Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Tesis Doctoral

Análisis y diseño de procesos de minería de datos astrofísicos sobre catálogos fotométricos múltiple época

Juan Bautista Cabral

Director: Dr. Pablo Granitto
Co-Director: Dr. Sebastián Gurovich

Tesis presentada en cumplimiento parcial para optar por el título de Doctor en Informática

Fecha: Marzo 2019

Certifico que el trabajo incluido en esta tesis es el resultado de tareas de investigación originales y que no ha sido presentado para optar a un título de posgrado en ninguna otra Universidad o Institución.

Juan Bautista Cabral

Reconocimientos

Esta Tesis fue desarrollada gracias a la ayuda económica del Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) mediante la Beca de Finalización de Doctorado (abril 2017 - abril 2019).

Agradecimientos

El primer lugar de mis agradecimientos corresponde a Pablo Granitto y Sebastián Gurovich, mis directores. Sus aportes y conocimientos fueron la fuerza principal detrás de la realización de este trabajo.

También, agradezco, a todo el Instituto de Astronomía Teórico Experimental que me brindo el mejor ambiente para desarrollar mi formación, y a la *FCEIA* y su departamento de postgrado, que han posibilitado la existencia de la carrera en una universidad pública y gratuita.

Quisiera agradecer fuertemente la desinteresada colaboración de Bruno, El Chuti, Mariano, Felipe, Schmidt, Marcelo, Guillermo Grinblat, David, Lucas Uzal por los consejos, discusiones, opiniones y trabajo aportado que elevaron la calidad de esta tesis.

No quiero dejar de mencionar a los revisores de las publicaciones que conforman el contenido de esta tesis; los cuales se han convertido en autores anónimos de muchas mejoras en el conocimiento obtenido y volcado en este trabajo.

Mención especial a mis compañeros de almuerzo de todos los días en "la cocina del IATE": Julián, Mario, Hernán, Horacio, Valeria, Ornela, Valotto, Víctor, Carlos, Dante, Mariana, Manuel, Anny, Viviana, Dario, Vicky, otra Vicky, el Polaco, Andrea, Yami, Anita O'mill, Cristiani, Rubén y Charly.

Va también mi agradeciendo a los que dieron el soporte emocional en estos años de trabajo: a mi familia directa: Nadia, Ada, Arqui, Mamá, Edi, Alicia y Alexis; Mis amigos de toda la vida: Anita, Salva, Mauri J, Gonza, Diego, Juan–Mochila, mi ahijado Mauricio, mi compadre Alejando; a Alejandro Garcia, Pablo Duboue, Xevi; y a toda la XVIII del Liceo Storni.

Finalizo agradeciendo al **Contador**: Aquiles Alberto Cabral, mi papá. Lamentablemente no pudo acompañarme; ya que se nos fue durante las etapas finales de mi tesis.

Resumen

El desarrollo de modernos telescopios terrestres y satelitales ha impulsado la realización de grandes relevamientos astronómicos, los cuales a su vez han generando un crecimiento gigantesco en la cantidad y calidad de datos a ser procesados, almacenados y analizados. Ante esta situación las técnicas de minería de datos y aprendizaje automático han empezado a jugar un rol importante en el resumen y presentación de la información para astrónomos.

La presente tesis tiene como objetivo la clasificación de fuentes astronómicas y detección de errores observacionales en el moderno relevamiento astronómico Vista Variables in the Via Lactea (VVV), comenzando por introducir conceptos elementales, sobre astronomía, aprendizaje automatizado y minería de datos que serán utilizados frecuentemente durante todo el trabajo. A continuación, se presenta una exposición de los datos del relevamiento astronómico VVV junto con la descripción del pre-procesamiento necesario para la utilización de estas técnicas automáticas; seguido de la presentación del diseño teórico e implementación de dos herramientas necesarias para el procesamiento de datos descriptos en el capítulo anterior. Los dos capítulos siguientes encaran primero la problemática de la clasificación de fuentes astronómicas en general y estrellas variables en particular frente al gran desbalance de observaciones existente en relevamiento; mientras que el segundo se enfoca en la detección automática de errores observacionales dentro los datos relevados.

Finalmente, se exponen las conclusiones y se discuten ideas de trabajo a futuro para continuar con las líneas de estudio dentro del VVV, así como futuros relevamientos astronómicos.

Abstract

The development of modern earth-based and space-born telescopes has prompted the realization of large astronomical surveys that in turn have not only generated a gigantic growth in the quantity and quality of data needed to be processed, stored but also analyzed. In this context data mining and machine learning tools growingly play important roles in the analysis and presentation of astrophysical information.

This thesis deals with the classification of astronomical sources and the detection and treatment of observational errors for the astronomical survey. It begins introducing the tools machine-learning and data mining. Both are used frequently throughout the work. The nature of the Vista Variables in the Via Lactea (VVV) survey data is then detailed, together with an explanation of the pre-processing required to use this data with automatic techniques; followed by the presentation of the theorical design and implementation of two tools used for data processing, as described in the previous chapter. The next two chapters deal, first, with the problem of unbalanced observations in the context of the classification of astronomical sources in general and variable stars in particular; while the second one focuses on the automatic detection of observational errors within the surveyed data.

Finally, conclusions are presented and ideas for future work discussed to continue with this line of research within the VVV as well within future astronomical surveys.

Índice general

Reconocimientos **I**

Agradecimientos **I**

Resumen **III**

Abstract **IV**

Glosario **IV**

Índice de figuras **IX**

Índice de tablas **XV**

1. Introducción **2**
1.1. Alcances y Objetivos 3
1.1.1. Aplicaciones 4
1.2. Resultados Originales Presentados 4
1.3. Organización de la tesis 5

2. Estado del Arte **7**
2.1. Objetivo del capítulo 7
2.2. La astronomía como ciencia de datos. 7
2.2.1. El surgimiento de la Astro-estadística y la Astro-informática 9
2.3. Aprendizaje automático en el contexto de la búsqueda de conocimiento en datos 10
2.4. Cómo aprenden las máquinas 11
2.5. Diferentes tipos de aprendizaje 12
2.6. Medición de errores 13
2.6.1. Curva *ROC* 14
2.7. Consideraciones Importantes 14
2.7.1. Sobreajuste 14
2.7.2. Desbalance de clases 16
2.7.3. Selección de características 16
2.8. Métodos de aprendizaje automático 17
2.8.1. Árboles de decisión 17
2.8.2. Random Forest 18
2.8.3. Máquinas de vectores de soporte (SVM) 19
2.8.4. KNN 20

2.9. Contribuciones del aprendizaje automatizado a la astronomía . . 21
2.9.1. *Palomar Transient Factory* (PTF) 21
2.9.2. Simulaciones cosmológicas 21
2.9.3. Detección de ondas gravitacionales en tiempo real 22
2.9.4. Aprendizaje automático en el *VVV* 24
2.10. Conclusiones del capítulo 25

3. Extracción de características 26
3.1. Objetivos del capítulo 26
3.2. El relevamiento "*Vista Variables in the Via Lactea*" 26
3.3. Catálogos fotométricos múltiple-época 28
3.4. Reconstrucción de la serie temporal para la generación de características 30
3.4.1. Emparejamiento por proximidad (*Cross-Matching*) 30
3.4.2. Corrección de Fechas 31
3.4.3. Período 31
3.5. Generación de los conjuntos de datos 31
3.6. Características 33
3.6.1. Características extraídas con *feets* 34
3.6.2. Características propuestas en este trabajo 36
3.7. Conclusiones del capítulo 40

4. Infraestructura 41
4.1. Objetivos del capítulo 41
4.2. Corral - Marco de trabajo para el procesamiento secuencial de datos 41
4.2.1. Pipelines astronómicos 43
4.2.2. Frameworks 44
4.2.3. Resultados: Un *framework Python* para implementar *pipelines* reproducibles basado en Modelos, Etapas y Alertas 46
4.2.4. Ejemplo simple de pipeline para convertir coordenadas (x, y) en ecuatoriales (RA, Dec) 51
4.3. feets - Extractor de Características de Series Temporales 57
4.3.1. Ingeniería de características 58
4.3.2. FATS 58
4.3.3. Ventajas y desventajas de FATS 61
4.3.4. Hacia la mejora en FATS 69
4.3.5. Resultado: Extractor de características para series temporales (*feets*) 70
4.4. Carpyncho: Pipeline para procesamiento de datos del *VVV* . . . 72
4.4.1. Modelos 72
4.4.2. Carpyncho Loader y Steps 74
4.5. Conclusiones y trabajo a futuro del capítulo 78

5. Generación de catálogos 79
5.1. Objetivos del capítulo 79
5.2. Las estrellas *RR-Lyrae* 79
5.3. Diseño experimental 80
5.4. Pre–Procesado 81
5.5. Comparación de clasificadores 82

5.6. Generalización 84
5.7. Balance de clases 86
5.8. Agregado de *OGLE-IV* y *VizieR* 88
5.9. Predicción de nuevos *tiles* 88
5.10. Conclusiones y trabajo a futuro 91

6. Detección de ruido experimental **93**
6.1. Objetivos del capítulo 93
6.2. Susceptibilidades al ruido observacional en las observaciones del VVV 93
6.3. Muestreo 94
6.4. Selección del modelo 94
6.5. Selección de características 95
6.6. Generalización 98
6.7. Conclusiones y trabajo a futuro 98

7. Conclusiones **101**

A. Guía rápida de Corral **104**
A.1. Instalación y configuración 104
A.2. Procesando los datos de IRIS 106
A.3. *Pipeline* de Exoplanetas 110

B. Carpyncho QA **117**
B.1. Carpyncho Pipeline Quality Report 117
B.1.1. Summary 117
B.1.2. About The Corral Quality Assurance Index (QAI) 117
B.1.3. About The Qualification 118
B.1.4. Full Output 118

C. Experimentos para la generación de catálogos **120**
C.1. *Notebooks* 120

D. Experimentos para la detección de ruido experimental **122**
D.1. *Notebooks* 122

Bibliografía **124**

Índice de figuras

1.1. Diagrama adaptado de Minniti et. al. 2010 que expone el número esperado de fenómenos astrofísicos que espera detectar VVV en sus catálogos. 3

2.1. Telescopios reflectores con diámetros mayores a 3 metros. El eje horizontal indica cual fue su año de construcción, y el vertical el tamaño de su espejo principal expresado en metros. A países con varios de esos telescopios se les asignó un código de color. Autor: Miket Wardos, Licencia CC-BY-SA 3.0 8
2.2. La brecha de datos: Crecimiento de datos vs. número de analistas. Adaptado del trabajo de Grossman et al. (2013). 9
2.3. Diagrama del proceso de Descubrimiento de conocimiento en bases de datos (KDD) adaptado del trabajo de Fayyad et al. (1996) 11
2.4. Curva Curva de característica operativa del receptor (ROC) de ejemplo con cinco clasificadores, además del clasificador trivial en línea punteada. La tasa de verdaderos positivos es otro nombre dado al *Recall*. 15
2.5. Polinomios de distintos grados aproximando datos ruidosos (Grieco, 2018). 16
2.6. Imagen ilustrativa de 3 hiper planos separando dos clases (puntos negros y puntos blancos) en dos dimensiones. Los puntos marcados como A y B son los vectores de soporte para H_3. 20
2.7. Ejemplo ilustrativo de como una transformación dada por una función kernel puede transformar ejemplos que no son linealmente separables a una dimensión mayor donde sí lo son. Adaptado del trabajo de Shia et al. (2017) 20
2.8. Fritz Zwicky mirando a través del telescopio Schmidt de 18 pulgadas (luego renombrado a Telescopio "Samuel Oschin"), alrededor de 1936. (Archivos de Caltech) 22
2.9. Campo de densidad de masa de la simulación "*Millenium-XXL*", centrado en el halo más masivo a tiempo presente ($z = 0$). Cada recuadro se aleja por un factor de 8 respecto al anterior; la longitud del lado varía desde $4{,}1Gpc$ hasta $8{,}1Mpc$. Todas estas imágenes son proyecciones de una rebanada $8Mpc$. Fuente `https://goo.gl/M1EbmY` 23

2.10. Foto aérea del interferómetro Virgo, se observan el edificio central, el edificio del Mode-Cleaner, los 3 km totales del brazo oeste y parte del principio del brazo norte (a la derecha). Los demás edificios son zonas de trabajo varias y la sala de control. Fuente https://www.ligo.caltech.edu 24

3.1. Telescopio VISTA (siglas en inglés de "*Visible and Infrared Survey Telescope for Astronomy*" o "Telescopio astronómico de rastreo de espectro visible e infrarrojo"). Crédito: ESO `https://www.eso.org` 27
3.2. Vista panorámica del Observatorio Paranal. Al frente se observa el telescopio VISTA y al fondo el sistema de cuatro telescopios del "*Very Large Telescope Project*" (VLT - literalmente "Telescopio Muy Grande"). Crédito: ESO `https://www.eso.org` 28
3.3. El telescopio VISTA observando. Crédito: ESO `https://www.eso.org` 29
3.4. Mapa de enrojecimiento de Schlegel adaptado de Minniti et al. (2010), mostrando el área del relevamiento en el cielo en coordenadas galácticas y las baldosas que conforman el VVV. En escala de grises está la densidad estelar proyectada de este mismo campo tomado del relevamiento astronómico "Two Micron All Sky Survey" (Skrutskie et al., 2006b). 29
3.5. A la izquierda el diagrama de los sensores infrarrojos de VIRCAM y a la derecha una foto de la VIRCAM. Adaptado de `http://www.vista.ac.uk` 30
3.6. Observaciones de magnitud de una estrella de tipo variable *RRLyrae* AB del trabajo "Bulge RR Lyrae stars in the VVV tile b201" (Gran et al., 2015) identificada con el ID `VVV J2703536.01-412829.4`. El eje X representa la fecha de medición y el Y la magnitud en orden inverso. 32
3.7. Observaciones de magnitud en fase de la misma estrella correspondiente a la figura 3.6. El eje X representa la fase de medición y el Y la magnitud en orden inverso. Por comodidad para la observación de periodicidad se presentan dos ciclos completos. . . . 32
3.8. En azul se observa la ubicación de *tiles* utilizados en el trabajo para *tiles* del Bulbo. 33
3.9. Mapa de extinción del bulbo galáctico utilizado por *BEAM* segun la ley de Cardelli et al. (1989), adaptado del trabajo de Gonzalez et al. (2012); donde: b y l son la latitud y longitud galáctica respectivamente, y A_{K_s} es la extinción en banda K_s. 37

3.10. Gráfico adaptado del trabajo de Angeloni et al. (2014), en el cual se presenta el objeto `858993759856` del *WFCAM Science Archive* (`http://wsa.roe.ac.uk/`) (Hambly et al., 2008; Cross et al., 2007, 2012) en bandas Z, J y K_s, las tres puestos en fase respecto a la banda K_s. El eje horizontal muestra dos fases para facilitar la percepción de la simétrica; mientras que el eje vertical ajusta las magnitudes de la banda Z y J respecto a la K_s, sumándoles una constante para facilitar su comparación. El objeto es una *RR-Lyrae* con un periodo de 0.51961138 días. Nótese como las amplitudes son más pronunciadas en bandas cercanas al óptico (Z). 38

4.1. Flujo de datos en la arquitectura tradicional de *pipelines* (izquierda) y el modelo de *pipeline* de *Corral* (derecha). En los modelos de *pipelines* clásicos, los datos se procesan secuencialmente, de un paso (*step*) por vez, con una eventual ramificación en paralelo; mientras que en el modelo propuesto en este trabajo las etapas son entidades independientes que interactúan por compartir un contenedor central de datos. En ambas arquitecturas se pueden extraer datos cuando se desee dada una serie de condiciones, y pueden ser compartidas en cualquier momento a los usuarios. . . 48
4.2. Caso donde todos las pruebas unitarias no-fallan con una cobertura de código total ideal. El eje horizontal son los errores de estilo entre 0 y 40 ($N_{SErr} \in [0, 40]$), y el vertical es la inversa de la penalización (γ^{-1}). Puede observarse una caída en la pendiente de error a medida que *tau* crece. 51
4.3. Esquema básico del flujo de trabajo para construir una pipeline utilizando Corral 52
4.4. Diagrama de clases de los modelos utilizados en Carpyncho para orquestar la transformación de datos, desde los catálogos hasta el resultado final en características utilizadas en aprendizaje automático. Este diagrama es auto generado por Corral. 75
4.5. Flujo de datos del *pipeline* Carpyncho. En azul se aprecia el *Loader* y en rojo el último paso que se encarga de extraer las características correspondientes de cada curva de luz. Cada nodo circular es un paso de Corral implementado como una clase python. 77

5.1. Curvas ROC de los clasificadores Bosques Aleatorios (RF), K vecinos más cercanos (KNN) y Máquinas de Vectores de Soporte (SVM) (con *kernels* polinómico, lineal y Función de base radial (RBF), para el experimento de entrenar y predecir sobre el *tile b278* utilizando 10 *K-folds*. Puede observarse cómo el clasificador RF mejora la eficiencia de clasificación respecto a KNN y los tres SVM, los cuales poseen curvas de formas similares y Área bajo la curva (AUC) equivalentes. 82

5.2. Curvas ROC resultantes de clasificar las fuentes de los *tiles* $b261$ y $b278$ sobre muestras de distinto tamaño. En el panel de la izquierda se encuentran las curvas calculadas utilizando al *tile* $b261$ como entrenamiento y el *b278* como prueba; mientras que en el panel de la izquierda los roles de entrenamiento y prueba de los *tiles* se invierten. En ambos paneles se presentan tres curvas, las cuales provienen de clasificar siempre la muestra de 20000 fuentes desconocidas (muestra grande) y entrenar con las muestras de 2500 (pequeña), 5000 (mediana) y 20000 (grande) fuentes. Los resultados demuestran un rendimiento similar en todos los casos. 87
5.3. Valores de *precision* (columna izquierda) y *recall* (columna derecha), para las clasificaciones en las muestras pequeña (fila superior), mediana (fila intermedia) y grande (fila inferior), para el punto de trabajo $P^* = 0{,}1$. Cada fila de las seis tablas corresponde al *tile* que se uso para entrenar y las columnas el que fue usado para probar; mientras que las diagonales principales contienen los valores obtenidos con *K-folds* sobre el mismo *tile* de entrenamiento. 90
5.4. Valores de máxima *precision* corregida P^* menor que 1 (columna izquierda) y *recall* (columna derecha), para las clasificaciones en las muestras pequeña. Cada una de las filas corresponde al *tile* que se usó para entrenar y las columnas e cual fue usado para probar; mientras que las diagonales principales contienen los valores obtenidos con *K-folds* sobre el mismo *tile* de entrenamiento. . . . 91

6.1. Curvas ROC de los clasificadores RF y SVM con *kernel* RBF, para el experimento de entrenar y predecir sobre el *tile* de una fuente dada sobre el conjunto de datos comprendido por las fuentes de los *tiles* $b278$ y $b261$. Puede observarse como el clasificador RF mejora la eficiencia de clasificación considerablemente respecto a SVM. 96
6.2. Subconjunto de características seleccionadas (eje horizontal) contra puntaje obtenido en promedio en la validación cruzada. En rojo se marca el máximo puntaje que corresponde al subconjunto de 15 características. 97
6.3. Comparación entre las curvas ROC de las clasificadores RF entrenados con las 57 características totales y con las 15 características seleccionadas con Eliminación recursiva de características (RFE), ambos entrenados son 10 *K-Folds* y utilizando el subconjunto de datos compuesto de los *tiles* $b278$ y $b261$. Se nota una leve mejoría en el comportamiento del clasificador entrenado con las características extraídas con RFE. 97

6.4. Resultados de las tres métricas de clasificación: *Precision* a la izquierda arriba, *recall* derecha arriba y AUC izquierda abajo; de utilizar las 15 características seleccionadas con RFE en todas las posibles combinaciones de *2-en-2* de todos los *tiles* disponibles. Además se acompaña una tabla de distancias entre *tiles* (derecha abajo). Cada celda de cada una de las tres primeras tablas contienen la métrica para la ejecución de una clasificación sobre un conjunto de datos compuesto únicamente por los *tiles* identificados en la fila y la columna, utilizando 10 *k-folds*. Puede observarse como las clasificaciones son consistentes con promedios de *precision* de 0,86, de *recall* de 0,87 y de AUC de 0,93. 99

A.1. Gráfico de masa (eje vertical) vs periodo (eje horizontal) de planetas encontrados en zona habitable por el *pipeline* de exo-planetas. El gráfico se presenta tal cual como es generado por la *alert*, es por esto que carece de etiquetas para los ejes. 115

Índice de tablas

2.1. Matriz de confusión para dos clases donde en las filas se muestras las clases reales, mientras que en las columnas las clases predichas. **TP** son los verdaderos positivos, **TN** los verdaderos negativos, mientras **FN** y **FP** son falsos positivos y falsos negativos (clasificaciones erróneas) 13

3.1. Cantidad de estrellas variables identificadas en cada *tile* bajo estudio en este trabajo. La columna **Tamaño** indica el total de fuentes, y **Variables** la cantidad de estrellas variables usando cortes pre-establecidos para identificarlas. 33

4.1. Análisis estadístico de los resultados de las características *AndersonDarling*, *StetsonJ*, y *StetsonK* calculadas por FATS, realizado con 100,000 curvas de luz Gaussianas generadas al azar. 69

5.1. Distribución de 1553 estrellas tipo *RR-Lyrae* identificadas con el catálogo de *OGLE-III*, sobre los *tiles b261*, *b262*, *b263*, *b264* y *b278*. Las columnas *%(G)*, *%(M)*, *%(P)* muestran el porcentaje de estas estrellas frente a las muestras de tamaño Grande (20 mil), Mediana (5 mil) y Pequeña (2500) respectivamente. Finalmente, la columna *T.Total* indica la cantidad total de fuentes presentes en esa baldosa, mientras que *T.Útil* indica cuántas de estas fuentes son útiles para nuestro análisis (esto es, fuentes con más de 30 observaciones y cuyas magnitud media no esté saturada ni sea muy tenue). 80

5.2. Características seleccionadas para la creación de catálogos de estrellas tipo *RR-Lyrae* sobre baldosas del VVV 81

5.3. Distribución final de las clases positivas (*RR-Lyrae*) frente las negativas (fuentes desconocidas) en las baldosas del VVV que son incluídas en este trabajo. 81

5.4. Tablas con los índices de errores de los clasificadores RF, KNN y SVM (con *kernels* polinómico (poli), lineal y RBF, para el experimento de entrenar y predecir sobre el *tile b278* utilizando 10 *K-folds* en la muestra **pequeña**. La columna Clase indica a que clase pertenecen los indicadores siendo FD la clase negativa de fuentes desconocidas, y RR la clase positiva o estrellas tipo "RR-Lyrae". La columna N indica la cantidad de observaciones de esa clase. Se puede observar de los resultados que las métricas de calidad para KNN y todos los SVM son similares, mientras que el RF mejora los otros índices para ambas clases. 83

5.5. Precision (Prec), Recall (Rec) y AUC para los clasificadores SVM con *kernels* lineal (*SVM-L*), polinómico (*SVM-P*) y RBF (*SVM-R*); RF y KNN para las muestras pequeña, mediana y grande, realizando 10 *K-Fold* sobre el *tile* $b278$ sólo para la clase de estrellas tipo $RR - Lyrae$. Puede observarse como los RF poseen métricas superiores a todos los demás clasificadores, los cuales entre sí tienen valores similares. 84

5.6. Precision (Prec), Recall (Rec) y AUC para RF sobre todos los puntos del experimento descrito en esta sección. La columna *Ent.* indica el *tile* (o conjunto de *tiles*) que se utilizó como entrenamiento para el experimento, y la columna *Prueba* indica con cuáles de ellos se realizó la prueba. 85

5.7. Precision (Prec), Recall (Rec) y AUC para RF entrenando primero en $b261$ y probando en $b278$; y entrenando en $b278$ y probando en $b261$ en segundo lugar. Para ambos casos las comparaciones radican en entrenar en muestras de tamaño 2500 (pequeña), 5000 (mediana) y 20000 (grande); y realizando las pruebas en el *tile* de la muestra de tamaño 20 mil (grande). Nótese que si bien el *recall* se mantiene en valores similares, la *precision* sobre la muestra disminuye a medida que se entrena con menor cantidad de fuentes desconocidas. 86

5.8. Resumen de estrellas tipo *RR-Lyrae* encontrados en los *tiles* utilizados en este estudio utilizando los catálogos provenientes de los relevamientos *OGLE-III*, *OGLE-IV* y la colección *VizieR*, en comparación con las fuentes encontradas al utilizar solamente el catálogo del relevamiento *OGLE-III*. La columna *T.Total* indica la cantidad de fuentes en el *tile*, *T.Útil* indica cuántas de estas fuentes son útiles para este análisis (esto es fuentes con más de 30 observaciones y cuya magnitud media no esté ni saturada ni sea muy tenue), *T. Catálogos* muestra cuántas estrellas tipo *RR-Lyrae* fueron identificadas con los tres catálogos; y finalmente la columna *OGLE-III* identifica cuántas estrellas tipo *RR-Lyrae* fueron detectadas solo con el catálogo *OGLE-III*. Se puede observar como incluso *tiles* extremos del relevamiento, como el $b396$, ahora poseen *RR-Lyrae* conocidas, mientras que los que ya poseían observaciones de este tipos de estrellas las han incrementado. 89

6.1. Cantidad de fuentes observadas por *tile*. 95

6.2. Métricas de clasificación de los modelos SVM con *kernel* RBF y RF sobre el conjunto de datos comprendido por las fuentes de los *tiles* $b278$ y $b261$. Puede observarse la superioridad de RF en todas las medidas. 95
6.3. Sub-conjunto de 15 características seleccionadas por el RFE con *k-fold*. En negrita las características creadas en particular para este trabajo, mientras que las demás son las provistas por *feets*. . 96
6.4. Métricas de clasificación del subconjunto de datos compuesto de los *tiles* $b278$ y $b261$, utilizando el clasificador RF con la colección total de 57 (RF) características, y solo con las 15 características seleccionadas con RFE. En ambos casos se usaron 10 *K-Folds* para obtener estos resultados. Puede observarse una muy leve mejora en el resultado obtenido con el subconjunto seleccionado con RFE. 96
6.5. Mediana de métricas de clasificación, según las distancias de *Manhattan*, entre la grilla *tiles* del bulbo galáctico (ver figura *3.8*). Los valores de distancias que fueron elegidos para este resumen son los que poseen suficiente frecuencia para tener una mediana confiable. Puede observarse como la mediana de las métricas mejora a medida que la distancia crece. 98

Capítulo 1

Introducción

El desarrollo de modernos telescopios y satélites ha impulsado la realización de grandes relevamientos astronómicos, los cuales a su vez han generando un crecimiento gigantesco en la cantidad y calidad de datos a ser procesados, almacenados y analizados. Actualmente se destaca el ya finalizado el relevamiento llamado *VVV* (Minniti et al., 2010) y su continuación el *VVV eXtended Survey (VVVx)* (Minniti, 2018), cuyo ambicioso objetivo es producir un mapa tridimensional de gran parte del centro galáctico de la Vía Láctea y de una parte del Disco Galáctico interno. Este mapa tridimensional esta siendo obtenido a partir de un censo de estrellas, posibilitado gracias a un monitoreo sistemático de estas regiones de la Vía Láctea; el cual ya completo para el caso del *VVV* un total de 1.929 horas de observación realizadas durante un período de 5 años (iniciados en el 2010). Para ello, el equipo de astrónomos a cargo del proyecto *VVV/VVVx* está utilizando el moderno telescopio VISTA, ubicado en Cerro Paranal, II Región, Chile. Actualmente el proyecto está generando 300 GB de datos por noche. Es de suma importancia hacer notar que esta información se hace pública a través de los archivos del Observatorio Europeo Austral (ESO) luego de pasado el "tiempo propietario", es decir, al año de haber sido realizada la observación. Esto permitirá que todos los interesados, ya sean instituciones o personas, profesionales o amateurs, puedan hacer uso de estos datos.

Los datos de estos relevamientos se presentan en una unidad llamada "*tile*" (baldosa en inglés), la cual es una zona rectangular del cielo relevada a través del tiempo. Por cada relevamiento de cada baldosa se generan varias imágenes en alta resolución para diferentes tipos de filtros de frecuencias lumínicas. Asimismo, por cada imagen existe una base de datos numérica con los valores de posición, magnitud y color de las fuentes de luz presentes en la imagen, llamada "catálogo fotométrico".

Una vez obtenidas las imágenes y los catálogos, los astrónomos las utilizan para variados análisis, entre los que se encuentran: búsqueda de exo-planetas (planetas fuera del sistema solar), búsqueda de galaxias de fondo, búsqueda de anomalías en las galaxias detectadas que puede llevar al descubrimiento de super-novas o galaxias de núcleo activo. Además de esto, es de interés astronómico la generación de catálogos de cualquier tipo de objeto que posea el relevamiento, ya que el *VVV* en su barrido total de la zona bajo estudio identificará como fuentes de luz además de planetas y galaxias un número bastante amplío (Ver figura 1.1) de diferentes fenómenos.

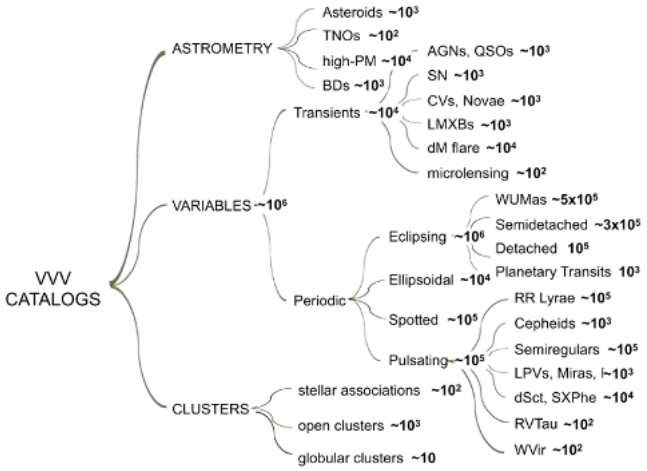

Figura 1.1: Diagrama adaptado de Minniti et. al. 2010 que expone el número esperado de fenómenos astrofísicos que espera detectar VVV en sus catálogos.

Dada esta masividad de datos, las técnicas de Minería de datos (DM) y Aprendizaje automático (ML) sobre catálogos fotométricos astronómicos adquieren un valor de piedra angular para el resumen y presentación de la información para astrónomos (Cavuoti, 2013).

Finalmente a partir del año 2020 se pondrá en marcha el proyecto *Large Synoptic Sky Telescope (LSST)*, el cual censará el cielo cada tres noches, generando aproximadamente 3 GB de datos astronómicos por cada segundo de observación. Esto generará una base de datos de varios Petabytes (Ivezic et al., 2008).

Esta situación provee la base y motivación para el desarrollo del presente trabajo.

1.1. Alcances y Objetivos

Esta tesis propone el estudio de la especialización o creación de técnicas de Aprendizaje Automático ML para la extracción y validación de conocimiento de relevamientos astronómicos provenientes de catálogos astrofísicos múltiple época; así como el diseño de procesos de integración de las mismas, para lograr estandarización, estabilidad y optimización de recursos. El objeto principal de estudio consiste en la clasificación y generación de catálogos de estrellas variables periódicas tipo *RR-Lyrae* en conjunto con la detección de errores observacionales en los datos. La importancia astronómica de este tipo de estrellas radica en que son estrellas pulsantes, viejas de población II y por lo tanto excelentes predictores de distancia (candelas-estándar). Esta característica las hace sumamente importante en un relevamiento que busca mapear el centro galáctico en 3D.

Además, se propone la producción de bases de datos de características de fuentes en el infrarrojo cercano; catálogos de estrellas tipo *RR-Lyrae* sobre baldosas del relevamiento *VVV*; así como publicación de un pipeline de extracción de características que soporte la magnitud de datos de un relevamiento como el

estudiado. Todo esto será útil para futuros trabajos tanto en la comunidad de aprendizaje automático como la de astronomía.

Finalmente todos los resultados serán validados contra catálogos ya existentes de estrellas variables, que solapan el área de relevamiento de *VVV/* pero utilizan la banda óptica de observación.

1.1.1. Aplicaciones

El trabajo desarrollado posee diversas aplicaciones en el campo del la astronomía, y el aprendizaje automático. Entre ellos, pueden mencionarse:

Mapeo 3D del núcleo y el bulbo galáctico: Como se menciono anteriormente, dado que las estrellas bajo estudio (*RR-Lyrae*) se utilizan para calcular distancias; son útiles para le creación de un mapa en 3D de la zona de observación.

Detección de galaxias de fondo: En la misma línea que el ejemplo anterior. Una de las técnicas de búsqueda de galaxias de fondo, es justamente buscar *clusters* de estrellas *RR-Lyrae*. Si todas estas estrellas se encuentran a una distancia mayor para ser contenidas dentro de la galaxia es probable que se encuentren en una galaxia de fondo.

Análisis de series temporales estacionarias: Todas las técnicas de extracción de características, estudiadas e implementadas en esta tesis, son aplicables a priori a cualquier estudio relacionado con otro tipo de series temporales estacionarias.

1.2. Resultados Originales Presentados

En la obtención de los mencionados objetivos, se tomaron en consideración métodos de identificación de estrellas variables ya existentes en astronomía, así como el uso de técnicas y algoritmos para la reconstrucción de curvas de luz. En muchos de los casos estos algoritmos estaban disponibles pero no se encontraban listos para operar con la eficiencia requerida en un ambiente de tantos datos; por lo cual debieron ser oportunamente re-diseñados o reescritos en su totalidad.

Las contribuciones más preponderantes, de manera resumida, incluyen:

- El diseño teórico e implementación de un ambiente de procesamiento de datos en *pipeline* basados en el patrón Modelo-Vista-Controlador apto para el procesamiento en etapas de grandes volúmenes de datos.
- La re-escritura, documentación y extensión de los cambios a la herramienta de facto para la extracción de características de series temporales estacionarias.
- Un conjunto de datos con los características más importantes para la clasificación de estrellas variables dentro del VVV (Este conjunto al momento de la redacción de esta tesis incluye ~ 8 millones de observaciones).
- Catálogo de estrellas variables tipo *RR-Lyrae* para tiles del *VVV*.

- Un método de para detectar cuáles son las características independientes del error instrumental.

Los resultados de la investigación de la presente tesis doctoral fueron presentados en las siguientes publicaciones Cabral et al. (2014, 2016a,b, 2017, 2018b).

Revistas Internacionales

- Cabral, J. B., Sánchez, B., Beroiz, M., Domínguez, M., Lares, M., Gurovich, S., & Granitto, P. (2017). **Corral framework: Trustworthy and fully functional data intensive parallel astronomical pipelines**. Astronomy and Computing, 20, 140-154.
- Cabral, J. B., Sánchez, B., Ramos, F., Gurovich, S., Granitto, P., & Vanderplas, J. (2018). **From FATS to feets: Further improvements to an astronomical feature extraction tool based on machine learning**. Astronomy and Computing.

Conferencias

- Cabral, J. B., Granitto, P. M., Gurovich, S., & Minniti, D. (2016, November). **Generación de *features* en la búsqueda de estrellas variables en el relevamiento astronómico VVV.** In Simposio Argentino de Inteligencia Artificial (ASAI 2016)-JAIIO 45 (Tres de Febrero, 2016).

Pósters y Resúmenes

- Cabral, J; Gurovich, S.; Good, J; Medel, R; Garcia Lambas, D; Amôres, E.; Clariá, J.; Minniti, D., ***Community Science with the VVV*** IAU Latin American Regional Meeting Lugar: Florianopolis; Año: 2013;
- Juan Bautista Cabral; Sebastián Gurovich; Felipe Gran; Dante Minniti **Carpyncho The VVV band-merged catalogue and data mining/machine learning facility** 7th VVV Science Workshop, Año: 2016;

Herramientas

- Cabral, J., Sanchez, B., Beroiz, M., Dominguez, M., Lares, M., Gurovich, S., & Granitto, P. (2018). **CPF: Corral Pipeline Framework**. Astrophysics Source Code Library.
- Cabral, J., Sanchez, B., Ramos, F., Gurovich, S., Granitto, P., & VanderPlas, J. (2018). **feets: feATURE eXTRACTOR for tIME sERIES**. Astrophysics Source Code Library.

1.3. Organización de la tesis

Los capítulos de este trabajo se organizan de la siguiente manera: en el Capítulo 2 se describe el estado del arte en técnicas de ML. El capítulo 3 aborda la reconstrucción de las series temporales del relevamiento *VVV* y la correspondiente extracción de características de dichas series; continuando con

el Capítulo 4 el cual hace hincapié en todo lo relacionado a la infraestructura de análisis de datos desde el punto de vista de software y hardware. Los capítulos 5 y 6 introducen el uso de las características calculadas para la generación de catálogos de estrellas variables tipo *RR-Lyrae* en el *VVV* a través de técnicas de ML tradicional; y la detección de cuales son las características más sensibles al error instrumental. Finalmente el Capítulo 7 explica las conclusiones y los trabajos a futuro.

Capítulo 2

Aprendizaje automático en general y en astronomía en particular

2.1. Objetivo del capítulo

En este capítulo se presenta el estado del arte de la ciencia de datos en astronomía y cómo se ha integrado con la informática en general y el aprendizaje automatizado en particular.

La primera sección hace referencia al contenido astronómico que está involucrado en este trabajo, realizando un repaso de la astronomía, que se ha vuelto una ciencia de datos intensivos ya hace más de 40 años (después de la invención y desarrollo del CCD), lo cual llevo al surgimiento de las sub-disciplinas conocidas como *Astro-Estadística* y *Astro-Informática*.

A continuación se dedica unos párrafos al aprendizaje automatizado repasando su teoría, algoritmos más comunes, métricas de calidad del rendimiento de esos algoritmos y una visión global sobre el proceso de descubrimiento de conocimiento en bases de datos. Finalmente, se presenta una colección de casos de aplicación del minería de datos en diferentes ramas de la astronomía.

2.2. La astronomía como ciencia de datos.

Si se observa la Figura 2.1, es evidente que el campo de la astronomía se encuentra en una carrera para construir los telescopios más grandes, los cuales hacen posible extraer cada vez más y mejores datos. Cabe aclarar que aunque el tamaño del telescopio es un factor diferenciador al momento de extraer datos, esto no es lo único a tener en cuenta si no también el campo de visión. Telescopios espaciales corren con ventaja al no tener que lidiar con problemas atmosféricos, así como los infrarrojos ganan en profundidad (como el utilizado en este trabajo, que será explicado con más detalle en Capítulo 3)

Este hecho no debe ser ignorado ya que implica un cambio radical en la forma de investigación en toda ciencia, como lo expone el libro seminal “El cuarto paradigma” (Hey et al., 2009).

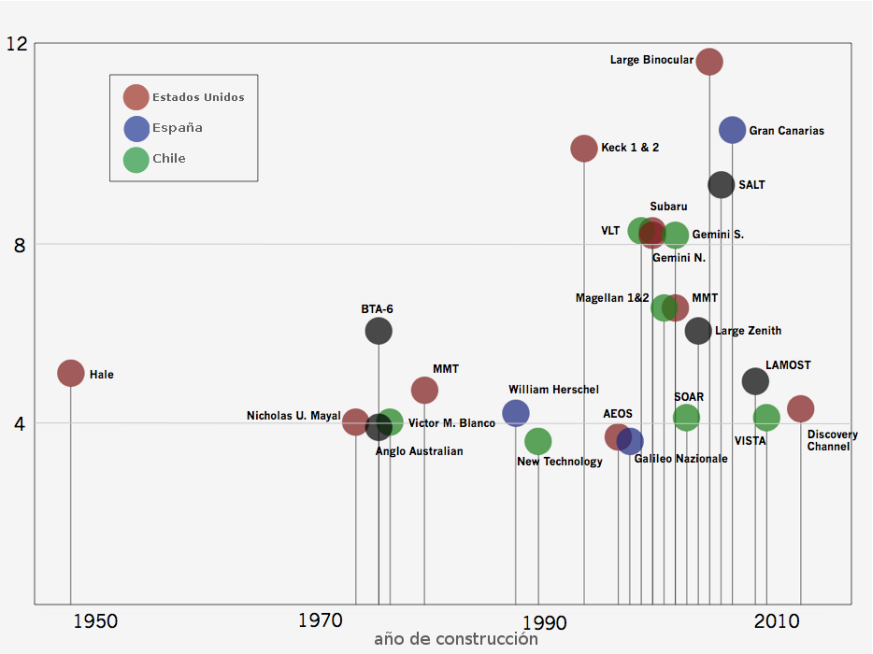

Figura 2.1: Telescopios reflectores con diámetros mayores a 3 metros. El eje horizontal indica cual fue su año de construcción, y el vertical el tamaño de su espejo principal expresado en metros. A países con varios de esos telescopios se les asignó un código de color. Autor: Miket Wardos, Licencia CC-BY-SA 3.0

"El cuarto paradigma" plantea un nuevo punto de vista independiente en la ciencia moderna, ya que cuando el volumen de datos supera la capacidad humana para analizar y evaluar cada punto independientemente, el científico debe confiar más y más en metodologías automáticas computacionales capaces de encontrar patrones o casos particulares en una dimensionalidad alta.

La otra diferencia es que en este paradigma, en vez de elegir una hipótesis y recolectar datos para probarla, ahora los datos se recolectan continuamente y luego se evalúa qué se puede probar con ellos; en otras palabras ahora el suceso de una investigación radica en la capacidad de minar conocimiento desde los datos (Karpatne et al., 2017).

Es decir, los nuevos astrónomos están siendo obligados a familiarizarse con términos como: aprendizaje automático, reconocimiento de patrones, minería de datos o reducción de dimensionalidad; todas ellas técnicas que han sido agrupadas en el campo de la *Astro-informática* (Borne, 2010).

La astro-informática, no sólo responde a una necesidad de crecimiento exponencial en datos astronómicos, sino que además se suma el problema de que esta tendencia no es seguida por la cantidad de nuevos analistas para esos datos. En la Figura 2.2 puede observarse que mientras los datos crecen exponencialmente el número de analistas se mantiene virtualmente constante.

Este es el contexto actual de la astronomía, en el cual trabajos antiguamente manuales como la identificación de un tipo de estrella o separar galaxias de

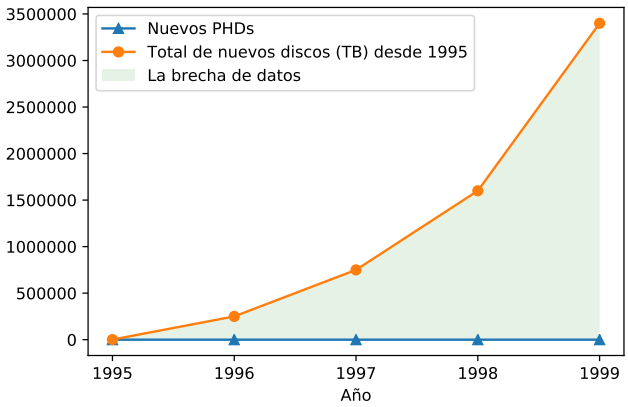

Figura 2.2: La brecha de datos: Crecimiento de datos vs. número de analistas. Adaptado del trabajo de Grossman et al. (2013).

otras fuentes se está apoyando firmemente en disciplinas como el aprendizaje automático.

2.2.1. El surgimiento de la Astro-estadística y la Astro-informática

Hasta hace un tiempo cercano la astronomía, como todas las otras ciencias, se dividía en ramas según su objeto de estudio, como la astronomía estelar (que estudia estrellas en particular), galáctica, sistemas planetarios, plasmas astrofísicos, astronomía instrumental (encargada históricamente del diseño de telescopios, actualmente también involucra el software de procesamiento de datos) y cosmología; además se le ha sumado la astro estadística como sub-disciplina.

En el siglo XIX, los astrónomos y los estadísticos eran las mismas personas: Legendre, Laplace y Gauss desarrollaron el método de mínimos cuadrados basados en los errores de la distribución normal, para resolver problemas de mecánica celeste. La separación entre ambos grupos de profesionales es un fenómeno que se ha dado recién en el siglo XX, e hizo que los astrónomos se hayan perdido métodos más sofisticados como estimación con *maximum-likelihood*, modelos estocásticos auto-regresivos para variaciones aperiódicas en series temporales, re-muestro *bootstraping* o inferencia bayesiana entre otros. Anteriormente, la práctica de estas técnicas estaban basadas en asunciones restrictivas, las cuales se han relajado por el desarrollo de técnicas computacionales intensivas y complejas, utilizadas en los mega-conjuntos de datos actuales. En este sentido, la astro-estadística ha creado una relación cercana con la astro-informática. (Feigelson, 2012).

Hay que tener en cuenta que la astro-informática, no está sola. El ya mencionado cuarto paradigma, está generando un gran conjunto de sub-disciplinas informáticas, que están lidiando con organizar, acceder, integrar y extraer información de un volumen de datos creciente, para obtener algún tipo de soporte de decisión (Fox, 2011).

En nuestro caso, la astro-informática es un nuevo modo de hacer astronomía que se ha habilitado gracias a que hoy disponemos de bases de datos, observatorios virtuales, cómputo distribuido y de alto desempeño, sistemas de descubrimientos inteligente y formas de visualización innovadoras; todo en un ambiente de relevamientos astronómicos que está yendo de giga-bytes a cientos de peta-bytes. Es así como la astro-informática como nueva sub-disciplina, está enfrentando el desafío de ser desarrollada, promocionada y reconocida en pleno derecho en los programas académicos por sus propios méritos. Justamente, es que deben disponerse de recursos significativos (medidos en términos de horas de científicos, programas educativos y fondos) en orden de crear y aplicar algoritmos de ciencia de datos específicos para la astronomía (Borne, 2012).

2.3. Aprendizaje automático en el contexto de la búsqueda de conocimiento en datos

El conocimiento proviene de tres fuentes diferentes: La información embebida en nuestra carga genética producto de la evolución; la experiencia que hizo exitoso a los mamíferos y, finalmente, la cultura que nos ha llevado a ser la especie dominante en el planeta (Domingos, 2015). La actualidad nos ha brindado las computadoras como nuevas fuentes de conocimiento al punto que como dice *Yann LeCun*, Director de investigación en inteligencia artificial de Facebook[1]:

> En el futuro, la mayor parte del conocimiento del mundo será extraído y almacenado dentro de máquinas.

Este es el contexto en el cual la inteligencia artificial se hace útil para que las máquinas auto-analicen sus propios conjuntos de datos. Dentro de las diferentes ramas que integran a la disciplina, la que se mantiene en vigencia postula que la forma de imitar la inteligencia humana por parte de una máquina se puede lograr a través del aprendizaje. Esto es brindar ejemplos de los datos para que un programa entienda su forma, y pueda generalizar el comportamiento para una tarea dada.

En nuestro caso además estamos interesados en el proceso completo de extracción de conocimiento de una colección dada de datos; este proceso es conocido como "Descubrimiento de conocimiento en bases de datos" (KDD). Tiene su origen en el método científico mismo, pero fue establecido como disciplina para las modernas bases de datos y sus volúmenes en el método descripto por el trabajo de Fayyad et al. (1996). KDD consiste en pasos sucesivos que van reduciendo y ordenando los datos para la aplicación de diferentes técnicas (manuales y automáticas) con el objetivo de facilitar el descubrimiento de nueva información (ver Figura 2.3).

Todos los capítulos que se incluyen en esta tesis son guiados por este proceso, así los Capítulos 3 y 4 describen los datos objetivos, el procesamiento y sus transformaciones; mientras que los Capítulos 5 y 6 se encargan de la parte de minería de datos, interpretación y evaluación de la información extraída.

[1] `http://facebook.com`

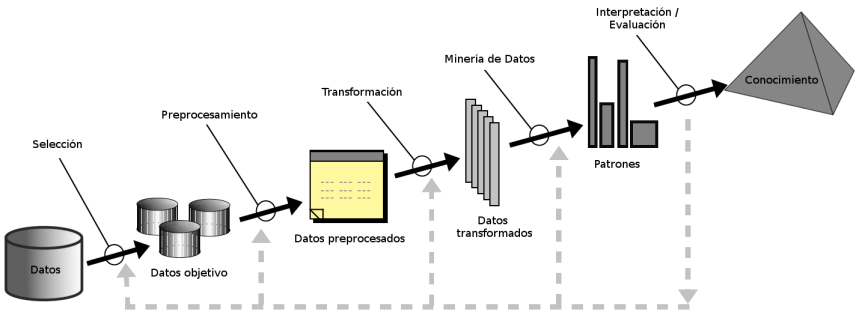

Figura 2.3: Diagrama del proceso de KDD adaptado del trabajo de Fayyad et al. (1996)

2.4. Cómo aprenden las máquinas

Formalmente Tom Mitchell en su libro "*Machine Learning*" (Mitchell et al., 1997) define al aprendizaje automático como:

> Se dice que un programa de computadora aprende de la experiencia E respecto a una tarea T y una medida de desempeño P, si el desempeño medido con P en una tarea T, mejora con la experiencia E.

Por ejemplo, un programa que aprenda a ganar Damas puede mejorar su habilidad para ganar el juego, a través de la experiencia de jugar contra sí mismo. En general, para tener un problema de aprendizaje automático bien definido es necesario identificar: las tareas T, la medidas de desempeño P y la experiencia E.

En el ejemplo de jugar Damas esto sería:

- **Tarea T:** En nuestro caso "jugar damas".
- **Medida de desempeño P:** Porcentaje de victoria frente a oponentes.
- **Experiencia E:** Jugar juegos de práctica contra sí mismo.

En el caso de la astronomía podemos imaginar un programa que busca galaxias en una imagen del cielo:

- **Tarea T:** Identificar galaxias.
- **Medida de desempeño P:** Porcentaje de identificaciones correctas.
- **Experiencia E:** Tratar de identificar galaxias en una sección del cielo en la cual se conozcan las galaxias a priori.

En resumen, el ML es un mecanismo inductivo de búsqueda de conocimiento, por lo tanto busca información general partiendo de observaciones particulares. Así, este mecanismo limita sus hipótesis dentro de este conocimiento particular para tener una base razonable para generalizar, esta limitación es llamada "**Sesgo inductivo**".

Finalmente el éxito de un método de ML está dado por qué tan correcta resultan estas hipótesis, lo cual discutimos más adelante en la Sección 2.6, y por el grado de entendimiento de la hipótesis inducida. En la práctica, muchas veces el entendimiento es mucho más importante que la exactitud, esto lleva a que siempre se prefieran hipótesis simples frente a las complejas; esta preferencia es llamada "*Navaja de Ockham*"[2]

2.5. Diferentes tipos de aprendizaje

Todo aprendizaje automatizado tienen algo en común: necesitan datos de entrenamiento de donde aprender. Sin embargo diferentes tipos de tarea T, experiencia E o desempeño P generan diferentes especializaciones. La primera clasificación más popular es aquella que divide a los algoritmos de aprendizaje automatizado según su tipo de supervisión:

Aprendizaje Supervisado Utiliza información precisa que usa a modo de etiqueta durante todo el aprendizaje. Intenta obtener una aproximación razonable de la función $F(X) = Y$, donde X son los datos de entrada e Y sus etiquetas. Esta aproximación puede realizarse a una serie de etiquetas discretas finitas, o a algún valor real tratando de predecir alguna magnitud.

En el caso de intentar aproximar a etiquetas discretas, el problema se llama **Clasificación**, y equivale por ejemplo a predecir si una observación de un telescopio de un evento transitorio fue real o simplemente un error de medición (Bloom et al., 2012), o si una fuente es una estrella o una galaxia (Kim and Brunner, 2016).

Por otra parte, la búsqueda de una aproximación a un valor continuo es llamada **Regresión**, como puede ser dado características de una propiedad, determinar su precio (Harrison Jr and Rubinfeld, 1978).

Aprendizaje No-Supervisado Acumula conocimiento durante la aplicación de la tarea, sin utilizar etiquetas. Tareas de este tipo pueden ser la reducción de la dimensionalidad, la visualización de datos (conservando propiedades intrínsecas), o el agrupamiento en grupos con características similares. Un ejemplo sencillo es agrupar las galaxias según su forma (Ordovás-Pascual and Almeida, 2014).

Sin importar el tipo de supervisión, todo aprendizaje automático tiene una fase de entrenamiento y una de predicción. La fase de entrenamiento se basa en los datos que le son suministrados a los modelos para ajustar sus parámetros internos. Asimismo, hay varios modelos que poseen hiper-parámetros que suelen configurar previamente a la fase de entrenamiento (determinan número de parámetros a aprender y tolerancia a errores, entre otras cosas). Luego, en la fase de predicción, los modelos pueden usarse para predecir alguna información de datos desconocidos.

[2]El principio metodológico conocido como *Navaja de Ockham*, atribuido al fraile franciscano Guillermo de Ockham (1280–1349), dice: *En igualdad de condiciones, la explicación más sencilla suele ser la más probable.* Así si dos teorías en igualdad de condiciones tienen las mismas consecuencias, es más probable que la teoría más simple sea la correcta.

	Clase Predicha	
Clase Real	**Positiva**	**Negativa**
Positiva	TP	FN
Negativa	FP	TN

Tabla 2.1: Matriz de confusión para dos clases donde en las filas se muestras las clases reales, mientras que en las columnas las clases predichas. **TP** son los verdaderos positivos, **TN** los verdaderos negativos, mientras **FN** y **FP** son falsos positivos y falsos negativos (clasificaciones erróneas)

2.6. Medición de errores

Supongamos que se desea identificar cuáles son las galaxias frente a unas observaciones que además tienen estrellas. En este caso, se dice que la clase positiva son las galaxias (son lo que se busca), y las negativas las estrellas. Un algoritmo de aprendizaje sólo puede cometer dos errores en este caso:

- Clasificar una clase positiva como negativa, o
- Clasificar una clase negativa como positiva.

Además de esto, es obvio que el algoritmo puede simplemente no confundirse en la clasificación de las fuentes. Con toda esta información se puede crear una **matriz de confusión** como la que se presenta en Tabla 2.1

Esta matriz es fácilmente extensible a más clases que solamente dos (positivos vs negativos), con sólo agregar una fila y una columna por clase.

La matriz de confusión brinda información sobre el desempeño de un clasificador, ya que identifica si el algoritmo confundió una clase con otra. Varios índices pueden ser calculados a partir de esta matriz. Los más sencillos e intuitivos son la tasa de error

$$Err = \frac{FP + FN}{TP + FN + FP + TN}$$

o la taza de aciertos

$$Acc = \frac{TP + TN}{TP + FN + FP + TN}$$

Estas dos métricas son útiles sobre todo en casos de clases balanceadas; ya que de estar en presencia de clases muy desbalanceadas (explicada en la subsección 2.7.2) los errores obtenidos en la clase minoritaria simplemente puede ser insignificantes frente a los aciertos en la clase mayoritaria. Es por esto que los índices más utilizados, dados con sus nombres en ingles, son:

Precision el cual determina cuánto de lo que seleccioné ($FP + TP$) era realmente relevante (TP):

$$Precision = \frac{TP}{TP + FP}$$

Recall Cuánto de lo relevante ($TP + FN$) fue seleccionado (TP)

$$Recall = \frac{TP}{TP + FN}$$

$\mathbf{F_1}$ La media armónica del *Precision* y *Recall*

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Falsa-alarma o Tasa de Falsos–Positivos (FPR)

$$FPR = \frac{FP}{FP + TN}$$

Todas estas medidas varían entre los valores mayores o iguales que 0 y menores o iguales que 1, e indican una mejora en clasificación cuanto más altos son, a excepción del FPR que es mejor al tender a 0.

2.6.1. Curva *ROC*

Algunos clasificadores entregan un puntaje que representa "en qué grado" o con qué probabilidad cada ejemplo es miembro de una clase, el cual puede ser utilizado para producir muchos clasificadores variando el umbral de pertenencia de una clase a la otra. Si se varían los umbrales y se grafican estos puntos en relación al *Recall* y al FPR correspondientes, lo que se obtiene es una potente herramienta de evaluación llamada "Curva Característica Operativa del Receptor" (Curva ROC).

Las ROC pueden utilizarse para caracterizar el rendimiento de un modelo de clasificación binaria sobre todos los posibles *trade-off* entre el *Recall* y las falsas alarmas; independientemente del desbalanceo de clases. Además, el Área bajo la curva (AUC) representa el rendimiento en un único valor, el cual es equivalente estadísticamente a la prueba de Wilcoxon para rangos (Wilcoxon, 1945). Las ROC permiten en un solo gráfico comparar el rendimiento de muchos clasificadores de una manera bastante sencilla, por ejemplo en la figura 2.4 puede observarse como el clasificador en violeta (un Random-Forest) es significativamente mejor que el resto.

2.7. Consideraciones Importantes

El sobre-ajuste y el desbalanceo de datos, son dos problemas que potencialmente afectan al aprendizaje automático durante la fase de entrenamiento. Por ello existen técnicas y metodologías destinadas a solucionar estos problemas para realizar predicciones con más precisión en un espectro más grande de aplicaciones.

2.7.1. Sobreajuste

El sobre-ajuste es un problema presente en **todos** los modelos de aprendizaje automático; el cual se produce cuando dicho modelo intenta explicar patrones en los datos de entrenamiento que decrementan la efectividad en la predicción de otros datos nunca vistos. El problema normalmente se percibe como una estimación muy optimista del error en los datos de entrenamiento.

El problema puede ser mitigado (Bishop, 2006) dividiendo el conjunto de entrenamiento en:

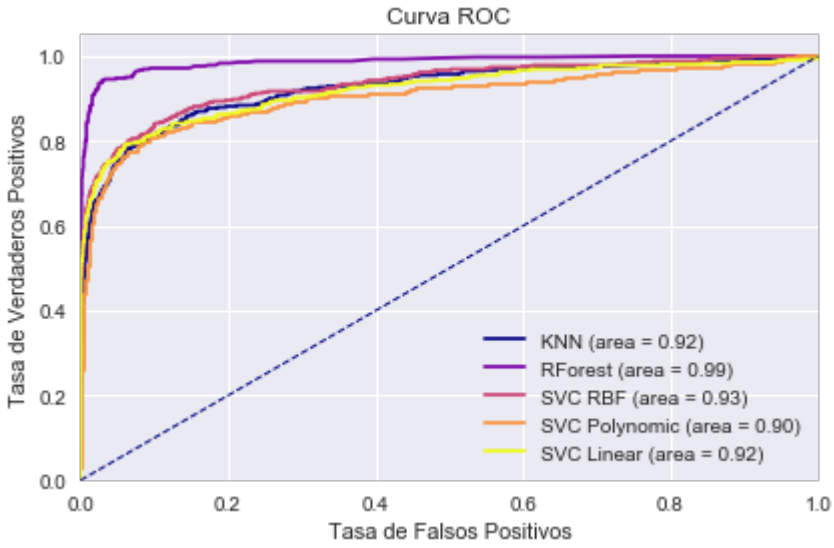

Figura 2.4: Curva ROC de ejemplo con cinco clasificadores, además del clasificador trivial en línea punteada. La tasa de verdaderos positivos es otro nombre dado al *Recall*.

Conjunto de entrenamiento Destinado a ajustar los parámetros del modelo supervisado.

Conjunto de prueba Utilizándolo es posible obtener una estimación del error de predicción sin el sesgo causado por el sobre ajuste de entrenar y probar con el mismo conjunto de entrenamiento.

Conjunto de validación De estar utilizando un modelo que tuviera hiperparámetros a determinar, este conjunto es necesario para estimar los hiperparámetros más útiles sin producir sobrea-juste.

Por ejemplo, si se utilizan modelos iterativos de aprendizaje, los datos de validación se utilizan para llevar control sobre el error de clasificación a lo largo del entrenamiento, es decir para determinar el número de iteraciones del método. Es muy común utilizar la técnica de frenar el entrenamiento cuando el error de validación alcance un mínimo, para evitar el sobre-ajuste de los datos de aprendizaje. Esta técnica es llamada detención temprana.

También hay que tener en cuenta que existe el problema del sub-ajuste, el cual se produce cuando el modelo es muy rígido para poder aprender a generalizar los patrones presentes en el conjunto de entrenamiento. Tanto sobre-ajuste como sub-ajuste pueden apreciarse visualmente en la Figura 2.5: en el panel de la izquierda un polinomio de 1er grado resulta en un modelo sub-ajustado, mientras que en el panel de la derecha el polinomio de 15avo grado ajusta perfectamente a la gran mayoría de los datos disponibles para entrenamiento, aunque en dicho modelo en nuevos datos, el error de predicción se incrementaría por sobre-ajuste. Finalmente, en el panel central, el polinomio de 4to grado re-

sulta un balance razonable entre descripción de los datos de entrenamiento y generalización .

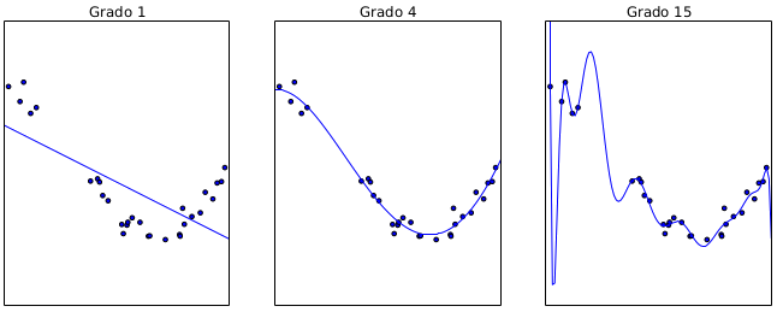

Figura 2.5: Polinomios de distintos grados aproximando datos ruidosos (Grieco, 2018).

2.7.2. Desbalance de clases

Un problema importante en el ML es el desbalance de clases. Esto sucede cuando la distribución de etiquetas de entrenamiento no es uniforme, con lo cual se pasa a tener una clase mayoritaria y una minoritaria. En presencia de este problema hay que tomar recaudos al momento de entrenar y evaluar un algoritmo.

Si se entrena un clasificador con un conjunto de datos **altamente desbalanceado**, suele derivar en la creación de un "*clasificador trivial*", el cual simplemente tiende a aprender a predecir la clase mayoritaria. Un ejemplo es el caso de la detección de transacciones bancarias fraudulentas con tarjetas de crédito, presentado en el trabajo de Dal Pozzolo et al. (2015). En este caso las operaciones fraudulentas son 0,172 %, por lo cual el clasificador tendería a indicar que todas las transacciones son legítimas.

Para disminuir este problema, existen dos conjuntos de técnicas ampliamente utilizadas:

Sobre-muestro aleatorio : en este caso se repiten aleatoriamente los casos de la clase minoritaria hasta que su conteo se asemeje al de la clase mayoritaria (He and Garcia, 2008).

Sub-muestro aleatorio Esta técnica consiste en seleccionar aleatoriamente un subconjunto de los datos de la clase mayoritaria de tamaño semejante al de la clase minoritaria (Japkowicz, 2000).

En ambos casos, solamente se balancea el conjunto de entrenamiento y **nunca** los de validación y prueba.

2.7.3. Selección de características

Los métodos de selección de características se utilizan dentro de la etapa de reducción de dimensión. Su utilidad es la de obtener un subconjunto de características más relevantes del conjunto completo de características, según

una función de criterio determinada (Violini, 2014). Dentro de estos métodos se destaca el llamado "*Recursive Feature Elimination*" o "Eliminación recursiva de características" (RFE) (Guyon et al., 2002).

El objetivo de RFE es seleccionar recursivamente un conjunto de características más pequeño, según su "importancia". Esta importancia es calculada con algún otro método supervisado (por ejemplo un clasificador SVM) el cual brinda un puntaje para cada característica.

El método opera de la siguiente forma:

1. Se calcula la importancia de todas las características utilizando el clasificador.
2. Se eliminan las características con menor puntaje.
3. Se repiten los pasos anteriores hasta que el número de características sea el deseado o se cumpla alguna condición de parada dada.

Las condiciones de corte pueden ser varias, como por ejemplo que una métrica dada (*precision*, *recall* o AUC) disminuya o se les asigne una cota mínima o máxima.

Respecto a cómo se calculan los puntajes, esto depende del método que se utiliza. Por ejemplo, RF analiza cuánto cambia la performance del método al alterar la característica dada de manera de transformarla en ruido sin alterar el problema de ninguna otra manera. Intuitivamente, una característica que cambia los resultados del clasificador es más importante que otra cuyo cambio no produce ningún efecto medible en el comportamiento del mismo.

2.8. Métodos de aprendizaje automático

A lo largo de este trabajo se utilizan diferentes métodos de ML, los cuales se describen a continuación.

2.8.1. Árboles de decisión

Los Árboles de decisión (DT) son un método para aproximar a una función objetivo discreta, en el cual la función aprendida se representa como un árbol. Los árboles aprendidos pueden representarse también como una serie de reglas *si-entonces* para hacerlas más legibles.

Los DT clasifican instancias ordenándolas en un árbol partiendo desde la raíz hasta las hojas. Cada nodo especifica un atributo, y cada rama descendente de ese nodo corresponde a un valor de ese atributo.

Para construir un DT la mayoría de los algoritmos desarrollados, siguen una estrategia de *arriba-abajo* buscando en un espacio de hipótesis constituido de todos los árboles de decisión posibles. Este enfoque fue desarrollado en el algoritmo *ID3* (Quinlan, 1986) y su sucesor *C4.5* (Quinlan, 2014).

ID3 construye los DT de arriba hacia abajo empezando con la pregunta: *¿Cual es el atributo que debe ser evaluado en la raíz del árbol?*. Para responder esto cada atributo es evaluado a través de una prueba estadística que determina qué tan bien clasifica él solo a los ejemplos de entrenamiento. El mejor atributo es entonces asignado a la raíz y entonces es creada una rama por valor posible

de este atributo; finalmente todos los ejemplos se asignan a las correspondiente ramas según su valor. A partir de allí se realiza una recursión, repitiendo el procedimiento descrito en cada rama. El proceso se detiene cuando no hay más ejemplos, cuando todos los ejemplos de una rama son de una sola clase, o cuando no hay más atributos para evaluar.

El test estadístico que se utiliza para determinar el mejor atributo es el llamado Ganancia de información (IG), la cual es una métrica derivada de la entropía:

$$Entropía \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus} (2.1)$$

dónde $p_{\oplus}$ y $p_{\ominus}$ son la proporción de ejemplos positivos y negativos en el conjunto de entrenamiento S. Nótese que la entropía es 0 si todos los miembros de S pertenecen a la misma clase y 1 cuando la colección contiene la misma cantidad de ejemplos positivos y negativos. Una de las interpretaciones de la entropía es el nivel de impureza de S, qué tan "mezclado" está S, o qué tan predecible es S.

Ahora IG es cuánto disminuye la entropía de S si retiramos un atributo A:

$$IG(S, A) \equiv Entropía(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} Entropía(S_v) (2.2)$$

donde $V(A)$ es el conjunto de todos los valores posibles para el atributo A, y S_v es el subconjunto de S para el cual el atributo A tiene el valor v.

2.8.2. Random Forest

Los DT poseen dos ventajas importantes: son fáciles de entrenar y son fáciles de interpretar. Por desgracia, un árbol de decisión suele no alcanzar para capturar lo complejo de muchos tipos de datos, así que son raramente utilizados.

Dentro de este contexto surge la idea de utilizar un conjunto grande de modelos que se usan juntos como un "meta modelo" o ensamble, para lograr usar conocimiento de distintas fuentes al tomar decisiones; así es que en el año 2001, Leo Breiman, definió el método de *Random-Forest* RF (Breiman, 2001). Este clasificador utiliza un conjunto de árboles de decisión, cada uno de ellos convenientemente entrenados sobre un sub-conjunto aleatorio de datos. Con esto, cada árbol es especialista en ciertos patrones de datos, de modo que hace posible que una buena predicción suceda cuando trabajan en conjunto. Este es un modelo robusto muy utilizado en diversos problemas de clasificación debido a una gran resistencia al sobreajuste y su efectividad en la predicción (Caruana et al., 2008).

RF trata de resolver el dilema del sesgo y la varianza (Friedman et al., 2001), utilizando DT individuales, los cuales son muy flexibles y ajustan muy bien a los datos pero que tienen una mala estimación de los parámetros óptimos. Para disminuir la varianza, utiliza muchos predictores promediados, los cuales construye seleccionando el mejor atributo entre una muestra al azar (pequeña) de todos los atributos disponibles en cada rama.

Desde el punto de vista de sus hiper-parámetros, solo dos de ellos son importantes:

Cantidad de árboles No suele alterar mucho el funcionamiento del meta-predictor mientras sean muchos (500, 1000, 2000). Hay que tener en cuenta que todos los árboles generados utilizan todas las variables que se les provee, y **no** son post-procesados para que generalicen mejor.

Cantidad de variables elegidas al azar por cada predictor El meta-clasificador es mucho más sensible a este parámetro. Por defecto se utiliza la raíz cuadrada de la cantidad total de variables ($\sqrt{p}$), la cual suele dar buenos resultados en la mayoría de las aplicaciones.

2.8.3. Máquinas de vectores de soporte (SVM)

La idea intuitiva de un SVM es muy simple, el modelo representa a los ejemplos como puntos en un espacio vectorial, de tal forma que diferentes clases están divididas por una superficie con un claro margen, es decir lejana a todos los puntos a la vez. Entonces, nuevos ejemplos son mapeados al mismo espacio y su pertenencia a una clase queda determinada por "de que lado" de la superficie son ubicados. Esta idea fue propuesta originalmente por Vladimir Vapnik y su actual encarnación (*Soft Margins*) por Corinna Cortes y Vladimir Vapnik (Boser et al., 1992; Cortes and Vapnik, 1995)

Desde un punto de vista más formal, los SVM construyen un hiper-plano en un dado espacio, que puede ser de dimensión muy alta o infinita. Intuitivamente una buena separación se puede lograr por el hiper-plano que tenga la mayor distancia con respecto a los puntos más cercanos del conjunto de entrenamiento (esto es llamado margen funcional y los puntos más cercanos son los llamados vectores de soporte), dado que grandes margenes generalizan mejor. Asi puede verse en la figura 2.6 como el hiper-plano H_1 no separa linealmente las clases, mientras que H_2 y H_3 si lo hacen; así mismo, H_3 es el hiper-plano con mayor distancia a los puntos más cercanos y por lo tanto el que mejor generaliza. En la práctica, los métodos actuales de SVM, utilizan el llamado "*Soft-margin*" o "margen débil", haciendo uso de una constante C (determinada al momento de validación), que regula la tolerancia a puntos que quedan del lado incorrecto del hiper-plano, lo que está directamente asociado a la flexibilidad y capacidad de generalización del modelo encontrado.

Los problemas enfrentados con SVM, empiezan en un espacio de dimensión finita y es común que en este espacio las clases no sean linealmente separables. Por esta razón se propone mapear el espacio original a un espacio de dimensión mayor donde presumiblemente será más sencillo realizar esta división (Figura 2.7). En lugar de buscar una transformación apropiada para convertir las observaciones al espacio de dimensiones mayor, Vapnik sugirió seleccionar una función *kernel* $K(x_i, x_j)$ adecuada, ya que el algoritmo sólo necesita conocer el valor del producto interno entre pares de vectores para encontrar el hiper-plano óptimo.

Dentro de los *kernels* más utilizados encontramos:

Lineal $K(x_i, x_j) = x_i^T x_j$
donde x_i y x_j son los puntos en el espacio original.

Polinómico de orden p $K(x_i, x_j) = (1 + x_i^T x_j)^p$
donde x_i y x_j son los puntos en el espacio original y p es un hiper parámetro con el orden del polinomio.

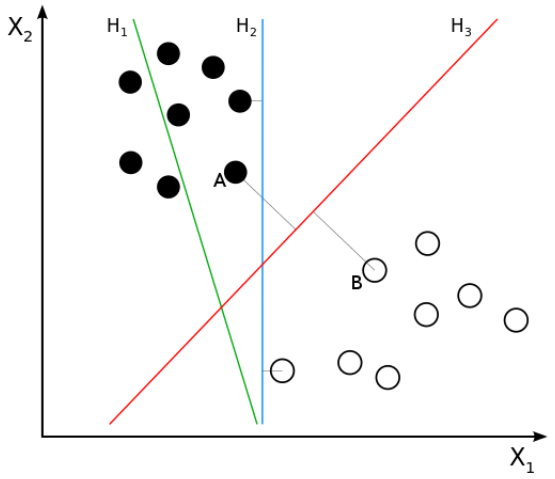

Figura 2.6: Imagen ilustrativa de 3 hiper planos separando dos clases (puntos negros y puntos blancos) en dos dimensiones. Los puntos marcados como A y B son los vectores de soporte para H_3.

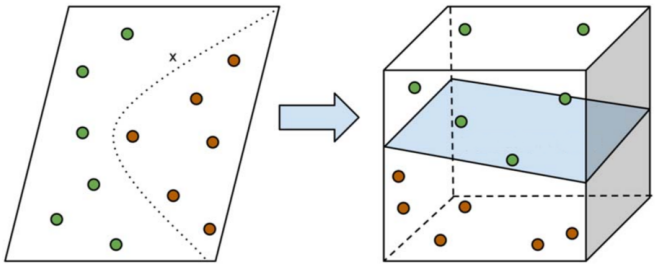

Figura 2.7: Ejemplo ilustrativo de como una transformación dada por una función kernel puede transformar ejemplos que no son linealmente separables a una dimensión mayor donde sí lo son. Adaptado del trabajo de Shia et al. (2017)

Función de base Radial (RBF) $K(x_i, x_j) = \exp(\frac{|x_i - x_j|^2}{2\sigma^2})$ donde X_i y x_j son los puntos en el espacio original y σ es un hiper parámetro que por defecto es la la varianza de x_i.

2.8.4. KNN

"K-nearest neighbors algorithm" o "K-vecinos más cercanos" (KNN) es un método no paramétrico usado para clasificación. Su idea es muy simple: la clase asignada a una observación es la que con más frecuencia se encuentre dentro de los K vecinos más próximos del conjunto de entrenamiento.

K es un parámetro decidido por el usuario, y la métrica de distancia utilizada es normalmente la distancia Euclídea.

KNN es un tipo de algoritmo conocido como "basado en instancias" donde la función sólo es aproximada localmente y toda la computación del resultado se

retrasa al momento de la clasificación. KNN es uno de los métodos más simples de aprendizaje automático.

2.9. Contribuciones del aprendizaje automatizado a la astronomía

En esta sección se ejemplifica el impacto del aprendizaje automático en el campo de la astronomía. Para esto se seleccionan algunos ejemplos de la literatura.

2.9.1. *Palomar Transient Factory* (PTF)

"*Palomar Transient Factory*" ("Fábrica de transitorios del palomar" – PTF) es un proyecto de exploración sistemático y automático del cielo en espectro óptico, en busca de eventos transitorios [3].

PTF ya ha culminado en el año 2010 y su éxito ha generado dos sucesores: el también finalizado IPTF (PTF intermedio) (Kulkarni, 2013a); y el actual ZTF ("*Zwicky Transient Facility*", Infraestructura de transitorios Zwiki). En cada caso se aumentó la calidad de la cámara y el tamaño del telescopio.

Lo interesante de regresar sobre el proyecto original, fue que con aprendizaje automático, una cámara modesta y un telescopio que era una pieza de museo, lograron una revolución en el campo de la búsqueda de transitorios con más de ~ 196 trabajos publicados a la fecha entre las 3 iteraciones [4].

La inspección transitoria se realizó utilizando una cámara de 8.1 grados cuadrados instalada en el telescopio "Samuel Oschin" (ver Figura 2.8) de 48 pulgadas en el Observatorio Palomar[5]; mientras que los colores y las curvas de luz para los transitorios detectados se obtienen con el telescopio automático Palomar de 60 pulgadas. PTF proporciona clasificación vía aprendizaje automático y seguimiento en tiempo real de transitorios, así como una base de datos que incluye todas las fuentes detectadas (Law et al., 2009).

2.9.2. Simulaciones cosmológicas

La cosmología es el estudio de las estructuras y dinámicas a mayor escala del Universo y se ocupa de preguntas fundamentales sobre su origen, evolución, destino final y estructura.

El estudio de esta rama de la astronomía suele realizarse a través de simulaciones (Kuhlen et al., 2012) en las cuales se validan hipótesis de evolución respecto a diferentes condiciones iniciales de diferentes parámetros cosmológicos.

Las simulaciones usan descripciones discretas del universo, donde colocan una gran cantidad de partículas (N) que trazan la distribución e interacción de materia y energía del universo.

Computacionalmente, calcular todas las fuerzas de interacción entre cada partícula tiene un coste en tiempo del orden $O(N^2)$, o $O(Nlog(N))$ como es el caso del "*Particle-Mesh*" (Klypin and Shandarin, 1983). Hay códigos que

[3] Fenómeno cuya duración puede ser de segundos a días, semanas o incluso varios años

[4] Lista privada de publicaciones del PTF `http://adsabs.harvard.edu/cgi-bin/nph-abs_connect`

[5] `http://www.astro.caltech.edu/palomar/homepage.html`

Figura 2.8: Fritz Zwicky mirando a través del telescopio Schmidt de 18 pulgadas (luego renombrado a Telescopio "Samuel Oschin"), alrededor de 1936. (Archivos de Caltech)

mejoran aun más estos tiempos pero el orden de magnitud de los "*snapshots*" [6] de las simulaciones es gigantesco (ver Figura 2.9).

En números, por ejemplo, la simulación "*Millenium-XXL*", resolvió la interacción de más de 300 mil millones de partículas equivalente a más de 13 mil millones de años, al mismo tiempo que hacía predicciones de distribución de masa a gran y pequeña escala. Este cómputo fue realizado en más de 12 mil núcleos, y 30 TB de RAM en la super-computadora "*JUROPA*" del "*Centro de Super-Computacion de Jülich*" en Alemania. El resultado fueron más de 100 TB de datos (Angulo et al., 2012).

El exigente cómputo que conllevan estas simulaciones dio lugar a que en la actualidad hayan surgido líneas de investigación las cuales intentan por medio de aprendizaje automático, acelerar o mejorar estos trabajos. Como por ejemplo el trabajo de Rodriguez et al. (2018) utiliza redes adversarias (Goodfellow et al., 2014) para generar observaciones independientes de las "*snapshots*" de las simulaciones, logrando producir en fracciones de segundo y con una calidad similar, lo que en la simulación tardaría varias horas.

2.9.3. Detección de ondas gravitacionales en tiempo real

Las ondas gravitacionales de la colisión de dos agujeros negros (Willke et al., 2018; Collaboration et al., 2016) y la posterior unión de dos estrellas de neutrones (Virgo et al., 2017), junto con sus observaciones electromagnéticas (Abbott et al., 2016b, 2017, 2016a; Díaz et al., 2017, 2016), detectadas por los inter-

[6] Como se dijo las simulaciones son discretas, se les llama "*snapshot*" (instantánea) a un tiempo-espacio el cual fue simulado.

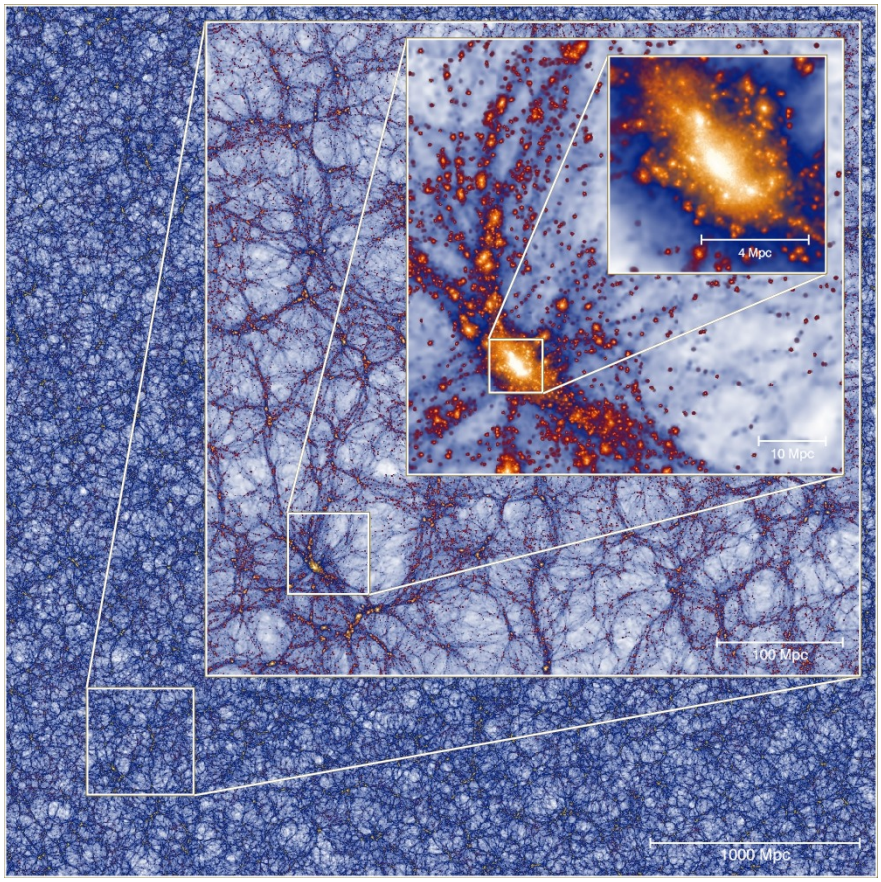

Figura 2.9: Campo de densidad de masa de la simulación "*Millenium-XXL*", centrado en el halo más masivo a tiempo presente ($z = 0$). Cada recuadro se aleja por un factor de 8 respecto al anterior; la longitud del lado varía desde 4,1Gpc hasta 8,1Mpc. Todas estas imágenes son proyecciones de una rebanada 8Mpc. Fuente `https://goo.gl/M1EbmY`

ferómetros (ver Figura 2.10) (Abramovici et al., 1992), fueron reconocidas en 2017 con el Premio Nobel de Física [7]

Sobre este tópico, el trabajo de George and Huerta (2018), demostró la capacidad del aprendizaje automático (LeCun et al., 1998) para la descripción y detección muy rápida de ondas gravitacionales utilizando datos reales de LIGO. Los resultados del trabajo también demuestran sensibilidades similares y errores más bajos en comparación con el filtrado combinado. Además, es mucho más eficiente desde el punto de vista informático y más tolerante a fallos; esto permite en tiempo real procesar señales débiles de series de tiempo en ruido no estacionario gaussiano con muy pocos recursos. Otra capacidad de este enfoque es la detección de nuevas clases de fuentes de ondas gravitacionales las cuales pasan desapercibidas con los algoritmos de detección existentes.

[7] `https://www.nobelprize.org/prizes/physics/2017/press-release/`

Figura 2.10: Foto aérea del interferómetro Virgo, se observan el edificio central, el edificio del Mode-Cleaner, los 3 km totales del brazo oeste y parte del principio del brazo norte (a la derecha). Los demás edificios son zonas de trabajo varias y la sala de control. Fuente https://www.ligo.caltech.edu

La técnica finalmente es aplicable a detección coincidente de ondas gravitacionales y sus contra-partes electromagnéticas multi-mensajeras[8] (Allen et al., 2018) en tiempo real.

Lo interesante de este trabajo es que reúne el estado del arte de la astronomía, y el estado del arte en aprendizaje automático.

2.9.4. Aprendizaje automático en el *VVV*

Para cerrar esta sección presentamos dos trabajos relacionados con el relevamiento que utilizamos en esta tesis donde se aplica la técnica de aprendizaje automático:

- *VVV Templates* El objetivo perseguido por este proyecto, es el desarrollo y prueba de los algoritmos de aprendizaje automático para clasificar automáticamente curvas de luz del *VVV*. Como se está posicionado frente al primer estudio masivo y multi-época de variabilidad estelar en el infra-rojo cercano, se requiere ejemplos para entrenar los algoritmos de clasificación los cuales a priori no están disponibles. Así este trabajo propone la creación de esta base de datos integral de ejemplos de variabilidad estelar infrarroja. (Angeloni et al., 2014).

- Las estrellas variables del tipo *RR Lyrae* son muy importantes para medir las distancias a las poblaciones estelares antiguas en la Vía Láctea. Dado que el *VVV* busca como uno de sus objetivos primordiales el mapeo de como se formo el bulbo galáctico, estas distancias son muy útiles

[8] La astronomía multi-mensajera es aquella que utiliza observaciones electromagnéticas y de onda gravitacionales al mismo tiempo

para esta tarea. El trabajo de Elorrieta et al. (2016) propone un mecanismo automático para la búsqueda de este tipo de estrellas describiendo pasos clave como son la generación de características y la construcción y evaluación de los clasificadores.

2.10. Conclusiones del capítulo

En este capítulo, se presentó el estado de la ciencia de datos orientado a la astronomía en particular; dando importancia a la transformación de la astronomía en una ciencia de datos, por la necesidad del crecimiento en calidad y cantidad de datos generados por los nuevos instrumentos astronómicos; lo cual finalizó con el surgimiento de las sub-disciplinas de "Astro-Estadística" y "Astro-Informática".

A continuación, se explicó al aprendizaje automático en contexto de la extracción de conocimiento para el soporte a otras disciplinas; continuando con una corta introducción teórica al aprendizaje automático en general acompañado de descripciones del funcionamiento de sus algoritmos más populares.

Finalmente, se presentaron cuatro ejemplos del uso de métodos computacionales en diferentes ramas de la astronomía, que están generando saltos cualitativos significativos en la forma de encarar esos problemas.

Capítulo 3

Reconstrucción de curvas de luz a partir de catálogos fotométricos múltiple-época y extracción de sus características

3.1. Objetivos del capítulo

Este capítulo persigue el fin de presentar al relevamiento astronómico fotométrico (terrestre) *"Vista Variables in the Via Lactea" (VVV)* (Minniti et al., 2010), sus objetivos, el telescopio que utiliza, particularidades de sus observaciones y los formatos de los datos que generan que son relevantes a este trabajo. Luego, se explica cómo reconstruir curvas de luz de las observaciones del *VVV* teniendo en cuenta su fotometría y su astrometría. Se finaliza con una descripción de las características extraídas de las curvas de luz antes reconstruidas.

3.2. El relevamiento "*Vista Variables in the Via Lactea*"

Como ya se mencionó el *VVV* y su sucesor el *"VVV Survey eXtended" VVVx*, persiguen el objetivo de producir un mapa tridimensional de una gran parte del centro galáctico (Bulbo) de la Vía Láctea y de una fracción del Disco Galáctico interno. En adelante durante todo el trabajo se refiere a los dos relevamientos indistintamente como VVV, excepto cuando sea necesario diferenciarlos.

Este mapa tridimensional se obtiene a partir de un censo de estrellas, posibilitado gracias a un monitoreo sistemático de estas regiones de la Vía Láctea. En el caso del *VVV*, por ejemplo, completó un total de 1.929 horas de observación realizadas durante un período de cinco años (iniciados en el 2010). Para ello, el equipo de astrónomos a cargo del proyecto *VVV* utilizó el moderno telescopio

Figura 3.1: Telescopio VISTA (siglas en inglés de "*Visible and Infrared Survey Telescope for Astronomy*" o "Telescopio astronómico de rastreo de espectro visible e infrarrojo"). Crédito: ESO `https://www.eso.org`

VISTA (Figura 3.1), ubicado en Cerro Paranal, II Región, Chile (Figura 3.2 y Figura 3.3); el cual generó mas de 300 GB de datos por noche.

Los datos del *VVV* se presentan en una unidad llamada "baldosa" (*tile*, en inglés), la cual es una zona rectangular del cielo de $1{,}501 deg^2$ relevada a través del tiempo. Cada baldosa se compone de varias imágenes en alta resolución para diferentes tipos de filtros (bandas anchas) de frecuencias lumínicas en el infrarrojo cercano (cinco en total). Asimismo, por cada imagen existe una base de datos con los valores de posición, magnitud y color de las fuentes de luz presentes en la imagen, llamada "catálogo fotométrico". En el caso del *VVV*, la totalidad de las baldosas que constituyeron el relevamiento (Área del disco de longitudes galácticas de 250 a 297 grados y el área del bulbo galáctico de 350 a 10 grados) puede apreciarse en la Figura 3.4.

Dentro del barrido total del relevamiento, es de interés astronómico la generación de catálogos de cualquier tipo de objeto. Para el *VVV* mas allá de estrellas periódicas variables y otras, en la zona bajo estudio también se identificará fuentes de luz de planetas y galaxias o sea de un número bastante amplio de fenómenos distintos, Minniti et al. (2010) (ver figura *1.1* en el Capítulo 1).

Resultan de interés, las estrellas de tipo variable *RR-Lyrae*, las cuales poseen características muy bien definidas y estables:

- Pulsación periódica de entre $\sim 0{,}2$ y $\sim 1{,}2$ días.
- Variaciones de magnitud desde $\sim$ 0,1 hasta $\sim$ 0,5 en NIR durante las pulsaciones.
- Tipo espectrales del A al F (esto determina el color, temperatura, composición química y masas de la estrella).

Estas características tan estables las hace útiles para la determinación de distancias, las cuales a su vez son de provecho para cumplir con uno de los objetivos primordiales del VVV: mapear el bulbo galáctico. Además la existencia

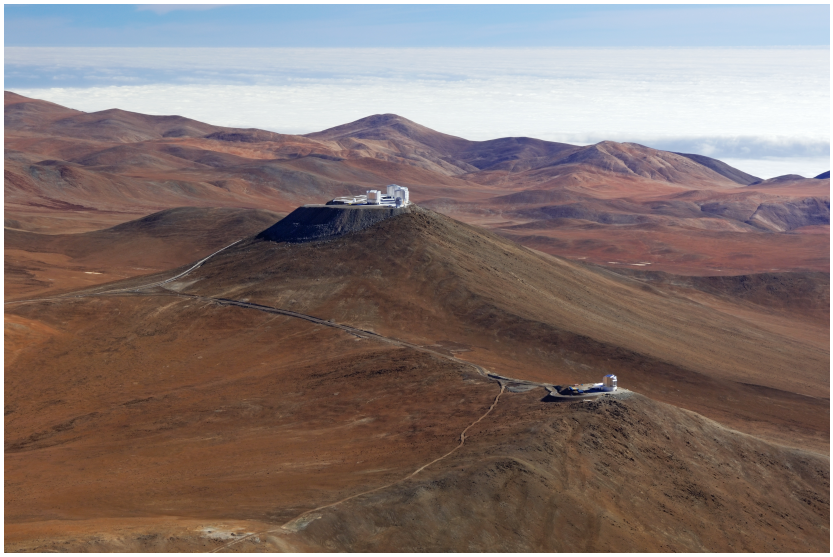

Figura 3.2: Vista panorámica del Observatorio Paranal. Al frente se observa el telescopio VISTA y al fondo el sistema de cuatro telescopios del "*Very Large Telescope Project*" (VLT - literalmente "Telescopio Muy Grande"). Crédito: ESO `https://www.eso.org`

de publicaciones que identifican algunós cientos de ellas dentro del relevamiento (Gran et al., 2015) (Gran et al., 2016), facilita la creación de los primeros conjuntos de entrenamiento y prueba.

Como aclaración adicional es conveniente explicar que la "magnitud" a la cual nos referimos es llamada formalmente "magnitud aparente". Este valor indica la medida del brillo y cantidad de energía por segundo por metro cuadrado que se recibe de un objeto celeste por un observador en la Tierra. Como dicha cantidad recibida depende de la transmisión de la atmósfera en dichas bandas, las magnitudes aparentes se normalizan a un valor fuera de la atmósfera terrestre. La escala de magnitudes es una relación inversa logarítmica por la cual la estrella más brillante es la que tiene menor magnitud: El sol tiene una magnitud aparente de aproximadamente -27 en la banda V. La estrella *Vega* se suele utilizar como referencia para el cálculo de magnitudes.

3.3. Catálogos fotométricos múltiple-época

El *pipeline* de *VVV* (Emerson et al., 2004), además de preprocesar cada imagen, brinda por cada una de ellas una base de datos de archivos con los valores de posición, magnitud y color de las fuentes de luz presentes en la imagen, llamada "catálogo fotométrico", como ya se mencionó. Es sobre estos catálogos donde se enfocará esta tesis.

Hay que tener en cuenta que existen diferentes tipos de catálogos. En este trabajo son importantes dos de ellos:

1. ***Pawprint Stack***. Los sensores infrarrojos de la cámara VIR-CAM (Figura 3.5) del telescopio VISTA tienen márgenes entre ellos. Es por esto, que

Figura 3.3: El telescopio VISTA observando. Crédito: ESO `https://www.eso.org`

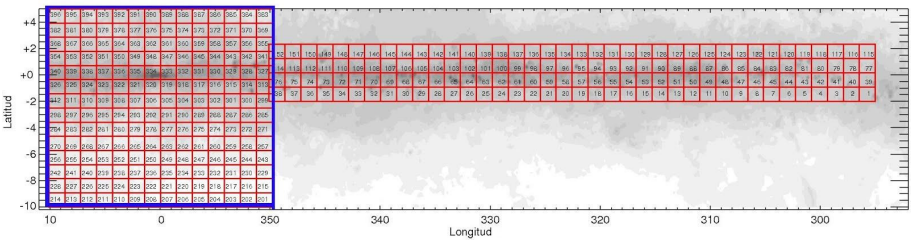

Figura 3.4: Mapa de enrojecimiento de Schlegel adaptado de Minniti et al. (2010), mostrando el área del relevamiento en el cielo en coordenadas galácticas y las baldosas que conforman el VVV. En escala de grises está la densidad estelar proyectada de este mismo campo tomado del relevamiento astronómico "Two Micron All Sky Survey" (Skrutskie et al., 2006b).

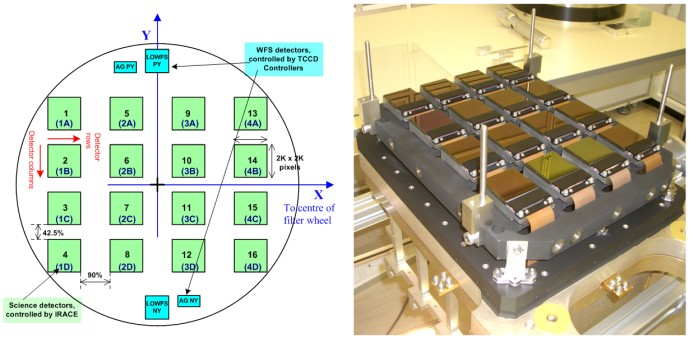

Figura 3.5: A la izquierda el diagrama de los sensores infrarrojos de VIRCAM y a la derecha una foto de la VIRCAM. Adaptado de `http://www.vista.ac.uk`

se necesita desplazar el telescopio tres veces variando el eje X y tres veces el eje Y para completar la imagen de un *tile*. Cada exposición es llamada *Pawprint* y la suma de las seis es llamada *Pawprint Stack* o *época*, por ser la observación de un *Tile* en una fecha dada.

2. ***Band-Merge***. Este catálogo se utiliza para tener una lista confiable de estrellas en un *tile*. Es necesario comprender que dos *Pawprint Stack* pueden no tener exactamente la misma cantidad de fuentes, ya sea por motivos meteorológicos o instrumentales que hacen que cada observación sea ligeramente diferente. Es por esto que los catálogos *Band-Merge* se crean manualmente utilizando solo la primera época, que es la única disponible en varias bandas de observación, y se extraen de ella las fuentes que solo están presentes al mismo tiempo en filtros infrarrojos J, K_s y H. En adelante se utiliza este catálogo como la lista definitiva de fuentes en un *tile* dado.

Dadas las características técnicas del relevamiento, todas las fuentes en los *Band-Merge* de magnitud[1] $\lesssim 12.$ (muy brillantes que saturan el telescopio) y $\gtrsim 16{,}5$ (muy débiles) son ignoradas (Gran et al., 2015).

3.4. Reconstrucción de la serie temporal para la generación de características

Como se mencionó antes, para identificar y tipificar una estrella variable-periódica de cualquier tipo, es necesario determinar su variación de magnitud en un período dado. Es entonces una necesidad identificar cada fuente presente en un *Band-Merge* de un *Tile* con todas las observaciones existentes en todos los *Pawprint Stack* disponibles para dicho *Tile*, y así reconstruir una serie temporal.

3.4.1. Emparejamiento por proximidad (*Cross-Matching*)

De cada *Pawprint Stack* se conoce: a qué *Tile* pertenece, a qué fecha corresponde la medición, posición (en coordenadas esféricas) y magnitud para

[1]Las fuentes son más brillantes mientras menor es la magnitud

cada fuente. Lamentablemente, no hay una forma unívoca de determinar que un objeto observado en un *Band-Merge* es el mismo que uno observado en el *Pawprint-Stack* ya que no comparten ningún identificador, y si bien las posiciones medidas son cercanas, no son iguales para la misma fuente.

Para superar esta dificultad se identificaron las fuentes a través del método *cross-matching*. Este método consiste en verificar cuáles son las fuentes más cercanas en posición entre los catálogos A y B, y viceversa dentro de un intervalo D. Sólo se asume que la fuente a del catálogo A y la fuente b del catálogo B son las mismas si a es la más cercana a b, b es la más cercana a a y además están dentro del intervalo D.

El intervalo D elegido en nuestro estudio fue $1/3$ *de segundo de arco* (Gray et al., 2006).

3.4.2. Corrección de Fechas

Otro punto importante, a tener en cuenta es que las fechas de los *Pawprint-Stack* están definidas como Días Julianos Medios (MJD del inglés *Mean Julian Date*); que son el promedio de días julianos de todos los *Pawprints* involucrados en el *stack*. Asimismo, un día juliano es la cantidad de días transcurridos desde el mediodía del 1º de enero del año 4713 A. C. En este estudio, para definir las series temporales, el MJD acarrea el problema de que es un formato de fecha *geocéntrico*. Esto lleva a que la misma fuente (observada en dos épocas distintas), dado que la velocidad de la luz es finita, dependa de la posición del observador en el sistema solar cuando es realizada.

Para subsanar la dificultad descrita, se utiliza el Día Juliano Heliocéntrico (HJD del inglés *Heliocentric Julian Date*) el cual corrige el MJD utilizando las diferencias en la posición de la Tierra con respecto al Sol (Eastman et al., 2010).

3.4.3. Período

Ya identificadas todas las observaciones de la misma estrella y corregidas sus fechas de observación, el siguiente paso consiste en determinar su período. La planificación de observaciones en *VVV* hace que los *tiles* no se muestreen uniformemente en un período dado. Por consiguiente, lo más cómodo es hacer el supuesto de que la frecuencia de observaciones es aleatoria. Si se grafican los datos directamente como serie temporal no se percibirá ningún período evidente (Figura 3.6).

Para recuperar el período se utiliza el método de *Fast Lomb-Scargle* (Lomb, 1976; Scargle, 1982; VanderPlas, 2018) el cual mide el ajuste de mínimos cuadrados de sinusoides a los datos muestreados. Poniendo en fase los datos sobre este período obtenido queda en evidencia la curva de luz de la fuente periódica (Figura 3.7)

3.5. Generación de los conjuntos de datos

El conjunto de datos de estrellas variables (incluidas las *RR-Lyrae*), fue obtenido al realizar un *cross-matching* (con la misma tolerancia con la cual recreamos las curvas de luz) con los catálogos de estrellas variables del relevamiento *OGLE-III* (Soszynski et al., 2011), *OGLE-IV* (Udalski et al., 2015) y *VizieR*

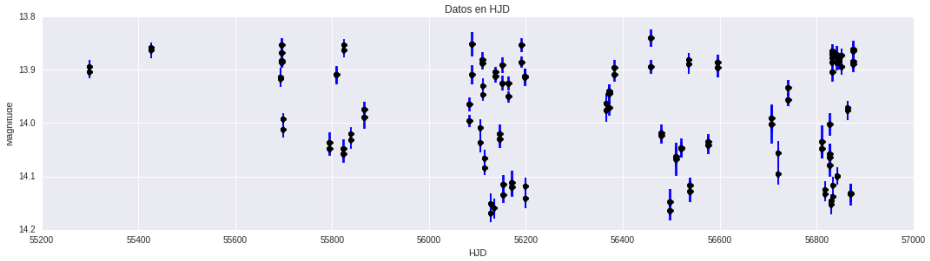

Figura 3.6: Observaciones de magnitud de una estrella de tipo variable *RRLyrae* AB del trabajo "Bulge RR Lyrae stars in the VVV tile b201"(Gran et al., 2015) identificada con el ID `VVV J2703536.01-412829.4`. El eje X representa la fecha de medición y el Y la magnitud en orden inverso.

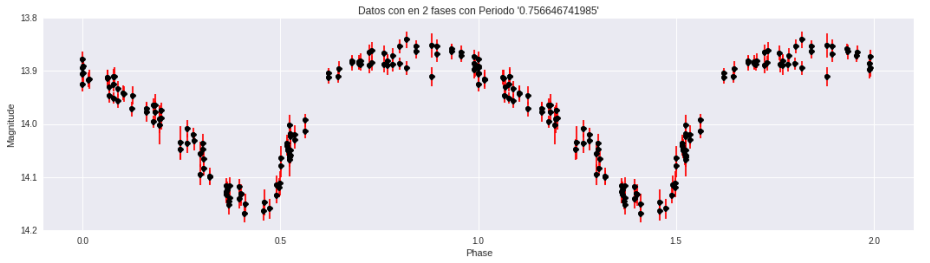

Figura 3.7: Observaciones de magnitud en fase de la misma estrella correspondiente a la figura 3.6. El eje X representa la fase de medición y el Y la magnitud en orden inverso. Por comodidad para la observación de periodicidad se presentan dos ciclos completos.

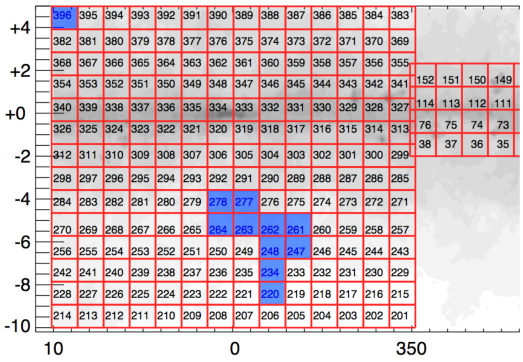

Figura 3.8: En azul se observa la ubicación de *tiles* utilizados en el trabajo para *tiles* del Bulbo.

Nombre	Tamaño	Variables
b220	691713	149
b234	820452	1352
b247	939320	2601
b248	1081662	3485
b261	955119	4677
b262	1087417	5678
b263	920350	5503
b264	999947	4747
b277	979349	8686
b278	1018874	10179
b396	1075212	172
Total	10569415	47229

Tabla 3.1: Cantidad de estrellas variables identificadas en cada *tile* bajo estudio en este trabajo. La columna **Tamaño** indica el total de fuentes, y **Variables** la cantidad de estrellas variables usando cortes pre-establecidos para identificarlas.

(Ochsenbein et al., 2000), los cual poseen zonas de observación superpuestas, aunque carentes de profundidad, como la del *VVV*. Así, en los *tiles* utilizados (ver figura 3.8) logramos detectar la distribución de estrellas variables que se aprecia en la Tabla 3.1,

3.6. Características

La extracción de características que se llevó adelante en este trabajo (incluido el período descrito en detalle en la Subsección 3.4.3) fue llevada adelante con la herramienta *feets* (Cabral et al., 2018b). A estas, se agregó otra colección de características solo calculables en el *VVV*, las cuales en su mayoría son libres de distorsión del polvo interestelar.

En las siguientes dos sub-secciones se presenta una breve descripción de las características extraídas con *feets* (acompañadas de referencias si el lector desea

ahondar conocimiento sobre algunas de ellas); y se continúa con una explicación más detallada de características calculadas en particular con este trabajo.

3.6.1. Características extraídas con *feets*

La versión de *feets* utilizada fue la *0.4*, con la cual se extrajo un total de 38 características de las curvas de luz compuestas por tiempos, magnitudes y errores estimados en esas magnitudes. A continuación, se presenta una breve descripción de las mismas.

1 - `Amplitude` **(A_{K_s})** o Amplitud, es la mitad de la diferencia entre la mediana del 5 % de los valores mayores y la mediana 5 % menores de las magnitudes en la banda K_s (Kim et al., 2011).

2 - `Autocor_length` del inglés "*Auto-Correlation Length*" o "Largo de la autocorrelación cruzada". Se define como correlación cruzada de la señal consigo misma. Es una herramienta matemática útil para encontrar patrones repetidos, como pueden ser señales periódicas ocultas por ruido (Kim et al., 2011).

4, 5, 6 - `Índices CAR` **(CAR_μ, CAR_σ, CAR_τ)** del inglés "*Continuous time Auto Regressive model*" o "Modelo continuo auto-regresivo". Provee una forma natural y consistente de estimar tiempos medios y varianzas de una curva de luz (Pichara et al., 2012).

7 - `Con` del inglés "*Consecutive*" o "Puntos consecutivos". Cantidad de tres puntos consecutivos que son mayores o menores que 2σ y normalizados por $N - 2$. Este índice fue introducido para la selección de estrellas variables en la base de datos del relevamiento OGLE (Wozniak, 2000; Kim et al., 2011).

8 - `Eta_e` **(η^e)** Índice utilizado para evaluar si existe alguna tendencia en los datos mediante la verificación de la dependencia de observaciones con respecto a las anteriores (Kim et al., 2014).

9, 10, 11, 12, 13 - `FluxPercentileRatioMid` del inglés "*Middle flux percentiles Ratio*" o "Proporción de flujo de percentiles centrales". Para caracterizar las distribuciones de magnitud ordenadas se usan proporciones de percentiles de sus flujos [2] (Richards et al., 2011b). Si decimos que $F_{i,j}$ es la diferencia entre el percentil j y el percentil i, entonces calculamos:

- $FluxPercentileRatioMid20 = F_{40,60}/F_{5,95}$
- $FluxPercentileRatioMid35 = F_{32,5,67,5}/F_{5,95}$
- $FluxPercentileRatioMid50 = F_{25,75}/F_{5,95}$
- $FluxPercentileRatioMid65 = F_{17,5,82,5}/F_{5,95}$
- $FluxPercentileRatioMid80 = F_{10,90}/F_{5,95}$

[2] Flujo o luminosidad, es la cantidad de energía que brinda una estrella en una unidad de tiempo

14 a 20 - `Componentes Fourier` Son los tres primeros componentes Fourier del primer periodo candidato. Las características *Freq1_harmonics_amplitude_i* y *Freq1_harmonics_rel_phase_j* representan la *i-ésima* amplitud y la *j-ésima* fase de la señal respectivamente (Richards et al., 2011b).

21 - `LinearTren` Tendencia lineal. Es la pendiente del ajuste lineal de los datos (Richards et al., 2011b).

22 - `MaxSlope` Pendiente máxima absoluta entre dos observaciones consecutivas en tiempo (Richards et al., 2011b).

23 - `Mean` Media de magnitudes (Kim et al., 2014).

24 - `MeanVariance` La media de variación, es la división entre la desviación estándar de las magnitudes σ, con la media de magnitudes $\bar{x}$. Si la serie tiene una gran variabilidad este valor es alto (Kim et al., 2011).

25 - `MedianAbsDev` Mediana de las desviaciones absolutas, este valor se define como la mediana de la diferencia de cada magnitud observada con la mediana de la serie, o en símbolos $MedianAbsoluteDeviation = median(|mag - median(mag)|)$ (Richards et al., 2011b).

26 - `MedianBRP` - Del inglés "*Median Buffer Range Percentage*" lo cual puede interpretarse como porcentaje alrededor de la mediana. Proporción de magnitudes dentro de la décima parte del rango total alrededor de la mediana (Richards et al., 2011b).

27 - `PairSlopeTrend` Tendencia de pendiente de pares. Se calcula considerando solamente las últimas 30 magnitudes (ordenadas por tiempo). Es la fracción de diferencias crecientes menos la fracción de diferencias decrecientes (Richards et al., 2011b).

28 - `PercentAmplitude` Porcentaje de amplitud. Máximo porcentaje de diferencia entre la máxima o la mínima magnitud y su mediana (Richards et al., 2011b).

29 - `PercentDifferenceFluxPercentile` $F_{5,95}$ convertido a magnitud (Richards et al., 2011b).

30 - `PeriodLS` o Período de *Lomb-Scargle*. Es primer período candidato extraído con el método de *Lomb-Scargle* (Kim et al., 2011, 2014; VanderPlas, 2018).

31 - `Period_fit` "*Period Fitness*" o "Ajuste del período". Probabilidad de falsa alarma de `PeriodLS`. Mientras más cerca de 1 esté el valor de esta característica, el período estimado es menos confiable (Kim et al., 2011, 2014; VanderPlas, 2018).

32 - `Psi_CS` (Ψ_{CS}) Índice R_CS aplicado a la curva en fase, utilizando el período propuesto en `PeriodLS` (Kim et al., 2011, 2014).

33 - `Psi_eta` (Ψ_η) Índice η^e calculado con la curva de luz puesta en fase con el período candidato obtenido en `PeriodLS` (Kim et al., 2011, 2014).

34 - `Q31` (Q_{3-1}) Es la diferencia entre la magnitud del tercer cuartil y el primer cuartil (Kim et al., 2014).

35 - `Rcs` (R_CS "*Range Cumulative Sum*" es literalmente el rango obtenido luego de calcular la suma acumulada de magnitudes (Kim et al., 2011).

36 - `Skew` Literalmente asimetría de los valores de las magnitudes de curva de luz (Kim et al., 2011).

37 - `SmallKurtosis` Literalmente kurtosis pequeña. Es un índice que en el caso de estar en presencia de una distribución normal el valor es cercano a 0 (Richards et al., 2011b).

38 - `Std` Desviación estándar de las magnitudes (Richards et al., 2011b).

3.6.2. Características propuestas en este trabajo

Las estrellas variables además de por su período, variación de magnitudes y la morfología de su curva de luz; se definen por su color. Esto lo hace una característica importante para clasificarlas. El color es esencialmente una medida de temperatura la cual se obtiene restando dos magnitudes de dos bandas distintas de la misma estrella. El problema que se presenta en los datos fotométricos, y sobre todo dentro del bulbo galáctico, es la presencia de polvo interestelar que distorsiona cualquier índice de color que se desee calcular. Este polvo genera un fenómeno llamado extinción (por la pérdida de luminosidad de la fuente), el cual afecta principalmente a los componentes azules de la luz dejando pasar las frecuencias más bajas, es por esto que se dice que las fuentes se "enrojecen" (en inglés "*reddening*"). En el caso de los datos del *VVV* sus datos no están corregidos por enrojecimiento.

Para eliminar este fenómeno es necesario recurrir a mapas de extinción. En este sentido, los trabajos Gonzalez et al. (2011) y Gonzalez et al. (2012) fueron de utilidad para estimar la extinción de cada una de las fuentes, al utilizar el proyecto "*BEAM - A VVV and 2MASS Bulge Extinction And Metallicity Calculator*" [3]. Lamentablemente al momento de realización de este trabajo no se encontraban disponibles mapas de las zonas correspondientes al VVVx, por lo cual si bien toda la metodología es a priori aplicable, nos centramos en analizar catálogos del relevamiento original.

El último detalle es que los mapas disponibles no tienen la resolución suficiente para encontrar las extinciones de las posiciones de todas las fuentes involucradas en esta tesis. En estos casos se optó por usar como reemplazo el promedio de la extinción de los cien vecinos más cercanos pesados por su distancia.

Así para cada fuente se obtienen índices de extinción, útiles para calcular el enrojecimiento con la fotometría del relevamiento *2MASS* (Skrutskie et al., 2006a) y convertibles a la fotometría del *VVV* (González-Fernández et al., 2017), de modo que se siguen dos leyes de extinción propuestas por los trabajos de Cardelli et al. (1989) y Nishiyama et al. (2009) respectivamente. Estas leyes son relaciones matemáticas entre el brillo de una fuente y la extinción de una zona.

[3] `http://mill.astro.puc.cl/BEAM/calculator.php`

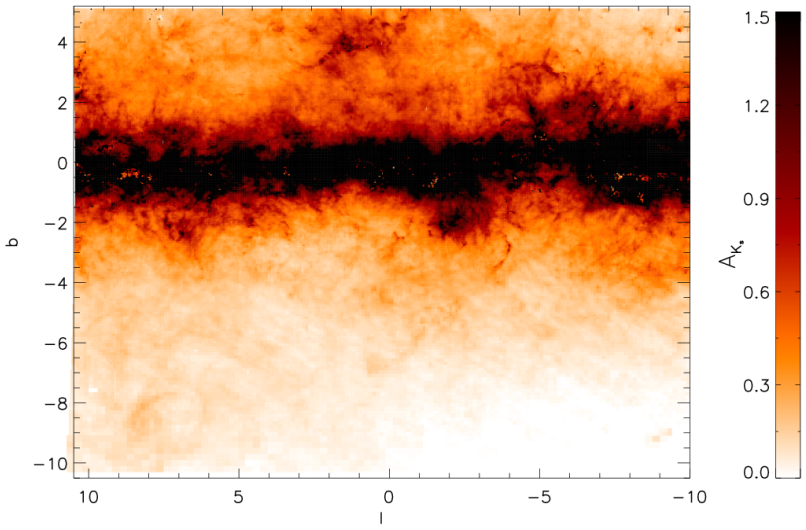

Figura 3.9: Mapa de extinción del bulbo galáctico utilizado por *BEAM* segun la ley de Cardelli et al. (1989), adaptado del trabajo de Gonzalez et al. (2012); donde: b y l son la latitud y longitud galáctica respectivamente, y A_{K_s} es la extinción en banda K_s.

Finalmente, el último escollo para calcular color es que lamentablemente en *VVV* sólo se observa la multi-banda en la primera época, por lo cual esta alternativa no es posible. Esto genera dos problemas:

1. No se puede utilizar la forma clásica de cálculo de índices color de restar dos curvas de luz (implementada en *feets*); ya que requiere varias observaciones en ambas bandas a restar.

2. La primera época de dos curvas de luz, pueden haber sido observadas en fases diferentes, y las relaciones entre la resta de dos puntos de diferentes fases en dos bandas, de incluso la misma estrella, no es constante. (Ver Figura 3.10).

En estas condiciones se optó por seguir la siguiente estrategia:

- Primero se calcularon los colores y se utilizó solo la primer época de observación, de modo que se aplican ambas leyes de extinción.

- Además, se calculó una Pseudo-fase multi-banda, que brinda una idea sobre en qué lugar de la fase se encuentra la primera época de la fuente acompañado del conteo de épocas de la fuente.

- También, se implementaron los pseudo-magnitudes, y pseudo-colores (con ambas leyes de extinción) propuestas en el trabajo de Catelan et al. (2011). Estos valores son realmente índices libres de enrojecimiento.

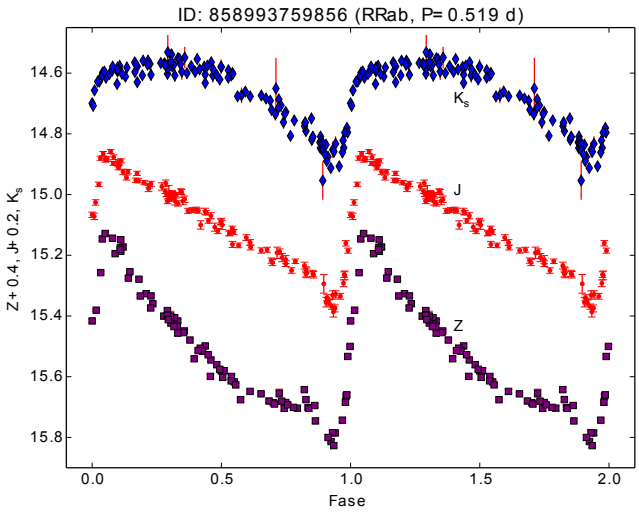

Figura 3.10: Gráfico adaptado del trabajo de Angeloni et al. (2014), en el cual se presenta el objeto `858993759856` del *WFCAM Science Archive* (`http://wsa.roe.ac.uk/`) (Hambly et al., 2008; Cross et al., 2007, 2012) en bandas Z, J y K_s, las tres puestos en fase respecto a la banda K_s. El eje horizontal muestra dos fases para facilitar la percepción de la simétrica; mientras que el eje vertical ajusta las magnitudes de la banda Z y J respecto a la K_s, sumándoles una constante para facilitar su comparación. El objeto es una *RR-Lyrae* con un periodo de 0.51961138 días. Nótese como las amplitudes son más pronunciadas en bandas cercanas al óptico (Z).

- Por último, dado que las amplitudes son magnitudes, se aprovechó la relación calculada para los índices libres de enrojecimiento en el trabajo de Catelan et al. (2013), para convertir la característica `Amplitude` (el cual está en banda K_s) a las bandas J y H, además de calcular sus diferencias. Las cuales son un índice de color enrojecido pero libre de fase.

De este modo, se obtuvieron las siguientes características[4]:

39 - `AmplitudeH` **(A_H)** Amplitud en la banda H derivada del valor de la amplitud en la banda K_s (característica `Amplitude`). Esta relación entre las amplitudes de las diferentes bandas está dada por la fórmula $A_H = 0{,}11 + 1{,}65 \times (A_{K_s} - 0{,}18)$.

40 - `AmplitudeJ` **(A_J)** Amplitud en la banda J derivada del valor de la amplitud en la banda K_s (característica `Amplitude`). Esta relación entre las amplitudes de las diferentes bandas está dada por la fórmula $A_J = -0{,}02 + 3{,}6 \times (A_{K_s} - ,18)$.

41 - `AmplitudeJH` **(A_JH)** Diferencias de amplitud de las bandas J y H. Se puede interpretar a esta característica como un color enrojecido.

42 - `AmplitudeJK` **(A_JK)** Diferencias de amplitud de las bandas J y K. Se puede interpretar a esta característica como un color enrojecido.

43 - `c89_c3` Pseudo-color $C3$ usando la ley de extincion de Cardelli et al. (1989) (Catelan et al., 2011).

44, 45, 46 - `c89_`*`ab`*`_color` Diferencias de magnitud en la primer época entre la banda a y la b usando la ley de extinción de Cardelli et al. (1989). Donde a y b pueden ser las bandas H, J y K_s

47, 48 - `c89_m2, c89_m4` Pseudo-magnitudes $m2$ y $m4$ usando la ley de extinción Cardelli et al. (1989) (Catelan et al., 2011).

49 - `n09_c3` Pseudo-color $C3$ usando la ley de extinción de Nishiyama et al. (2009) (Catelan et al., 2011).

50, 51, 52 - `n09_`*`ab`*`_color` Diferencias de magnitud en la primera época entre la banda a y la b al utilizar la ley de extinción de Nishiyama et al. (2009). Dónde a y b pueden ser las bandas H, J y K_s

53, 54 - `n09_m2, n09_m4` Pseudo-magnitudes $m2$ y $m4$ al usar la ley de extinción Nishiyama et al. (2009) (Catelan et al., 2011).

55 - `cnt` o Conteo. Cantidad de épocas/observaciones de la curva de luz.

56 - `ppmb` Del inglés
"*Pseudo-Phase Multi-Band*" o "Pseudo-Fase Multi-Banda". Este índice pone en fase a la primera época respecto al promedio de tiempos en todas las bandas, utilizando el período calculado por *feets*.

En símbolos:

[4]La numeración continua de la lista de características de *feets*.

$$PPMB = \frac{|frac(mean(HJD_H, HJD_J, HJD_{K_s}) - T_0)|}{P}$$

Dónde HJD_H, HJD_J y HJD_{K_s} son los tiempos de observación en la banda H, J y K_s respectivamente; T_0 es el tiempo de observación en banda K_s de la máxima magnitud; $mean$ es la función que calcula el promedio; $frac$ es una función que retorna solo parte decimal del valor; y P es el período calculado por *Lomb-Scargle*.

3.7. Conclusiones del capítulo

En este capítulo, se presenta el relevamiento astronómico VVV y sus particularidades observacionales; haciendo hincapié en la forma en la cual se utiliza el telescopio VISTA y la cámara VIR-CAM para observar el cielo, con el objetivo de almacenar los datos en imágenes y catálogos fotométricos.

Asimismo, se describió la reconstrucción de curvas de luz de las fuentes observadas que contempla cuestiones relacionadas a la fotometría y astrometría que la afectan.

Finalmente se detalló la extracción de 56 características útiles para el aprovechamiento de las curvas de luz, en algoritmos de aprendizaje automático utilizando dos métodos: uno brindado por la herramienta *feets* Cabral et al. (2018b) (útil para cualquier curva de luz de cualquier relevamiento) y otro que aprovecha las particularidades del VVV (a la vez que se explican conceptos como el enrojecimiento y la extinción en el Bulbo galáctico).

Capítulo 4

Problemas y soluciones relacionados al ambiente de procesamiento de datos del *VVV*

4.1. Objetivos del capítulo

Este capítulo presenta toda la tecnología e ingeniería relacionada con el tratamiento de datos del *VVV*, dado el poder de cómputo disponible para llevar adelante este proyecto.

En primer lugar se presenta un diseño teórico e implementación de un innovador marco de trabajo para la creación de procesamientos de datos secuenciales, que se llevó adelante con el doble objetivo de servir de ambiente de reducción de observaciones para el telescopio TOROS (Diaz et al., 2014) y este trabajo. Después, se contínua con la explicación de un proceso de re-ingeniería sobre un proyecto de extracción de características de curvas de luz, que culminó con la publicación de una nueva herramienta llamada "*feATURES eXTRACTOR for tIME sERIES - feets*" (del inglés "Extractor de Características de Series Temporales") (Cabral et al., 2018b). Finalmente, se presenta la integración de las dos primeras secciones en un único ambiente de trabajo para la extracción de características de series temporales del *VVV*.

4.2. Corral - Marco de trabajo para el procesamiento secuencial de datos

Como ya se mencionó, estamos en una época en la cual el desarrollo de modernos telescopios terrestres y espaciales, que observan todo en el espectro electromagnético, y un creciente interés sobre la variabilidad en el dominio del tiempo, han planteado la necesidad de grandes bases de datos de observaciones astronómicas.

Este fenómeno es una manifestación de la profunda transformación que está atravesando la Astronomía, junto con el desarrollo de nuevas tecnologías en la era de grandes volúmenes de datos; con crecientes demandas sobre: calidad; necesidades de almacenamiento y herramientas de análisis.

Así, el desarrollo de *pipelines*[1] de procesamiento de información es una natural consecuencia de proyectos científicos involucrados en la adquisición de datos y su pre–procesamiento para un posterior análisis.

Algunos incluyen:

- "*The Dark Energy Survey Data Management System*" del inglés "Sistema de procesamiento de datos para el relevamiento de Materia Oscura"(Mohr et al., 2008) diseñado para explotar la naturaleza de la aceleración cósmica, utilizando una cámara de 74 CCDs[2] instalada en el "Telescopio Blanco".

- "*Infrared Processing and Analysis Center*" o "El Centro de Análisis y Procesamiento Infrarrojo"(Masci et al., 2016) una utilidad para el descubrimiento de fuentes transitorias en el observatorio Palomar (iPTF Kulkarni, 2013b).

- Pipeline de procesamiento de imágenes Pan-STARRS PS1(Magnier et al., 2006) que realiza el procesamiento y análisis de imágenes para los datos del telescopio prototipo *Pan-STARRS PS1*.

- Pipeline del Relevamientos del Telescopio Vista (Emerson et al., 2004) el cual genera los datos consumidos en este trabajo.

De hecho, la implementación de pipelines en Astronomía es una tarea común al desarrollo de relevamientos (e.g. Marx and Reyes, 2015; Hughes et al., 2016; Hadjiyska et al., 2013), e incluso es utilizada para la operación remota de telescopios (Kubánek and Jelínek, 2010). En este contexto, herramientas estándar para la generación de *pipelines* se han desarrollado. Algunos ejemplos son Luigi[3], la cual implementa un método para la creación de *pipelines* distribuidos; OPUS (Rose et al., 1995), concebido por el *Instituto de Ciencia del Telescopio Espacial*; y más recientemente Kira(Zhang et al., 2016), una herramienta distribuida centrada en el análisis de imágenes astronómicas.

En esta sección, se presenta un *framework* del lenguaje de programación *Python* para la creación de pipelines astronómicos desarrollados en el contexto de la colaboración TOROS ("Transient Optical Robotic Observatory of the South", Diaz et al., 2014).

El proyecto *TOROS* se dedica a la búsqueda de contrapartes electromagnéticas para eventos de ondas gravitacionales (GW), como respuesta al incipiente desarrollo de la Astronomía de ondas gravitacionales.

TOROS participó en la primera ejecución de observación del interferómetro *LIGO GW*(Abramovici et al., 1992; Abbott et al., 2016) desarrollado desde septiembre de 2015 hasta enero de 2016 con resultados prometedores (Beroiz et al., 2016). Actualmente se intenta desplegar un telescopio óptico de campo amplio en Cordón Macón, en la meseta de Atacama, en el noroeste de Argentina(Renzi

[1]La traducción literal de "*pipeline*" al castellano es "tubería", pero es muy extraño encontrar el termino traducido, así que hemos optado por mantener la palabra original

[2]Dispositivo de carga acoplada (en inglés charge-coupled device, conocido también como CCD) es un componente utilizado en la fotografía digital

[3]Luigi: `https://luigi.readthedocs.io/`

et al., 2009; Tremblin et al., 2012). La colaboración enfrentó el desafío de iniciar un observatorio robótico en el ambiente extremo del Cordón Macón. Dado el aislamiento de esta ubicación geográfica (el sitio está a aproximadamente 4,600 metros sobre el nivel del mar y la ciudad más cercana está a 300 km), la interacción humana y la conectividad a Internet no están disponibles, esto impone una fuerte demanda de requerimientos de procesamiento y almacenamiento de datos mediante pipelines in-situ junto con tolerancia a fallos.

Para lograr esto, proveemos de una formalización de un *framework* para la creación de pipelines basado en el patrón de diseño *Modelo–Vista–Controlador* (MVC), y un paquete de Python de código abierto de licencia BSD-3[4] llamado Corral. El paquete es capaz de crear una capa de abstracción de alto rendimiento sobre un almacén de datos, con manipulación multiproceso, y generación de informes de control de calidad. El *framework* proporciona estructuras orientadas a objetos simples, que aprovechan la potencia del cómputo multiproceso de manera trasparente. Además, Corral extrae las métricas de aseguramiento de la calidad para la ejecución de la *pipeline* que resultan útiles para la depuración de errores.

4.2.1. Pipelines astronómicos

La arquitectura de pipelines típica involucra cadenas de procesos que consumen un flujo de datos, en la cual cada etapa de procesamiento es una salida dependiente de una etapa previa.

De acuerdo con Bowman-Amuah (2004), cualquier formalismo de pipelines debe incluir las siguientes entidades:

Stream (flujo): el *data stream* o flujo de datos representa un flujo continuo de datos producidos por un experimento que necesita transformar y almacenar dichos datos.

Filtros: un punto donde se está realizando una acción atómica en los datos y se puede resumir como etapas de la pipeline donde el flujo de datos sufre transformaciones.

Conectores: los puentes entre dos filtros. Varios conectores pueden converger en un solo filtro, uniendo una etapa de procesamiento con una o más etapas previas.

Ramas: los datos en el flujo pueden ser de diferentes naturaleza y servir a diferentes propósitos, lo que significa que las pipelines pueden alojar grupos de filtros en los que debe pasar todo tipo de datos, así como un conjunto específicos para diferentes tipos de datos Este concepto permite a las *pipelines* la capacidad de procesar datos en paralelo siempre que los datos sean independientes.

Esta arquitectura se usa comúnmente en proyectos experimentales que necesitan manejar grandes cantidades de datos. En opinión del autor es la forma adecuada para gestionar el flujo de datos de los telescopios desde el almacenamiento en su base de datos de observaciones hasta sus etapas finales de análisis

[4] `https://opensource.org/licenses/BSD-3-Clause`

En general, la mayoría de los telescopios u observatorios dedicados tienen al menos un pipeline a cargo de la captura, transformación y almacenamiento de datos a analizar en el futuro (Klaus et al., 2010; Tucker et al., 2006; Emerson et al., 2004). Esto también es importante porque muchos de los grandes relevamientos astronómicos venideros (por ej. LSST, Ivezic et al., 2008) se espera que estén en la escala del PetaByte de información cruda de datos[5]. Esta es la motivación a desarrollar algún tipo de infraestructura para *pipelines*. Esto es tan importante que el *LSST* actualmente mantiene su propia fundación para la creación de *pipelines* de datos (Axelrod et al., 2010).

4.2.2. Frameworks

La mayoría de los grandes proyectos en la industria del *software* parten de una línea de base común definida por un *framework*[6] ya existente.

La idea principal detrás de un *framework* es ofrecer una teoría-metodología que reduce significativamente la repetición de código, de modo que permite al desarrollador extender una funcionalidad ya existente, a la vez que optimiza el tiempo, los costos y otros recursos (Bäumer et al., 1997; Pierro, 2011). Un *framework* también ofrece su propio control de flujo y reglas para escribir extensiones de una manera común, lo que facilita la mantenibilidad del código.

El patrón Modelo-Vista-Controlador (MVC)

Modelo-Vista-Controlador (MVC) es un formalismo originalmente diseñado para definir software con interfaces visuales alrededor de un entorno "dirigido por datos" (DD) (Krasner et al., 1988).

El enfoque MVC fue adoptado con éxito por la mayoría de los *frameworks* para desarrollo web modernos como *Django*[7] (Forcier et al., 2008) o *Ruby on Rails*[8]. La idea principal detrás de este patrón es dividir el problema en tres partes principales:

- los *modelos*, que definen la estructura **lógica** de los datos,
- *controladores*, que definen la lógica para **acceder** y **transformar** los datos por parte el usuario, y
- las *vistas*, a cargo de **presentar** al usuario los datos almacenados en el modelo, administrados por el controlador.

En general, estas tres partes fueron definidas inicialmente por el *Paradigma Orientado a objetos* (OOP, Coad, 1992). La implementación de MVC proporciona módulos autónomos, bien definidos, reutilizables y menos redundantes, todas estas características necesarias para cualquier proyecto de *software* de una gran colaboración, como las pipeline de datos astronómicos.

[5]LSST System & Survey Key Numbers: `https://www.lsst.org/scientists/keynumbers` LSST Petascale R & D Retos: `https://www.lsst.org/about/dm/petascale`

[6]Así como se hizo con la palabra "pipeline", se optó por dejar la palabra original en inglés ya que las traducciones como "marco de trabajo" son inexistente en la bibliografía en español

[7]Django: `https://www.djangoproject.com/`

[8]Ruby on Rails: `https://rubyonrails.org`

Multi-procesamiento y paralelización de *pipelines*

Como se indicó anteriormente, las *pipelines* pueden ser ramificadas cuando la cadena de procesamiento de datos se divide en varias tareas independientes. Esto se puede explotar fácilmente para que la pipeline aproveche al máximo los recursos disponibles en una máquina con varios núcleos. Además, con este enfoque la distribución de tareas dentro de un entorno de red, como en un *cluster* moderno, es simple y directo.

Aseguramiento de calidad de código

La calidad se ha convertido en un componente clave para el desarrollo de software. Según Feigenbaum (1961),

> "La calidad es una determinación del **cliente**, no la determinación de un ingeniero, no una determinación de *marketing*, ni una determinación de la gerencia general. Se basa en la experiencia real del cliente con el producto o servicio, medido contra sus requerimientos - explícito o implícito, consciente o meramente percibido, técnicamente operacional o totalmente subjetivo - y siempre representando un blanco móvil en un mercado competitivo".

En nuestro contexto, un *cliente* no es una sola persona sino el *rol* de quienes definen nuestros requisitos científicos, y los *ingenieros* son los responsables del diseño y desarrollo de una *pipeline* capaz de satisfacer la funcionalidad definida por esos requerimientos. Medir la calidad del software es una tarea que implica la extracción de métricas cualitativas y cuantitativas.

Una de las formas más comunes para medir la calidad del software es con Covertura de código (CC), el cual se base en el concepto de *pruebas unitarias*. El objetivo de las pruebas unitarias es aislar y mostrar que cada parte del programa es correcta (Jazayeri, 2007), mientras que el CC es el porcentaje de código ejecutado por las pruebas unitarias (Miller and Maloney, 1963).

Otra métrica interesante está relacionada con la mantenibilidad del *software*. Aunque esto puede parecer un parámetro subjetivo, puede medirse utilizando una estandarización de estilo de código, y el número de desviaciones de estilo como trazador de mantenibilidad de código.

También se puede definir la calidad en términos del uso y rendimiento del hardware. Esto se conoce comúnmente como *profiling*[9]. Un *profile* de software ayuda a encontrar cuellos de botella en la utilización de recursos de la computadora, como puede ser el uso del procesador y la memoria, los dispositivos de E/S o incluso el consumo de energía (Gorelick and Ozsvald, 2014).

El *profiling* se divide en *profiling de aplicación* y *profiling de sistema*. El primero se restringe al software desarrollado, mientras que el segundo prueba el sistema subyacente (bases de datos, sistema operativo y hardware) en busca de errores de configuración que pueden causar ineficiencias o sobre-consumo (Gregg, 2013).

Existen diferentes técnicas para obtener esta información dependiendo de la Unidad de análisis y método de muestreo de datos. Hay *profilers* que evalúan la aplicación en general (*profile de nivel de aplicación*), en cada llamada a función

[9]La traducción de *profiling* es "perfilado" pero el término no es usado normalmente

(*profile de nivel de función*) o cada línea de código (*profile de línea*) (Gregg, 2013).

Otra clasificación divide al *profiling* en *determinista* y *estocástico* (Roskind, 2007) (Schneider, 2000). Los evaluadores deterministas muestrean los datos en todo momento, mientras que los estocásticos registran los datos a intervalos regulares, tomando nota de la función actual y la línea de ejecución (si algo es muy lento se va a muestrear más veces que algo que rara vez va a "caer" en el intervalo de evaluación por su velocidad).

A diferencia de la unidad de análisis, que se decide de antemano (y generalmente se modifica durante el análisis) la elección de un *profiler* determinista se basa en la necesidad de recuperar mediciones precisas a costa de la velocidad, ya que pueden disminuir el tiempo de ejecución de la aplicación hasta en un factor de diez. Por otro lado, los estocásticos ejecutan el software a una velocidad casi normal.

4.2.3. Resultados: Un *framework Python* para implementar *pipelines* reproducibles basado en Modelos, Etapas y Alertas

Diseñamos un proceso dirigido por datos basado en MVC para generar *pipelines* para aplicaciones en astronomía que soportan métricas de calidad mediante pruebas de unidad y cobertura de código. Está compuesto por varias partes, cada una consistente con una funcionalidad establecida por *pipelines* astronómicas tradicionales, con soporte de bifurcaciones para el procesamiento en paralelo de forma natural. Este diseño fue implementado sobre el paradigma OO, el lenguajePython y bases de datos relacionales. El proyecto se denomina *Corral*[10] y posee una licencia BSD-3 de código abierto.

Las herramientas utilizadas

Como se mencionó anteriormente, *Corral* se implementa en el lenguaje de programación Python, el cual tiene un vasto ecosistema de bibliotecas científicas como NumPy, SciPy (Van Der Walt et al., 2011), *Scikit-Learn* (Pedregosa et al., 2011), acompañado de una sintaxis muy potente y muy simple.

La mayoría de los astrónomos eligen *Python* como su herramienta principal para el procesamiento de datos, de modo que favorecen la existencia de bibliotecas como *AstroPy* (Robitaille et al., 2013a), *CCDProc*[11], *PhotUtils*[12] (Tollerud, 2016), etc. También vale la pena mencionar que *Python* alberga una serie de bibliotecas para el paralelismo, herramientas de interacción con línea de comando y diseño de casos de prueba, que son útiles para una traducción de nuestras ideas a software real.

Otro requisito clave es el almacenamiento y recuperación de datos en múltiples procesos, lo que lleva a utilizar *Sistemas de gestión de bases de datos relacionales* (RDBMS). Los RDBMS son un estándar probado para la gestión de datos y han existido durante más de treinta años. Son compatibles con un número importante de implementaciones y vale la pena mencionar que muchas de las más utilizadas son *Open Source*, por ejemplo, *PostgreSQL*.

[10]Página de inicio de Corral: `https://gitub.com/toros-astro/corral`
[11]CCDProc: `http://ccdproc.readthedocs.io`
[12]PhotUtils:`http://photutils.readthedocs.io/en/stable/`

SQL es un potente lenguaje de programación para RDBMS y ofrece ventajas en la consistencia de los datos y en las consultas de búsqueda. *SQL* tiene un amplio espectro de implementaciones: desde más pequeñas, aplicaciones locales, accesibles desde un solo proceso, como *SQLite* (Owens and Allen, 2010), a soluciones distribuidas en *clusters* de computadoras, capaces de servir a miles de millones de peticiones, como *Apache-Hive* (Thusoo et al., 2009). Esta variedad de opciones permite flexibilidad en la creación de pipelines: desde pequeñas que corren en una computadora personal, hasta otras desplegadas en los *clusters* de computación que alojan grandes volúmenes de datos y múltiples usuarios. Dentro del ecosistema de *Python*, la biblioteca *SQLAlchemy* [13] ofrece la posibilidad de diseñar esquemas de modelos de una manera bastante simple y al mismo tiempo ofrecer suficiente flexibilidad para no causar problemas relacionados con dialectos de *SQL* específicos. Por lo tanto, ofrece un buen compromiso para satisfacer las diferentes necesidades del desarrollador.

Desde el Modelo-Vista-Controlador al Modelo-Etapas-Alertas

Para cerrar la brecha entre la terminología tradicional de MVC y la que se usa para describir la arquitectura de las *pipelines*, algunos términos (por ejemplo, Vistas y Controladores) se han redefinido en este trabajo, para hacer que el código sea más descriptivo tanto para los programadores como para los científicos.

Modelos define protocolos para el *flujo* de la *pipeline*. Definen la estructura de datos para la ingesta inicial de información, productos intermedios y conocimiento final del proceso. Dado que el marco es dirigido por datos, cada etapa del proceso consume y carga datos a través de los modelos que actúan como canales de comunicación entre los diferentes componentes de la *pipeline*. Los modelos pueden almacenar datos o meta-datos.

Etapas (Internamente, en el código, a las etapas las llamamos "*Steps*") Al igual que los modelos, los *Steps* están definidos por clases, y en este caso actúan como *filtros* y *conectores*. Hay dos tipos diferentes de etapas utilizadas por Corral: *loaders* (o cargadores), y *steps* (o pasos) .

- **Loaders** son responsables de alimentar el *pipeline* en su etapa más temprana. Por elección de diseño, una proyecto construido sobre *Corral* solo puede tener un cargador.
- **Steps** selecciona datos del *stream* imponiendo restricciones, y carga los datos nuevamente después de alguna transformación. Es evidente que estos pasos son filtros y conectores al mismo tiempo, que además incluyen implícitamente el concepto ramificación.

Alertas Para definir *vistas* tomamos los conceptos de Alertas tal como se utilizan en algunas aplicaciones astronómicas, las cuales son utilizadas para experimentos de seguimiento, como inspiración para diseñar las *vistas*. En *Corral* las *views* se llaman *Alerts* y son eventos especiales desencadenados por un estado particular de los datos.

[13] SQLAlchemy: url http://www.sqlalchemy.org/

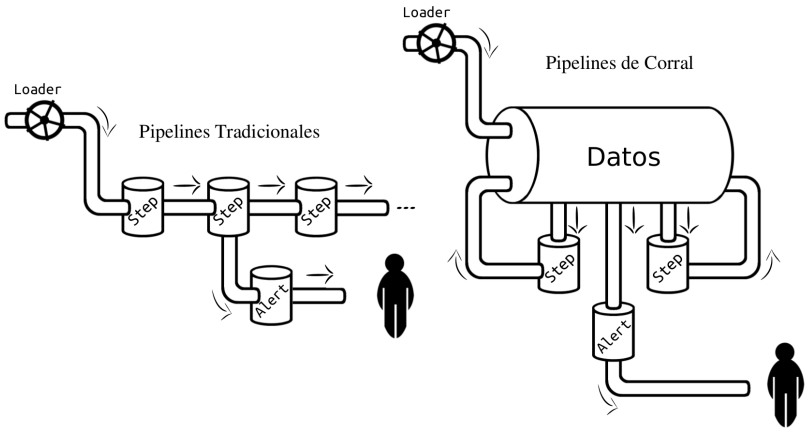

Figura 4.1: Flujo de datos en la arquitectura tradicional de *pipelines* (izquierda) y el modelo de *pipeline* de *Corral* (derecha). En los modelos de *pipelines* clásicos, los datos se procesan secuencialmente, de un paso (*step*) por vez, con una eventual ramificación en paralelo; mientras que en el modelo propuesto en este trabajo las etapas son entidades independientes que interactúan por compartir un contenedor central de datos. En ambas arquitecturas se pueden extraer datos cuando se desee dada una serie de condiciones, y pueden ser compartidas en cualquier momento a los usuarios.

Algunos ejemplos de códigos *Python* relacionados con cada uno de estos "procesadores" se pueden encontrar en el apéndice A el cual implementa dos *pipelines* simples a modo ilustrativo. Una imagen completa del *framework* y los elementos definidos por el usuario de la *pipeline* junto con sus interacciones, se muestra en la Figura 4.1.

Algunas palabras sobre multiproceso y cómputo distribuido

Como se mencionó anteriormente, nuestro proyecto está construido sobre un RDBMS, que por diseño se puede acceder simultáneamente desde una computadora local o de red. Cada etapa accede solo a la parte filtrada del flujo, así que con el mínimo esfuerzo uno puede desplegar el *pipeline* en uno o más ordenadores y ejecutar las etapas en paralelo. Dado que las RDBMS garantizan la propiedades ACID (Atomicidad, consistencia, aislamiento y durabilidad) de las transacciones de la base de datos, se evitan todos los peligros de la corrupción de datos. Este enfoque funciona bien en la mayoría de los escenarios, pero hay algunos inevitables inconvenientes, que surgen, por ejemplo, en el procesamiento de *tiempo real*, donde la consistencia es menos importante que la disponibilidad. Corral aprovecha esta propiedad de estas tecnologías para iniciar un proceso para cada etapa o alerta dentro de la *pipeline*, ya que permite que varios agentes interactúen al mismo tiempo con los datos almacenados en la base de datos sin generar problemas. De ser necesario, los grupos de procesos se pueden distribuir manualmente en los nodos de un *cluster* ya que pueden interactuar con la base de datos de forma remota.

Vale la pena señalar que cualquier comunicación directa entre procesos está prohibida por diseño, y la única forma de intercambiar mensajes es a través de la base de datos.

Calidad - *Pipelines* confiables

Un requisito importante para una *pipeline* es la confiabilidad de sus resultados. Una verificación manual de los datos es prácticamente imposible cuando su volumen se acerca al Tera-Byte. En nuestro enfoque, sugerimos un enfoque simple de pruebas unitarias para verificar el estado de la secuencia antes y después de cada *Step*, *Loader* o *Alert*.

Debido a que las pruebas son unitarias, *Corral* garantiza el aislamiento de cada una, al crear y destruir la base de datos del flujo antes y después de la ejecución de cada prueba. Si se alimenta el flujo de datos de la *pipeline* con una cantidad suficiente de datos heterogéneos, se puede verificar la mayor parte de la funcionalidad de la *pipeline* antes de desplegarlo en producción científica.

Finalmente, *Corral* proporciona capacidades para crear informes con toda la información de calidad y estructural del proyecto desarrollado, al igual que brinda una pequeña herramienta para ejecutar un *profile* de rendimiento de la CPU durante su ejecución.

Índice de Aseguramiento de Calidad (QAI)

Se reconoce la necesidad de un único valor para cuantificar la calidad de un *pipeline*. Así, se utilizan diferentes estimadores para la estabilidad y la capacidad de mantenimiento del código. En este caso, se llegó al siguiente Índice de Calidad QAI (del inglés "*Quality Assurance Index*" que es literalmente Índice de Aseguramiento de Calidad):

$$QAI = \frac{\Theta \times \Lambda_{Cov} \times R_{PT}}{\gamma}$$

donde γ es un factor de penalización definido como:

$$\gamma = \frac{1}{2} \times \left(1 + exp\left(\frac{N_{SError}}{\tau \times N_f}\right)\right)$$

Θ es 1 si todos las pruebas unitarias pasan o 0 si alguna falla, R_{PT} es la proporción de procesadores probados (siendo los procesadores: *Loader, Steps and Alerts*) dividido el total de procesadores, Λ_{Cov} es la cobertura de código (entre 0 y 1), N_{SError} es el número de errores de estilo, τ es la tolerancia de errores de estilo, y N_f es la cantidad de archivos en el proyecto.

El factor Θ es un parámetro crítico del índice, ya que es discreto, y si una sola prueba unitaria falla, establecerá el QAI en cero, en el espíritu de que si tus propias pruebas fallan, entonces ningún resultado está garantizado para ser reproducible. El factor R_{PT} es una medida de cuántas de las diferentes etapas de procesamiento están siendo probadas, un valor bajo de este parámetro debe interpretarse como una necesidad de escribir nuevas pruebas para cada etapa del *pipeline*. Λ_{Cov} e el porcentaje de código ejecutado por las pruebas unitarias (CC). De alguna manera es equivalente a la calidad de las pruebas en sí, ya que bajos valores de este índice indican que las pruebas en realidad ejecutan pocas partes del código total de proyecto; o que hay código sin utilizar. En general si se

poseen muchas pruebas pero un Λ_{Cov} bajo se debe a que no se están probando más que los flujos habituales de ejecución.

$N_{NSerr}/(\tau \times N_f)$ es el factor de escala para la función exponencial. Involucra la información sobre errores de estilo, atenuada por la tolerancia τ multiplicada por el número de archivos en el proyecto N_f.

La función exponencial expresa el hecho de que un cierto número de los errores de estilo no son críticos, pero después de algún punto esto compromete seriamente la mantenibilidad de la *pipeline*, y en esta situación γ penaliza fuertemente el índice de calidad.

El factor $1/2$ es una constante de normalización, por lo que $QAI \in [0, 1]$. Este índice apunta a codificar en una sola figura de mérito qué tan bien la *pipeline* cumple los requisitos especificados por el usuario. Observamos que este índice representa una métrica de confianza. Por lo tanto, una *pipeline* podría ser completamente funcional incluso si todas las pruebas fallan (o si aún no hay pruebas escritas para ello). El índice QAI intenta responder la pregunta de la confiabilidad-subjetiva de la *pipeline* en un solo valor.

Determinación de la tolerancia de errores de estilo (τ) por defecto en un proyecto Python

Corral, como se mencionó anteriormente, está escrito en *Python*, que ofrece una serie de bibliotecas de terceros para validación de estilo. El estilo de código del lenguaje está estandarizado por Documento *PEP 8*[14].

Para la medición de QAI de *Corral*, un detalle clave estuvo en la determinación de la cantidad de errores de estilo que los desarrolladores registraron con mayor frecuencia. Para esto, recolectamos datos de casi 4000 archivos públicos de código fuente *Python*, a los cuales se les cálculo la cantidad de errores de estilos utilizando la herramienta *Flake8*[15], y a estos valores se le calculó la media inter-quartil. Se obtuvo con esto el valor ~ 13 el cual es propuesto por nosotros como nuestro τ por defecto.

Es importante tener en cuenta que este valor puede ser cambiado por el usuario si se requiere un QAI más estricto (Fig. 4.2).

Informes estructurales y de calidad

Corral incluye la posibilidad de automáticamente generar documentación, creando un manual y reportes de calidad en formato Markdown[16]. Este formato es fácilmente convertible a *PDF* (ISO., 2005), LaTeX(Lamport, 1994) o *HTML* (Price, 2000). También se ofrece una herramienta para generar los diagramas de clase (Booch et al., 2006) para los modelos de la *pipeline*, utilizando *Graphviz*[17].

Profiling

Corral ofrece una herramienta determinista para el análisis del rendimiento de uso de la CPU a nivel de función, que muestra los tiempos y frecuencia de las llamadas de función, así como la cadena de llamadas, durante la ejecución de pruebas unitarias. Vale la pena señalar que en caso de que una *pipeline* muestre

[14]PEP8: `https://www.python.org/dev/peps/pep-0008`
[15]Flake8: `http://flake8.pycqa.org/`
[16]Markdown: `https://daringfireball.net/projects/markdown/`
[17]Graphviz: `http://www.graphviz.org/`

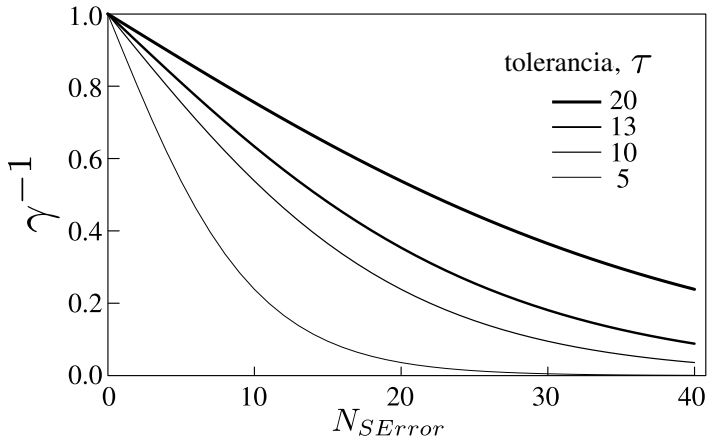

Figura 4.2: Caso donde todos las pruebas unitarias no-fallan con una cobertura de código total ideal. El eje horizontal son los errores de estilo entre 0 y 40 ($N_{SErr} \in [0, 40]$), y el vertical es la inversa de la penalización (γ^{-1}). Puede observarse una caída en la pendiente de error a medida que *tau* crece.

signos de retraso o al disminuir la velocidad, ejecutar un perfilador en una sesión de prueba unitaria puede ayudar a localizar cuellos de botella. Sin embargo, para análisis rigurosos, se deben usar datos reales y procesamiento real.

Palabras finales sobre la calidad de Corral

El *framework* no contiene ninguna funcionalidad para analizar causas de error, o reintentar etapas fallidas. Cada procesador debe poder manejar correctamente sus datos y su ciclo de vida y condiciones. Las herramientas de calidad se ofrecen con la intención de que si el desarrollador de la *pipeline* logra una alta cobertura de código y puede probar con una heterogeneidad de datos, los posibles errores de software pueden disminuir considerablemente, hasta 80 % (Jeffries and Melnik, 2007).

4.2.4. Ejemplo simple de pipeline para convertir coordenadas (x, y) en ecuatoriales (RA, Dec)

A continuación se dan algunos ejemplos para ilustrar cada parte de una *pipeline* rudimentaria construida sobre *Corral*. Además se encuentra disponible el tutorial completo del proyecto ubicado en `http://corral.readthedocs.io`.

Instalación de Corral

El método de instalación recomendado para obtener Corral es utilizar *pip*[18]:

```
$ pip install -U corral-pipeline
```

También son posibles otros métodos, y se detallan en la documentación[19].

[18] Pip es la herramienta estandar para la instalación de paquetes en *Python* `https://pip.pypa.io`

[19] http://corral.readthedocs.io/en/latest/install.html

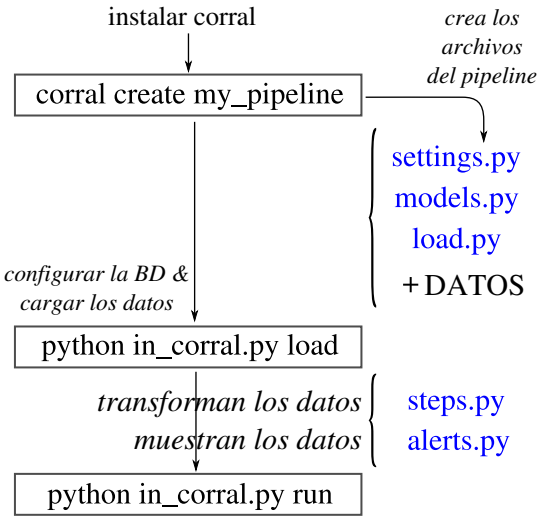

Figura 4.3: Esquema básico del flujo de trabajo para construir una pipeline utilizando Corral

Pasos para crear una *pipeline* con *Corral*

El flujo de trabajo para crear una *pipeline* simple se puede resumir de la siguiente manera (Figura 4.3):

- Crear la pipeline
- Definir los modelos
- Crear la base de datos para los datos.
- Cargar los datos
- Definir los pasos
- Ejecutar la pipeline

Creación de la *pipeline*

Si se ejecuta:

```
$ corral create my_pipeline
[INFO] Creating pipeline in '/home/juan/proyectos/corral/my_pipeline'...
[INFO] Created 'my_pipeline/in_corral.py'.
[INFO] Created 'my_pipeline/my_pipeline/steps.py'.
[INFO] Created 'my_pipeline/my_pipeline/models.py'.
[INFO] Created 'my_pipeline/my_pipeline/pipeline.py'.
[INFO] Created 'my_pipeline/my_pipeline/tests.py'.
[INFO] Created 'my_pipeline/my_pipeline/load.py'.
[INFO] Created 'my_pipeline/my_pipeline/settings.py'.
[INFO] Created 'my_pipeline/my_pipeline/alerts.py'.
```

```
[INFO] Created 'my_pipeline/my_pipeline/__init__.py'.
[INFO] Created 'my_pipeline/my_pipeline/commands.py'.
[INFO] Created 'my_pipeline/my_pipeline/migrations/alembic.ini'.
[INFO] Created 'my_pipeline/my_pipeline/migrations/env.py'.
[INFO] Created 'my_pipeline/my_pipeline/migrations/versions/README.txt'.
[INFO] Pipeline my_pipeline Ready!!!
```

Se creará una estructura de directorios como la siguiente:

```
my_pipeline
├── my_pipeline
│   ├── migrations/
│   │   └── ...
│   ├── settings.py
│   ├── models.py
│   ├── load.py
│   ├── steps.py
│   ├── alerts.py
│   ├── tests.py
│   └── ...
└── in_corral.py
```

Cada uno de estos archivos (más algunos que se han obviado por propósitos didácticos) contienen una parte de la *pipeline*, la cual es necesario modificar.

Ejemplo de *Modelos*

En el siguiente ejemplo se muestra un modelo para una fuente astronómica. Usa las coordenadas (x, y) y (RA, Dec) y un campo para almacenar la magnitud aparente. El campo de identificación (ID) se resuelve automáticamente.

```
# este código se ubica dentro de mypipeline/models.py
from corral import db

class PointSources(db.Model):
  "Model for star sources"
  __tablename__ = "PointSources"

  id = db.Column(
    db.Integer, primary_key=True)
  x = db.Column(db.Float, nullable=False)
  y = db.Column(db.Float, nullable=False)
  ra = db.Column(db.Float, nullable=True)
  dec = db.Column(db.Float, nullable=True)

  app_mag = db.Column(db.Float, nullable=False)
```

Ejemplo de un *Loader*

En el siguiente ejemplo se muestra un cargador - *loader*, donde el modelo del ejemplo anterior (`PointSource`) es llenado con nuevos datos. Nótese

que los campos (RA, Dec) en el Modelo de `PointSource` pueden ser nulos (`nullable=True`), y por lo tanto no es necesario establecerlos a priori.

```
# este código se aloja en mypipeline/load.py
from astropy.io import ascii
from corral import run

from mypipeline import models

class Load(run.Loader):
  "Cargador del pipeline"
  def setup(self):
    self.cat = ascii.read(
        'point_sources_cat.dat')

  def generate(self):
    for source in self.cat:
      src = models.PointSource()
      src.x = source['x']
      src.y = source['y']
      src.app_mag = source['app_mag'])
      yield src
```

Ejemplo de *Step*

A continuación se muestra un ejemplo de *step*, donde se realiza una transformación de datos simple. El *step* toma un conjunto de fuentes y transforma sus coordenadas (x, y) a (RA, Dec). Las líneas 10-12 muestran definiciones de atributos de nivel de clase las cuales son los responsables de realizar la consulta. Luego, *Corral* recupera los datos y los sirve al método `process` (línea 14), que ejecuta la transformación.

```
1  # este código se ubica dentro del módulo mypipeline/step.py
2  from corral import run
3  from mypipeline import models
4
5  import convert_coord
6
7  class StepTransform(run.Step):
8    "Step para transformar from x,y to ra,dec"
9
10   model = models.PointSource
11   conditions = [model.ra == None,
12                 model.dec == None]
13
14   def process(self, source):
15     x = source.x
16     y = source.y
17
18     source.ra, source.dec = convert_coord(x, y)
```

Ejemplo de *Alert*

En el siguiente ejemplo, se activa una alerta cuando un estado de un modelo satisface una condición particular de los datos en el flujo.

En este caso particular un correo electrónico y un telegrama astronómico (Rutledge, 1998) serán emitidos siempre que se detecte una fuente en la vecindad de Sgr A* [20], cerca del centro de la galaxia.

```
# código dentro de mypipeline/alert.py
from corral import run
from corral.run import endpoints as ep

from mypipeline import models

class AlertNearSgrA(run.Alert):

  model = models.PointSource
  conditions = [
    model.ra.between(266.4, 266.41),
    model.dec.between(-29.007, -29.008)]
  alert_to = [ep.Email(["sci1@sci.edu",
                        "sci2@sci.edu"]),
              ep.ATel()]
```

Ejecución de la *Pipeline*

Ya definido el protocolo para la transmisión y las acciones a realizar para transformar las coordenadas; se puede apreciar que nunca escribimos el código que corre la *pipeline*.

Sin embargo, por haber diseñado esta *pipeline* con el patrón equivalente al MVC provisto por *Corral*; se tienen garantías de que todas las etapas se ejecutarán de manera independiente, ya que que todas las tareas se realizarán siempre que haya datos pendientes de procesamiento.

En cada *Loader / Step / Alert* hay una garantía de consistencia de los datos, ya que cada uno utiliza una sesión de *SQLAlchemy*, por lo que cada tarea está vinculada a una transacción de base de datos. Como cada transacción de SQL es atómica, entonces el proceso se ejecuta o falla, lo que reduce el riesgo de corrupción de datos en la transmisión.

Si deseamos entonces correr toda la *pipeline*:

```
$ python in_corral.py run-all
[mypipeline-INFO@2017-01-12 18:32:54,850]
    Executing Loader
    '<class 'mypipeline.load.Load'>'
[mypipeline-INFO@2017-01-12 18:32:54,862]
    Executing Alert
    '<class 'mypipeline.alert.AlertNearSgrA'>'
[mypipeline-INFO@2017-01-12 18:32:54,874]
    Executing Step
    '<class 'mypipeline.load.StepTransform'>'
```

[20] Sgr A* es el agujero negro super-masivo en el centro de nuestra galaxia

```
[mypipeline-INFO@2017-01-12 18:33:24,158]
    Done Alert
    '<class 'mypipeline.alert.AlertNearSgrA'>'
[mypipeline-INFO@2017-01-12 18:34:57,158]
    Done Step
    '<class 'mypipeline.load.StepTransform'>'
[mypipeline-INFO@2017-01-12 18:36:12,665]
    Done Loader
    '<class 'mypipeline.load.Loader'>' #1
```

Se puede ver en las marcas de tiempo de las ejecuciones y finalizaciones de tareas para cada *Loader / Step / Alert*, que no hay un orden relevante entre ellos y que todos corren en paralelo.

Comprobando la calidad del *pipeline*

Pruebas unitarias Siguiendo las pautas mencionadas de Feigenbaum (1961) que declara que la calidad es una especificación definida por el usuario; en *Corral* esta especificación se logra al escribir una prueba unitaria. A continuación se incluye un ejemplo que muestra un caso de prueba básico para nuestro *Step*. La prueba alimenta el flujo con algunos datos simulados definidos por el usuario, luego ejecuta el *Step* y finalmente verifica si el estado del resultante de los datos es el esperado.

```
1  # código de mypipeline/tests.py
2  from corral import qa
3
4  from mypipeline import models, steps
5
6  import convert_coord
7
8  class TestTransform(qa.TestCase):
9    "Test the StepTransform step"
10   subject = steps.StepTransform
11
12   def setup(self):
13     src = models.PointSource(
14       x=0, y=0, app_mag=0)
15     self.ra, self.dec = convert_coord(0, 0)
16     self.save(src)
17
18   def validate(self):
19     self.assertStreamHas(
20       models.PointSource,
21       models.PointSource.ra==self.ra,
22       models.PointSource.dec==self.dec)
23     self.assertStreamCount(
24       1, models.PointSource)
```

Como se muestra en las líneas 12-16, el método `setup()` está a cargo de crear nuevos datos, cuyo resultado transformado ya se conoce, enton-

ces `validate()` evalúa el resultado de `StepTransform`. Este proceso se repetiría si se definieran más pruebas. Hay que recordar que los datos simulados son privados para cada prueba de unidad, así nunca colisionarán entre ellos de modo que produzcan resultados inesperados.

Reporte de calidad y *profiling* Corral proporciona funcionalidades incorporadas para comunicar información de control de calidad:

1. `create-doc`: este comando produce en versión *Markdown* un manual generado automáticamente para la *pipeline*. Incluye información sobre modelos, etapas, alertas e interfaz de línea de comandos; esto lo logra utilizando las cadenas de documentación provistas en el propio código.
2. `create-models-diagram`: Esto crea un Diagrama de clases de los modelos (Booch et al., 2006) en formato de *Graphviz* (Ellson et al., 2001).
3. `qareport`: ejecuta cada prueba y evaluación de cobertura de código, y utiliza los resultados de esto para crear un documento de *Markdown* que detalla los estados resultantes de cada etapa de prueba, y finalmente calcula el índice QAI.
4. `profile`: ejecuta todas las pruebas existentes y despliega una interfaz web interactiva para evaluar el rendimiento de diferentes partes de la *pipeline*.

Con estos cuatro comandos, el usuario puede obtener un informe detallado sobre estructuras, estado, y una medición global del nivel de calidad de la *pipeline*.

4.3. feets - Extractor de Características de Series Temporales

El aprendizaje automático (ML) ha demostrado ser una herramienta importante para el análisis de datos en astronomía. Numerosos proyectos como *AstroML* (VanderPlas et al., 2014), *UPSILoN* (Kim and Bailer-Jones, 2016), *MeSsI* (de los Rios et al., 2016), y "*Features Análisis for time Series*" (del ingles análisis de características para series de tiempo, FATS) (Nun et al., 2015) se han desarrollado para ayudar a los astrónomos a utilizar el enfoque ML. Lenguajes de programación como *R*[21] y *Python*[22] proporcionan un gran número de paquetes listos para usar en estos contextos. Sin embargo, si realizamos un análisis en profundidad de estos sistemas, es evidente que algunos proyectos carecen de un diseño de ingeniería de software adecuado (Cowling, 1998), y por lo tanto, a menudo son difíciles de probar, mantener y extender, y algunos tienen un pobre rendimiento en términos de velocidad y memoria. Para complejizar más la situación, muchos de estos proyectos se ejecutan en entornos críticos y procesan grandes volúmenes de datos, por lo cual pueden ser ineficientes si son

[21] `https://www.r-project.org/`
[22] `https://www.python.org/`

optimizados incorrectamente. Por lo tanto, en términos simples, el código de muchos sistemas científicos "huelen mal" (Tufano et al., 2015).

Según Fowler, un mal olor en el código es:

> ... una indicación superficial que usualmente corresponde a un problema más profundo en el software; y actualmente se sabe que *la mayoría de las veces los artefactos de código se ven afectados por los llamados "malos olores" desde su creación* ... (Fowler, 2006).

En estos casos, la solución más pragmática es reemplazar la mayoría del código por una implementación superior, pero evitando cualquier cambio funcional. Este tipo de proceso se llama *refactorización de código* (Fowler and Beck, 1999).

En esta sección, empleamos un proceso de refactorización de código para proporcionar una biblioteca de extracción de características de series de tiempo más robusta basada en el proyecto *FATS* (Nun et al., 2015). La versión actual de FATS (1.3.6) [23] está escrita enteramente en *Python* y basada en las bibliotecas numéricas *Numpy* (Walt et al., 2011), *Scipy* (Oliphant, 2007), y *StatsModels* (Seabold and Perktold, 2010).

FATS puede extraer hasta 64 características de las series temporales, y también incluye funciones de preprocesamiento e importación del relevamiento *MACHO* (Cook et al., 1995). Asimismo, el proyecto cuenta con un tutorial[24] muy bueno, pero carece de documentación interna del código, lo cual es crucial cuando se desea agregar funciones a esta herramienta. En particular, identificamos varias limitaciones, cuando intentamos usar *FATS* para clasificar estrellas variables periódicas en nuestro conjunto de datos, por lo cual se desarrolló una actualización, que fue diseñada cuidadosamente para aprovechar los puntos fuertes del proyecto original reutilizando lo más posible el código y documentación.

4.3.1. Ingeniería de características

Los algoritmos de aprendizaje automático pueden aplicarse a grandes volúmenes de datos para mejorar su rendimiento en una tarea determinada (Samuel, 1959), entre las cuales se encuentran clasificación, regresión, optimización o agrupación(Michalski et al., 2013). Los datos empleados pueden provenir de una amplia gama de fuentes, y se representan en valores llamados características. El proceso realizado para definirlas es simplemente llamado *ingeniería de características*

Es importante notar la naturaleza altamente específica del proceso de ingeniería, que hace que sea costoso, difícil y lento; además de que se necesita una amplia experiencia en el área de aplicación. Todas estas cuestiones hacen que la ingeniería de características sea el paso más crítico en una proyecto de aprendizaje automático (Ng, 2013).

4.3.2. FATS

La herramienta FATS se utiliza para extraer características de los datos de series temporales. En particular, el proyecto tiene como objetivo estandarizar el proceso de extracción de características de curvas de luz astronómicas. Como se

[23] `https://pypi.org/project/FATS`

[24] `http://isadoranun.github.io/tsfeat/FeaturesDocumentation.html`

mencionó, está construido sobre la pila científica de *Python* (*Numpy* y *Scipy*) agregando la librería *StatsModels* para análisis estadísticos adicionales.

Ejemplo de uso de FATS

Se agrega a continuación fragmentos de la documentación traducidas del original para ilustrar los componentes y manera de operar de la librería.

> La biblioteca recibe como entrada los datos de la serie de tiempo y regresa como salida una matriz con las características calculadas la que depende de la entrada disponible.
>
> Por ejemplo, si el usuario tiene solo los parámetros referentes a la magnitud y al tiempo, solo se podrán computar las características que **solo dependan** de estos dos datos.
>
> Para calcular todas las características posibles los siguientes vectores son necesarios por curva de luz:
>
> - magnitude
> - time
> - error
> - magnitude2
> - aligned_magnitude
> - aligned_magnitude2
> - aligned_time
> - aligned_error
> - aligned_error2
>
> Donde **2** se refiere a una banda de observación diferente.
> Cabe señalar que el vector de magnitud es la única entrada que es estrictamente requerida por la biblioteca, ya que es requerida por todas las características. Por lo tanto, si el usuario no tiene estos datos adicionales o se están analizando series de tiempo distintas, todavía es posible calcular algunas de las características...

Ilustramos este punto con el siguiente código de Python, que calcula características para una curva de luz generada aleatoriamente.

Sólo se utilizan datos de magnitud y tiempo, por lo que se obtendrá el conjunto de características más pequeño. Este código también se basa en el tutorial de *FATS*.

```
>>> import numpy as np
>>> import FATS

# Se generan los valores al azar
>>> magnitude_ex = np.random.rand(30)
>>> time_ex = np.arange(0, 30)

# Se crea el arreglo de la curva de luz con el mismo
# orden que en la lista antes descrita.
```

```
>>> lc_example = np.array([magnitude_ex, time_ex])


# Se establece el feature-space (esto objeto sirve
# como punto de entrada para extraer todas las características)
# de modo que se especifican los datos disponibles
# by specifying the available data
>>> fs = FATS.FeatureSpace(
...     Data=['magnitude','time'])
Warning: the feature Beyond1Std could not be
   calculated because
  ['magnitude', 'error'] are needed.
  ...
Warning: the feature CAR_mean could not be
   calculated because
  ['magnitude', 'time', 'error'] are needed.

# Se calculan las características
>>> fs.calculateFeature(
...   lc_example).result("dict")
{'Amplitude': 0.46422830004583993,
 'AndersonDarling': 0.69055170152838952,
 'Autocor_length': 1.0,
 'Con': 0.0,
 'Eta_e': 1.2660816816234817,
 ...
 'Rcs': 0.21319202165853643,
 'Skew': 0.19709543035122007,
 'SmallKurtosis': -0.93310208660425609,
 'Std': 0.28904604267318268}
```

Funcionalidades

Además del ejemplo anterior, *FATS* proporciona funcionalidades para la extracción de características, el preprocesamiento de series de tiempo e importación de curvas de luz del relevamiento MACHO. A continuación, se detallan los alcances de dichas funcionalidades:

Marco de trabajo para la extracción de características FATS incluye una herramienta simple para crear su propio extractor de características.

Cada extractor de características es una clase de python dentro del módulo `FATS.FeatureFunciontLib`. Esta clase, debe contener el objeto *atributes* que define cuáles de los componentes del vector de datos son requeridos, y un método `fit()` utilizado para calcular la característica.

El siguiente código es un ejemplo de una clase que devuelve el número de observaciones en una serie de tiempo

```
class Count(Base):
  def __init__(self):
    self.Data = ['time']
```

```
  def fit(self, data):
    # Se calculan los tiempos
    # del vector de datos
    time = data[1]

    # Se retorna la longitud
    # del vector de tiempo
    return len(time)
```

A continuación, el extractor se puede utilizar de la siguiente manera:

```
# Se especifica sólo el extractor
# recién creado
>>> fs = FATS.FeatureSpace(
...   featureList=["Count"])

# Se calculan las características
>>> fs.calculateFeature(
...   lc_example).result("dict")
{'Count': 30}
```

Preprocesamiento de series de tiempo: Dos funciones están integradas para el preprocesamiento de datos de la curva de luz: La primera es una funcionalidad de *sigma-clipping* (la cual elimina elementos de la curva de luz iterativamente hasta que las magnitudes no se excedan más allá de un número específico de desviaciones estándar) implementada en la clase `FATS.Preprocess_LC`) y la segunda es una clase llamada `FATS.Align_LC` que permite alinear dos series de tiempo. La documentación sugiere utilizar estas funcionalidades antes de extraer las características.

Importador de curvas de luz del relevamiento MACHO `FATS.ReadLC_MACHO` recupera la magnitud, el tiempo y el error de un objeto *MACHO-id* dado (la identificación es asignada por el relevamiento *MACHO*). Esta implementación no busca ninguno de los datos en el relevamiento, sino que el usuario es responsable de descargar la curva de luz y ubicarla en el directorio actual de trabajo.

4.3.3. Ventajas y desventajas de FATS

Desde una perspectiva de ingeniería de software, muchas decisiones de diseño en el proyecto *FATS* son muy buenas, mientras que otras son inadecuadas. A continuación, se describen brevemente las "buenas" decisiones que son resaltadas antes de poner foco en las más problemáticas.

Buenas decisiones en diseño

Una buena decisión de diseño en *FATS* es haber separarado la API[25] en dos partes:

[25]Interfaz de programación abstracta: la colección de funciones, clases, y objetos que el programador puede usar en la biblioteca.

1. Una de extracción que se configura en la clase `FATS.FeatureSpace`.
2. Otra para la creación de extractores que se implementa como una jerarquía de clases dentro del módulo `FATS.FeatureFunciontLib`.

Es decir, la API se dividió entre la funcionalidad para el análisis (`FATS.FeatureSpace`) y la funcionalidad para la creación de extractores. Esto mantiene la simplicidad en la utilización del sistema, pero permite la posibilidad de implementar extractores de características complejos. Además, como se señaló anteriormente, el preprocesamiento y la manipulación de la curva de luz de *MACHO* no son asociados directamente con la funcionalidad principal de la biblioteca, y se proporcionan como un adicional en el proyecto.

Criticas

Consideramos varios aspectos débiles de la implementación actual de *FATS*, donde algunos problemas son simples errores de estilo; mientras otros están relacionados con el proceso de desarrollo y diseño, que pueden causar errores y limitaciones durante la extracción de características.

El experimento completo que forma la base de muchas de estas críticas puede ser encontrado en: `https://github.com/carpyncho/feets/blob/master/paper/reports/FATS_tests.ipynb` e incluye las siguientes cuestiones:

Estilo y mantenibilidad *Python* tiene un estilo de codificación estricto definido en el documento *PEP-8*[26]. Este documento define las pautas para hacer que el código sea fácil de entender por cualquier Desarrollador Python. Cuando un proyecto sigue estas pautas, así como otras, como por ejemplo el PEP-20[27] (relacionado con la filosofía detrás del diseño de *Python*), la comunidad del lenguaje se refiere al mismo como "Pythonico"(fácil de entender y mantener). Los errores de PEP-8 se pueden verificar fácilmente con varias herramientas como *flake8*[28] y pylint[29].

FATS no se adhiere a las recomendaciones y 828 problemas de estilo se encontraron en 1249 líneas de código.

Configuraciones globales *FATS* posee una falla producto de almacenar cálculos intermedios en variables globales en todos los extractores de características. Los errores provenientes de almacenar información en estos espacios de memoria compartidos por la aplicación están bien documentados en la literatura y deben evitarse a toda costa (Wulf and Shaw, 1973).

El problema es un tanto complicado de explicar, por esto se desarrolla a continuación un ejemplo completo, partiendo desde la documentación del proyecto (que explica el requerimiento de funcionalidad que lleva al error), hasta la la demostración del error en sí.

La documentación de *FATS* establece lo siguiente:

> **Nota:** Algunas características dependen de otras características, y en consecuencia deben ser computadas juntas. Por ejemplo, `Period_fit` devuelve la probabilidad de falsa alarma del

[26] `https://www.python.org/dev/peps/pep-0008/`
[27] `https://www.python.org/dev/peps/pep-0020/`
[28] `http://flake8.pycqa.org`
[29] `https://www.pylint.org/`

período estimado. Por lo tanto, también es necesario calcular el período `PeriodLS`.

Estas idea de que "**una característica depende de otra**" está implementada con variables globales.

Por ejemplo, se puede verificar lo descripto en el código de la clase `StructureFunction_index_21` dentro del módulo `FATS.FeatureFunctionLib`

```
1  class StructureFunction_index_21(Base):
2
3    def __init__(self):
4      self.Data = ['magnitude', 'time']
5
6    def fit(self, data):
7        magnitude = data[0]
8        time = data[1]
9
10       global m_21
11       global m_31
12       global m_32
13
14       # Existe más código aquí
15
16       m_21, b_21 = np.polyfit(sf1_log, sf2_log, 1)
17       m_31, b_31 = np.polyfit(sf1_log, sf3_log, 1)
18       m_32, b_32 = np.polyfit(sf2_log, sf3_log, 1)
19
20       return m_21
```

Según este ejemplo, `m_21`, `m_31` y `m_32` (líneas 16 - 18) se calculan y almacenan en el entorno, es decir a nivel de módulo (líneas 10–12); pero sólo se devuelve el valor de `m_21` (línea 20).

Además, si se revisa la clase `StructureFunction_index_31` en el mismo módulo:

```
1  class StructureFunction_index_31(Base):
2
3    def __init__(self):
4      self.Data = ['magnitude', 'time']
5
6    def fit(self, data):
7       try:
8         return m_31
9       except:
10        print("error: please run "
11              "StructureFunction_index_21 "
12              "first...")
```

Se aprecia que esta clase se usa para recuperar la variable `m_31` del entorno global (línea 31), o se imprime un error en la consola (línea 10).

Esta opción de diseño crea un error, que puede ser explotado para recuperar valores incorrectos si se realiza el siguiente procedimiento.

1. Primero, hay que importar los módulos y crear dos curvas de luz sintéticas: `normal_lc` (las magnitudes se generan a partir de una distribución de valores Gaussianos) y `uniform_lc` (las magnitudes se generan a partir de una distribución de valores uniformes).

```
>>> import numpy as np
>>> import FATS

>>> mag = np.random.normal(size=10000)
>>> time = np.arange(10000)
>>> normal_lc = [mag, time]

>>> mag2 = np.random.uniform(size=10000)
>>> time2 = np.arange(10000)
>>> uniform_lc = [mag2, time2]
```

2. En segundo lugar, creamos un espacio de características del cual extraeremos solamente `StructureFunction_index_21` y `StructureFunction_index_31` de la curva de luz normal (`normal_lc`).

```
>>> fs_normal = FATS.FeatureSpace(
... featureList=[
...   'StructureFunction_index_21',
...   'StructureFunction_index_31'])

# extraemos
>>> fs.calculateFeature(normal_lc)
>>> result = fs.result(method='dict')

# imprimimos los resultados
>>> print "Normal LC:"
>>> for f, v in result.items():
...   f = f.split("_", 1)[-1]
...   print "  {} = {}".format(f, v)
Normal LC:
  index_21 = 1.97547953389
  index_31 = 3.05091739197
```

3. Creamos un espacio de características **nuevo** e intentamos extraer solo `StructureFunction_index_31` de la curva de luz uniforme (`uniform_lc`). El valor obtenido será el mismo que para el `StructureFunction_index_31` de la curva de luz normal.

```
>>> fs2 = FATS.FeatureSpace(
... featureList=[
...   'StructureFunction_index_31'])

# Se extrae
>>> fs2.calculateFeature(uniform_lc)
>>> result = fs2.result(method='dict')
```

```
>>> print "Bad Uniform LC:"
>>> for f, v in result.items():
...   f = f.split("_", 1)[-1]
...   print "  {} = {}".format(f, v)
Bads Uniform LC:
  index_31 = 3.05091739197
```

La única manera de evitar este problema es calcular siempre en conjunto `StructureFunction_index_21` para todas las curvas de luz de entrada.

```
>>> fs3 = FATS.FeatureSpace(
... featureList=[
...   'StructureFunction_index_21',
...   'StructureFunction_index_31'])

>>> fs3.calculateFeature(uniform_lc)
>>> result = fs3.result(method='dict')

>>>  print "Uniform LC:"
>>> for f, v in result.items():
...   f = f.split("_", 1)[-1]
...    print "  {} = {}".format(f, v)
Uniform LC:
  index_21 = 1.89689583705
  index_31 = 2.74650784403
```

Nótese como ahora el resultado de index_31 en la última línea sí cambió.

Finalmente, se aprecia que el mismo error afecta a todas las características almacenadas utilizando el espacio global: `Periodo_fit`, `Psi_CS`, `CAR_tau`, `CAR_mean`, los componentes *Fourier*, y `StructureFunction_index_31` y `StructureFunction_index_32`, como se mencionó anteriormente.

Python exit El lenguaje Python define a los errores como excepciones[30], así que en presencia de cualquier mala configuración se crean estados excepcionales para informar al código llamador: "algo salió mal".

Por ejemplo, si se desea escribir una función de división que falla con un divisor igual a 0, podríamos escribir lo siguiente:

```
>>> def division(a, b):
...   if b == 0:
...           raise Exception("b no puede ser 0")
...     return a / b
```

El siguiente resultado se obtiene si se llama a esta función.

```
>>> result = division(1, 2.)
>>> print result
0.5
```

[30]Condiciones anómalas o excepcionales que requieren un procesamiento especial

```
>>> result = division(1, 0)
Traceback (most recent call last):
  ...
  raise Exception("b no puede ser 0")
Exception: b no puede ser 0
```

Si deseamos imprimir el valor nulo de *Python* `None` cuando se produce una excepción, es posible gestionar el error con la estructura `try-except`.

```
>>> try:
...   result = division(1, 0)
... except Exception:
...   result = None
>>> print result
None
```

Este ejemplo simple muestra cómo *Python* puede ser usado por el programador para gestionar correctamente estados excepcionales: *si no sabe cómo manejar una configuración, entonces lance una excepción e ignórela sin detener el sistema.*

Las excepciones proporcionadas al programador por *Python* se pueden reducir a dos tipos básicos: `BaseExceptions` y `Exceptions`. El primer tipo comprende excepciones que solo se pueden gestionar en casos inusuales, como `SystemExit`. Estos se generan cuando la función `sys.exit()` es llamada. Con eso se apaga la máquina virtual y se envía el código de salida al sistema operativo. Por ejemplo, la siguiente pieza de código:

```
>>> import sys
>>> sys.exit()
```

finaliza la máquina virtual *Python* y envía un valor 0 (sin error) al sistema operativo. En *FATS*, esto ocurre cuando un `FeatureSpace` se configura incorrectamente: simplemente la máquina virtual es apagada. Si esto sucede, por ejemplo, en un entorno de multiprocesamiento (como un servidor web, *pipeline*, o simple cálculo en múltiples núcleos) se puede al menos esperar que el sistema se vuelva inestable. Los siguientes dos códigos reproducen este error.

1- Pedir una característica inválida

```
$ python
Python 2.7.6 (default, ...)
>>> import FATS
>>> FATS.FeatureSpace(
...   featureList=['Foo'])
could not find feature Foo
# python termina aqui
```

2- Enviar una configuración incorrecta a un extractor

```
$ python
Python 2.7.6 (default, ...)
```

```
>>> import FATS
>>> FATS.FeatureSpace(
...   featureList=['Std'], Std=(1,2,3))
error in feature Std
# python termina aqui
```

El siguiente código puede gestionar este error.

```
>>> import FATS
>>> try:
>>>   FATS.FeatureSpace(
...     featureList=['Std'], Std=(1,2,3))
...  except:
...      # hacemos algo con el error
```

Pero como ya se dijo: `SystemExit` no está diseñado para ser tratado.

Python 3 *Python 3* es el nuevo integrante de la familia *Python* que reemplazará la rama *2.7.x* para el 2020 [31]. Esta versión es **incompatible** con Python 2.x pero incluye varias mejoras en términos de expresividad y velocidad[32]

Actualmente, todos los cimientos de la pila científica de *Python*, en base a los cuales se construye *FATS*, ya han sido portados a la rama 3.x, pero el proyecto todavía está solamente disponible *Python 2.x*[33]. Este problema representa esencialmente una sentencia de muerte inminente para el paquete en los próximos años.

Orden de curva de luz Este es un problema menor. La mayoría de los conjuntos de datos de curva de luz representan los datos en el siguiente formato: *tiempo / magnitud / error-de-magnitud*. *FATS*, en cambio, utiliza *magnitud / tiempo / error-de-magnitud*. Esto implica un preproceso en varios catálogos.

Rutinas ineficientes En *FATS*, algunos problemas de rendimiento están vinculados con el cálculo de dos características: `MaxSlope` y `PeriodLS`.

El código en la clase `MaxSlope` se muestra a continuación para su análisis.

```
1  class MaxSlope(Base):
2    """
3    Examining successive (time-sorted)
4    magnitudes, the maximal first difference
5    (value of delta magnitude over delta time)
6    """
7    def __init__(self):
8      self.Data = ['magnitude', 'time']
9
10   def fit(self, data):
```

[31] https://www.python.org/dev/peps/pep-0373/
[32] https://speed.python.org/comparison/
[33] Este problema ya se informó a los autores en: https://github.com/isadoranun/FATS/issues/7

```
11      magnitude = data[0]
12      time = data[1]
13      slope = (
14        np.abs(magnitude[1:] - magnitude[:-1]) /
15        (time[1:] - time[:-1]))
16      np.max(slope)
17      return np.max(slope)
```

La función `np.max` se llama dos veces en las líneas 16 y 17. El resultado de la línea 16 no se utiliza, por lo que este cálculo puede eliminarse.

El problema es más complicado para `PeriodLS`, porque la implementación del método de *Lomb-Scargle* (VanderPlas, 2018), que se incluye con *FATS*, es una re-escritura de una versión programada en el lenguaje *IDL*, (Landsman, 1995)[34] basado en la rutina del libro "*Numerical Recipes*" (Press, 2007). Esto hace que el código sea difícil de mantener y posea algunos problemas de rendimiento debido al uso incorrecto de la biblioteca *Numpy*. Sumado a esto, este código se aplica iterativamente para calcular las nueve características de Fourier.

Se realizó la experiencia de extraer un gran número de características para una curva de luz particular en el relevamiento *MACHO*, y el tiempo computacional disminuyó en un factor 20 cuando se quitaron todas las características que se calculan utilizando el método de *Lomb-Scargle*.

Pruebas y Cobertura de código La medición de las métricas cualitativas y cuantitativas para un proyecto de software implica al menos realizar **pruebas unitarias** y *cobertura de código*.

En el tutorial de *FATS*, un resultado estático es presentado en base a una prueba de invarianza utilizando datos muestreados irregularmente[35]. El proyecto cuenta con 19 casos de pruebas unitarias automatizadas y solo uno actualmente falla[36]. Desafortunadamente, todo el conjunto de pruebas solo ejecuta el 62 % del código completo, que es significativamente menor que el requisito del 90 % adherido por otros proyectos de astronomía como *Astropy* (Robitaille et al., 2013b).

Algunas características no producen los valores esperados La documentación de FATS indica lo siguiente:

- Para una distribución normal el estadístico de *Anderson-Darling* debe tomar valores cercanos a 0,25.
- Para una distribución gaussiana de magnitudes, *StetsonJ* debe tomar un valor cercano a cero.
- La característica *StetsonK* para una distribución gaussiana de magnitudes debe tomar un valor cercano a $2/\pi = 0{,}798$.

Para validar la documentación, calculamos estas tres características para 100,000 curvas de luz gaussianas generadas al azar y los resultados no

[34] `https://github.com/isadoranun/FATS/blob/master/FATS/lomb.py`

[35] `http://isadoranun.github.io/tsfeat/FeaturesDocumentation.html#Appendix`

[36] Este problema ha sido reportado a los autores en: `https://github.com/isadoranun/FATS/issues/9`

	AndersonDarling	StetsonJ	StetsonK
conteo	100000	100000	100000
promedio	0.6052	595386	0.2047
std	0.2576	377575	0.0662
min	0.0930	244228	0.0342
25 %	0.3796	420438	0.1564
50 %	0.5993	510419	0.2036
75 %	0.8427	661672	0.2510
max	1.0000	40561220	0.4484

Tabla 4.1: Análisis estadístico de los resultados de las características *AndersonDarling*, *StetsonJ*, y *StetsonK* calculadas por FATS, realizado con 100,000 curvas de luz Gaussianas generadas al azar.

fueron los esperados, ya que las tres características antes mencionadas difieren notoriamente con los valores documentados. Puede verse el resultado completo del análisis en la Tabla 4.1

4.3.4. Hacia la mejora en FATS

Como ya se discutió, la forma se subsanar problemas en malas decisiones de diseño es la refactorización de código. Entonces, para poder modificar *FATS* sin cambiar su funcionalidad se optó por la siguiente estrategia:

1. Primero, se ejecutaron todos los extractores de características en *FATS* para una curva de luz de ejemplo y los resultados se almacenaron en disco.
2. En segundo lugar, se realiza un caso de prueba de unidad, que ejecuta los mismos extractores sobre la misma curva de luz de ejemplo y verifica si los resultados son los mismos que los almacenados en el archivo.
3. A continuación, la conversión de la nueva arquitectura se inicia progresivamente mientras se verifica si la prueba todavía se pasa.
4. Si un extractor cambia por decisión intencional de un cambio de diseño, entonces la prueba se modifica para tomar en cuenta este cambio.

Este enfoque donde se especifica una prueba y luego se construye el código es llamado desarrollo dirigido por pruebas. Esta técnica hace que sea más fácil confiar en que cada nueva pieza de código escrita durante el proceso de refactorización no genera nuevos errores de regresión[37].

Después de la prueba inicial, el proceso de refactorización del código se divide en las siguientes seis tareas secuenciales:

1. Se actualiza cada función de extracción de características mientras mantenemos el mismo comportamiento[38].
2. Se reemplaza la clase `FeatureSpace` por una que no guarde ningún tipo de estado proveniente de cómputos anteriores.

[37] Se le llama regresión en este contexto a algo que solía funcionar, pero ya no lo hace

[38] los resultados deben ser los mismos para la misma entrada para pasar el test antes definido

3. Se agrega una prueba para asegurar que cada extractor de características devuelva valores razonables, al menos para los casos esperados y habituales.

4. Se continúa con la incorporación de pruebas a casos más raros en los extractores, hasta que se logre la cobertura del código de 90 %.

5. Se incluye la documentación para las características y extractores dentro del código Python.

6. Se porta el tutorial. Este proceso incluye auto-generación de "Documentación" para cada extractor de características.

4.3.5. Resultado: Extractor de características para series temporales (*feets*)

feets (del inglés *feATURES eXTRACTOR for tIME sERIES*), es el resultado de un rediseño radical e incompatible con la anterior infraestructura que nos llevo a decidir por crear un nuevo proyecto.

Todas las funcionalidades de *FATS* se pueden encontrar dentro de esta nueva biblioteca, mientras que se ha abordado todos los problemas antes descriptos. Se aplicaron estrategias de manejo de errores en los casos en que esto no fue posible, con el fin de informar al usuario sobre cualquier posible problema con los resultados.

Soporte para Python 3.x

La versión actual de *feets*[39] (0.4) es compatible con las versiones 2.7, 3.5, 3.6 y 3.7 de *Python*, utilizando la librería *six*[40].

Mejor encapsulamiento de extractores

Los extractores fueron rediseñados para evitar variables globales y ahora devuelven un conjunto fijo de características y es el `FeatureSpace` el encargado de descartar las no solicitadas. La nueva infraestructura es capaz de hacer verificaciones en tiempo de compilación para cualquier extractor introducido por un usuario para evitar comportamientos inesperados durante la extracción de características.

Finalmente, se proporciona una función `feets.register_extractor`, que es capaz de incluir clases de extractor definidas por el usuario en las funcionalidades de *feets*.

Se pueden encontrar explicaciones más detalladas de estos temas en el tutorial (`http://feets.readthedocs.io/en/latest/#Library-structure`.

Excepciones y advertencias

Si el usuario configura mal el `feets.FeatureSpace`, se genera una excepción en lugar de `sys.exit()`, que no apaga toda la máquina virtual de *Python*.

[39] `https://pypi.org/project/feets/#history`
[40] `https://pypi.org/project/six/`

Además, se muestra una advertencia cuando el usuario solicita una función de cualquiera de los extractores con comportamiento inconsistente: (*StetsonK*, *StetsonJ* (Richards et al., 2011a), y *Anderson-Darling* (Kim et al., 2009)).

Integración con Astropy y otras dependencias

La página de inicio de *Astropy*[41] (Robitaille et al., 2013b) indica lo siguiente.

> El Proyecto *Astropy* es un esfuerzo comunitario para desarrollar un paquete central para la astronomía utilizando el lenguaje de programación *Python* para mejorar la reusabilidad, interoperabilidad y colaboración entre los paquetes astronómicos de *Python*.

La decisión de incluir el paquete *Astropy* dependía de dos objetivos: reemplazar la implementación de *Lomb-Scargle*[42] incluida en *FATS*, por la distribuida por el proyecto *Astropy*, para mejorar el rendimiento del extractor de características del período; y además *feets* fue aceptado como parte de los proyectos afiliados a *Astropy*[43] con el fin de demostrar un compromiso con los objetivos de Astropy para los paquetes de astronomía y astrofísica de Python.

Además de *Astropy*, cada dependencia en *feets* está incluido en su instalador, por lo que el proyecto está listo para usar con un solo comando `pip install feets`.

Finalmente, se eliminó *matplotlib* como una dependencia.

Otras mejoras

- Se indexó el proyecto en el repositorio de código astronómico ASCL[44] (Cabral et al., 2018a).
- El orden de los parámetros de entrada se actualizó a *Time-Magnitude-Magnitude Error*, que ahora es consistente con la mayoría de los conjuntos de datos existentes.
- Las funciones de preproceso fueron renombradas para hacerlas más intuitivas. Por ejemplo,

  ```
  FATS.Preprocess_LC(
    mag, time, error).Preprocess()
  FATS.Align_LC(
    time, time2, mag, mag2, error, error2)
  ```

  fue renombrado de la siguiente manera:

  ```
  feets.preprocess.remove_noise(
    time, mag, error)
  feets.preprocess.align(
    time, time2, mag, mag2, error, error2)
  ```

[41] `http://www.astropy.org`

[42] `https://github.com/isadoranun/FATS/blob/6fcd852adf213a477fda878650be5b467a5dd0d8/FATS/lomb.py`

[43] `http://www.astropy.org/affiliated/index.html`

[44] `http://ascl.net/`

- El módulo `feets.dataset` fue incluido para recuperar curvas de luz de los relevamientos *MACHO* (Alcock et al., 2000), *OGLE-III* (Udalski, 2004), así como para crear curvas de luz sintética basadas en un conjunto de distribuciones aleatorias de parámetros (por ejemplo, períodos).

4.4. Carpyncho: Pipeline para procesamiento de datos del *VVV*

La aplicación directa de lo expuesto en las secciones anteriores de este capítulo fue la creación de un pipeline de extracción de características, que decidimos llamar *Carpyncho* (Cabral et al., 2017) por ningún motivo en particular.

Carpyncho está construido sobre Corral y consta de nueve pasos que se encargan de procesar y almacenar los datos de los catálogos del *VVV* de manera eficiente. Como base de datos de producción científica utiliza un PostgreSQL[45] y un conjunto de archivos binarios para almacenar los resultados de las operaciones. Todo el código del proyecto está disponible en `https://github.com/carpyncho/carpyncho`.

Como dato de color, el *Carpyncho* descrito en este capítulo es en realidad la tercera iteración del *pipeline*, siendo los dos primeros:

1. Fue una base de datos construida, que aprovechaba las abstracciones sobre bases de datos y la posibilidad de extender comandos del *framework web Django* (Forcier et al., 2008); el cual fue abandonado por la complejidad de desarrollar aplicaciones multi-procesos. En esta versión se inspiró la creación de *Corral* (Cabral et al., 2017).

2. Un pipeline con Corral pero que almacenaba toda su información en la base de datos. En este caso, se fracasó en obtener un rendimiento razonable dado el hardware disponible; de todas formas el conjunto de modelos utilizados por esta versión es la misma utilizada en Carpyncho 3. Algo a recalcar de esta iteración es que poseía una interfaz web para su consulta, la cual implementaba un lenguaje propio para la generación de los conjuntos de datos de sus características (Cabral et al., 2016a).

El código de ambas versiones abandonadas aun está disponible en `https://github.com/carpyncho/yeolde_carpyncho`

4.4.1. Modelos

Carpyncho utiliza 4 modelos (ver Figura 4.4 para el completo diagrama de clases de los modelos) para orquestar la totalidad de información que almacena. Estos modelos indexan ficheros binarios en un formato útil para el computo vectorial[46] en el sistema de archivos. Además, mantienen una serie de campos los cuales gestionan estados de estos ficheros. Estos modelos son:

Modelo Tile Indexa una baldosa y contiene el *link* a la versión cruda del *band-merge* y a una versión binaria. Posee cinco estados:

[45] `https://www.postgresql.org/about/`
[46] `https://www.numpy.org/devdocs/reference/generated/numpy.lib.format.html`

1. `raw` - El *band-merge* está guardado en el formato de entrada y ningún procesamiento se ha realizado. Está listo para transformarlo en un formato binario más cómodo para su manipulación.
2. `ready-to-tag` - El *band-merge* está listo para ser apareado con los catálogos de estrellas variables. También se ha creado el archivo binario y se han corregido las fechas en formato HJD. Asimismo, se asigna a cada fuente del *band-merge* un id de catorce dígitos con el formato `PTTTOOOOOOOOOO`; donde `P` indica la posición del tile en el relevamiento (3 es bulge y 4 disc), `TTT` es el número de la baldoza o *tile*, y `OOOOOOOOOO` es el número de orden de la fuente dento del catálogo *band-merge*. Por ejemplo el id *40010000000130* (4-001-0000000130) indica que es la 130ª fuente dentro del *tile* d001.
3. `ready-to-unred` - Las fuentes del *band-merge* binario ya se han identificado en los catálogos de estrellas variables.
4. `ready-to-match` - La baldoza está lista para buscar coincidencia de sus fuentes con las observaciones provenientes de su *Pawprints-Stacks*. Así también, en esta etapa ya se corrigió el enrojecimiento de las magnitudes [47] utilizando los mapas de extinción de *BEAM* [48] (Gonzalez et al., 2012). Además, se calcularon colores para las primera épocas de cada fuente.
5. `ready-to-extract-features` - Todas las fuentes del *band-merge* están identificadas en todos los *pawprint-stacks* (en otras palabras las curvas de luz están reconstruidas) y se puede proceder a aplicar *feets* para extraer las características del tile.

Modelo *Pawprint-Stack* Indexa una época de observación del relevamiento. Contiene la ruta a dos archivos: La versión cruda del catálogo (en formato *FITS* (Hanisch et al., 2001)) y la versión procesada en formato binario. Posee dos estados:

1. `raw` - El *Pawprint-Stack* está guardado en el formato de entrada y ningún procesamiento se ha realizado. Está listo para transformarlo en un formato binario más cómodo para su manipulación.
2. `ready-to-match` - Se ha creado el archivo binario y el *Pawprint* está listo para ser apareado con un *band-merge* en busca de todas las fuentes que tienen en común para reconstruir las curvas de luz. También se han corregido las fechas por cada fuente utilizando los días heliocéntricos medios (HJD); a la vez que se ha asignado a cada fuente un id único de dieciseis dígitos con el formato `3PPPPPPPOOOOOOOO` donde: `3` es siempre el número 3 (sirve para garantizar el mismo número de dígitos en todos los ID), `PPPPPPP` es el ID del *pawprint-stack* en Carpyncho, y `OOOOOOOO` es el número de orden de la fuente dento del catálogo del *pawprint-stack*. Por ejemplo, el ID *3000003500003708* (3-0000035-00003708) hace referencia a la 3708ª fuente dentro del pawprint con ID 35.

[47] Corrimiento al rojo del color de la luz de las fuentes producto de polvo interestelar.
[48] A VVV and 2MASS Bulge Extinction And Metallicity Calculator `http://mill.astro.puc.cl/BEAM/calculator.php`

Modelo PawprintStackXTile Dado que las baldosas en *VVV* se solapan, hay algunos *pawprint-stack* que pertenecen a más de una baldosa. Es por ello que el modelo intermedio se encarga de enlazar un *Pawprint-Stack* a una baldosa de modo que se evita almacenar archivos duplicados. Por otra parte, este modelo almacena un único archivo binario el cual conoce cuáles son las fuentes en común del *band-merge* con el *pawprint-stack*. Para gestionar esta información maneja tres estados diferentes:

1. `raw` - Solo se ha enlazado la baldosa con el *Pawprint-Stack*.
2. `ready-to-match` - Este estado permite al *pipeline* buscar las fuentes en común en la baldosa y el *Pawprint-Stack*, en consecuencia este estado indica que ambos están en el estado *ready-to-match*
3. `matched` - Indica que se ha realizado la operación para determinar cuáles son las fuentes en común que poseen el *Pawprint-Stack* y la baldosa, de forma que se almacenan en un archivo binario los ID de ambas fuentes.

Modelo LightCurves Cada baldosa tiene una sola *LightCurve* asociada. Se encarga de dos responsabilidades. Reunir todos los entrecruzamientos entre la baldosa y sus *Pawprint-Stacks* almacenados en todos los *PawprintStackXTile* para reconstruir las curvas de luz en una sola estructura de datos, y almacenar las características extraídas de cada una de esas curvas de luz.

4.4.2. Carpyncho Loader y Steps

La transformación de los modelos y sus archivos asociados están orquestados por el *Loader* o cargador y siete *"steps"* o pasos que van transformando de manera simple y trazable los datos. Estos pasos están diseñados de manera que puedan ser resumidos[49] en caso de algún tipo de falla; además de disminuir la cantidad de memoria utilizada al mínimo.

Carpyncho loader

El *Loader* de carpyncho es bastante simple, analiza un directorio con una estructura fija, para crear instancias de los modelos explicados en el punto anterior y enlazarlos a los archivos que fueron encontrados en la carpeta. Dicha estructura está compuesta de la siguiente manera:

1. Una carpeta por baldosa que lleva el nombre de dicha baldosa.
2. Un archivo con extensión `.dat` por cada carpeta de la baldosa, que contiene la información del *band-merge* en formato tabular.
3. Un sub-directorio por cada carpeta de la baldosa llamado `pawprints` que contiene los catálogos *PawprintStack* en formato fits.

[49] Tienen memoria de todo lo que procesaron y evitan procesar dos veces la misma información

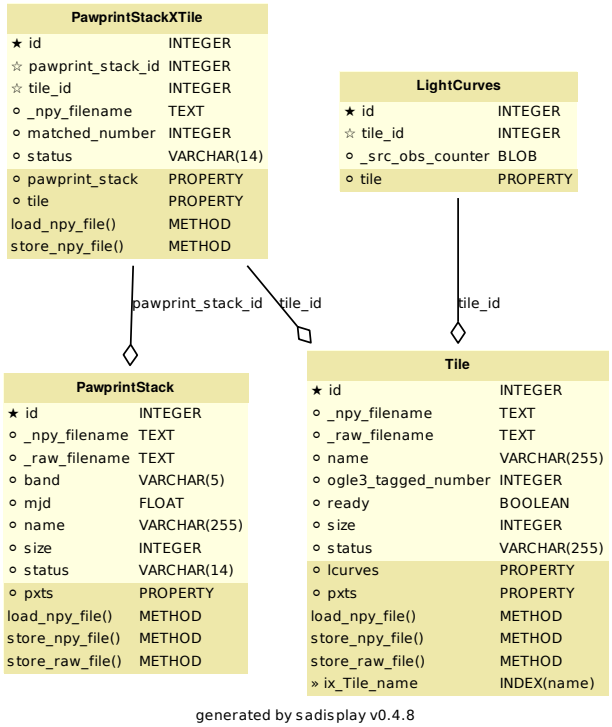

Figura 4.4: Diagrama de clases de los modelos utilizados en Carpyncho para orquestar la transformación de datos, desde los catálogos hasta el resultado final en características utilizadas en aprendizaje automático. Este diagrama es auto generado por Corral.

Por ejemplo una jerarquía de directorios como la siguiente

```
data
├── b396
│   ├── bandmerge.dat
│   └── pawprints
│       ├── pwp00.fits
│       └── pwp01.fits
└── d001
    ├── bandmerge.dat
    └── pawprints
        ├── pwp02.fits
        └── pwp03.fits
```

generaría dos *Tile* (`b396` y `d001`), cuatro *PawprintStack* y cuatro enlaces *PawprintStackXTile* entre cada *PawprintStack* y sus respectivos *Tile*.

Carpyncho Steps

Ya cargados los datos por el *Loader* el procesamiento se lleva adelante por los pasos, los cuales se encuentran conectados a través de los estados que alteran los *Tile, PawprintStack* y *PawprintStackXtile*. Este flujo de trabajo puede observarse en la Figura 4.5.

Así, los steps tienen las siguientes responsabilidades:

Step ReadTile Procesa *Tiles* o baldosas en status `raw`. Se encarga de leer el archivo .dat con el band merge, asignarle IDs a cada una de las fuentes allí encontradas y calcula las fechas corregidas en días julianos heliocéntricos medios. Toda la información se almacena en un archivo binario. Finalmente asigna a la baldosa el status `ready-to-tag`.

Step VSTagTile Obtiene las baldosas con el status `ready-to-tag` y aparea cada fuente con los catálogos de estrellas variables; luego asigna a la baldosa el status `ready-to-unred`.

Step Unred Procesa baldosas con el status `ready-to-unred`; y luego de corregir sus magnitudes según el mapa de enrojecimiento de BEAM y calcular colores para las primeras épocas de cada fuente; asigna a la baldosa el status de `ready-to-match`.

Step ReadPawprintStack Obtiene todos los PawprintStack que tengan el status `raw`; de cuyos catálogos fits computa magnitudes y errores para cada fuente utilizando una versión modificada de la herramienta *fitsio_cat_list* creada por CASU. A estos valores les agrega la fecha de observación correjida a HJD y la posición en grados. Finalmente, almacena el *PawprintStack* en el status `ready-to-match`.

Step PrepareForMatch Es probablemente el paso más simple, se encarga de cambiar el status de todos lo PawprintStackXTile que estén en el status `raw` al status `ready-to-match`; siempre y cuando tanto la baldosa como el PawprintStack que enlazan estén ambos en status `ready-to-match`.

Step Match Encargado de tomar todo los PawprintStackXTile que estén en status `ready-to-match` y encontrar por proximidad la observación de cada fuente en el *pawprint-stack*. Finalmente, deja al *PawprintStackXTile* en status `matched`.

Step CreateLightCurves Este paso es crítico para evitar que la memoria de los nodos de procesamiento colapsen al momento de extraer las características en el último paso. *CreateLightCurves* obtiene todas las baldosas que tenga todos sus *PawprintStackXTile* en status `matched` y crea un objeto `LigthCurve` para la baldosa con un catálogo binario con los valores de tiempo (en HJD), magnitudes y errores de magnitud para todas las observaciones de cada fuente en el *band-merge*. Esto evita que el paso de extracción de características tenga que levantar en memoria cada *Pawprint-Stack* con sus múltiples columnas. El estado final con el cual marca a la baldosa es de `ready-to-extract-features`.

Step FeaturesExtractor Este paso calcula todas las características de las baldosas que tengan status `ready-to-extract-features` al utilizar *feets*

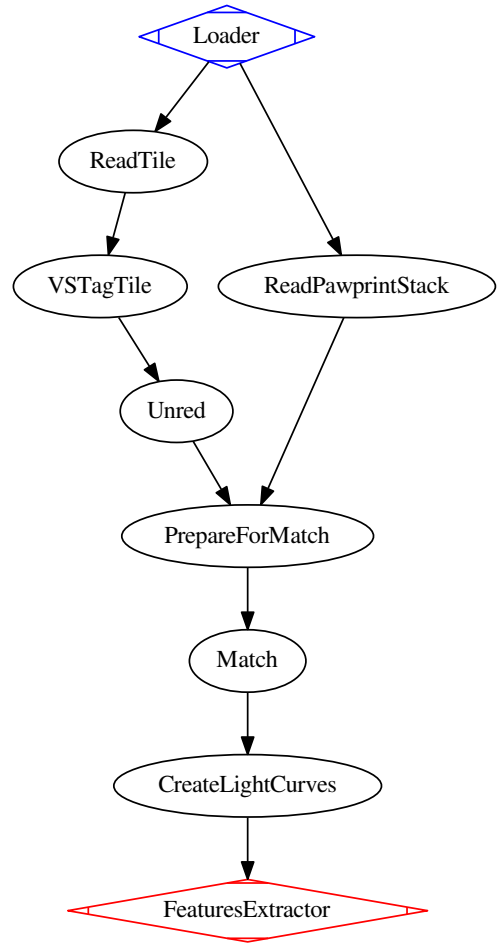

Figura 4.5: Flujo de datos del *pipeline* Carpyncho. En azul se aprecia el *Loader* y en rojo el último paso que se encarga de extraer las características correspondientes de cada curva de luz. Cada nodo circular es un paso de Corral implementado como una clase python.

(Cabral et al., 2018b). El resultado es almacenado en un archivo binario asociado a *light-curve* de la baldosa. Finalmente, pone a la baldosa en estado `ready`.

Calidad

Para finalizar, aprovechamos las herramientas de calidad diseñadas en *Corral* para evaluar iterativamente el resultado final de *Carpyncho*. Él posee un total de 16 pruebas unitarias, que involucran la ejecución del 80,46 % del código.

Así con un $\tau = 13$, se obtiene un $QAI = 80{,}46$ con una calificación de $B-$. Todo el informe de calidad más el reporte del proyecto se puede encontrar en el apéndice B.

4.5. Conclusiones y trabajo a futuro del capítulo

Este capítulo presentó a *Corral*: una alternativa para el procesamiento de un gran volumen de datos que además, de ser usado correctamente, brinda una idea de calidad de todo el ambiente creado sobre él, al basarse en conceptos teóricos del área de ingeniería de software.

Adicionalmente, se brindó un análisis sobre cómo mejorar una herramienta existente basándose en refactorización de código y pruebas unitarias, con lo cual se logró una herramienta, *feets*, de mejor rendimiento para la extracción de características de curvas de luz de estrellas variables. Igualmente, la herramienta es acompañada por un conjunto de pruebas más grande con una cobertura de código de 90 %, tiene mayor extensibilidad, mejor integración con más conjuntos de datos, a la vez que se encuentra en proceso de ser integrada como paquete afiliado al estándar de-facto de herramientas de análisis astronómico (*Astropy*). Además, el proyecto fue desarrollado en un repositorio público, con más de doscientos treinta y tres *"commit"* y cinco contribuyentes.

Finalmente, se mostró el uso en conjunto de las dos tecnologías desarrolladas teórica y empíricamente para la extracción de características de curvas de luz del *VVV* en el *pipeline* llamado *Carpyncho*. Sumado a esto, la arquitectura desarrollada utiliza una variedad de datos sobre enrojecimiento y catálogos de estrellas variables conocidas, para generar un conjunto de datos más completo que aprovecha las cualidades propias del relevamiento.

Como trabajo a futuro quedan pendientes ideas no implementadas tanto en Corral como en feets. Para el primero será interesante reeimplementar todo el lanzador de procesos sobre una plataforma de cómputo distribuido como *Spark* (Zaharia et al., 2010) o *Dask* (Rocklin, 2015) de modo que se intente mantener la mayor compatibilidad con la sintaxis actual. Eso permitiría un manejo trasparente de procesos y la posibilidad de despliegue de *pipelines* sobre *clusters* de cómputo. Para el segundo proyecto, se pensó en brindar la posibilidad de agregar herramientas de análisis interactivo, como la capacidad de realizar gráficos de las características extraídas y agregar nuevos accesos a conjuntos de datos de curvas de luz. Relacionado a `Carpyncho` sería interesante retomar la idea de brindar a los astrónomos y usuarios una herramienta web para la descarga de las curvas de luz así como de las características extraídas que ellos deseen.

Capítulo 5

Generación de catálogos de estrellas *RR-Lyrae* en el relevamiento astronómico *VVV*

5.1. Objetivos del capítulo

El capítulo describe el proceso para la creación de catálogos de estrellas variables tipo *RR-Lyrae* en las *tiles* pertenecientes al VVV. Se comenzará con un repaso de la importancia de la búsqueda de *RR-Lyraes* en la astronomía en general, a la vez que se describen globalmente: experimentos a realizar en cada etapa, métodos de aprendizaje a probar, conjunto de datos utilizados, y métricas de error de clasificación para evaluar el rendimiento de los diferentes clasificadores. Luego, se describirán los resultados obtenidos al comparar los diferentes tamaños de muestra para evaluar el efecto del desbalance de clases. Finalmente, se utilizará el método con mejores métricas de clasificación para realizar una selección de características y sugerir cómo generar un catálogo de estrellas candidatas en el VVV.

5.2. Las estrellas *RR-Lyrae*

Desde que Shapley usó las estrellas *RR-Lyrae* para medir la distancia de algunos cúmulos globulares y la posición del sol con respecto al centro galáctico (Shapley, 1936), estas estrellas han adquirido un papel prominente en la determinación de la estructura de nuestra galaxia; y como se explicó en el Capítulo 3, la búsqueda de este tipo de estrellas es uno de los objetivos principales del *VVV*.

También en el Capítulo 3 se explicó que la identificación de este tipo de estrellas en *VVV* se obtuvo al realizar un *cross-matching* con los catálogos de estrellas variables de los relevamientos *OGLE-III* (Udalski, 2004), *OGLE-IV* (Udalski et al., 2015) y *VizieR* (Ochsenbein et al., 2000); pero para el caso de

este experimento sólo se utilizó el primero de ellos (*OGLE-III*), ya que a la fecha de su realización los otros dos aún no existían.

De todas formas, para aprovechar la actual disponibilidad de los catálogos *OGLE-IV* y *VizieR*, al finalizar este experimento se evaluaron los resultados de nuestros Falsos positivos (FP) (estrellas que se creían que eran *RR-Lyraes* pero según *OGLE-III* no lo son) contra los nuevos datos disponibles.

5.3. Diseño experimental

El conjunto de datos original seleccionado para la generación de catálogos fue de los *tiles b261*, *b262*, *b263*, *b264* y *b278*. Fueron elegidos por dos motivos: el primero fue la presencia de la "Ventana de Baade" en el *tile b278*, la cual es una zona con poco polvo intergaláctico en la línea de visual de la tierra al núcleo galáctico (Baade, 1946) que fue utilizado históricamente para estudiar la poblacion de RR-Lyrae; y en segundo lugar porque la zona de observación de *OGLE-III* se superpone bien con estas baldosas.

La cantidad de estrellas variables detectadas en esta zona está en el orden de 300 por baldosa, sobre un total de fuentes de otro tipo del orden de 1000000. Para trabajar con muestras de tamaño aceptable, se procedió a extraer como clases negativas a tres muestras de estas fuentes de tipo desconocido con tamaños de 20000, 5000 y 2500, que llamaremos grande, mediana y pequeña, respectivamente. Así, en las muestras más grandes las estrellas de tipo *RR-Lyrae* representan el 1,5 % del total, en las de tamaño mediano el 5,8 % y en las chicas alrededor del 11 %. Un resumen de estos datos puede observarse en la Tabla 5.1.

Tile	RR-Lyrae	%(G)	%(M)	%(P)	T.Total	T.Útil
b261	221	1.09	4.23	8.12	955119	575075
b262	296	1.46	5.59	10.59	1087417	591770
b263	305	1.50	5.75	10.87	920350	585661
b264	294	1.45	5.55	10.52	999947	614967
b278	423	2.07	7.80	14.47	1018874	781612
Promedio	308	1.51	5.78	10.91	996341	629817

Tabla 5.1: Distribución de 1553 estrellas tipo *RR-Lyrae* identificadas con el catálogo de *OGLE-III*, sobre los *tiles b261*, *b262*, *b263*, *b264* y *b278*. Las columnas *%(G)*, *%(M)*, *%(P)* muestran el porcentaje de estas estrellas frente a las muestras de tamaño Grande (20 mil), Mediana (5 mil) y Pequeña (2500) respectivamente. Finalmente, la columna *T.Total* indica la cantidad total de fuentes presentes en esa baldosa, mientras que *T.Útil* indica cuántas de estas fuentes son útiles para nuestro análisis (esto es, fuentes con más de 30 observaciones y cuyas magnitud media no esté saturada ni sea muy tenue).

Los modelos elegidos para realizar los experimentos fueron SVM con un *kernel* lineal, polinómico y de función de base radial (RBF); KNN con $k = 50$ y RF con 500 árboles utilizando IG como métrica de selección de características.

Los hiper parámetros para todos los modelos fueron escogidos con una evaluación en *k-folds* a partir de una búsqueda en una grilla de valores; como no se encontró una fuerte dependencia de sus valores, se decidió dejarlos fijos para el resto de los experimentos por simplicidad.

Para realizar la evaluación de los experimentos se optó por utilizar las medidas *Precision*, *Recall*, $F1$ y AUC (acompañada de su correspondiente curva ROC).

5.4. Pre–Procesado

A continuación, se realizó una limpieza de valores inválidos (nulos e infinito); y se determinó que las características *GSkew* y *Period_fit* son afectadas por ellos. En el caso de *GSkew* fue eliminada de las muestras por poseer gran número de valores inválidos. *Period_fit*, por otro lado, es una característica muy útil ya que determina la confianza del período calculado (*PeriodLS*), el cual también se utiliza para obtener los componentes de *Fourier* de la serie temporal, por lo cual perder la información brindada por esta característica es algo indeseable. Afortunadamente, sólo un número muy pequeño de fuentes tenían problemas con esta *Period_fit*, por lo que se decidió eliminarlas.

Finalmente, se obtuvo un conjunto de cincuenta y siete características resumidas en la Tabla 5.2; y con una distribución final de clases presentadas en la Tabla 5.3.

Características			
Amplitude	AmplitudeH	AmplitudeJ	AmplitudeJH
AmplitudeJK	Autocor_length	Beyond1Std	CAR_mean
CAR_sigma	CAR_tau	Con	Eta_e
FluxPercentileRatioMid20	FluxPercentileRatioMid35	FluxPercentileRatioMid50	FluxPercentileRatioMid65
FluxPercentileRatioMid80	Freq1_harmonics_amplitude_0	Freq1_harmonics_amplitude_1	Freq1_harmonics_amplitude_2
Freq1_harmonics_amplitude_3	Freq1_harmonics_rel_phase_0	Freq1_harmonics_rel_phase_1	Freq1_harmonics_rel_phase_2
Freq1_harmonics_rel_phase_3	LinearTrend	MaxSlope	Mean
Meanvariance	MedianAbsDev	MedianBRP	PairSlopeTrend
PercentAmplitude	PercentDifferenceFluxPercentile	PeriodLS	Period_fit
Psi_CS	Psi_eta	Q31	Rcs
Skew	SmallKurtosis	Std	c89_c3
c89_hk_color	c89_jh_color	c89_jk_color	c89_m2
c89_m4	cnt	n09_c3	
n09_hk_color	n09_jh_color	n09_jk_color	
n09_m2	n09_m4	ppmb	

Tabla 5.2: Características seleccionadas para la creación de catálogos de estrellas tipo *RR-Lyrae* sobre baldosas del VVV

Tile	Pequeña	Mediana	Grande	RR-Lyrae
b261	2718	5212	20193	221
b262	2791	5288	20247	296
b263	2805	5302	20293	305
b264	2792	5292	20289	294
b278	2912	5406	20354	423

Tabla 5.3: Distribución final de las clases positivas (*RR-Lyrae*) frente las negativas (fuentes desconocidas) en las baldosas del VVV que son incluídas en este trabajo.

5.5. Comparación de clasificadores

En una primera instancia se trata de determinar el rendimiento de los diferentes clasificadores respecto a los diferentes desbalances de clase muestreados. Así el rendimiento fue obtenido con la evaluación de la clasificación de *b278*, al dividir los datos en 10 usando *K-Folds* estratificados para todos los tamaños de muestra.

El resultado de ejecutar el experimento sobre la muestra pequeña puede observarse en la Tabla 5.4 y la Figura 5.1. En estos dos resúmenes se hace notar que tanto KNN como las tres versiones de SVM (lineal, polinómico y RBF) dan resultados bastante similares, siendo levemente superior el SVM con un *kernel* RBF. Todos estos clasificadores poseen valores similares en las métricas de *precision*, *recall*, F_1 y AUC.

El caso diferente es el del clasificador RF, el cual muestra mejores valores en las tres métricas analizadas en la tabla. Si se observan los resultados de este experimento para todos los tamaños de muestra (ver Tabla 5.5), se verifica que la superioridad de RF se repite en todos los otros casos. La Figura 5.1 muestra que este comportamiento se observa también para el AUC de las curvas ROC correspondientes. Estos resultados se repiten al utilizar otros *tiles* (los resultados completos de todos los experimentos de este capítulo se encuentran accesibles en el Apéndice C). Por este motivo, sólo nos centraremos en analizar este clasificador de aquí en adelante.

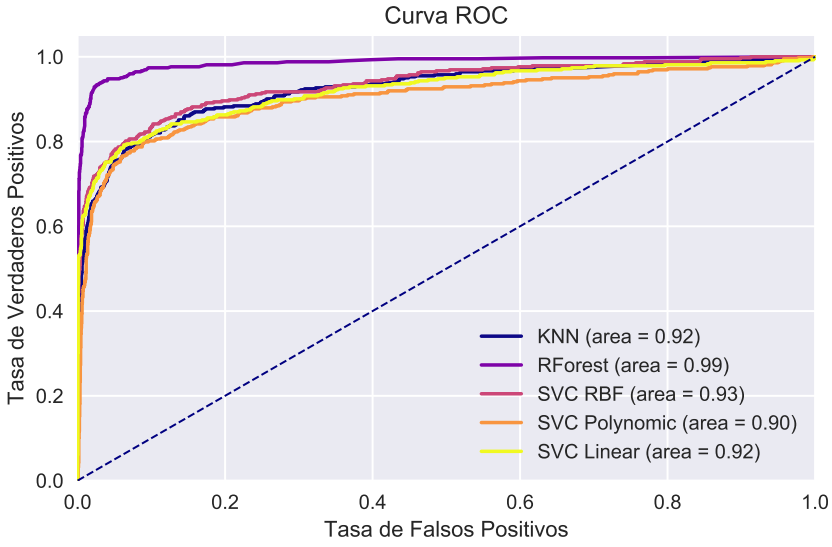

Figura 5.1: Curvas ROC de los clasificadores RF, KNN y SVM (con *kernels* polinómico, lineal y RBF, para el experimento de entrenar y predecir sobre el *tile b278* utilizando 10 *K-folds*. Puede observarse cómo el clasificador RF mejora la eficiencia de clasificación respecto a KNN y los tres SVM, los cuales poseen curvas de formas similares y AUC equivalentes.

Finalmente, es importante notar que para realizar un catálogo lo que se busca es tener una **alta detección de la clase positiva** (alto *recall* en *RR*-

SVM - Lineal				
Clase	Precision	Recall	F1	N
FD	0.95	0.98	0.96	2489
RR	0.86	0.68	0.76	423
Promedio/Total	0.93	0.94	0.93	2912

SVM - Poli				
Clase	Precision	Recall	F1	N
FD	0.93	0.99	0.96	2489
RR	0.88	0.53	0.66	423
Promedio/Total	0.92	0.92	0.91	2912

SVM - RBF				
Clase	Precision	Recall	F1	N
FD	0.95	0.99	0.97	2489
RR	0.91	0.66	0.77	423
Promedio/Total	0.94	0.94	0.94	2912

Random-Forest				
Clase	Precision	Recall	F1	N
FD	0.98	0.99	0.98	2489
RR	0.94	0.85	0.89	423
Promedio/Total	0.97	0.97	0.97	2912

KNN				
Clase	Precision	Recall	F1	N
FD	0.93	0.99	0.96	2489
RR	0.89	0.60	0.72	423
Promedio/Total	0.93	0.93	0.93	2912

Tabla 5.4: Tablas con los índices de errores de los clasificadores RF, KNN y SVM (con *kernels* polinómico (poli), lineal y RBF, para el experimento de entrenar y predecir sobre el *tile b278* utilizando 10 *K-folds* en la muestra **pequeña**. La columna Clase indica a que clase pertenecen los indicadores siendo FD la clase negativa de fuentes desconocidas, y RR la clase positiva o estrellas tipo "RR-Lyrae". La columna N indica la cantidad de observaciones de esa clase. Se puede observar de los resultados que las métricas de calidad para KNN y todos los SVM son similares, mientras que el RF mejora los otros índices para ambas clases.

Lyrae), de modo que además se trata de conseguir la **menor cantidad posible de falsos positivos** (una *precision* alta en la clase *RR-Lyrae*). Las medidas de calidad sobre la clase *FD* no son relevantes para nuestro propósito. El AUC es la única otra medida que agrega información adicional sobre la calidad del

Clasificador	M. Pequeña			M. Mediana			M. Grande		
	Prec	**Rec**	**AUC**	**Prec**	**Rec**	**AUC**	**Prec**	**Rec**	**AUC**
SVM-L	0.86	0.68	0.92	0.91	0.55	0.92	0.91	0.43	0.91
SVM-P	0.88	0.53	0.91	0.85	0.43	0.88	0.86	0.40	0.88
SVM-R	0.91	0.66	0.93	0.91	0.53	0.93	0.95	0.43	0.90
RF	0.94	0.85	0.99	0.93	0.79	0.99	0.93	0.65	0.99
KNN	0.89	0.60	0.92	0.85	0.46	0.91	0.87	0.33	0.89

Tabla 5.5: Precision (Prec), Recall (Rec) y AUC para los clasificadores SVM con *kernels* lineal (*SVM-L*), polinómico (*SVM-P*) y RBF (*SVM-R*); RF y KNN para las muestras pequeña, mediana y grande, realizando 10 *K-Fold* sobre el *tile* *b*278 sólo para la clase de estrellas tipo $RR-Lyrae$. Puede observarse como los RF poseen métricas superiores a todos los demás clasificadores, los cuales entre sí tienen valores similares.

catálogo que se crea. Por lo tanto, en el resto del análisis nos concentraremos en estas dos medidas de calidad (*precision* y *recall*) sólo para la clase positiva, y el AUC.

5.6. Generalización

Ya definidas las métricas de calidad, y el clasificador a utilizar (RF); surge la inquietud de analizar qué tan bien se pueden predecir los datos de un *tile* del VVV, partiendo de otros *tiles*. Para esto diseñamos el siguiente experimento que permite evaluar esta capacidad en distintos aspectos:

1. Comenzamos con el experimento de la sección anterior de evaluar la detección en *b278*, al dividir los datos en 10 *K-Folds* estratificados.

2. Usar como conjunto de entrenamiento *b278* y probar el modelo entrenado con *b261*, *b262*, *b263* y *b264*, para evaluar si realmente los clasificadores son aplicables a otros *tiles*.

3. Usar como conjunto de entrenamiento *b261* y probar el modelo entrenado con *b278*, *b262*, *b263* y *b264*, para apreciar si los resultados dependen fuertemente del conjunto con que se entrena.

4. Usar como conjunto de entrenamiento *b278* $\cup$ *b261* y probar el modelo entrenado con *b262*, *b263* y *b264*, para evaluar si el resultado mejora al agrandar el conjunto de entrenamiento.

5. Usar como conjunto de entrenamiento *b278* $\cup$ *b261* $\cup$ *b264* y probar el modelo entrenado con *b262* y *b263*, para continuar el punto anterior.

Los resultados de estos experimentos son presentados en la Tabla 5.6. Se observa globalmente en esta tabla que todas las clasificaciones funcionan en su límite de eficiencia con AUC muy cercanas a 1. Al ser un problema altamente desbalanceado, el AUC está dominado por la clase mayoritaria y no produce mucha información sobre la otra clase. También, y más importante, es notar que a lo largo de cualquier combinación de *entrenamiento-test* el *Recall* disminuye en las muestras grandes, mientras que la *precision* es en gran medida la misma.

En términos de problema, el clasificador usa un punto de trabajo que en general prefiere elegir que una observación es de la clase desconocida (clase mayoritaria) para garantizar que lo seleccionado como positivo es efectivamente una *RR-Lyrae*.

Sobre los puntos 1 y 2 del experimento, se observa que la estimación de las medidas de calidad dada por el *K-Folds* interno es consistente con los valores que se obtienen al usar nuevos *tiles* como test, por lo que se puede considerar válida como medida de la capacidad de generalización del modelo. Se observa también que hay diferencias entre los *tiles* incluidos. Por ejemplo, el *tile* b264 parece ser más simple de clasificar que otros, siendo sus valores de recall consistentemente mayores. Está información se corrobora en el punto 3, ya que al entrenar con el *tile* b261 se repite que los valores sobre el b264 son mayores que el resto. También se observa en este punto que la información que el clasificador aprende en el *tile* b261 es ligeramente distinta al b278, ya que las métricas de calidad varían en todos los casos, aunque sólo en unos pocos es importante cuantitativamente el cambio. Es interesante notar que los valores de *recall* son consistentemente menores que para el clasificador entrenado con b278. Esto parece indicar que en realidad hay una falta de información en el *tile* b261 más que información distinta. Esta hipótesis parece corroborarse con el punto 4 del experimento, ya que al unir la información de los *tiles* b278 y b261 para entrenar, los resultados que se obtienen son similares, aunque levemente superiores, a los que se ven al utilizar solamente el *tile* b278. El punto 5 afirma estos resultados. Sólo existe una mejora muy leve si se utiliza como entrenamiento la información de varios *tiles*, aunque la leve mejora justificaría el mayor tiempo de entrenamiento.

		M. Pequeña			**M. Mediana**			**M. Grande**		
Ent.	**Prueba**	**Prec**	**Rec**	**AUC**	**Prec**	**Rec**	**AUC**	**Prec**	**Rec**	**AUC**
b278	K-fold	0.94	0.85	0.99	0.93	0.79	0.99	0.93	0.65	0.99
	b261	0.94	0.88	0.99	0.93	0.84	0.99	0.96	0.74	0.98
	b262	0.97	0.85	0.99	0.97	0.78	0.99	0.97	0.67	0.99
	b263	0.96	0.86	0.99	0.97	0.77	0.99	0.92	0.67	0.99
	b264	0.95	0.92	0.99	0.96	0.85	0.99	0.93	0.75	0.99
b261	b278	0.97	0.69	0.99	0.97	0.67	0.99	0.97	0.57	0.99
	b262	0.98	0.77	0.99	0.98	0.73	0.99	0.98	0.63	0.99
	b263	0.98	0.80	0.99	0.99	0.73	0.99	0.96	0.64	0.99
	b264	0.98	0.85	0.99	0.99	0.76	0.99	0.96	0.69	0.99
b261 ∪	b262	0.98	0.85	0.99	0.97	0.79	0.99	0.97	0.69	0.99
b278	b263	0.97	0.85	0.99	0.98	0.79	0.99	0.93	0.70	0.99
	b264	0.97	0.92	0.99	0.97	0.84	0.99	0.94	0.74	0.99
b261 ∪	b262	0.98	0.86	0.99	0.97	0.81	0.99	0.97	0.70	0.99
b278 ∪	b263	0.97	0.87	0.99	0.97	0.80	0.99	0.94	0.70	0.99
b264										

Tabla 5.6: Precision (Prec), Recall (Rec) y AUC para RF sobre todos los puntos del experimento descrito en esta sección. La columna *Ent.* indica el *tile* (o conjunto de *tiles*) que se utilizó como entrenamiento para el experimento, y la columna *Prueba* indica con cuáles de ellos se realizó la prueba.

5.7. Balance de clases

La siguiente etapa de la generación de catálogos aborda la pregunta de cómo los experimentos antes realizados diferirán en sus resultados si se altera el balance entre clases durante el entrenamiento. En otras palabras, cómo cambiará la *precision* y el *recall* a medida que se incrementa el número de estrellas desconocidas a los conjuntos de entrenamiento y prueba. En la Tabla 5.6 se muestran resultados variando tanto el conjunto de entrenamiento como el de prueba al mismo tiempo. Se observa en general que el punto de trabajo elegido por el clasificador tiende a mantener una *precision* que prácticamente no cambia con el tamaño de la muestra, mientras que el *recall* tiende a disminuir al aumentar el tamaño de las muestras, lo que sugeriría en principio que es conveniente trabajar con muestras que sean lo más balanceadas posibles.

Muestra	Entrenamiento	Prueba	Prec	Rec	AUC
Pequeña	b278	b261	0.889	0.697	0.986
	b261	b278	0.735	0.878	0.988
Mediana	b278	b261	0.921	0.693	0.987
	b261	b278	0.845	0.842	0.987
Grande	b278	b261	0.969	0.582	0.984
	b261	b278	0.971	0.747	0.986

Tabla 5.7: Precision (Prec), Recall (Rec) y AUC para RF entrenando primero en $b261$ y probando en $b278$; y entrenando en $b278$ y probando en $b261$ en segundo lugar. Para ambos casos las comparaciones radican en entrenar en muestras de tamaño 2500 (pequeña), 5000 (mediana) y 20000 (grande); y realizando las pruebas en el *tile* de la muestra de tamaño 20 mil (grande). Nótese que si bien el *recall* se mantiene en valores similares, la *precision* sobre la muestra disminuye a medida que se entrena con menor cantidad de fuentes desconocidas.

La Tabla 5.7 y la Figura 5.2 muestran resultados que permiten una mejor evaluación de esta situación. En primer lugar, la figura muestra que globalmente los clasificadores son muy similares, y que las diferencias son mínimas entre ellos. La tabla muestra qué valores de métricas selecciona el clasificador al entrenar con un tile con cierta cantidad de estrellas desconocidas y probar en otro con el mismo conjunto de estrellas. En la tabla puede observarse un *trade-off* entre el tamaño de la muestra a entrenar y la *precision* resultante, así, a medida que se entrena con muestras más chicas, la *precision* disminuye mientras que el *recall* se mantiene relativamente constante. Esto se debe a que el clasificador para mantener el "recupero" de estrellas tipo *RR-Lyrae* hace crecer los grupos de clasificación seleccionados como positivo incrementando los FP.

Llegado este punto, la comparación se torna dificultosa. Todos los resultados anteriores son métricas tomadas de las muestras que se usan como prueba, pero que en realidad no tienen el tamaño, ni el desbalance que tiene un *tile* completo. Al mantener el razonamiento del párrafo anterior, puede sostenerse que la cantidad de FP simplemente va a crecer a medida que aumenta el conjunto de prueba para llegar a la situación real. Sin embargo, los valores obtenidos a partir de las muestras reducidas pueden ser utilizados para estimar los valores

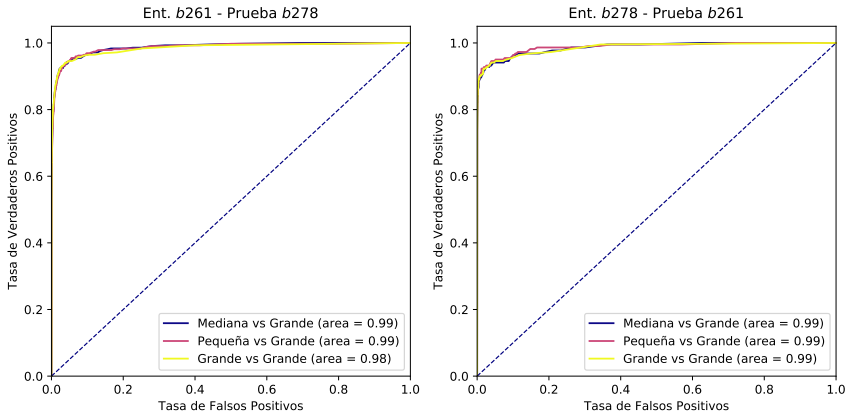

Figura 5.2: Curvas ROC resultantes de clasificar las fuentes de los *tiles* *b*261 y *b*278 sobre muestras de distinto tamaño. En el panel de la izquierda se encuentran las curvas calculadas utilizando al *tile* *b*261 como entrenamiento y el *b278* como prueba; mientras que en el panel de la izquierda los roles de entrenamiento y prueba de los *tiles* se invierten. En ambos paneles se presentan tres curvas, las cuales provienen de clasificar siempre la muestra de 20000 fuentes desconocidas (muestra grande) y entrenar con las muestras de 2500 (pequeña), 5000 (mediana) y 20000 (grande) fuentes. Los resultados demuestran un rendimiento similar en todos los casos.

que se verían en un *tile* completo, siempre y cuando el conjunto que se usa sea tomado al azar de forma justa, ya que fuera de que se incrementa la cantidad de observaciones, no cambiaría la densidad relativa en cada zona del *tile*. Entonces es posible estimar el número de FP en un *tile* completo como:

$$FP^* = FP \times \frac{TR}{TM} \tag{5.1}$$

donde FP^* es la cantidad de FP esperados en el *tile* completo, TR es el tamaño real del *tile*, TM es el tamaño de la muestra sobre la que fue estimada la cantidad FP, y FP son los falsos positivos medidos en la muestra.

A partir de este valor se puede estimar la *precision* en todo el *tile* con:

$$P^* = \frac{TP}{TP + FP^*} \tag{5.2}$$

donde P^* es la *precision* estimada a obtener al clasificar un *tile* entero, y la cantidad Verdaderos positivos (TP) se mantiene constante porque la clase minoritaria/positiva siempre se utiliza en su totalidad en los clasificadores.

Por ejemplo, al entrenar un clasificador en un *tile* A, probando sobre una sub-muestra de 2500 fuentes de un *tile* B que posee un total de 575075 fuentes, se obtienen las siguientes métricas:

- $FP = 12$
- $TP = 197$
- $Precision = 0{,}94$

con las que se puede calcular FP^* como

$$FP^* = 12 \times \frac{575075}{2500} = 2760,36$$

y en consecuencia obtener $precision^*$ como

$$P^* = \frac{197}{197 + 2760{,}36} = 0{,}067$$

o sea que la *precision* de 0,94 medida en la muestra equivaldría a una *precision* real de apenas 0,067 en la totalidad de datos del *tile*.

Con este valor corregido, y debido a que se dispone de todos los FP, TP, *precision* y *recall* de todo los clasificadores para todos sus umbrales de decisión; se puede estimar la *precision* real para un *tile*, para cualquier umbral de clasificación, y realizar una comparación más justa.

5.8. Agregado de *OGLE-IV* y *VizieR*

Como se mencionó en la introducción de este capítulo, al iniciar este experimento sólo se poseía información de estrellas variables provenientes del catálogo del relevamiento *OGLE-III*. Esto fue un limitante bastante fuerte ya que este relevamiento solo superpone su zona de observación en algunos *tiles* centrales del bulbo galáctico. Estas restricciones fueron el motivo principal de elección de los *tiles* utilizados hasta el momento en este capítulo.

Afortunadamente, se hizo disponible a lo largo del trabajo de los catálogos de estrellas variables del relevamiento *OGLE-IV*, así como un subconjunto de estrellas de la colección *VizieR* (solo se utilizaron *RRLyrae* de esta colección por ser una base de datos ambigua en algunos casos). Con estos dos relevamientos se logró obtener muestras de estrellas variables mucho más completas por *tile* y a lo largo de todo el bulbo galáctico. La Tabla 5.8 muestra un resumen de los nuevos datos agregados.

5.9. Predicción de nuevos *tiles*

Como se explicó al final de la Sección 5.7, se puede estimar la *precision* de un *tile* dado, completo, basándose en los obtenidos al entrenar y clasificar muestras de menor tamaño. Esto es útil para fijar un punto de trabajo común a lo largo de todos los clasificadores y poder realizar una evaluación correcta. Para comparar entonces los entrenamientos realizados con muestras de distinto tamaño sobre la base de datos ampliada se fijó un valor de $P^* = 0{,}1$

Así, el procedimiento para encontrar *precision* y *recall* de este punto de trabajo consiste en:

1. Entrenar un clasificador por cada tile en cada tamaño de muestra.
2. Por cada clasificador extraer las métricas de clasificación realizando pruebas en *k-folds* sobre sí mismo, y con cada uno de los demás *tiles*.
3. De cada una de estas métricas extraer todos los FP y TP de para todos los umbrales de decisión.

			RR-Lyrae	
Tile	**T.Total**	**T.Útil**	**T. Catálogos**	**OGLE-III**
b220	691713	281150	65	0
b234	820452	297302	126	0
b247	939320	414497	192	0
b248	1081662	426369	222	0
b261	955119	575075	253	221
b262	1087417	591770	318	296
b263	920350	585661	319	305
b264	999947	614967	312	294
b277	979349	753146	434	0
b278	1018874	781612	441	423
b396	1075212	494646	15	0
Total	10569415	5746895	2697	1553
Promedio	960855.9	522445	245.18	139.91

Tabla 5.8: Resumen de estrellas tipo *RR-Lyrae* encontrados en los *tiles* utilizados en este estudio utilizando los catálogos provenientes de los relevamientos *OGLE-III*, *OGLE-IV* y la colección *VizieR*, en comparación con las fuentes encontradas al utilizar solamente el catálogo del relevamiento *OGLE-III*. La columna *T.Total* indica la cantidad de fuentes en el *tile*, *T.Útil* indica cuántas de estas fuentes son útiles para este análisis (esto es fuentes con más de 30 observaciones y cuya magnitud media no esté ni saturada ni sea muy tenue), *T. Catálogos* muestra cuántas estrellas tipo *RR-Lyrae* fueron identificadas con los tres catálogos; y finalmente la columna *OGLE-III* identifica cuántas estrellas tipo *RR-Lyrae* fueron detectadas solo con el catálogo *OGLE-III*. Se puede observar como incluso *tiles* extremos del relevamiento, como el *b*396, ahora poseen *RR-Lyrae* conocidas, mientras que los que ya poseían observaciones de este tipos de estrellas las han incrementado.

4. Calcular los FP^*.
5. Calcular la P^* para cada umbral de decisión.
6. Buscar el valor P^{**} el cual es el valor más cercano a 0,1 entre todos los P^*.
7. Extraer los valores de *precision* y *recall* correspondiente al valor de P^{**}.

Los resultados de estos experimentos pueden observarse en la Figura 5.3.

Las matrices de *precision* muestran los valores observados en el punto de trabajo seleccionado, y sólo sirven para mostrar que en algunos *tiles* los valores alcanzables son menores que en otros, por diferencias propias de los datos.

Las matrices de *recall* son las que proveen la información buscada. Lo primero que puede observarse es que los resultados son muy buenos en general, con altos valores de *recall*, que parecen ser más altos para las muestras más pequeñas. También que el *tile b*396 es muy diferente de los demás, y sus valores de ambas métricas difieren completamente del resto

Para comparar cuantitativamente las matrices de *recall* de la Figura 5.3, se procede a generar un *ranking* de mayor a menor, por cada celda con su homóloga

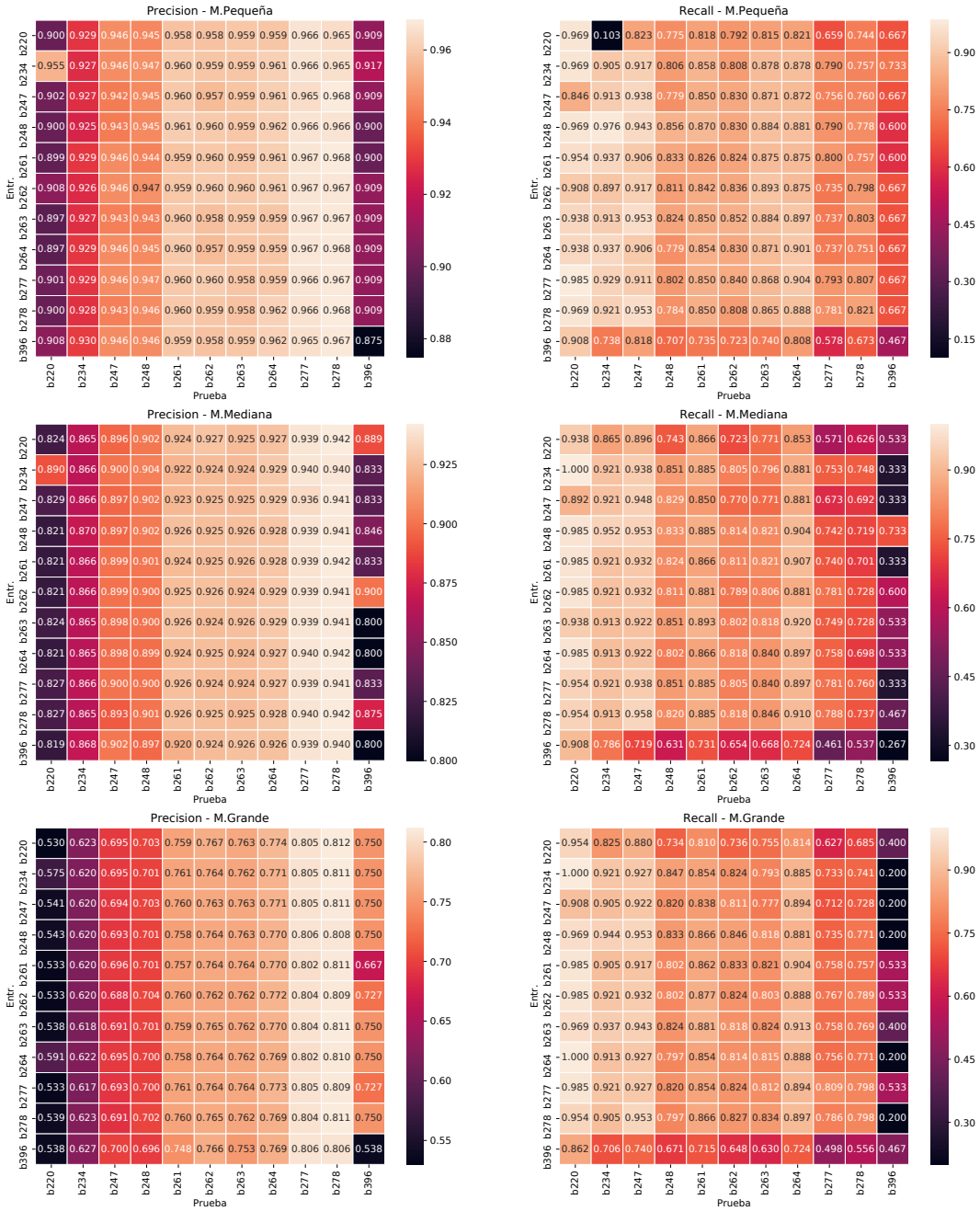

Figura 5.3: Valores de *precision* (columna izquierda) y *recall* (columna derecha), para las clasificaciones en las muestras pequeña (fila superior), mediana (fila intermedia) y grande (fila inferior), para el punto de trabajo $P^* = 0{,}1$. Cada fila de las seis tablas corresponde al *tile* que se uso para entrenar y las columnas el que fue usado para probar; mientras que las diagonales principales contienen los valores obtenidos con *K-folds* sobre el mismo *tile* de entrenamiento.

en las otras matrices, excluyendo la diagonal principal y el *tile* *b*396. Así, si una celda sale primera sumará 2 puntos a la matriz, y si sale segunda solamente 1. Al sumar todos los puntos obtenidos por las matrices se obtiene:

Muestra Pequeña - 125 *puntos*.

Muestra Mediana - 118 *puntos*.

Muestra Grande - 87 *puntos*.

Este resultado confirma que es conveniente utilizar muestras lo más balanceadas posibles para generar el catálogo. Este resultado es consistente con la bibliografía sobre problemas desbalanceados.

Como último paso, para generar el catálogo, se utilizarán las muestras pequeñas (2500), y se seleccionará el punto de trabajo que provea la mayor *precision* posible. Los resultados se pueden ver en la Figura 5.4. Exceptuando el *tile* *b*396, los valores obtenidos son muy alentadores. El valor de *precision* medio (sin diagonal y *b*396) es de 0,48, lo que implica que en promedio, cada dos estrellas indicadas por nuestro sistema como *RR-Lyrae*, una realmente lo será. Por otro lado, el *recall* medio es de 0,71, lo que implica que se recuperan más del 70 % de las *RR-Lyrae* detectadas en los catálogos con nuestro sistema automático.

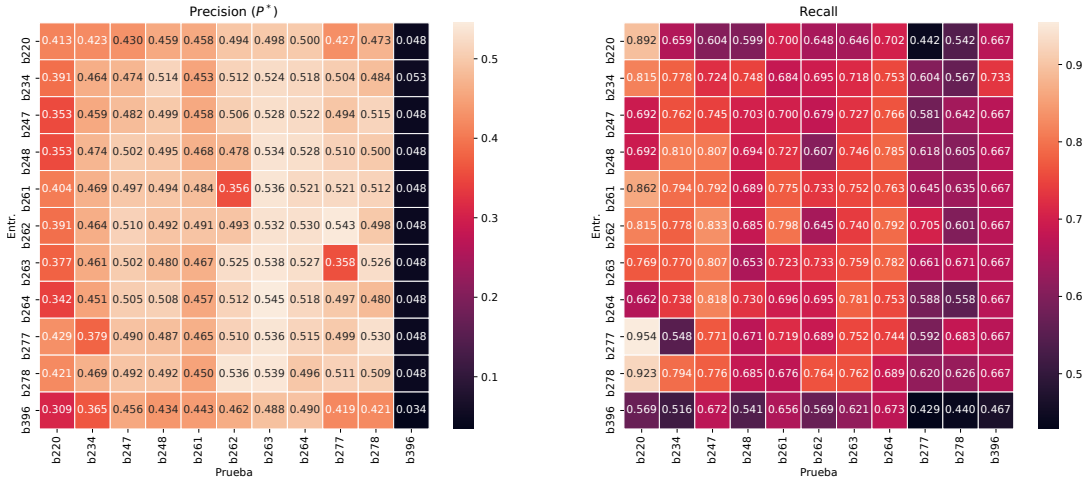

Figura 5.4: Valores de máxima *precision* corregida P^* menor que 1 (columna izquierda) y *recall* (columna derecha), para las clasificaciones en las muestras pequeña. Cada una de las filas corresponde al *tile* que se usó para entrenar y las columnas e cual fue usado para probar; mientras que las diagonales principales contienen los valores obtenidos con *K-folds* sobre el mismo *tile* de entrenamiento.

5.10. Conclusiones y trabajo a futuro

En este capítulo se han analizado las implicancias de entrenar un clasificador con diferentes tamaños de muestra en la clase negativa (fuentes de tipo desconocido) para intentar encontrar fuentes de un tipo *RR-Lyrae* en el VVV;

siendo este un problema muy desbalanceado. Se comenzó con la exposición de la importancia de las estrellas tipo *RR-Lyrae*, y la descripción del diseño experimental que define los modelos y métricas de calidad que se utilizaron en todo el capítulo. Seguido a esto se explicó el pre-procesado de datos y cuáles fueron los *features* tomados en cuenta para este experimento. La 5ta y 6ta secciones llevaron adelante la explicación de la selección del clasificador (RF), y cómo este generaliza a diferentes tamaños de muestra y diferentes *tiles*. A continuación se analizó cómo afecta el desbalance de clases el entrenamiento y la clasificación de diferentes *tiles* y se explicó una forma de corregir las métricas para, a partir de una muestra, estimar la calidad de clasificación en el *tile* completo. La ante-última sección es un interludio que presenta los datos de *OGLE-VI* y *VizieR*, los cuales son utilizados en la última parte del capítulo para realizar predicciones, es decir un catálogo de estrellas de tipo *RR-Lyrae*, y estimar sus métricas de error. Así, finalmente, se explica dado los datos que se poseen cómo se debe generar un catálogo de estrellas variables en el VVV.

Capítulo 6

Detección de características sensibles al ruido experimental

6.1. Objetivos del capítulo

Este capítulo intenta, por medio de la selección de características, determinar si un sub-conjunto de éstas son influenciadas o no por ruidos experimentales.

Se iniciará explicando por qué el *VVV* tiene ruidos experimentales así como cuáles fuentes son las que poseen el mismo tipo de ruido.

A continuación, se presentan las muestras sobre las cuales se realizan la selección de hiper-parámetros para los modelos elegidos, y finalmente se elige el modelo a utilizar durante el resto del experimento.

Las últimas dos secciones consisten en la selección de características sensibles al ruido experimental y la extracción de métricas de clasificación para su evaluación y análisis.

6.2. Susceptibilidades al ruido observacional en las observaciones del VVV

Como se explicó en el Capítulo 3 el VVV utiliza el telescopio vista y su cámara la *VIR-CAM* para producir un mapa tridimensional de una gran parte del centro galáctico (Bulbo) de la Vía Láctea y de una fracción del Disco Galáctico interno.

Los datos del *VVV* se presentan en una unidad llamada "baldosa" (*tile*, en inglés), la cual es una zona rectangular del cielo relevada a través del tiempo. Por cada *tile* el *pipeline* de VVV/VVVx (Emerson et al., 2004) brinda una imagen pre-procesada y una base de datos de archivos con los valores de posición, magnitud y color de las fuentes de luz presentes en la imagen, llamada "catálogo fotométrico". Son estos catálogos los que son objeto de este estudio.

Hay que entender que la información contenida, tanto dentro de la imagen como de sus catálogos derivados, está sujeta a ciertos errores experimentales

que permean todos los datos recolectados. Así, fases lunares, condiciones atmosféricas, mantenimientos de la cámara o el telescopio y modificaciones en el software pueden influir en las detecciones de fuentes astronómicas. Teniendo esto en cuenta, hay que considerar entonces que las mediciones derivadas que se utilizaron como características son también proclives, en diferentes medidas, a estos errores.

Así, es interesante preguntarse si se puede detectar qué características de las extraídas con *feets* y las calculadas en base a des-enrojecimiento (pseudo-colores, pseudo-magnitudes, pseudo-fases y amplitudes multi-banda) son más sensibles al ruido experimental. Es decir, interesa detectar cuáles son las características que son más afectados por efectos propios de la observación y que, por tanto, omiten reflejar propiedades astro-físicas y astro-métricas del objeto observado.

En general, un *tile* siempre es observado en conjunto, por lo cual sus fuentes comparten estados atmosféricos e instrumentales. Cabe aclarar que se dice "en general", porque algunas fuentes pueden observarse inconsistentemente, aunque éstas son eliminadas por nuestro *pipeline* que exige que todo objeto sea avistado al menos 30 veces.

Con lo expuesto en esta sección, se propone un experimento que consiste en determinar si hay un subconjunto de las características medidas que permitan identificar si una fuente pertenece a un *tile*. Si se puede identificar el origen de la fuente, se pone como hipótesis que lo que permite la identificación es ruido experimental, por lo que las características que más aporten al identificar el *tile* deberían ser las más ruidosas. Finalizando, dado que los tipos de errores son productos de diferentes etapas de la adquisición de datos, se refiere a cualquiera de ellos como "Ruido experimental".

6.3. Muestreo

Como se explicó en la sección anterior, la idea del experimento es encontrar qué características representan mejor las diferencias entre los *tiles*. No es necesario en este caso utilizar ningún tipo particular de fuente más allá de que éstas deben ser confiables en su fotometría.

Por estos motivos se optó por extraer uniformemente 1000 fuentes no saturadas (magnitud media > 12) y no difusas (magnitud media $< 16{,}5$), de cada uno de los *tiles* disponibles. De estas selecciones se eliminaron fuentes con valores inválidos y se obtuvo la muestra final (ver Tabla 6.1). Las características a utilizar son las mismas que se presentaron en la Tabla *5.2* del Capítulo 5.

El análisis completo de ésta y las siguientes secciones, puede encontrarse en el Apéndice D.

6.4. Selección del modelo

Los modelos de clasificación evaluados fueron SVM y RF. Para determinar sus mejores hiper-parámetros se utilizó un análisis con *k-folds* sobre un conjunto de datos con las fuentes de los *tiles* $b278$ y $b261$. Estos *tiles* fueron elegidos por dos motivos: por un lado no son extremos en el conjunto de datos disponibles como son $b396$ y $b220$ ni tampoco son contiguos; y por el otro se han utilizado

Tile	**Tamaño**
b220	1000
b234	1000
b247	1000
b248	998
b261	998
b262	996
b263	1000
b264	1000
b277	996
b278	994
b396	999

Tabla 6.1: Cantidad de fuentes observadas por *tile*.

Modelo	Precision	Recall	AUC
SVM-RBF	0.7363	0.7665	0.8068
RF	0.8876	0.8387	0.9412

Tabla 6.2: Métricas de clasificación de los modelos SVM con *kernel* RBF y RF sobre el conjunto de datos comprendido por las fuentes de los *tiles* $b278$ y $b261$. Puede observarse la superioridad de RF en todas las medidas.

como evaluadores en el capítulo 5. En esta etapa se determinó que el mejor *kernel* para SVM para este caso es el RBF.

Con los hiper-parámetros determinados se evaluaron los modelos sobre los mismos set de datos utilizando *10 k-folds*, los resultados pueden observarse en la Tabla 6.2 y la Figura 6.1.

Con estos resultados se opta por continuar el experimento utilizando unicamente el clasificador RF.

6.5. Selección de características

La siguiente etapa del experimento consiste en realizar una selección de características, eligiendo las que den la mejor predicción del *tile* de origen. Por el contrario, si no se encuentra claramente este subconjunto, se espera que todas las características sean igual de afectadas por el ruido experimental.

Así, para llevar adelante esta etapa se procedió a ejecutar un RFE sobre la totalidad de características, eliminándolas de a una por vez, sin poner un límite a la cantidad mínima de características seleccionadas. El algoritmo identificó un subconjunto de 15 características útiles, resumidas en la Tabla 6.3 y en la Figura 6.2.

Con este subconjunto de características, se realizó el mismo experimento de la sección anterior (entrenar en $b278$ y probar en $b261$); el cual mejoró levemente en todos sus índices (ver Tabla 6.4 y Figura 6.3), como es típico al utilizar RFE.

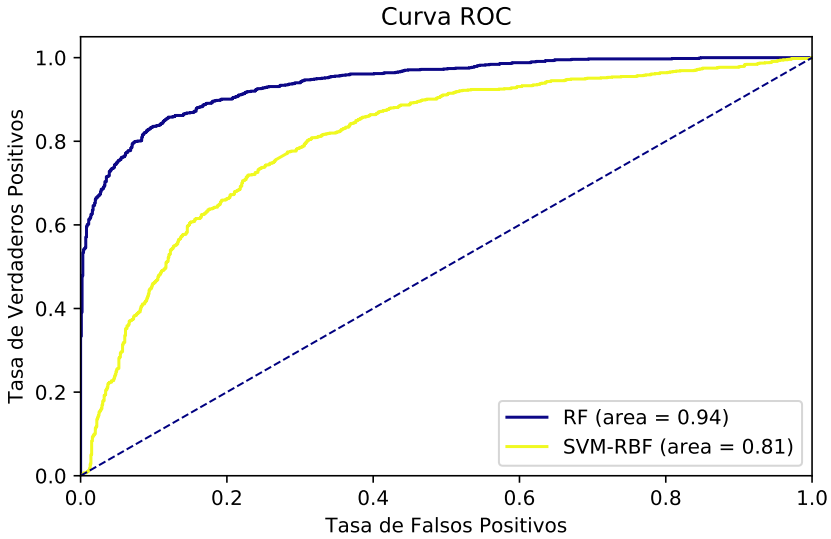

Figura 6.1: Curvas ROC de los clasificadores RF y SVM con *kernel* RBF, para el experimento de entrenar y predecir sobre el *tile* de una fuente dada sobre el conjunto de datos comprendido por las fuentes de los *tiles* $b278$ y $b261$. Puede observarse como el clasificador RF mejora la eficiencia de clasificación considerablemente respecto a SVM.

Beyond1Std	Eta_e	Freq1_harmonics_amplitude_0
LinearTrend	MaxSlope	Mean
Meanvariance	Psi_eta	Rcs
Skew	**c89_m2**	**cnt**
n09_c3	**n09_hk_color**	**n09_m2**

Tabla 6.3: Sub-conjunto de 15 características seleccionadas por el RFE con *k-fold*. En negrita las características creadas en particular para este trabajo, mientras que las demás son las provistas por *feets*.

Modelo	Precision	Recall	AUC
RF	0.8876	0.8387	0.9412
RFE-RF	0.8908	0.8497	0.9473

Tabla 6.4: Métricas de clasificación del subconjunto de datos compuesto de los *tiles* $b278$ y $b261$, utilizando el clasificador RF con la colección total de 57 (RF) características, y solo con las 15 características seleccionadas con RFE. En ambos casos se usaron 10 *K-Folds* para obtener estos resultados. Puede observarse una muy leve mejora en el resultado obtenido con el subconjunto seleccionado con RFE.

Estos resultados evidencian que estas 15 características poseen la suficiente información para diferenciar con la misma calidad (y un poco mejor) las fuentes entre los *tiles* $b278$ y $b261$.

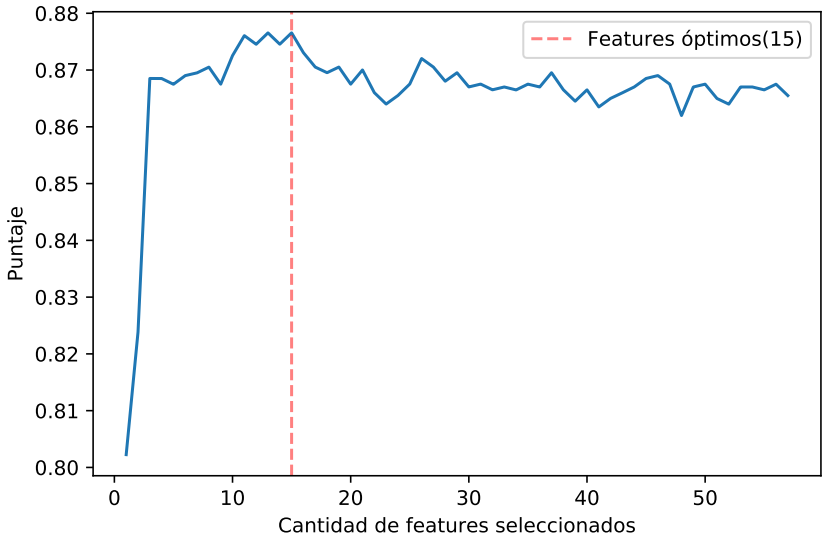

Figura 6.2: Subconjunto de características seleccionadas (eje horizontal) contra puntaje obtenido en promedio en la validación cruzada. En rojo se marca el máximo puntaje que corresponde al subconjunto de 15 características.

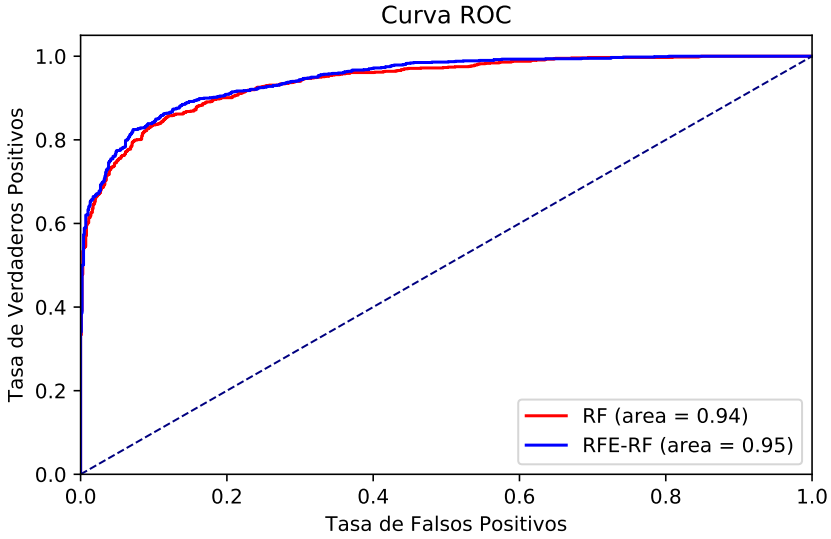

Figura 6.3: Comparación entre las curvas ROC de las clasificadores RF entrenados con las 57 características totales y con las 15 características seleccionadas con RFE, ambos entrenados son 10 *K-Folds* y utilizando el subconjunto de datos compuesto de los *tiles* $b278$ y $b261$. Se nota una leve mejoría en el comportamiento del clasificador entrenado con las características extraídas con RFE.

Distancia	Frecuencia	Precision	Recall	AUC
1.0	11	0.837	0.882	0.944
2.0	11	0.889	0.888	0.962
3.0	10	0.887	0.901	0.962
4.0	8	0.897	0.908	0.967
5.0	4	0.908	0.926	0.977

Tabla 6.5: Mediana de métricas de clasificación, según las distancias de *Manhattan*, entre la grilla *tiles* del bulbo galáctico (ver figura *3.8*). Los valores de distancias que fueron elegidos para este resumen son los que poseen suficiente frecuencia para tener una mediana confiable. Puede observarse como la mediana de las métricas mejora a medida que la distancia crece.

6.6. Generalización

Dado los resultados obtenidos en el caso de prueba del subconjunto de datos comprendido entre las fuentes de los *tiles* $b278$ y $b261$, es razonable preguntarse si estas mediciones se mantienen con todas las combinaciones de *tiles*. Para esto se procedió a crear 55 conjuntos de datos combinando de *2-en-2* todos los *tiles* disponibles en el trabajo, se entrenaron clasificadores y se extrajeron las mismas 3 métricas de clasificación (*Precision*, *Recall* y AUC) utilizando 10 *K-Folds*.

Los resultados de este experimento se encuentran disponibles en la Figura 6.4. Notoriamente, los resultados muestran que en la mayoría de los casos este grupo de características puede distinguir entre los *tiles* originales. Sólo en un par de casos los valores de las métricas se acercan a 0,5, que sería el valor esperado para características independientes del *tile*. También es importante notar que en todos los casos se percibe una mejora en las métricas de calidad a medida que los *tiles* son más distantes (ver Tabla 6.5), lo que indica que hay mayores diferencias en las observaciones, y por lo tanto menor predictibilidad para generar buenos catálogos, al aumentar la distancia entre los *tiles*. Vale la pena recordar que dado que los *tiles* observados se encuentran todos en la región del bulbo galáctico, su distancia determina en gran medida zonas muy diferentes de observación. Así, *tiles* como el $b278$ tienen una población estelar y extinción muy alta; mientras que $b396$ o $b220$ son zonas mucho menos pobladas.

Los resultados obtenidos también se pueden utilizar como una medida de la calidad del catálogo que se genera. La Figura 6.4 se puede relacionar con el resultado final del capítulo anterior, Figura 5.4. Sería razonable que modelos generados con un *tile* dado den predicciones de baja calidad sobre *tiles* que son muy diferentes (o sea, donde se puede diferenciar claramente el origen de las fuentes). Este es el caso, por ejemplo, de los pares de *tiles* $b264$ y $b278$, o $b277$ y $b234$.

6.7. Conclusiones y trabajo a futuro

En este capítulo se presentó una forma de evaluar qué tan independientes son las características extraídas de los catálogos, respecto a la metodología de observación y reducción de datos. El proceso consiste en intentar determinar cuáles son las características más importantes a la hora de determinar **dónde**

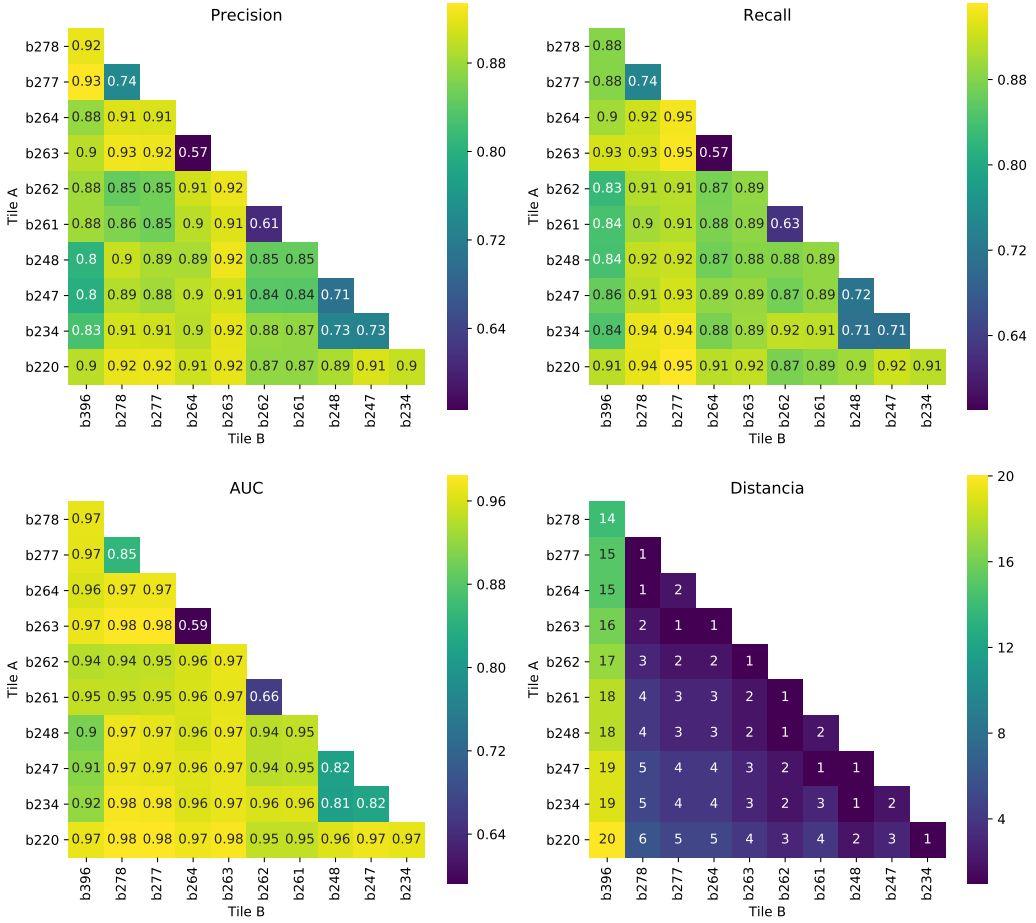

Figura 6.4: Resultados de las tres métricas de clasificación: *Precision* a la izquierda arriba, *recall* derecha arriba y AUC izquierda abajo; de utilizar las 15 características seleccionadas con RFE en todas las posibles combinaciones de *2-en-2* de todos los *tiles* disponibles. Además se acompaña una tabla de distancias entre *tiles* (derecha abajo). Cada celda de cada una de las tres primeras tablas contienen la métrica para la ejecución de una clasificación sobre un conjunto de datos compuesto únicamente por los *tiles* identificados en la fila y la columna, utilizando 10 *k-folds*. Puede observarse como las clasificaciones son consistentes con promedios de *precision* de 0,86, de *recall* de 0,87 y de AUC de 0,93.

se está observando (o qué *tile* en el contexto del VVV), obviando el **qué** se está observando.

Se logró extraer un conjunto de 15 características, que poseen un *precision* promedio del 86 % para determinar cuál es el *tile* al que pertenece una fuente, y que podrían entonces estar afectadas en mayor medida por el ruido. Así mismo, se percibe un incremento en las métricas de calidad a medida que los *tiles* son más distantes y por ende más distintos observacionalmente, que podría utilizarse como indicio de una peor calidad en la generación de catálogos.

Como trabajo a futuro queda pendiente el análisis de estas características, desde el punto de vista astronómico-observacional, así como la comparación de las medidas de calidad con más *tiles* para lograr una mejor apreciación de las mismas.

Capítulo 7

Conclusiones y trabajo a futuro

Si bien se realizaron conclusiones por capítulo a lo largo de todo el trabajo, estos párrafos serán utilizados para sumarizar y discutir brevemente los principales hallazgos y resultados en conjunto.

Por el lado del Capítulo 3 se presentó el relevamiento astronómico VVV y sus particularidades observacionales; haciendo hincapié en la forma en la cual se utiliza el telescopio *VISTA* y la cámara *VIR-CAM* para observar el cielo, y cómo con estos datos se pueden reconstruir curvas de luz de las fuentes observadas para luego extraer características útiles en métodos automáticos. En este caso, quedan pendientes el diseño de características extraídas directamente de la morfología de la curva puesta en fase; o en otras palabras tratar a la curva como una foto y sus observaciones como píxeles; este tipo de características serían en extremo útiles para utilizar aprendizaje profundo en este contexto.

El Capítulo 4 presenta una herramienta alternativa para el procesamiento de un gran volumen de datos llamada *Corral*, que además, de ser usada correctamente, brinda una idea de calidad de todo el ambiente creado sobre ella, al basarse en conceptos teóricos del área de ingeniería de software. Además, se presentó una segunda herramienta, *feets*, responsable de la extracción de características de curvas de luz, siendo esta una extensión de una herramienta existente, la cual fue totalmente rediseñada y puesta a punto para lidiar con un gran volumen de datos en menos tiempo. En el caso de *Corral* se ha pensado como trabajo a futuro extender su funcionalidad (o reescribirla) sobre una plataforma real de cómputo distribuido como puede ser *Apache Spark* (Zaharia et al., 2010) o *Dask* (Rocklin, 2015). En el caso *feets*, ya se está trabajando en extender sus funcionalidades más allá del proceso por lotes que plantea su diseño, para permitir el análisis interactivo de las características extraídas.

Continuando con el Capítulo 5, se determinó el modo más eficiente para generar catálogos de estrellas variables tipo *RR-Lyrae* de forma automática de los *tiles* del VVV. Esto se realizó teniendo especial cuidado en el balance de los datos utilizados para entrenamiento y prueba, así como una estimación de como será el funcionamiento de clasificadores entrenados con muestras mucho menores en tamaño de las que serán utilizadas para generar los catálogos. Como trabajo a futuro se generarán y publicarán catálogos de estrellas *RR-Lyrae* candidatas

para todos los *tiles* comprendidos en este estudio, haciendo principal énfasis en $b396$, por ser el menos estudiado.

Finalmente el Capítulo 6 presentó una forma de evaluar qué tan independientes son las características extraídas de los catálogos, respecto a la metodología de observación y reducción de datos. Contrariamente al capítulo anterior, el cual asume que los *tiles* son homogéneos entre si pero internamente son heterogéneos (y por eso pueden identificarse las fuentes); en este caso se plantea que tan heterogéneos son *tiles* de contenido homogéneo. Así el procedimiento consiste en intentar determinar cuáles son las características más importantes a la hora de determinar "**dónde**" se está observando (o qué *tile* en el contexto del VVV) obviando el "**qué**" se está observando. Se logró extraer un conjunto de 15 características, que poseen un *precision* promedio del 86 % para determinar cuál es el *tile* al que pertenece una fuente, y que podrían entonces estar afectadas en mayor medida por el ruido, el cual se incrementa cuando los *tiles* son más cercanos entre si. Como trabajo a futuro queda pendiente el análisis de estas características, desde el punto de vista astronómico-observacional, así como la comparación de las medidas de calidad con más *tiles* para lograr una mejor apreciación de las mismas.

Apéndice A

Guía rápida para crear *pipelines* con *Corral*

Aquí se presentan dos ejemplos de desarrollo de *pipelines* con *Corral*. El primero calcula estadísticas simples sobre el famoso conjunto de datos de flores *IRIS* (Fisher, 1936), mientras que el segundo detecta exo-planetas[1] dentro de la zona habitable[2] de su estrella utilizando el conjunto de datos de "Exoplanets Data Explorer"[3] (Han et al., 2014).

A.1. Instalación y configuración

El método de instalación recomendado es utilizar la herramienta *pip*:

```
$ pip install -U corral-pipeline
```

Otros métodos también son posibles, y se detallan en la documentación `http://corral.readthedocs.io/en/latest/install.html`.

El flujo de trabajo para crear un *pipeline* se puede resumir de la siguiente manera:

- Crear el *pipeline*.
- Definir los modelos.
- Crear la base de datos.
- Cargar los datos con el *loader*.
- Definir los *steps*.
- Ejecutar el *pipeline*.

Así el primer paso se lleva adelante ejecutando el comando

[1] Planetas fuera del sistema solar

[2] El área alrededor de una estrella en la cual estén dadas las condiciones para que los planetas que que allí se encuentren posean una temperatura que permita la existencia de agua líquida en su superficie

[3] `http://exoplanets.org/`

```
$ corral crear my_pipeline
```

Lo cual crea un archivo `in_corral.py` y un nuevo directorio `my_pipeline`. El archivo `in_corral.py` es el punto de acceso al *pipeline*, y permite ejecutar los dentro de su entorno.

`my_pipeline` puede reemplazarse por IRIS o EXO, para los ejemplos en datos IRIS o datos de exo–planetas, respectivamente. Así también el directorio `my_pypeline` es el directorio raíz del *pipeline*, además de ser un paquete *Python*. Este directorio contiene archivos cada uno con un propósito específico:

my_pipeline/__init__.py Archivo vacío que informa a *Python* que este directorio debe ser considerado como paquete.

my_pipeline/pipeline.py Configuraciones globales del *pipeline*.

my_pipeline/models.py Definiciones de entidades que serán almacenadas en la base de datos del *pipeline*.

my_pipeline/load.py Contiene la clase que define el puntos de entrada de los datos crudos al *stream* del pipeline.

my_pipeline/steps.py Contiene todos los *steps* del *pipeline*.

my_pipeline/alerts.py Definiciones de los canales de comunicación hacia el exterior del pipeline.

my_pipeline/commands.py Usado para agregar comandos de consola personalizados.

Así el primer paso a realizar luego de la creación del *pipeline* es editar el archivo `settings.py`, el cual contiene variables que deben ser configuradas para poner en marcha el proyecto. En particular unas de las variables más importantes es la que contiene la ubicación del proyecto:

```
PATH = os.path.abspath(os.path.dirname(__file__))
```

La cual es útil a la hora de definir ubicaciones relativas al *pipeline*. Por ejemplo si desea almacenar la ubicación del archivo que contiene los datos de *IRIS*, `iris.csv`[4] el cual se encuentra dentro del mismo pipeline, se podría hacer de la siguiente manera:

```
IRIS_PATH = os.path.join(PATH, "iris.csv")
```

Este archivo contiene 5 columnas que listan: el largo y el ancho de los *sépalos* y los *pétalos* y la especie de cada observación. Las 3 especies que componen este conjunto de datos son o de datos multi–variados el cual almacena 3 especies de flores "*Iris Setosa*", "*Iris Virginica*" e "*Iris Versicolor*".

Por el lado del ejemplo de los exo-planetas el subconjunto de datos obtenidos de `exoplanets.org`, se encuentra almacenada en el archivo `exoplanets.csv`[5], y contiene las siguientes columnas: nombre; periodo y masa del planeta; separación entre la estrella y el planeta; distancia del planeta a su estrella; y finalmente masa, radio, temperatura y metalicidad de la estrella anfitriona. En este caso la única diferencia en el archivo `settings.py` con respecto al ejemplo de IRIS es:

[4] `https://github.com/toros-astro/corral/raw/master/datasets/iris.csv`
[5] `https://github.com/toros-astro/corral/raw/master/datasets/exoplanets.csv`

```
EXO_PATH = os.path.join(PATH, "exoplanets.csv")
```

Al margen de estas configuraciones básicas, los nombres del *loader*, los *steps* y las *alerts*, deben ser listados en las variables `LOADER`, `STEPS` y `ALERTS` respectivamente. Por ejemplo para el *pipeline* de IRIS se crearon 4 *steps*.

```
STEPS = [
    "pipeline.steps.StatisticsCreator",
    "pipeline.steps.SetosaStatistics",
    "pipeline.steps.VirginicaStatistics",
    "pipeline.steps.VersicolorStatistics"]
```

A.2. Procesando los datos de IRIS

Como parte esencial del patrón MVC, el siguiente paso en la preparación del *pipeline* es definir los modelos. Los modelos determinan la estructuras de los datos, y se definen en el archivo `models.py`. A continuación se crearán dos modelos:

- `Name` que solo almacenara el nombre de cada especie.
- `Flower` el cual define una tabla con 4 columnas que almacenaran los datos del los largos y anchos de los pétalos y los sépalos, además de contener una referencia a la tabla `Name`.

Para que un modelo cree una tabla uno de los requerimientos es que herede de la clase `corral.db.Model`

```
from corral import db

class Name(db.Model):

    __tablename__ = 'Name'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), unique=True)


class Flower(db.Model):

    __tablename__ = 'Flower'

    id = db.Column(
        db.Integer, primary_key=True)

    name_id = db.Column(
        db.Integer, db.ForeignKey('Name.id'),
        nullable=False)
    name = db.relationship(
        "Name", backref=db.backref("flowers"))
    sepal_l = db.Column(
```

```
        db.Float, nullable=False)
    sepal_w = db.Column(
        db.Float, nullable=False)
    petal_l = db.Column(
        db.Float, nullable=False)
    petal_w = db.Column(
        db.Float, nullable=False)
```

En el ejemplo de IRIS, para calcular una estadística básica (la media para cada columna de datos), se debe crear una tabla adicional para almacenar los resultados (y cualquier otro valor que pueda ser requerido). Esta tabla en la base de datos está vinculada a la tabla `Flower` por su clave primaria, que se genera automáticamente.

```
class Statistics(db.Model):

    __tablename__ = 'Statistics'

    id = db.Column(db.Integer, primary_key=True)
    name_id = db.Column(
        db.Integer, db.ForeignKey('Name.id'),
        nullable=False, unique=True)
    name = db.relationship(
        "flowers", uselist=False,
        backref=db.backref("statistics"))

    mean_sepal_l = db.Column(
        db.Float, nullable=False)
    mean_sepal_w = db.Column(
        db.Float, nullable=False)
    mean_petal_l = db.Column(
        db.Float, nullable=False)
    mean_petal_w = db.Column(
        db.Float, nullable=False)

    def __repr__(self):
        return "<Statistics of '{}'>".format(
            self.name.name)
```

Con los modelos definidos, la base de datos es creada con el comando:

```
$ python in_corral.py createdb
```

En este punto la base de datos no contiene datos, por lo que deben definirse los *loaders* para la carga inicial con el comando

```
$ python in_corral.py load
```

El cual utiliza la clase definida en `load.py`, la cual lee la información del archivo `iris.csv` y la almacena dentro de la base de datos.

```
class Loader(run.Loader):

    def setup(self):
        self.fp = open(settings.IRIS_PATH)

    def get_name_instance(self, row):
        name = self.session.query(
            models.Name
        ).filter(
            models.Name.name == row["Name"]
        ).first()

        if name is None:
            name = models.Name(name=row["Name"])

            # we need to add the new
            # instance and save it
            self.save(name)
            self.session.commit()

        return name

    def store_observation(self, row, name):
        return models.Flower(
            name=name,
            sepal_l=row["SepalLength"],
            sepal_w=row["SepalWidth"],
            petal_l=row["PetalLength"],
            petal_w=row["PetalWidth"])

    def generate(self):
        for row in csv.DictReader(self.fp):
            name = self.get_name_instance(row)
            obs = self.store_observation(
                row, name)
            yield obs

    def teardown(self, *args):
        if self.fp and not self.fp.closed:
            self.fp.close()
```

En este ejemplo el método `setup()` se ejecuta justo antes del método `generate()` por lo cual es el mejor lugar para abrir el archivo de datos `iris.csv`. Por otro lado `teardown()` es ejecutado justo después de `generate()` y es el encargado de cerrar el archivo. La lectura real de cada línea del archivo de datos se divide en dos partes durante la ejecución de `generate()`:

1. El método llamado `get_name_instance` recibe a la fila como parámetro y retorna la instancia de `Name` correspondiente, creándola de ser necesario.

2. `store_observarion`, por otra parte, también recibe la fila como parámetro además de la instancia de `Name` y retorna una nueva instancia de `Observation`.

La línea final del *loader* `yields obs` entrega la nueva observación a corral para ser almacenada.

Una vez que los datos se han almacenado en la base de datos, todos los pasos de procesamiento se pueden escribir en el archivo `steps.py` Así el primer *step* se encargará de crear los objetos `Statistics` para cada tipo de *iris*:

```
from . import models

class StatisticsCreator(run.Step):

    model = models.Name
    conditions = []

    def process(self, name):
        stats = self.session.query(
            models.Statistics
        ).filter(
            models.Statistics.name_id == name.id
        ).first()
        if stats is None:
            yield models.Statistics(
                name_id=name.id,
                mean_sepal_l=0.,
                mean_sepal_w=0.,
                mean_petal_l=0.,
                mean_petal_w=0.)
```

Mientras que el segundo se encarga de realizar el cálculo estadístico para cada objeto `Statistic` que tenga los valores en 0.

```
class SetosaStatistics(run.Step):

    model = models.Statistics
    conditions = [model.mean_sepal_l==0.]

    def process(self, stats):
        sepal_l, sepal_w = [], []
        petal_l, petal_w = [], []
        for obs in stats.name.flowers:
            sepal_l.append(obs.sepal_l)
            sepal_w.append(obs.sepal_w)
            petal_l.append(obs.petal_l)
            petal_w.append(obs.petal_w)
        stats.mean_sepal_l = sum(sepal_l)
        stats.mean_sepal_w = sum(sepal_w)
        stats.mean_petal_l = sum(petal_l)
        stats.mean_petal_w = sum(petal_w)
```

```
        stats.mean_sepal_l = (
            stats.mean.sepal_l/len(sepal_l))
        stats.mean_sepal_w = (
            stats.mean.sepal_w/len(sepal_w))
        stats.mean_petal_l = (
            stats.mean.petal_l/len(petal_l))
        stats.mean_petal_w = (
            stats.mean.petal_w/len(petal_w))
```

Finalmente todo el pipeline se puede ejecutar con el comando

```
$ python in_corral run_all
```

La implementación completa de este *pipeline* puede encontrarse en `https://github.com/toros-astro/corral_iris`.

A.3. *Pipeline* de Exoplanetas

El *pipeline* para los exoplanetas tiene una estructura de datos diferentes, por ejemplo:

```
class Planet(db.Model):

    __tablename__ = 'Planet'

    id = db.Column(
        db.Integer, primary_key=True)

    nomb = db.Column(db.Float, nullable=False)
    per = db.Column(db.Float, nullable=False)
    mass = db.Column(db.Float, nullable=False)
    sep = db.Column(db.Float, nullable=False)
    dist = db.Column(db.Float, nullable=False)
    mstar = db.Column(db.Float, nullable=False)
    rstar = db.Column(db.Float, nullable=False)
    teff = db.Column(db.Float, nullable=False)
    fe = db.Column(db.Float, nullable=False)
```

La carga de datos se realiza con solamente una tabla; además de que la lógica es muy parecida al ejemplo de *IRIS*, exceptuando que ahora utilizamos la variable `settings.EXO_PATH`;

```
import csv
from corral import run
from corral.conf import settings
from exo import models

class Loader(run.Loader):
    """Extract data from the `exoplanets.csv` and feed
    the stream of the pipeline.
```

```
    """

    def setup(self):
        # we open the file and assign it to
        # an instance variable
        self.fp = open(settings.EXO_PATH)

    def float_or_none(self, value):
        try:
            return float(value)
        except (TypeError, ValueError):
            return None

    def generate(self):
        # now we make use of "self.fp"
        # for the reader
        for row in csv.DictReader(self.fp):
            di = {
                'name': row['NAME'],
                'per': self.float_or_none(
                    row['PER']),
                'mass': self.float_or_none(
                    row['MASS']),
                'sep': self.float_or_none(
                    row['SEP']),
                'dist': self.float_or_none(
                    row['DIST']),
                'mstar': self.float_or_none(
                    row['MSTAR']),
                'rstar': self.float_or_none(
                    row['RSTAR']),
                'teff': self.float_or_none(
                    row['TEFF']),
                'fe': self.float_or_none(
                    row['FE'])}
            yield models.Planet(**di)

    def teardown(self, *args):
        # checking that the
        # file is really open
        if self.fp and not self.fp.closed:
            self.fp.close()
```

donde la función `float_or_none` permite lidiar con los valores faltantes, lo cual es muy común en la base de datos de *exoplanets.org*. Este *pipeline* también puede ser extendido agregando *steps* y *alerts*.

Por ejemplo, un *step* se puede configurar para filtrar el conjunto de datos, calcular parámetros de correlación, o aplicar técnicas de aprendizaje automático para descubrir la agrupaciones o realizar clasificaciones. Además, el entorno *Python* permite escribir alertas, que pueden configurarse para producir gráficos,

enviar los resultados por por correo electrónico o reemplazar información en una página web. Aquí mostramos un ejemplo de un paso, que determina la lista de planetas incluidos en la zona habitable, y de un alerta, que realiza un diagrama de dispersión de la masa y el período de los planetas en la zona habitable.

Para agregar una lista de planetas en la zona habitable a la base de datos, creamos una nueva tabla:

```
class HabitableZone(run.Step):
    __tablename__ = "HabitableZoneStats"

    id = db.Column(
        db.Integer, primary_key=True)

    planet_id = db.Column(
        db.Integer, db.ForeignKey('Planet.id'),
        nullable=False)
    planet = db.relationship("Planet",
        backref=db.backref("hzones"))

    luminosity = db.Column(db.Float)
    radio_inner = db.Column(db.Float)
    radio_outer = db.Column(db.Float)

    in_habitable_zone = db.Column(db.Boolean)
```

Ahora es necesario agregar un *step* que busque a los planetas que se encuentren en zona habitable. Para este objetivo computamos los límites de la zona habitable como $rin = L/(1{,}1 * L_{sun})$ y $rout = L/(0{,}53 * L_{sun})$, siendo L

$$L = 4\pi R_*^2 * \sigma T_{eff}^4 \tag{A.1}$$

Donde L_{sun} es la luminosidad del sol (proporcionada en la librería *astropy* como `astropy.constants.L_sun`), σ es la contante de Stefan Boltzmann (también presente en *astropy* como `astropy.constants.sigma_sb`), R_* es el radio de la estrella, y T_{eff} es la temperatura efectiva de la estrella.

```
from corral import run
import numpy as np
from astropy import units as u, constants as c
from . import models

STEFAN_BOLTZMANN = c.sigma_sb
SUN_LUMINOSITY = c.L_sun

class HabitableZone(run.Step):
    """Compute some statistics of the star of
    a given planet and then determines if is in
    their habitable zone.
    """

    model = models.Planet
```

```
    conditions = [model.rstar != None,
                  model.teff != None]

    def process(self, planet):
        # habitable zone of the host star
        Rstar  = (planet.rstar * u.solRad).to('m')
        Teff = planet.teff * u.K
        luminosity = (
            STEFAN_BOLTZMANN * 4 * np.pi *
            (Rstar ** 2) * (Teff ** 4))
        lratio = luminosity / SUN_LUMINOSITY
        rin = np.sqrt(lratio / 1.1)
        rout = np.sqrt(lratio / 0.53)

        in_hz = (
            planet.sep >= rin and
            planet.sep <= rout)
        return models.HabitableZoneStats(
            planet=planet,
            in_habitable_zone=in_hz,
            luminosity=lratio.value,
            radio_inner=rin.value,
            radio_outer=rout.value)
```

Finalmente se presenta la *alert* que produce un *scatter-plot* de masa contra periodo de cada estrella en zona habitable (un ejemplo de la salida de esta alerta puede observarse en la figura A.1)

```
class LogScatter(ep.EndPoint):

    def __init__(self, path, xfield, yfield,
                 title, **kwargs):
        self.path = path
        self.xfield = xfield
        self.yfield = yfield
        self.title = title
        self.kwargs = kwargs
        self._x, self._y = [], []

    def process(self, hz):
        planet = hz.planet
        x = getattr(planet, self.xfield)
        y = getattr(planet, self.yfield)
        if x and y:
            self._x.append(x)
            self._y.append(y)

    def teardown(self, *args):
        plt.scatter(
            p.log(self._x),
```

```
            np.log(self._y), **self.kwargs)
        plt.title(self.title)
        plt.legend(loc="best")
        plt.savefig(self.path)
        super(LogScatter, self).teardown(*args)

class PlotAlert(run.Alert):
    """Store a list of planets in habitable
    zone in a log file and also generate a
    period vs mass plot of this planets
    """

    model = models.HabitableZoneStats
    conditions = [
        model.in_habitable_zone == True]
    alert_to = [
        ep.File("in_habzone.log"),
        LogScatter(
            "in_habzone.png",
            xfield="per", yfield="mass",
            title="Period Vs Mass",
            marker="*")]

    def render_alert(self, now, ep, hz):
        planet = hz.planet
        data = []
        for fn in planet.__table__.c:
            data.append([fn.name,
            getattr(planet, fn.name)])
        fields = ", ".join(
            "{}={}".format(k, v) for k, v in data)
        return "[{}] {}\n".format(
            now.isoformat(),  fields)
```

Para generar un informe sobre la calidad del *pipeline*, es necesario escribir algunas pruebas. A continuación, se presentan dos casos de prueba que verifican la consistencia en los datos.

El caso `HabitableZoneTest` crea una instancia de un planeta con valores $R_* = 1$ y $T_{eff} = 1$; esto generará una luminosidad de $4 * \pi * \sigma / L_\odot$ lo cual no ubicará al planeta en la zona habitable. Además se verifica que los límites de la zona habitable sean consistentes, esto es que el límite interior sea menor que el límite exterior.

```
class HabitableZoneTest(qa.TestCase):

    subject = steps.HabitableZone

    def setup(self):
        planet = models.Planet(
            name="foo", rstar=1, teff=1)
```

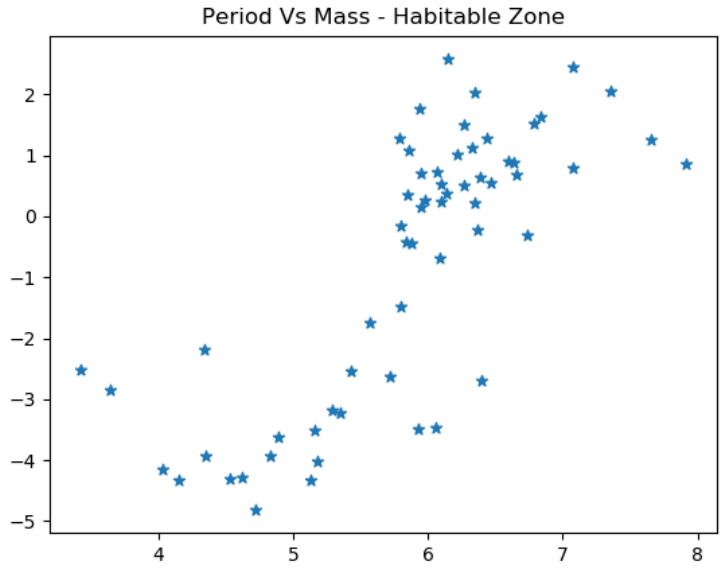

Figura A.1: Gráfico de masa (eje vertical) vs periodo (eje horizontal) de planetas encontrados en zona habitable por el *pipeline* de exo-planetas. El gráfico se presenta tal cual como es generado por la *alert*, es por esto que carece de etiquetas para los ejes.

```
        self.save(planet)

    def validate(self):
        planet = self.session.query(
            models.Planet).first()
        hzone = planet.hzones[0]
        self.assertAlmostEquals(
            hzone.luminosity, 8.96223797571e-10)
        self.assertLess(
            hzone.radio_inner, hzone.radio_outer)
        self.assertFalse(hzone.in_habitable_zone)
        self.assertStreamCount(
            1, models.HabitableZoneStats)
```

El otro caso de prueba verifica que si un planeta no posee las dos columnas necesarias para generar la *alert* (se recuerda que los datos faltantes son un caso común en *exoplanets.org*), no crea ninguna entrada en la tabla `HabitableZoneStats`

```
class HabitableZoneNoRstarNoTeffTest(qa.TestCase):

    subject = steps.HabitableZone

    def setup(self):
```

```
        planet = models.Planet(name="foo")
        self.save(planet)

    def validate(self):
        self.assertStreamCount(0,
                models.HabitableZoneStats)
```

Estas pruebas ejecutan el 81,94 % del código del *pipeline* y generan un $QAI =$ 27,31 % con calificación de F. Es evidente que para mejorar esta calidad, más pruebas deben ser escritas.

Este ejemplo y sus documentos estructurales asociados pueden encontrarse en:

- Repositorio de código: `https://github.com/toros-astro/corral_exoplanets`
- Reporte de calidad: `https://github.com/toros-astro/corral_exoplanets/blob/master/exo/docs/qareport.md`
- Diagrama de clases de los modelos: `https://github.com/toros-astro/corral_exoplanets/blob/master/exo/models.png`
- Documentación: `https://github.com/toros-astro/corral_exoplanets/blob/master/exo/doc.md`

Apéndice B

Reporte de calidad de *Carpyncho*

Se incluye en este apéndice la salida del informe de calidad auto-generado del código fuente de *Carpyncho*

B.1. Carpyncho Pipeline Quality Report

```
    * Created at: `2018-12-06 10:55:34.688758`
    * Corral Version: `0.3`
```

B.1.1. Summary

```
    * Tests Success: `Yes`
    * Tests Ran: `16`
    * Processors: `16`
    * Coverage: `80.46%`
    * Maintainability & Style Errors: `0`
    * QA Index: `80.46%`
    * QA Qualification: `B-`
```

B.1.2. About The Corral Quality Assurance Index (QAI)

```
QAI = 2 * (TP * (PT/PNC) * COV) / (1 + exp(MSE/tau))

Where:
    TP: If all tests passes is 1, 0 otherwise.
    PT: Processors and commands tested.
    PCN: The number number of processors
         (Loader, Steps and Alerts) and commands.
    COV: The code coverage (between 0 and 1).
    MSE: The Maintainability and Style Errors.
    tau: Tolerance of style errors per file
```

```
Current Value of Tau: `13.00` per file
```

B.1.3. About The Qualification

The Corral qualification is a quantitave scale based on QAI

```
* `QAI >= 00.00% -- F`
* `QAI >= 60.00% -- D-`
* `QAI >= 63.00% -- D`
* `QAI >= 67.00% -- D+`
* `QAI >= 70.00% -- C-`
* `QAI >= 73.00% -- C`
* `QAI >= 77.00% -- C+`
* `QAI >= 80.00% -- B-`
* `QAI >= 83.00% -- B`
* `QAI >= 87.00% -- B+`
* `QAI >= 90.00% -- A-`
* `QAI >= 93.00% -- A`
* `QAI >= 95.00% -- A+`
```

B.1.4. Full Output

```
Tests

    runTest (carpyncho.tests.LoaderTestCase) ... ok
    runTest (carpyncho.tests.CreateLightCurvesTestCase) ... ok
    runTest (carpyncho.tests.FeaturesExtractorTestCase) ... ok
    runTest (carpyncho.tests.MatchTestCase) ... ok
    runTest (carpyncho.tests.PrepareForMatchTestCase) ... ok
    runTest (carpyncho.tests.ReadPawprintStackTestCase) ... ok
    runTest (carpyncho.tests.ReadTileTestCase) ... ok
    runTest (carpyncho.tests.UnredTestCase) ... ok
    runTest (carpyncho.tests.VSTagTileTestCase) ... ok
    runTest (carpyncho.tests.LSPawprintTestCase) ... ok
    runTest (carpyncho.tests.SampleFeaturesTestCase) ... ok
    runTest (carpyncho.tests.LSTileTestCase) ... ok
    runTest (carpyncho.tests.SetTileStatusTestCase) ... ok
    runTest (carpyncho.tests.PathsTestCase) ... ok
    runTest (carpyncho.tests.BuildBinCase) ... ok
    runTest (carpyncho.tests.LSSyncTestCase) ... ok

    ----------------------------------------------------------------------
    Ran 16 tests in 912.936s

    SUCCESS


Coverage
```

```
Name                                                Stmts   Miss  Cover
-------------------------------------------------------------------
carpyncho/__init__.py                                   1      0   100%
carpyncho/alerts.py                                     7      7     0%
carpyncho/bin/__init__.py                               9      4    56%
carpyncho/bin/catalogs/__init__.py                     73     59    19%
carpyncho/bin/vvv_flx2mag/__init__.py                  11      2    82%
carpyncho/commands.py                                 128     13    90%
carpyncho/lib/__init__.py                               0      0   100%
carpyncho/lib/beamc.py                                109     75    31%
carpyncho/lib/context_managers.py                       8      0   100%
carpyncho/lib/matcher.py                                9      0   100%
carpyncho/lib/mppandas.py                              13      0   100%
carpyncho/load.py                                      50      0   100%
carpyncho/local_settings.py                             5      0   100%
carpyncho/models/__init__.py                            3      0   100%
carpyncho/models/pawprint_stack.py                     42      1    98%
carpyncho/models/psxt.py                               30      2    93%
carpyncho/models/tile.py                               94     10    89%
carpyncho/pipeline.py                                  25      8    68%
carpyncho/settings.py                                  38      5    87%
carpyncho/steps/__init__.py                             0      0   100%
carpyncho/steps/create_lc.py                           43      7    84%
carpyncho/steps/features_extractor.py                 222     35    84%
carpyncho/steps/match.py                               56     15    73%
carpyncho/steps/prepare_for_match.py                   10      0   100%
carpyncho/steps/read_pawprint_stack.py                 80      1    99%
carpyncho/steps/read_tile.py                           43      0   100%
carpyncho/steps/tag_tile.py                            51      3    94%
carpyncho/steps/unred.py                               91     45    51%
carpyncho/tests.py                                    259      3    99%
-------------------------------------------------------------------
TOTAL                                                1510    295    80%

MAINTAINABILITY & STYLE

    No-Errors
```

Apéndice C

Experimentos para la generación de catálogos de *RR-Lyrae* en el VVV

Se presentan a continuación los enlaces correspondientes a todos los experimentos realizados para la creación del Capítulo 5.

Todos los experimentos están organizados en *notebooks* del proyecto para cómputo científico interactivo *Jupyter*[1] (Kluyver et al., 2016) utilizando el lenguaje *Python*.

La totalidad de los experimentos están disponibles en el repositorio público ubicado en `https://github.com/carpyncho/vvv_gen_catalog/`.

C.1. *Notebooks*

Los archivos y directorios relevantes dentro del repositorio son:

data Directorio con archivos de datos en formato *numpy*(`*.npy`) (Kern, 2007) o *Pandas* (`*.pkl.bz2`) (McKinney, 2011). En este directorio se encuentran los conjuntos de datos pequeños (2.5 mil observaciones de estrellas desconocidas), medianos (5 mil) y grandes (20 mil).

lib utilidades para la realización de experimentos.

00_scaler.ipynb Se realiza la normalización y selección principal de características y datos de los valores de los tres conjuntos de datos (pequeño, mediano y grande).

01_sample20k.ipynb Se realiza la totalidad de los experimentos de entrenamiento y prueba (explicado en detalle en la sección **Generalización** del capítulo 5) para las muestras grandes.

02_sample5k.ipynb Se realiza la totalidad de los experimentos de entrenamiento y prueba (explicado en detalle en la sección **Generalización** del capítulo 5) para las muestras medianas.

[1] `https://jupyter.org/`

03_sample2.5k.ipynb Se realiza la totalidad de los experimentos de entrenamiento y prueba (explicado en detalle en la sección **Generalización** del capítulo 5) para las muestras pequeñas.

04_rf_trains_testb.ipynb Se realizan los experimentos de probar con un tamaño de muestra pequeña o grande y evaluar con la muestra grande.

05_resume_table_only_ogle3.ipynb *Notebook* intermedio para la generación de tablas comparativas entre clasificadores utilizando *k-folds* sobre el *tile* *b*278.

06_ogle3_train_ogle4_check.ipynb Chequea cuántos de los FP en las clasificaciones utilizando solo *OGLE-III* en realidad son estrellas variables identificadas en *OGLE-IV* o *VizieR*.

07_scaler-o3o4vZ.ipynb Se realiza la normalización y selección principal de características y datos de los valores de los tres conjuntos de datos (pequeño, mediano y grande). A diferencia de `00_scaler.ipynb` aquí los datos utilizados son los que poseen como casos positivos a todas las *RR-Lyrae* identificadas en *OGLE-III*, *OGLE-IV* y *VizieR*.

08_all_vs_all_vs.ipynb Se ejecuta el experimento de entrenar en cada *tile* y probar con todos los demás. Para evaluar la generalización de los resultados.

09_analysis_avs_vs.ipynb Generación de los gráficos y tablas del *notebook* anterior.

10_train5_testALL.ipynb Prueba de entrenar con las muestras medianas en *b*278 y probar con todo el *tile* en *b*261, y viceversa.

Apéndice D

Experimentos para la detección de ruido experimental en fuentes del VVV

Se presentan a continuación los enlaces correspondientes a todos los experimentos realizados para la creación del Capítulo 6.

Todos los experimentos están organizados en *notebooks* del proyecto para cómputo científico interactivo *Jupyter*[1] (Kluyver et al., 2016) utilizando el lenguaje *Python*.

La totalidad de los experimentos están disponibles en el repositorio público ubicado en `https://github.com/carpyncho/paper_inoise`.

D.1. *Notebooks*

Los archivos y directorios relevantes dentro del repositorio son

data Directorio con archivos de datos en formato *numpy*(`*.npy`) (Kern, 2007) o *Pandas* (`*.pkl.bz2`) (McKinney, 2011).

lib utilidades para la realización de experimentos.

plots figuras en formato *pdf* utilizadas para las publicaciones.

00_scaler.ipynb Se realiza la normalización y selección principal de características y datos de los valores de los tres conjuntos de datos (pequeño, mediano y grande).

01_HP_selection.ipynb Selección de hiper-parámetros.

02_models.ipynb Selección de modelos.

[1] `https://jupyter.org/`

03_FS.ipynb Selección de características, y evaluación del modelo elegido con ellas.

04_final.ipynb Comparación de todos los *tiles* con las características seleccionadas.

05_tables.ipynb Tablas finales y mapas de calor resumen, del *notebook* anterior.

Bibliografía

Abbott, B., Abbott, R., Abbott, T., Abernathy, M., Acernese, F., Ackley, K., Adams, C., Adams, T., Addesso, P., Adhikari, R., et al. (2016a). Localization and broadband follow-up of the gravitational-wave transient gw150914. *The Astrophysical journal letters*, 826(1):L13.

Abbott, B., Abbott, R., Adhikari, R., Ananyeva, A., Anderson, S., Appert, S., Arai, K., Araya, M., Barayoga, J., Barish, B., et al. (2017). Multi-messenger observations of a binary neutron star merger. *Astrophysical Journal Letters*, 848(2):L12.

Abbott, B. P., Abbott, R., Abbott, T. D., Abernathy, M. R., Acernese, F., Ackley, K., Adams, C., Adams, T., Addesso, P., Adhikari, R. X., and al, e. (2016b). Binary Black Hole Mergers in the First Advanced LIGO Observing Run. *Physical Review X*, 6(4):041015.

Abbott, B. P., Abbott, R., Abbott, T. D., Abernathy, M. R., Acernese, F., Ackley, K., Adams, C., Adams, T., Addesso, P., Adhikari, R. X., and et al. (2016). Binary Black Hole Mergers in the First Advanced LIGO Observing Run. *Physical Review X*, 6(4):041015.

Abramovici, A., Althouse, W. E., Drever, R. W. P., Gursel, Y., Kawamura, S., Raab, F. J., Shoemaker, D., Sievers, L., Spero, R. E., and Thorne, K. S. (1992). LIGO - The Laser Interferometer Gravitational-Wave Observatory. *Science*, 256:325–333.

Alcock, C., Allsman, R., Alves, D. R., Axelrod, T., Becker, A. C., Bennett, D., Cook, K. H., Dalal, N., Drake, A. J., Freeman, K., et al. (2000). The macho project: microlensing results from 5.7 years of large magellanic cloud observations. *The Astrophysical Journal*, 542(1):281.

Allen, G., Anderson, W., Blaufuss, E., Bloom, J. S., Brady, P., Burke-Spolaor, S., Cenko, S. B., Connolly, A., Couvares, P., Fox, D., et al. (2018). Multi-messenger astrophysics: Harnessing the data revolution. *arXiv preprint arXiv:1807.04780*.

Angeloni, R., Ramos, R. C., Catelan, M., Dékány, I., Gran, F., Alonso-García, J., Hempel, M., Navarrete, C., Andrews, H., Aparicio, A., Beamín, J. C., Berger, C., Borissova, J., Peña, C. C., Cunial, A., de Grijs, R., Espinoza, N., Eyheramendy, S., Lopes, C. E. F., Fiaschi, M., Hajdu, G., Han, J., Hełminiak, K. G., Hempel, A., Hidalgo, S. L., Ita, Y., Jeon, Y.-B., Jordán, A., Kwon, J., Lee, J. T., Martín, E. L., Masetti, N., Matsunaga, N., Milone, A. P.,

Minniti, D., Morelli, L., Murgas, F., Nagayama, T., Navarro, C., Ochner, P., Pérez, P., Pichara, K., Rojas-Arriagada, A., Roquette, J., Saito, R. K., Siviero, A., Sohn, J., Sung, H.-I., Tamura, M., Tata, R., Tomasella, L., Townsend, B., and Whitelock, P. (2014). The VVV Templates Project. Towards an Automated Classification of VVV Light-Curves. I. Building a database of stellar variability in the near-infrared. *Astronomy & Astrophysics*, 567:A100. arXiv: 1405.4517.

Angulo, R., Springel, V., White, S., Jenkins, A., Baugh, C., and Frenk, C. (2012). Scaling relations for galaxy clusters in the millennium-xxl simulation. *Monthly Notices of the Royal Astronomical Society*, 426(3):2046–2062.

Axelrod, T., Kantor, J., Lupton, R., and Pierfederici, F. (2010). An open source application framework for astronomical imaging pipelines. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 774015–774015. International Society for Optics and Photonics.

Baade, W. (1946). A search for the nucleus of our galaxy. *Publications of the Astronomical Society of the Pacific*, 58:249–252.

Bäumer, D., Gryczan, G., Knoll, R., Lilienthal, C., Riehle, D., and Züllighoven, H. (1997). Framework development for large systems. *Communications of the ACM*, 40(10):52–59.

Beroiz, M., Colazo, C., Diaz, M., Dominguez, M., Garcia Lambas, D., Gurovich, S., Lares, M., Macri, L., Penuela, T., Rodriguez, H., Sanchez, B., and Toros Collaboration (2016). Results of optical follow-up observations of advanced LIGO triggers from O1 in the southern hemisphere. In *APS Meeting Abstracts*.

Bishop, C. M. (2006). Pattern recognition and machine learning (information science and statistics) springer-verlag new york. *Inc. Secaucus, NJ, USA*.

Bloom, J., Richards, J., Nugent, P., Quimby, R., Kasliwal, M., Starr, D., Poznanski, D., Ofek, E., Cenko, S., Butler, N., et al. (2012). Automating discovery and classification of transients and variable stars in the synoptic survey era. *Publications of the Astronomical Society of the Pacific*, 124(921):1175.

Booch, G., Rumbaugh, J., and Jacobson, I. (2006). *UML: guia do usuário*. Elsevier Brasil.

Borne, K. (2012). Astroinformatics in a nutshell.

Borne, K. D. (2010). Astroinformatics: data-oriented astronomy research and education. *Earth Science Informatics*, 3(1-2):5–17.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM.

Bowman-Amuah, M. (2004). *Processing pipeline in a base services pattern environment*. Google Patents. US Patent 6,715,145.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Cabral, J., Gurovich, S., Good, J., Medel, R., Garcia Lambas, D., Amores, E., Clariá, J., and Minniti, D. (2014). Community science with the vvv. In *Revista Mexicana de Astronomia y Astrofisica Conference Series*, volume 44, pages 212–212.

Cabral, J., Gurovich, S., Gran, F., and Minnitti, D. (2016a). Carpyncho The VVV band-merged catalogue and data mining/machine learning facility. In *7th VVV Science Workshop*.

Cabral, J., Sanchez, B., Ramos, F., Gurovich, S., Granitto, P., and VanderPlas, J. (2018a). feets: feature extractor for time series. *Astrophysics Source Code Library*.

Cabral, J. B., Granitto, P. M., Gurovich, S., and Minniti, D. (2016b). Generación de features en la búsqueda de estrellas variables en el relevamiento astronómico vvv. In *Simposio Argentino de Inteligencia Artificial (ASAI 2016)-JAIIO 45 (Tres de Febrero, 2016)*.

Cabral, J. B., Sánchez, B., Beroiz, M., Domínguez, M., Lares, M., Gurovich, S., and Granitto, P. (2017). Corral Framework: Trustworthy and Fully Functional Data Intensive Parallel Astronomical Pipelines. *Astronomy and Computing*, 20:140–154. arXiv: 1701.05566.

Cabral, J. B., Sánchez, B., Ramos, F., Gurovich, S., Vanderplas, J., and Granitto, P. (2018b). From FATS to feets: Further improvements to an astronomical feature extraction tool based on machine learning. *Astronomy and Computing*.

Cardelli, J. A., Clayton, G. C., and Mathis, J. S. (1989). The relationship between infrared, optical, and ultraviolet extinction. *The Astrophysical Journal*, 345:245–256.

Caruana, R., Karampatziakis, N., and Yessenalina, A. (2008). An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 96–103. ACM.

Catelan, M., Minniti, D., Lucas, P., Alonso-Garcia, J., Angeloni, R., Beamin, J., Bonatto, C., Borissova, J., Contreras, C., Cross, N., et al. (2011). The vista variables in the via lactea (vvv) eso public survey: Current status and first results. *arXiv preprint arXiv:1105.1119*.

Catelan, M., Minniti, D., Lucas, P., Dékány, I., Saito, R., Angeloni, R., Alonso-García, J., Hempel, M., Helminiak, K., Jordán, A., et al. (2013). Stellar variability in the vvv survey. *arXiv preprint arXiv:1310.1996*.

Cavuoti, S. (2013). Data-rich astronomy: mining synoptic sky surveys. *arXiv:1304.6615 [astro-ph]*. arXiv: 1304.6615.

Coad, P. (1992). Object-oriented patterns. *Communications of the ACM*, 35(9):152–159.

Collaboration, L. S., Collaboration, V., et al. (2016). Gw151226: observation of gravitational waves from a 22-solar-mass binary black hole coalescence. *arXiv preprint arXiv:1606.04855*.

Cook, K. H., Alcock, C., Allsman, R. A., Axelrod, T. S., Freeman, K. C., Peterson, B. A., Quinn, P. J., Rodgers, A. W., Bennett, D. P., Reimann, J., Griest, K., Marshall, S. L., Pratt, M. R., Stubbs, C. W., Sutherland, W., and Welch, D. (1995). Variable stars in the macho collaboration database. *International Astronomical Union Colloquium*, 155:221–231.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Cowling, A. J. (1998). The first decade of an undergraduate degree programme in software engineering. *Annals of Software Engineering*, 6(1-4):61–90.

Cross, N., Hambly, N., Collins, R., Bryant, J., Mann, B., Read, M., Sutorius, E., and Williams, P. (2007). Synoptic data in the wfcam & vista science archives. In *Astronomical Data Analysis Software and Systems XVI*, volume 376, page 54.

Cross, N. J., Collins, R. S., Mann, R. G., Read, M. A., Sutorius, E. T., Blake, R. P., Holliman, M., Hambly, N. C., Emerson, J. P., Lawrence, A., et al. (2012). The vista science archive. *Astronomy & Astrophysics*, 548:A119.

Dal Pozzolo, A., Boracchi, G., Caelen, O., Alippi, C., and Bontempi, G. (2015). Credit card fraud detection and concept-drift adaptation with delayed supervised information. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE.

de los Rios, M., R, D., J, M., Paz, D., and Merchán, M. (2016). The MeSsI (merging systems identification) algorithm and catalogue. *Monthly Notices of the Royal Astronomical Society*, 458(1):226–232.

Díaz, M., Macri, L., Lambas, D. G., de Oliveira, C. M., Castellón, J. N., Ribeiro, T., Sánchez, B., Schoenell, W., Abramo, L., Akras, S., et al. (2017). Observations of the first electromagnetic counterpart to a gravitational-wave source by the toros collaboration. *The Astrophysical Journal Letters*, 848(2):L29.

Diaz, M. C., Benacquista, M., Belczynski, K., Branchesi, M., Brocato, E., DePoy, D. L., Diaz, M. C., Dominguez, M., Garcia Lambas, D., Macri, L. M., Marshall, J. L., Oelkers, R. J., and Torres, C. V. (2014). The TOROS Project. In Wozniak, P. R., Graham, M. J., Mahabal, A. A., and Seaman, R., editors, *The Third Hot-wiring the Transient Universe Workshop*, pages 225–229.

Diaz, M. C., Benacquista, M., Belczynski, K., Branchesi, M., Brocato, E., DePoy, D. L., Diaz, M. C., Dominguez, M., Garcia Lambas, D., Macri, L. M., Marshall, J. L., Oelkers, R. J., and Torres, C. V. (2014). The TOROS Project. In Wozniak, P. R., Graham, M. J., Mahabal, A. A., and Seaman, R., editors, *The Third Hot-wiring the Transient Universe Workshop*, pages 225–229.

Díaz, M. C., Beroiz, M., Penuela, T., Macri, L. M., Oelkers, R. J., Yuan, W., Lambas, D. G., Cabral, J., Colazo, C., Dominguez, M., et al. (2016). Gw150914: First search for the electromagnetic counterpart of a gravitational-wave event by the toros collaboration. *The Astrophysical Journal Letters*, 828(2):L16.

Domingos, P. (2015). *The master algorithm: How the quest for the ultimate learning machine will remake our world*. Basic Books.

Eastman, J., Siverd, R., and Gaudi, B. S. (2010). Achieving better than 1 minute accuracy in the Heliocentric and Barycentric Julian Dates. *Publications of the Astronomical Society of the Pacific*, 122(894):935.

Ellson, J., Gansner, E., Koutsofios, L., North, S. C., and Woodhull, G. (2001). Graphviz—open source graph drawing tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer.

Elorrieta, F., Eyheramendy, S., Jordán, A., Dékány, I., Catelan, M., Angeloni, R., Alonso-García, J., Contreras-Ramos, R., Gran, F., Hajdu, G., et al. (2016). A machine learned classifier for rr lyrae in the vvv survey. *Astronomy & Astrophysics*, 595:A82.

Emerson, J. P., Irwin, M. J., Lewis, J., Hodgkin, S., Evans, D., Bunclark, P., McMahon, R., Hambly, N. C., Mann, R. G., Bond, I., et al. (2004). Vista data flow system: overview. In *Optimizing scientific return for astronomy through information technologies*, volume 5493, pages 401–411. International Society for Optics and Photonics.

Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R. (1996). *Advances in knowledge discovery and data mining*. the MIT Press.

Feigelson, E. D. (2012). Astrostatistics in a nutshell.

Feigenbaum, A. (1961). *Total quality control*. McGraw Hill.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188.

Forcier, J., Bissex, P., and Chun, W. J. (2008). *Python web development with Django*. Addison-Wesley Professional.

Fowler, M. (2006). CodeSmell.

Fowler, M. and Beck, K. (1999). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

Fox, P. (2011). The rise of informatics as a research domain. In *Proceedings of WIRADA Science Symposium, Melbourne, Australia*, volume 15, pages 125–131.

Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning*, volume 1. Springer series in statistics New York, NY, USA:.

George, D. and Huerta, E. (2018). Deep learning for real-time gravitational wave detection and parameter estimation: Results with advanced ligo data. *Physics Letters B*, 778:64–70.

Gonzalez, O., Rejkuba, M., Zoccali, M., Valenti, E., and Minniti, D. (2011). Reddening and metallicity maps of the milky way bulge from vvv and 2mass-i. the method and minor axis maps. *Astronomy & Astrophysics*, 534:A3.

Gonzalez, O., Rejkuba, M., Zoccali, M., Valenti, E., Minniti, D., Schultheis, M., Tobar, R., and Chen, B. (2012). Reddening and metallicity maps of the milky way bulge from vvv and 2mass-ii. the complete high resolution extinction map and implications for galactic bulge studies. *Astronomy & Astrophysics*, 543:A13.

González-Fernández, C., Hodgkin, S. T., Irwin, M. J., González-Solares, E., Koposov, S. E., Lewis, J. R., Emerson, J. P., Hewett, P. C., Yoldaş, A. K., and Riello, M. (2017). The vista zyjhks photometric system: calibration from 2mass. *Monthly Notices of the Royal Astronomical Society*, 474(4):5459–5478.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Gorelick, M. and Ozsvald, I. (2014). *High Performance Python: Practical Performant Programming for Humans*. .º'Reilly Media, Inc.".

Gran, F., Minniti, D., Saito, R. K., Navarrete, C., Dékány, I., McDonald, I., Ramos, R. C., and Catelan, M. (2015). Bulge RR Lyrae stars in the VVV tile $b201$. *Astronomy & Astrophysics*, 575:A114. arXiv: 1501.00947.

Gran, F., Minniti, D., Saito, R. K., Zoccali, M., Gonzalez, O. A., Navarrete, C., Catelan, M., Ramos, R. C., Elorrieta, F., Eyheramendy, S., and Jordán, A. (2016). Mapping the outer bulge with RRab stars from the VVV Survey. *arXiv:1604.01336 [astro-ph]*. arXiv: 1604.01336.

Gray, J., Szalay, A., Budavári, T., Thakar, A. R., Nieto-Santisteban, M. A., and Lupton, R. (2006). Cross-Matching Multiple Spatial Observations and Dealing with Missing Data. Technical Report MSR-TR-2006-175, Microsoft Research.

Gregg, B. (2013). *Systems Performance: Enterprise and the Cloud*. Pearson Education.

Grieco, G. (2018). *Detección de Vulnerabilidades y Análisis de Fallos con Técnicas de Aprendizaje Automatizado*. PhD thesis, Universidad Nacional de Rosario.

Grossman, R. L., Kamath, C., Kegelmeyer, P., Kumar, V., and Namburu, R. (2013). *Data mining for scientific and engineering applications*, volume 2. Springer Science & Business Media.

Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422.

Hadjiyska, E., Hughes, G., Lubin, P., Taylor, S., Hartong-Redden, R., and Zierten, J. (2013). The transient optical sky survey data pipeline. *New Astronomy*, 19:99–108.

Hambly, N., Collins, R., Cross, N., Mann, R., Read, M., Sutorius, E., Bond, I., Bryant, J., Emerson, J., Lawrence, A., et al. (2008). The wfcam science archive. *Monthly Notices of the Royal Astronomical Society*, 384(2):637–662.

Han, E., Wang, S. X., Wright, J. T., Feng, Y. K., Zhao, M., Fakhouri, O., Brown, J. I., and Hancock, C. (2014). Exoplanet Orbit Database. II. Updates to Exoplanets.org. *PASP*, 126:827.

Hanisch, R. J., Farris, A., Greisen, E. W., Pence, W. D., Schlesinger, B. M., Teuben, P. J., Thompson, R. W., and Warnock, A. (2001). Definition of the flexible image transport system (fits). *Astronomy & Astrophysics*, 376(1):359–380.

Harrison Jr, D. and Rubinfeld, D. L. (1978). Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102.

He, H. and Garcia, E. A. (2008). Learning from imbalanced data. *IEEE Transactions on Knowledge & Data Engineering*, 21(9):1263–1284.

Hey, T., Tansley, S., Tolle, K. M., and others (2009). *The fourth paradigm: data-intensive scientific discovery*, volume 1. Microsoft research Redmond, WA.

Hughes, A. L. H., Jain, K., Kholikov, S., and the NISP Solar Interior Group (2016). Gong classicmerge: Pipeline and product. *ArXiv e-prints*.

ISO. (2005). *Document Management: Electronic Document File Format for Long-term Preservation*. ISO.

Ivezic, Z., et al., and for the LSST Collaboration (2008). LSST: from Science Drivers to Reference Design and Anticipated Data Products. *ArXiv e-prints*.

Japkowicz, N. (2000). The class imbalance problem: Significance and strategies. In *Proc. of the Int'l Conf. on Artificial Intelligence*.

Jazayeri, M. (2007). Some trends in web application development. In *Future of Software Engineering, 2007. FOSE'07*, pages 199–213. IEEE.

Jeffries, R. and Melnik, G. (2007). Guest editors' introduction: Tdd–the art of fearless programming. *IEEE Software*, 24(3):24–30.

Karpatne, A., Atluri, G., Faghmous, J. H., Steinbach, M., Banerjee, A., Ganguly, A., Shekhar, S., Samatova, N., and Kumar, V. (2017). Theory-guided data science: A new paradigm for scientific discovery from data. *IEEE Transactions on Knowledge and Data Engineering*, 29(10):2318–2331.

Kern, R. (2007). A simple file format for numpy arrays.

Kim, D.-W. and Bailer-Jones, C. A. L. (2016). A package for the automated classification of periodic variable stars. *Astronomy & Astrophysics*, 587:A18.

Kim, D.-W., Protopapas, P., Alcock, C., Byun, Y.-I., and Bianco, F. B. (2009). De-Trending Time Series for Astronomical Variability Surveys. In *Astronomical Data Analysis Software and Systems XVIII*, volume 411, page 247.

Kim, D.-W., Protopapas, P., Bailer-Jones, C. A., Byun, Y.-I., Chang, S.-W., Marquette, J.-B., and Shin, M.-S. (2014). The epoch project-i. periodic variable stars in the eros-2 lmc database. *Astronomy & Astrophysics*, 566:A43.

Kim, D.-W., Protopapas, P., Byun, Y.-I., Alcock, C., Khardon, R., and Trichas, M. (2011). Quasi-stellar object selection algorithm using time variability and machine learning: Selection of 1620 quasi-stellar object candidates from macho large magellanic cloud database. *The Astrophysical Journal*, 735(2):68.

Kim, E. J. and Brunner, R. J. (2016). Star-galaxy classification using deep convolutional neural networks. *Monthly Notices of the Royal Astronomical Society*, page stw2672.

Klaus, T. C., McCauliff, S., Cote, M. T., Girouard, F. R., Wohler, B., Allen, C., Middour, C., Caldwell, D. A., and Jenkins, J. M. (2010). Kepler science operations center pipeline framework. In *Software and Cyberinfrastructure for Astronomy*, volume 7740, page 774017. International Society for Optics and Photonics.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J. B., Grout, J., Corlay, S., et al. (2016). Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90.

Klypin, A. and Shandarin, S. (1983). Three-dimensional numerical model of the formation of large-scale structure in the universe. *Monthly Notices of the Royal Astronomical Society*, 204(3):891–907.

Krasner, G. E., Pope, S. T., and others (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49.

Kubánek, P. and Jelínek, M. (2010). RTS2 - The Remote Telescope System. *Advances in Astronomy, Advances in Astronomy*, 2010, 2010:e902484.

Kuhlen, M., Vogelsberger, M., and Angulo, R. (2012). Numerical simulations of the dark universe: State of the art and the next decade. *Physics of the Dark Universe*, 1(1-2):50–93.

Kulkarni, S. (2013a). The intermediate Palomar Transient Factory (iPTF) begins. *The Astronomer's Telegram*, 4807:1.

Kulkarni, S. (2013b). The intermediate palomar transient factory (iptf) begins. *The Astronomer's Telegram*, 4807:1.

Lamport, L. (1994). *Latex*. Addison-Wesley.

Landsman, W. (1995). The IDL Astronomy User's Library. In *Astronomical Data Analysis Software and Systems IV*, volume 77, page 437.

Law, N. M., Kulkarni, S. R., Dekany, R. G., Ofek, E. O., Quimby, R. M., Nugent, P. E., Surace, J., Grillmair, C. C., Bloom, J. S., Kasliwal, M. M., et al. (2009). The palomar transient factory: system overview, performance, and first results. *Publications of the Astronomical Society of the Pacific*, 121(886):1395.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lomb, N. R. (1976). Least-squares frequency analysis of unequally spaced data. *Astrophysics and Space Science*, 39(2):447–462.

Magnier, E., Kaiser, N., and Chambers, K. (2006). The pan-starrs ps1 image processing pipeline. In *The Advanced Maui Optical and Space Surveillance Technologies Conference*, volume 1, page 50.

Marx, R. and Reyes, R. d. l. (2015). A Prototype for the Cherenkov Telescope Array Pipelines Framework: Modular Efficiency Simple System (MESS). *arXiv:1509.01428 [astro-ph]*. arXiv: 1509.01428.

Masci, F. J., Laher, R. R., Rebbapragada, U. D., Doran, G. B., Miller, A. A., Bellm, E., Kasliwal, M., Ofek, E. O., Surace, J., Shupe, D. L., et al. (2016). The ipac image subtraction and discovery pipeline for the intermediate palomar transient factory. *Publications of the Astronomical Society of the Pacific*, 129(971):014002.

McKinney, W. (2011). pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, pages 1–9.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (2013). *Machine learning: An artificial intelligence approach*. Springer Science & Business Media.

Miller, J. C. and Maloney, C. J. (1963). Systematic mistake analysis of digital computer programs. *Communications of the ACM*, 6(2):58–63.

Minniti, D. (2018). Mapping the Milky Way in the Near-IR: The Future of the VVV Survey. In *The Vatican Observatory, Castel Gandolfo: 80th Anniversary Celebration*, Astrophysics and Space Science Proceedings, pages 63–71. Springer, Cham.

Minniti, D., Lucas, P., Emerson, J., Saito, R., Hempel, M., Pietrukowicz, P., Ahumada, A., Alonso, M., Alonso-Garcia, J., Arias, J. I., et al. (2010). Vista variables in the via lactea (vvv): The public eso near-ir variability survey of the milky way. *New Astronomy*, 15(5):433–443.

Mitchell, T. M. et al. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877.

Mohr, J. J., Adams, D., Barkhouse, W., Beldica, C., Bertin, E., Cai, Y. D., da Costa, L. A. N., Darnell, J. A., Daues, G. E., Jarvis, M., et al. (2008). The dark energy survey data management system. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 70160L–70160L. International Society for Optics and Photonics.

Ng, A. (2013). *Machine Learning and AI via Brain simulations*. Stanford University.

Nishiyama, S., Tamura, M., Hatano, H., Kato, D., Tanabé, T., Sugitani, K., and Nagata, T. (2009). Interstellar extinction law toward the galactic center iii: J, h, ks bands in the 2mass and the mko systems, and 3.6, 4.5, 5.8, 8.0 μm in the spitzer/irac system. *The Astrophysical Journal*, 696(2):1407.

Nun, I., Protopapas, P., Sim, B., Zhu, M., Dave, R., Castro, N., and Pichara, K. (2015). Fats: Feature analysis for time series. *arXiv preprint arXiv:1506.00010*.

Ochsenbein, F., Bauer, P., and Marcout, J. (2000). The vizier database of astronomical catalogues. *Astronomy and Astrophysics Supplement Series*, 143(1):23–32.

Oliphant, T. E. (2007). Scipy: Open source scientific tools for python. *Computing in Science and Engineering*, 9:10–20.

Ordovás-Pascual, I. and Almeida, J. S. (2014). A fast version of the k-means classification algorithm for astronomical applications. *Astronomy & Astrophysics*, 565:A53.

Owens, M. and Allen, G. (2010). *SQLite*. Springer.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830.

Pichara, K., Protopapas, P., Kim, D.-W., Marquette, J.-B., and Tisserand, P. (2012). An improved quasar detection method in eros-2 and macho lmc data sets. *Monthly Notices of the Royal Astronomical Society*, 427(2):1284–1297.

Pierro, M. D. (2011). web2py for Scientific Applications. *Computing in Science & Engineering*, 13(2):64–69.

Press, W. H. (2007). *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

Price, R. (2000). Iso/iec 15445: 2000 (e). hypertext markup language.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.

Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

Renzi, V., Vrech, R., Ferreiro, D., García Lambas, D., Solinas, M., Muriel, H., Viramonte, J., Sarazin, M., and Recabarren, P. (2009). Caracterización astronómica del sitio cordón macón en la provincia de salta. *Boletin de la Asociacion Argentina de Astronomia La Plata Argentina*, 52:285–288.

Richards, J. W., Starr, D. L., Butler, N. R., Bloom, J. S., Brewer, J. M., Arien Crellin-Quick, Higgins, J., Kennedy, R., and Rischard, M. (2011a). On Machine-learned Classification of Variable Stars with Sparse and Noisy Time-series Data. *The Astrophysical Journal*, 733(1):10.

Richards, J. W., Starr, D. L., Butler, N. R., Bloom, J. S., Brewer, J. M., Crellin-Quick, A., Higgins, J., Kennedy, R., and Rischard, M. (2011b). On machine-learned classification of variable stars with sparse and noisy time-series data. *The Astrophysical Journal*, 733(1):10.

Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A. M., Kerzendorf, W. E., et al. (2013a). Astropy: A community python package for astronomy. *Astronomy & Astrophysics*, 558:A33.

Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A. M., Kerzendorf, W. E., and others (2013b). Astropy: A community Python package for astronomy. *Astronomy & Astrophysics*, 558:A33.

Rocklin, M. (2015). Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th Python in Science Conference*, number 130-136 in Python in Science. Citeseer.

Rodriguez, A. C., Kacprzak, T., Lucchi, A., Amara, A., Sgier, R., Fluri, J., Hofmann, T., and Réfrégier, A. (2018). Fast cosmic web simulations with generative adversarial networks. *arXiv preprint arXiv:1801.09070*.

Rose, J., Akella, R., Binegar, S., Choo, T., Heller-Boyer, C., Hester, T., Hyde, P., Perrine, R., Rose, M., and Steuerman, K. (1995). The opus pipeline: a partially object-oriented pipeline system. In *Astronomical Data Analysis Software and Systems IV*, volume 77, page 429.

Roskind, J. (2007). The python profiler. *URL http://docs. python. org/lib/profile. html*.

Rutledge, R. E. (1998). The astronomer's telegram: A web-based short-notice publication system for the professional astronomical community. *Publications of the Astronomical Society of the Pacific*, 110(748):754.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.

Scargle, J. D. (1982). Studies in astronomical time series analysis. II-Statistical aspects of spectral analysis of unevenly spaced data. *The Astrophysical Journal*, 263:835–853.

Schneider, S. (2000). Statistical Profiling: An Analysis.

Seabold, S. and Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with python. In *Proceedings of the 9th Python in Science Conference*, volume 57, page 61. SciPy society Austin.

Shapley, H. (1936). Summary of investigations of variable stars. *Proceedings of the National Academy of Sciences*, 22(1):8–14.

Shia, W.-C., Huang, Y.-L., Wu, H.-K., and Chen, D.-R. (2017). Using flow characteristics in three-dimensional power doppler ultrasound imaging to predict complete responses in patients undergoing neoadjuvant chemotherapy. *Journal of Ultrasound in Medicine*, 36(5):887–900.

Skrutskie, M., Cutri, R., Stiening, R., Weinberg, M., Schneider, S., Carpenter, J., Beichman, C., Capps, R., Chester, T., Elias, J., et al. (2006a). The two micron all sky survey (2mass). *The Astronomical Journal*, 131(2):1163.

Skrutskie, M. F., Cutri, R. M., Stiening, R., Weinberg, M. D., Schneider, S., Carpenter, J. M., Beichman, C., R. Capps, Chester, T., Elias, J., Huchra, J., Liebert, J., Lonsdale, C., Monet, D. G., Price, S., Seitzer, P., T. Jarrett, Kirkpatrick, J. D., Gizis, J. E., Howard, E., Evans, T., Fowler, J., Fullmer, L., Hurt, R., Light, R., Kopan, E. L., Marsh, K. A., McCallon, H. L., Tam, R., Dyk, S. V., and Wheelock, S. (2006b). The Two Micron All Sky Survey (2mass). *The Astronomical Journal*, 131(2):1163.

Soszynski, I., Dziembowski, W. A., Udalski, A., Poleski, R., Szymanski, M. K., Kubiak, M., Pietrzynski, G., Wyrzykowski, L., Ulaczyk, K., Kozlowski, S., and Pietrukowicz, P. (2011). The Optical Gravitational Lensing Experiment. The OGLE-III Catalog of Variable Stars. XI. RR Lyrae Stars in the Galactic Bulge. *arXiv:1105.6126 [astro-ph]*. arXiv: 1105.6126.

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., and Murthy, R. (2009). Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629.

Tollerud, E. (2016). Jwst dadf (data analysis development forum) and photutils. psf. In *Python in Astronomy 2016*.

Tremblin, P., Schneider, N., Minier, V., Durand, G. A., and Urban, J. (2012). Worldwide site comparison for submillimetre astronomy. *Astronomy & Astrophysics*, 548:A65.

Tucker, D. L., Kent, S., Richmond, M. W., Annis, J., Smith, J. A., Allam, S. S., Rodgers, C. T., Stute, J. L., Adelman-McCarthy, J. K., Brinkmann, J., Doi, M., Finkbeiner, D., Fukugita, M., Goldston, J., Greenway, B., Gunn, J. E., Hendry, J. S., Hogg, D. W., Ichikawa, S.-I., Ivezic, Z., Knapp, G. R., Lampeitl, H., Lee, B. C., Lin, H., McKay, T. A., Merrelli, A., Munn, J. A., Neilsen, J., Newberg, H. J., Richards, G. T., Schlegel, D. J., Stoughton, C., Uomoto, A., and Yanny, B. (2006). The Sloan Digital Sky Survey Monitor Telescope Pipeline. *Astronomische Nachrichten*, 327(9):821–843. arXiv: astro-ph/0608575.

Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., and Poshyvanyk, D. (2015). When and Why Your Code Starts to Smell Bad. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, pages 403–414. IEEE.

Udalski, A. (2004). The optical gravitational lensing experiment. Real time data analysis systems in the OGLE-III survey. *arXiv preprint astro-ph/0401123*.

Udalski, A., Szymański, M., and Szymański, G. (2015). Ogle-iv: fourth phase of the optical gravitational lensing experiment. *arXiv preprint arXiv:1504.05966*.

Van Der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.

VanderPlas, J., Fouesneau, M., and Taylor, J. (2014). AstroML: Machine learning and data mining in astronomy. *Astrophysics Source Code Library*, page ascl:1407.018.

VanderPlas, J. T. (2018). Understanding the lomb–scargle periodogram. *The Astrophysical Journal Supplement Series*, 236(1):16.

Violini, M. L. (2014). *Selección de características*. PhD thesis, Facultad de Informática.

Virgo, L., Collaboration, S., Abbott, B., et al. (2017). Gw170817: Observation of gravitational waves from a binary neutron star inspiral. *Phys. Rev. Lett*, 119.

Walt, S. v. d., Colbert, S. C., and Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*, 13(2):22–30.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83.

Willke, B., Collaboration, L. S., et al. (2018). Observation of gravitational waves from a binary black hole merger–dawn of a new astronomy. *Symmetry: Culture and Science*, 29(2):257–264.

Wozniak, P. (2000). Difference image analysis of the ogle-ii bulge data. i. the method. *arXiv preprint astro-ph/0012143*.

Wulf, W. and Shaw, M. (1973). Global variable considered harmful. *ACM Sigplan notices*, 8(2):28–34.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95.

Zhang, Z., Barbary, K., Nothaft, F. A., Sparks, E. R., Zahn, O., Franklin, M. J., Patterson, D. A., and Perlmutter, S. (2016). Kira: Processing astronomy imagery using big data technology. *IEEE Transactions on Big Data*.