Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Clustering y visualización de curvas de luz de estrellas periódicas

Informe Preliminar

Equipo: 8A
Autores: Nicolás Canales V.
Matías Vergara S.
Profesor: Pablo Estevez V.
Auxiliar: Ignacio Reyes Jainaga
Tutor: Javier Molina F.
Fecha de entrega: 30 de octubre de 2021
Santiago de Chile

Índice de Contenidos

1. Introducción **1**

2. Datos a utilizar **2**

3. Preprocesamiento de datos **2**
3.1. Selección de características iniciales . . . 3
3.2. Filtrado de entradas . . . 3
3.3. *Data Augmentation* . . . 3
3.4. Extracción de características mediante Turbo-FATS . . . 4
3.5. Estandarización y limpieza de características avanzadas . . . 5

4. Algoritmos a utilizar **5**
4.1. UMAP . . . 6
4.1.1. Justificación . . . 6
4.1.2. Parámetros y criterio de detención . . . 6
4.2. Autoencoders . . . 7
4.2.1. Parámetros y criterio de detención . . . 7
4.3. DAGMM . . . 7
4.3.1. Parámetros y criterio de detención . . . 8

5. Software y recursos computacionales **8**
5.1. Entorno de programación y librerías . . . 8
5.2. Estimación de recursos computacionales . . . 9
5.3. Estimación de simulaciones a realizar . . . 9

6. Salidas deseadas y función objetivo **10**

7. Resultados esperados **10**
7.1. Resultados y medidas de desempeño . . . 10
7.2. Formato de los resultados . . . 11

8. Resultados preliminares **11**
8.1. UMAP . . . 11
8.2. *Autoencoders* . . . 13

9. Carta Gantt **15**

Índice de Figuras

1. Resultado UMAP, usando todas las *features*, todos los datos . . . 12
2. Resultado UMAP, usando el subconjunto $C1$ de *features*, todos los datos . . . 12
3. Resultado Autoencoder, usando todas las *features*, todos los datos . . . 14
4. Resultado Autoencoder, usando el subconjunto $C1$ de *features*, datos con *data-augmentation* y *subsampling* . . . 14

Índice de Tablas

1. Distribución de clases en filtered_alerts.csv 4
2. Arquitectura autoencoder fase intermedia 13

1. Introducción

Entender cómo funciona el espacio y sus objetos ha sido un desafío que ha acompañado al ser humano desde sus orígenes, dando lugar a lo que es la ciencia de la astronomía. Estudios en esta área han permitido comprender distintos fenómenos, sin embargo, existen aún muchos aspectos sin resolver, y su estudio requiere de recursos los cuales son limitados. En este sentido, el desarrollo de herramientas computacionales que permitan enfocar dichos recursos resulta fundamental.

Una de estas herramientas es ALeRCE (Automatic Learning for the Rapid Classification of Events)[1], un *broker* de alertas astronómicas que recibe y procesa observaciones provenientes del *survey* astronómico ZTF *(Zwicky Transient Facility)*[2]. Cada observación corresponde a una variación en el brillo de un objeto astrofísico, para lo cual interesa reconocer el tipo del objeto implicado a fin de levantar dicha alerta a los centros de estudio especializados. De esta manera, ALeRCE actúa como un intermediario entre ambas partes, generando una clasificación de la observación según si corresponde a una fuente de tipo transiente/estocástico/periódico, y luego según las subclases de cada tipo. Esta clasificación juega un rol esencial en el enfoque de recursos, pues permite enviar las alertas más prometedoras a estudio por los centros especializados, los cuales a su vez retroalimentan al *survey* para el rastreo de aquellos objetos de interés.

El proyecto a realizar corresponde a un trabajo semestral del curso Inteligencia Computacional, y se enmarca en dicho contexto. Su objetivo principal comprende la **generación y visualización de grupos de similitud o *clusters* de curvas de luz de objetos clasificados por ALeRCE como periódicos**, para lo cual se trabajará sobre un conjunto de observaciones provenientes del ZTF, extrayendo aquellas clasificadas como periódicas y calculándose para cada una las características inherentes a series temporales. Dichas características serán posteriormente procesadas por algoritmos de reducción de dimensionalidad, a fin de obtener una codificación 2D que pueda ser proyectada en búsqueda de dichos *clusters*. De esta forma, un desafío importante será **escoger características que conserven la estructura global de los datos**.

El desarrollo del proyecto permitirá tanto comprender con mayor detalle los objetos en estudio como la relación entre las distintas clases involucradas. En especial, se busca **estudiar la correspondencia entre los *clusters* resultantes y las clases y subclases presentes en los datos**, así como **caracterizar aquellas curvas de luz que se ubiquen en la periferia.**

El presente informe busca ser una entrega **preliminar** del proyecto, por lo cual no incorpora la totalidad del trabajo a realizar mas sí una parte importante de ello. Su estructura sigue el enfoque propuesto por el equipo docente: inicialmente se presenta el problema y los datos con los que se trabajará, seguido de los preprocesamientos realizados y la justificación de los algoritmos a utilizar, que corresponden a UMAP[3], *autoencoders*[4] y DAGMM[5]. Posteriormente se especifican las salidas deseadas y la función objetivo, así como el entorno en el cual se desarrollará el modelo y los recursos computacionales necesarios. Finalmente se comentaran los resultados esperados y se presentarán los resultados preliminares, cerrando con una planificación de las etapas próximas a la fecha de entrega de este documento.

2. Datos a utilizar

El proyecto toma como entrada un conjunto de 110 millones de observaciones astronómicas del ZTF, para las cuales se tienen las siguientes características:

- oid: identificación del objeto. Puede repetirse.
- candid: identificación única de la observación.
- dec, ra: *declination* y *right ascension*. Permiten reconocer la posición del objeto en el cielo.
- magpsf_corr: magnitud del brillo observado, obtenido mediante fotometría PSF-fit.
- sigmapsf_corr: desviación estándar del brillo observado, obtenido mediante PSF-fit photometry. [1].
- sigmapsf_corr_ext: desviación estándar del brillo observado, obtenido mediante PSF-fit photometry.
- fid: identificador de la banda de observación. El valor 1 indica que se trata del filtro (banda) g, 2 para el filtro r y 3 para ambos.
- mjd: *Modified Julian Date*. Fecha de la observación en día juliano modificado.

Dichos datos fueron facilitados por el equipo docente, y se encuentran disponibles para su descarga a través de Google Drive[6], bajo el nombre alerts.csv.

Además, se requiere también de un *dataset* con 123.000 clasificaciones realizadas por ALeRCE, para las cuales se cuenta con diversos atributos - de los cuales interesan el oid (identificador de objeto) y classALeRCE (clase asociada) -. Estos datos también fueron facilitados por el equipo docente, y se encuentran disponibles a través de GitHub bajo el nombre de labels_set.csv, en un repositorio del monitor[7].

Es importante mencionar que, si bien esta es la entrada inicial del proyecto, su manipulación puede resultar bastante compleja debido al gran volumen de datos que representan. Con esta consideración, los autores han enfocado el trabajo en distintas fases o etapas para cada una de las cuales se pone a disposición el *dataset* de entrada, a fin de que el lector pueda reproducir cada una por separado - sin tener que pasar por los procesamientos más extensos o computacionalmente complejos -. Dichos datos se encuentran disponibles a través de la carpeta Drive del proyecto[8], y una explicación detallada de cómo utilizarlos se incluye a través del *Readme* en el repositorio GitHub del proyecto[9].

3. Preprocesamiento de datos

Antes de aplicar los distintos algoritmos de *clustering*, resulta necesario realizar un preprocesamiento de los datos. Esto se realiza en cinco etapas: una primera de selección de las características

[1] Esta característica fue incorporada en una etapa más avanzada el proyecto (no formaba parte de los datos iniciales) tras discusiones con el equipo docente sobre problemas al extraer características con magpsf_corr.

iniciales de interés, seguida de una segunda etapa de filtrado de las entradas y una tercera de *data augmentation* sobre las clases infrarrepresentadas, una cuarta etapa de obtención de las características asociadas a series de tiempo, y una etapa final de estandarización y limpieza de dichas características.

3.1. Selección de características iniciales

En primer lugar resulta necesario escoger las características iniciales a mantener (listadas en la sección 2: Datos a utilizar). Para ello se considera la entrada requerida por las etapas siguientes, en especial para lo que es el cálculo de las características avanzadas. Se decide mantener las columnas de oid, magpsf_corr, sigmapsf_corr_ext y mjd. Estas representarán el índice, la magnitud, el error y el tiempo de los datos sobre los cuales se extraerán las características más avanzadas.

En este punto es importante mencionar que se optó por trabajar con sigmapsf_corr_ext en lugar de sigmapsf_corr por recomendación del equipo docente, tras iterar extensamente la etapa de extracción de características sin lograr obtener resultados.

3.2. Filtrado de entradas

Posterior a la selección de las características de interés se realiza el cruce de los datos en alerts.csv con aquellos en labels_set.csv, en búsqueda de mantener solamente aquellas filas del primero que correspondiesen a un objeto de tipo periódico en el segundo. Para ello en primer lugar resultó necesario identificar cuáles clases corresponden a objetos periódicos, para lo cual se estudió la publicación de ALeRCE: *The Light Curve Classifier*[10]. Se determinó que los objetos periódicos eran aquellos con las etiquetas CEP (*Cepheid*), E (*Eclipsing binary*), DSCT (delta *Scuti*), LPV (*Long Period Variable*), RRL (*RR Lyrae*) y Periodic-Other (otras curvas periódicas). Con esta información, se procedió a filtrar el archivo alerts.csv mediante un *match* del atributo oid con su par en labels_set.csv, manteniendo solamente aquellas filas que tuvieran un classALeRCE igual a alguna de las etiquetas ya mencionadas.

Como resultado de esta etapa se generó un nuevo archivo de alrededor de 9 millones de filas, el cual se encuentra disponible en el Drive del proyecto bajo el nombre de filtered_alerts.csv. Su peso no supera los 900MB, por lo cual es mucho más abordable localmente para el lector.

3.3. *Data Augmentation*

Con los datos ya filtrados, resultaba interesante estudiar cómo se distribuían las distintas etiquetas. Se realizó un análisis exploratorio de los datos a fin de resolver esta interrogante, estudiando cuantos oid diferentes había para cada clase. Los resultados fueron los siguientes (Tabla 1):

Tabla 1: Distribución de clases en filtered_alerts.csv

Clase	Entradas
E	37900
RRL	32464
LPV	14045
Periodic-Other	1256
DSCT	618
CEP	732

De inmediato se observó un desbalance enorme de clases, con tres de ellas (Periodic-Other, DSCT, CEP) representando solamente el 2.99 % de los oid disponibles. Esto representaría un problema grave, pues podría introducir sesgo en los algoritmos hacia las clases más representadas.

Para lidiar con esta dificultad, se decidió tomar dos enfoques simultáneamente: el primero, realizar *data augmentation* sobre las clases infrarrepresentadas para aumentar su cantidad. El segundo, realizar un *subsampling* de las clases sobrerrepresentadas para disminuir su cantidad, y así reducir el desbalance tanto como fuese factible.

En lo que respecta al *data augmentation*, se optó por generar entradas sintéticas en base a la integración de ruido sobre las curvas de luz reales, una por cada una. De esta forma, se duplicó la cantidad de entradas para cada etiqueta Periodic-Other, DSCT y CEP. El *notebook* con el proceso y su documentación se encuentra disponible en el repositorio del proyecto bajo el nombre data_augmentation.ipynb, mientras que los datos sintéticos se presentan bajo el nombre de sintetic_class.csv en la carpeta de drive, reemplazando class por el nombre de la etiqueta.

Por otro lado, para el *subsampling* de las clases sobrerrepresentadas se optó por un muestreo aleatorio, manteniendo alrededor de 3000 oid distintos por cada clase. El archivo resultante lleva el nombre de reduced_data.csv.

Es importante mencionar que, en todos los experimentos a realizar, se llevarán a cabo simulaciones con y sin los datos sintéticos/subsampleados, a fin de evaluar también la robustez de los resultados ante el ya mencionado desbalance.

3.4. Extracción de características mediante Turbo-FATS

Una vez que se tienen los datos originales filtrados a curvas periódicas con sus *features* de interés y lo mismo para los datos sintéticos, sigue la extracción de *features* propias de las series de tiempo. Para ello se trabaja con la librería Turbo-FATS[11], que recibe atributos tales como la magnitud, el error, el tiempo y la banda de un conjunto de observaciones de un mismo objeto, y calcula más de 100 características de interés para la curva de luz resultante. Al respecto, cabe mencionar que:

- Inicialmente, se indicó trabajar con la librería FATS, la cual estaría deprecada y con la cual no se consiguió obtener resultados pese a realizar las adaptaciones necesarias para Python 3.
- Una vez se cambia la librería a Turbo-FATS, aún resulta imposible obtener características debido a problemas en los datos. Esto motivó la inclusión del atributo sigma_psf_ext en reemplazo de

sigma_psf por parte del equipo docente.

- Finalmente, se logró obtener las características tras un procesamiento de más de 40 horas. Es importante mencionar que este procesamiento debe realizarse en un bloque de *try-catch* y objeto por objeto, pues de lo contrario el programa fallará.
- Otra condición importante para la extracción es que las fechas (el atributo mjd) deben ser desplazadas a iniciar en 0 (restando la menor de las fechas a todas las entradas) pues, de lo contrario, se perderán muchas filas por *Overflow Error*. Es importante destacar que para la extracción de características importa el tiempo transcurrido entre observaciones y no el tiempo calendario, por lo cual lo anterior no tiene impacto alguno sobre los resultados.

Entre las características calculadas se encuentran, por ejemplo, el periodo de la curva de luz en cada banda, así como su amplitud, frecuencias armónicas, índices de Welsh-Stetson, entre otros. Un listado completo de las *features* obtenidas para cada objeto y un análisis de su distribución y correlaciones se encuentra disponible en el repositorio del proyecto bajo el nombre profile_data_augmented.html.

3.5. Estandarización y limpieza de características avanzadas

Finalmente, una vez se tienen las 100 características avanzadas para cada curva de luz - resultado de la etapa anterior -, se debe realizar un proceso de **selección** y **estandarización** de las mismas:

- La selección se debe a que existen muchas características que tendrán principalmente valores nulos, especialmente aquellas que requieren observaciones tanto en la banda g y r.
- La estandarización, por otro lado, resulta necesaria puesto que las escalas de cada *feature* son muy diferentes (el power rate, por ejemplo, es una tasa entre 0 y 1, mientras que amplitude g alcanza valores sobre 50).

Para la estandarización se utilizan librerías destinadas para ello (StandardScaler y CuantileTransform de TensorFlow), lo cual no presenta mayor dificultad y se lleva a cabo como un paso previo a la aplicación de cada algoritmo (no como parte del dataset) para poder así evaluar los resultados obtenidos tanto con escalamiento por desviación estándar como por cuantiles. Para la selección de características, sin embargo, se estudia el ya mencionado *perfilador* de los datos (obtenido mediante Pandas Profiler en búsqueda de aquellos atributos con un alto porcentaje de valores nulos. Se decide descartar las columnas Con_g, Con_r.

Lo anterior termina la etapa de preprocesamiento de los datos. Un último comentario importante es notar que, si bien en este punto se tienen 98 *features* por cada curva de luz, al momento de llevar a cabo los experimentos podrá ser conveniente seleccionar un subconjunto de ellas. Esto se presentará como parte de las simulaciones, pues se busca probar con distintos subconjuntos y para ello se requiere contar con todas las columnas a priori.

4. Algoritmos a utilizar

Los algoritmos a utilizar son 3: UMAP, *Autoencoders* y *Mezcla de Gaussianas sobre Autoencoders Profundos (DAGMM)*. En esta sección se describirá brevemente cada uno y se comentará sobre porqué se considera que podrían resolver el problema.

4.1. UMAP

4.1.1. Justificación

UMAP son las siglas de *Uniform Manifold Approximation and Projection for Dimension Reduction*[3]. Se trata de un algoritmo de reducción de dimensionalidad que busca preservar la estructura global de los datos, sin necesidad de imponer restricciones sobre las dimensiones de entrada. Esto ofrece una ventaja importante para el problema a tratar, pues a priori no sabemos qué dimensionalidad tendrán los datos de entrada (no sabemos cuántas características usaremos como subconjunto de interés, y nos interesará probar con grupos que potencialmente serán de distinto tamaño).

UMAP guarda un potencial importante al asegurar una equivalencia teórica entre los espacios topológicos de los datos en alta y baja dimensión, es decir: aquellos datos que estén cerca en la 100-dimensión, deberían seguir haciéndolo en la 2-dimensión. Esto permitiría cumplir con el desafío más importante para lograr buenos *clusters*: respetar la estructura global de los datos.

El algoritmo incluye además diversos parámetros que dan sentido al gran número de simulaciones que se desean llevar a cabo. En particular, incorpora el concepto de *Nearest Neighbours* o vecinos más cercanos (NN), lo cual tendrá una influencia importante en el tamaño de los *clusters* generados, y el concepto de distancia mínima, que influirá en qué tan compactos serán dichos *clusters*. En particular, es importante mencionar que el parámetro relacionado a los NN viene a ser el equivalente de la *perplexity* de otros algoritmos tales como t-SNE (T-distributed Stochastic Neighbor Embedding)[**?**] [2].
Se pretende que UMAP entregue una visualización preliminar de los datos, la cual posteriormente se comparará con la salida de los otros modelos. Esto servirá además como guía para la selección de *features* mediante experimentos a un costo bajo, pues UMAP ha resultado ser - hasta el momento - el algoritmo más rápido.

4.1.2. Parámetros y criterio de detención

Como ya se adelantaba, los parámetros a definir y/o ajustar para UMAP serán en primer lugar los de n_neighbors y min_dist. También puede resultar útil ajustar el parámetro metric, que corresponde a la métrica a utilizar para medir la distancia entre objetos. Otros parámetros tales como n_components no interesan pues determinan la dimensión de salida, que para el caso del proyecto está fijada a 2.

En cuanto al criterio de detención a utilizar, UMAP se detiene una vez que alcanza la dimensión solicitada por n_components (2 por defecto). En consecuencia, no hay parámetros a ajustar en este sentido.

[2] En la presentación intermedia se aconsejó estudiar el parámetro de perplexity, sin embargo, la documentación de UMAP no incorpora tal parámetro. Investigando se llegó al paper citado, en donde se comenta que el equivalente de la *perplexity* en UMAP son los *n_neighbours*.

4.2. Autoencoders

Se buscará también generar la reducción de dimensionalidad a través de *Autoencoders*. Tales modelos corresponden a redes neuronales que se entrenan para replicar la entrada en la salida, a través de dos fases: una primera de codificación, donde la entrada es alimentada a una red neuronal con un número variable de capas que termina en una de menor tamaño, denominada *cuello de botella*, la cual da origen a un "código" de la entrada: una representación en dimensionalidad reducida de la misma, la cual debe capturar las características más importantes. Posteriormente le sigue la fase de decodificación, donde se busca reconstruir la entrada a partir del código, lo cual a su vez evidencia que dicho código es suficiente para representar y reconstruir la entrada.

Los autoencoders tienen algunas ventajas y desventajas a considerar. En particular es importante tener en cuenta que la representación a través del código siempre incluirá un cierto error de reconstrucción, el cual difícilmente será despreciable. Sin embargo, esto no es del todo negativo, pues podría estar perdiendo la capacidad de reconstruir aquellas *features* que no eran de verdadero interés.

Se pretende que dicho modelo permita abarcar el problema al utilizar una capa cuello de botella de dos dimensiones, a fin de obtener una codificación de la entrada que pueda ser visualizada en el plano en búsqueda de *clusters* y así también comparar con la salida de UMAP. Además, la salida de este modelo podrá ser utilizada como entrada para el algoritmo siguiente, que es una refinación del mismo.

4.2.1. Parámetros y criterio de detención

En cuanto a los parámetros a determinar para este modelo, los más importantes guardan relación con la cantidad de capas a utilizar, el tipo de capa, la cantidad de neuronas en cada una y la función de activación asociada, tanto para *encoder* como para *decoder*. Para ello se deberán realizar distintos experimentos, teniendo en cuenta también la dimensión de entrada (cuantas *features* le entregaremos al modelo). Como enfoque inicial se trabajó con 5 capas densas en el *encoder* y 3 en el *decoder* (cada una con 90 a 160 neuronas), sin embargo, se cree que existen otras configuraciones que podrían esconder un potencial mucho mayor y que se buscarán tanto mediante prueba y error como investigando aplicaciones en problemas similares.

En cuanto a los criterios de detención, para este modelo se consideró un *early stopping* con una *patience* de 5 iteraciones sin mejoría sobre el *loss*.

4.3. DAGMM

DAGMM corresponde a las siglas de *Deep Autoencoder Gaussian Mixture Model[5]*. Corresponde a la aplicación de una mezcla de Gaussianas sobre la codificación de un autoencoder profundo, como el que se obtendrá del algoritmo anterior. Se trata de un **modelo probabilístico** que asume que todos los datos provienen de la mezcla de un número finito de distribuciones de Gauss con parámetros desconocidos, las cuales se descubren a través de un entrenamiento sobre los datos a través del algoritmo de expectación-maximización.

La principal razón por la que se busca aplicar este modelo es porque se considera que, a través de la aplicación de mezclas de gaussianas, se logrará identificar aquellas entradas que recaen en más de

un *cluster*, lo cual los Resultados Preliminares (más adelante) han demostrado sucede con frecuencia. Reconocer estos puntos será de vital importancia para poder estudiarlos en búsqueda de una mejor comprensión del fenómeno. Este proceso suele recibir el nombre de **detección de anomalías**.

4.3.1. Parámetros y criterio de detención

Para este modelo existen al menos 4 parámetros a ajustar: comp_hiddens que corresponde al tamaño de las capas ocultas del *autoencoder*, comp_activation que corresponde a las funciones de activación de dichas capas, est_hiddens que corresponde a las capas ocultas de la red de estimación (modelo probabilístico) y est_activation, funciones de activación de la red de estimación. No se han evaluado criterios de detención para este modelo, pues aún no se ha estudiado en profundidad.

5. Software y recursos computacionales

5.1. Entorno de programación y librerías

El *software* a utilizar viene dictado principalmente por las librerías necesarias tanto para la extracción de características como para la implementación de los modelos y algoritmos de *clustering*. En particular se opta por usar el lenguaje de programación *Python* en su versión 3.10, esto debido a que es el lenguaje en el cual se encuentra la librería de *Turbo-FATS* y también por tratarse de un lenguaje para el cual existe una amplia gama de librerías y documentación relacionada con lo que es el aprendizaje de máquinas, además de ser la principal herramienta utilizada en el curso.

En cuanto al entorno de programación, se utilizan principalmente dos enfoques: *scripts* individuales para los procesamientos más pesados - como la extracción de características - destinados a ser ejecutados localmente en los equipos de los autores, y *notebooks* de Google Colab para la aplicación de algoritmos y modelos, entorno que permite además la posibilidad de utilizar aceleración de *hardware* (CPU, GPU Y TPU).

Las principales librerías involucradas en el trabajo son:

- ***pandas***, para el manejo de *dataframes* y archivos csv,
- ***numpy, numba***, como herramientas de soporte matemático y de optimización, necesarias para las librerías de *tensorflow* y *keras*,
- ***tensorflow, keras***, como librerías de aprendizaje automático y redes neuronales.
- **umap, umap[plot]**, librerías que implementan UMAP siguiendo la API de *tensorflow* y que resultaron esenciales para aplicar dicho algoritmo. En particular, umap[plot] permite la visualización de los resultados.
- ***bokeh***, como librería para la creación de visualizaciones interactivas.
- ***pandas-profiler***, necesaria para la generación de reportes o "perfiles" de los datos, los cuales se ven involucrados en el análisis exploratorio de los mismos, permitiendo reconocer aquellas *features* de mayor relevancia y descartar a su vez aquellas con poco o nulo aporte al problema.

- ***turbo-FATS***, librería que se empleó para extraer los *features* de las curvas de luz. Corresponde a una evolución de la librería ***FATS***, autoría de Isidora Nun, realizada por el auxiliar del curso y miembro de ALeRCE, Ignacio Reyes Jainaga.

  Desde luego, existen otras dependencias propias de cada librería, en especial para lo que es *turbo-fats*. No se entrará al detalle de ellas, pero son (principalmente): *C4J, scipy, GPy, abc,* entre otras.

5.2. Estimación de recursos computacionales

En cuanto a los recursos computacionales, estos variarán enormemente dependiendo de la etapa del proceso a replicar. Por ejemplo, aquellos a realizar en Colab - la aplicación de los modelos sobre la data ya procesada - son livianos tanto en entrada y salida como en carga de procesadores, por lo cual podrían ejecutarse en cualquier computador que cuente con las librerías y al menos 4GB de memoria RAM en un tiempo prudente (no más de 5 minutos cada simulación). Sin embargo, aquellos que se trabajaron de forma local - *scripts* individuales, enfocados al preprocesamiento de los datos y en especial a la extracción de *features* - podrían requerir de equipos con mayores prestaciones (16 GB de RAM en adelante, principalmente).

Teniendo en consideración que ninguno de los autores contaba con un equipo con las características antes mencionadas, se optó por intercambiar el requisito de memoria por un proceso **intensivo en I/O** (*input/output*). Esto trajo consigo una ventaja y una desventaja: por un lado, se logró extraer *features* y continuar con el trabajo. Por otro lado, se requirió de 42 horas de CPU para completar la tarea, aún con prioridad máxima y sin realizar ninguna actividad en paralelo.

Otras actividades, tales como la generación del perfil de los datos mediante *pandas-profiler*, se pueden realizar en Colab y no debiesen tomar más de 15 minutos en CPU. De utilizarse un entorno con aceleración por *hardware* - como el que Colab ofrece -, estos tiempos deberían ser aún menores.

5.3. Estimación de simulaciones a realizar

Se estima que, para cada modelo, será necesario realizar al menos 25 experimentos para probar con distintos subconjuntos de *features*, con y sin *data augmentation/subsampling* y con estandarización por desviación estándar o cuantiles. Al respecto:

- Para **UMAP**, cada experimento toma alrededor de 7 minutos en Colab (procesamiento + visualización), lo cual entregaría un total de 2 horas 55 minutos de procesamiento para este algoritmo.
- Para ***Autoencoders***, cada experimento toma alrededor de 10 minutos en Colab (procesamiento + visualización), lo cual agregaría un total de 4 horas 10 minutos de procesamiento.
- Para la **mezcla de Gaussianas sobre *autoencoder*,** se estima que cada experimento tomará alrededor de 15 minutos, sumando un total de 7 horas 15 minutos de tiempo de procesamiento.

Lo anterior suma un tiempo total de 13 horas 20 minutos invertidos en la ejecución de los algoritmos que se pretende sean suficientes para solucionar el problema. Si bien esta cifra es elevada - es más del

tiempo dedicado semanal que debería dedicarse al curso -, se corre con la ventaja de que dichos algoritmos no requieren demasiada supervisión (se pueden dejar ejecutando en segundo plano mientras se avanza con otras tareas). Se estima por ende que el tiempo restante del semestre será suficiente para llevar a cabo las simulaciones mencionadas y realizar los análisis correspondientes.

6. Salidas deseadas y función objetivo

Se busca que la salida de los dos primeros modelos, UMAP y *Autoencoders*, corresponda a una proyección en 2 dimensiones de los datos de entrada. Esta proyección se utilizará para generar visualizaciones en búsqueda de *clusters*.

Por cada *cluster*, se pretende además que sus miembros pertenezcan, mayormente, a una misma clase. Cabe recordar que los algoritmos no conocerán la clase de antemano (pues se trata de aprendizaje no supervisado), esta solo se utilizará para generar las visualizaciones y evaluar los *clusters* obtenidos. Se desea además que, si se asocia a cada *cluster* su clase predominante, el error de clasificación (los puntos de dicho grupo que no pertenecen a la clase asociada) sea bajo. Es decir, se busca optimizar el error de clasificación de los *clusters*.

Pese a lo anterior - las salidas deseadas -, se reconoce que el problema es desafiante y que difícilmente se lograrán óptimos muy favorables. Es acá donde el tercer modelo entra a jugar un rol importante, pues permitirá estudiar aquellos puntos que causen el error o el *overlapping* entre distintos *clusters*. Al respecto, se espera que la salida de DAGMM corresponda a un mapeo objeto - energía (entendiéndose energía como el índice de anomalía), lo cual debe permitir estudiar estos casos, por ejemplo, a través de un diagrama de energía.

7. Resultados esperados

7.1. Resultados y medidas de desempeño

Para los modelos de UMAP y *clustering*, se espera que la salida respete la estructura global de los datos, permitiendo identificar *clusters* asociados a cada clase en los datos mediante diagramas de dispersión. Se desea que estos *clusters* sean compactos, bien definidos y distantes entre sí. Para ello, se evaluará el resultado mediante dos métricas especializadas en la evaluación de algoritmos de *clustering*: el Coeficiente de Silhouette (ecuación 1) y el Índice de Dunn's (ecuación 2):

$$s(i) = \frac{b(i) - a(i)}{max\{dist(a(i), b(i)\}} \tag{1}$$

$$D = \frac{min\{dist(c_i, c_j\}}{max\{dist(c_i, c_j\}} \tag{2}$$

El coeficiente de Silhouette corresponde a una medida de cuán similar es un objeto a su propio cúmulo (cohesión) en comparación con otros cúmulos (separación) a través de un valor entre -1 y 1, donde un valor alto indica que el objeto está bien emparejado con su propio cúmulo y mal emparejado con los cúmulos vecinos. El índice de Dunn's, por su parte, es la relación entre el mínimo y el máximo

de distancia entre *clusters*. Un valor 0 indicaría que todos los *clusters* estan reunidos (lo cual es desfavorable). Un valor 1 indicaría que los clusters están bien separados.

Además de las métricas de *clustering*, al asociarle a cada *cluster* su clase predominante se pueden calcular también las métricas clásicas de *precision, accuracy, recall,* etc. Se dará especial atención a *recall* y *F1-Score* pues se trata de métricas más robustas ante el desbalance de los datos, que como ya se mencionó es bastante drástico.

Para el modelo de DAGMM, por otra parte, se espera que los valores que arroje para cada curva de luz permitan, efectivamente, identificar valores *outliers*. Para ello se considerará como *outlier* un punto que excede el percentil 99 en el índice de anomalia o *energía*.

7.2. Formato de los resultados

Como ya se adelantaba, los resultados se presentarán principalmente mediante gráficos: mientras para los modelos de MAPS y *Autoencoders* se generarán visualizaciones 2D de tipo *scatterplot* incorporando un mapeo del color a la clase de cada curva de luz según ALeRCE, para *DAGMM* se generará un diagrama de energía de cada punto. Las métricas obtenidas, por otra parte, se presentarán a través de tablas.

Parte del desafío propuesto por el equipo docente tras la presentación intermedia consta de incorporar *tooltips* o interacciones en las visualizaciones de *scatterplot*. Se intentará lograr este acometido mediante herramientas de la librería *Bokeh*, teniendo como alternativas librerías como *plotly* o *D3.js*.

8. Resultados preliminares

A la fecha de entrega del informe, se ha logrado obtener resultados preliminares para UMAP y Autoencoders. Estos resultados corresponden a la reducción de dimensionalidad sobre un conjunto de *features* reducidas, lo cual posteriormente se lleva al formato deseado para las visualizaciones (aún sin *tooltip*). Hasta el momento no se han calculado métricas sobre los *clusters* ni sobre el error de clasificación de los mismos.

A continuación se presentarán los resultados de cada modelo, así como las *features* utilizadas y demás parámetros de interés.

8.1. UMAP

En esta iteración se utilizó UMAP sin parámetros, salvo `random_state` para reproducibilidad. Es decir, se dejó al algoritmo decidir los mejores valores para `min_neighbors` y `min_dist`. Se realizaron diversos experimentos variando las *features* a utilizar y la cantidad de datos a trabajar (con o sin tratamiento al desbalance). Se utilizó `StandardScaler` como herramienta de estandarización.

La Figura 1 muestra el resultado de aplicar FATS sobre la totalidad de los datos con todas las *features*, es decir, con una dimensión de entrada igual a 99. La Figura 2, por otro lado, mues-

tra el resultado del mismo proceder más sobre un subconjunto de 26 características, que se denominará C1 = {Multiband_period, delta_period_g, delta_period_r, Harmonics_mag_1_g, Harmonics_mag_1_r, Harmonic_mse_r, Harmonics_mse_g, Power_rate 1/4, Anderson Darling_g, Anderson Darling_r, iqr_g, iqr_r, Amplitude_g, Mean_g, Meanvariance_g, Amplitude_r, Mean_r, Meanvariance_r, PairSlopeTrend_r, PairSlopeTrend_g, LinearTrend_r, LinearTrend_g, ExcessVar_r, ExcessVar_g}. La visualización inicial se incluye por corresponder al resultado más básico, mientras que la segunda corresponde al mejor resultado obtenido en esta fase del proyecto:

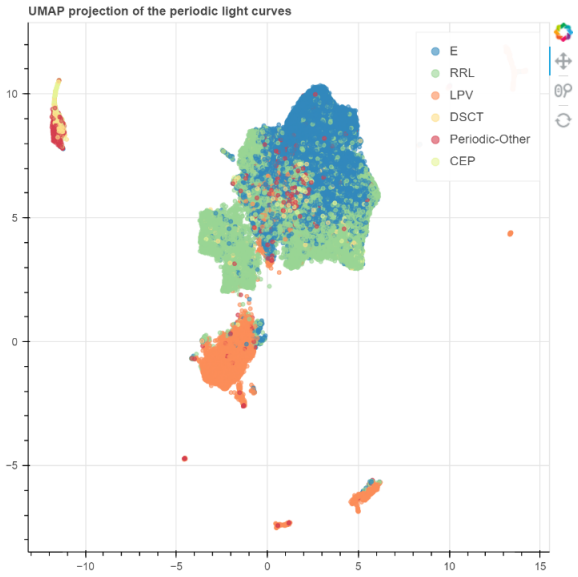

Figura 1: Resultado UMAP, usando todas las *features*, todos los datos

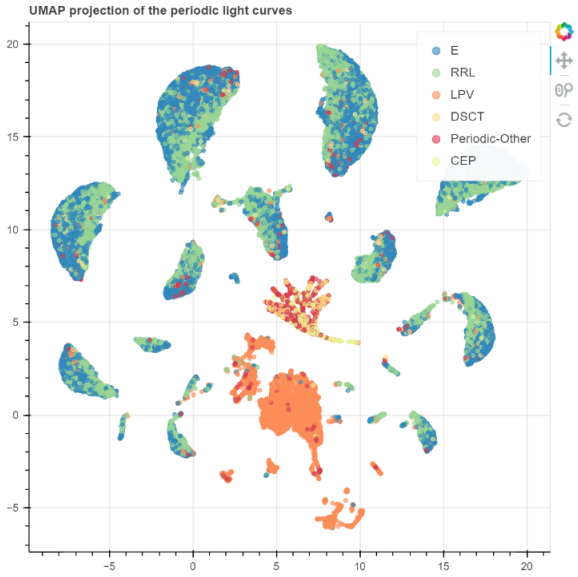

Figura 2: Resultado UMAP, usando el subconjunto $C1$ de *features*, todos los datos

Se observa que al utilizar todas las features, se generan cuatro grandes *clusters*: uno que reúne las

estrellas de tipo E y RRL, otro con las clases Periodic-Other, CEP y DSCT, y dos más que incorporan principalmente elementos de clase LPV y E. Si bien la cantidad de *clusters* es pequeña y se observan bastante compactos, ninguno captura bien una única clase, lo cual es sumamente desfavorable.

En vista de lo anterior, y siguiendo la recomendación del enunciado del proyecto, se opta por trabajar con el subconjunto C1 de features. El resultado para este nuevo experimento (Figura 2) incorpora muchos más *clusters*, alcanzando un total de al menos 13 agrupaciones, lo cual podría indicar un intento excesivo de *clusterizar*. Sin embargo, se reconoce una ventaja importante con respecto al caso anterior: ahora sí existen *clusters* con presencia mayoritaria de una sola clase, y corresponden a aquellos poblados principalmente por curvas de luz de tipo LPV (en naranja, bajo $y = 0$). Otro punto que llama la atención es que, aún con esta aparente *sobre-clusterización*, las curvas de luz de tipo E y RRL siguen presentando *overlapping* en cada uno de los *clusters* en que están presentes. Esto levanta de inmediato una necesidad para la próxima iteración: **encontrar las características que permiten diferenciar dichas clases**.

En cuanto al trabajo futuro con respecto a este modelo, se buscará obtener mejores resultados por medio de seguir las recomendaciones realizadas por el equipo docente: utilizar una estandarización por cuantiles y manipular el parámetro nearest_neighbors. También se buscará, para las próximas entregas, incorporar métricas que permitan evaluar de forma más objetiva los *clusters* que la simple inspección visual utilizada en este caso.

8.2. *Autoencoders*

En cuanto a la utilización de *autoencoders*, se generaron diversas pruebas variando la configuración del modelo y el subconjunto de *features* a utilizar, así como la aplicación o no de *data augmentation* y *oversampling.* Nuevamente, se utilizó un StandardScaler como herramienta de estandarización y no se logró incluir métricas de desempeño en razón del tiempo. Como consecuencia, el análisis a realizar es puramente visual.

El mejor resultado se logró mediante un subconjunto C2 de 21 *features*, aplicado a datos *subsampleados* y con *data augmentation*, sobre un modelo con la arquitectura presentada en la Tabla 2. La visualización de dicho resultado se presenta en la Figura 3.

Tabla 2: Arquitectura autoencoder fase intermedia

Tipo capa	**Neuronas**	**F. Activación**
Dense	99	relu
Dense	2048	relu
Dense	1024	relu
Dense	512	selu
Dense	256	sigmoid
Dense	512	relu
Dense	1024	relu
Dense	2048	relu
Dense	99	-

Figura 3: Resultado Autoencoder, usando todas las *features*, todos los datos

Se observa que, a diferencia de UMAP, con *autoencoders* se generan alrededor de 5 grandes *clusters*, dos de los cuales incorporan entradas de una única clase: LPV. Además, los puntos correspondientes a estrellas de tipo RRL se logran separar en mayor medida de aquellos de tipo E, generando un *cluster* exclusivo para el tipo E (lo cual representa un gran avance). Por otro lado, si bien las curvas de tipo CEP, Periodic-Other y DSCT siguen presentando un alto *overlapping*, se percibe una mejoría con respecto al resultado de FATS. En vista de ello, surge la interrogante si los buenos resultados se replicarán para el caso con el *dataset* completo (sin utilizar *subsampling*). Los resultados de este nuevo experimento se presentan en la Figura 2:

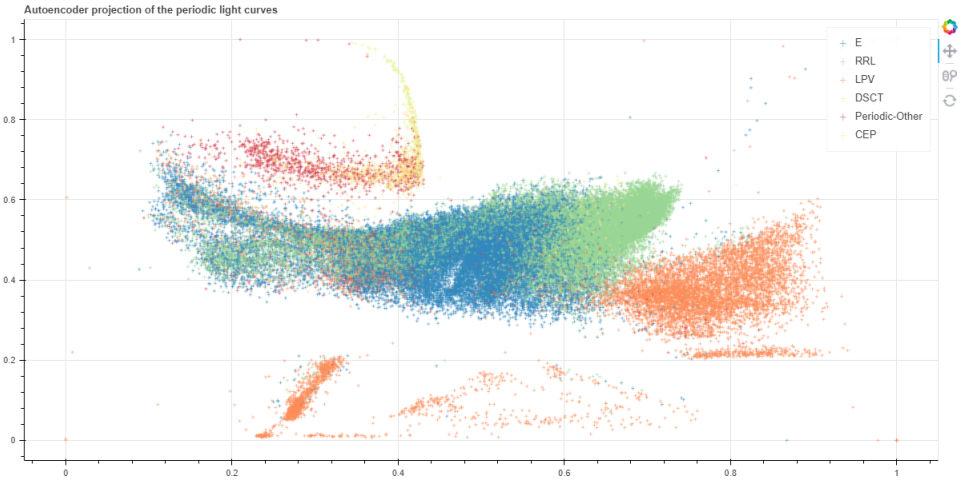

Figura 4: Resultado Autoencoder, usando el subconjunto $C1$ de *features*, datos con *data-augmentation* y *subsampling*

Se observa que, al incorporar todos los datos, las clases sobrerrepresentadas E y RRL vuelven a mostrar un gran *overlapping*. Sin embargo, otras clases tales como LPV (en naranja) y DSCT (en amarillo) dan lugar a *clusters* propios.

En lo que respecta al trabajo futuro para este modelo, la tarea principal constará de buscar arquitecturas que entreguen mejores resultados, y discutir con el equipo docente sobre la factibilidad de

trabajar solamente con los datos *subsampleados* o si resulta necesario trabajar con la totalidad de ellos. Así mismo, también será importante buscar herramientas de visualización que permitan incorporar *tooltips* y otras interacciones para alcanzar el formato deseado para los resultados.

9. Carta Gantt

A continuación se presenta la Carta Gantt acordada con el monitor del proyecto, Javier Molina, a inicios del mismo. En ella se distribuye el tiempo del semestre en las distintas tareas que se consideran necesarias para la consecución exitosa del proyecto. La carta se plantea como un punteo debido a la dificultad de plasmar, en el documento, la tabla completa.

- **FASE INTERMEDIA:**
  - Estudiar referencias y generar carta Gantt: **desde 01-10-2021 hasta 04-10-2021**
  - Exploración preliminar de datos con visualizaciones: **desde 04-10-2021 hasta 06-10-2021**
  - MP1: Planificación y Exploración de datos: **06-10-2021**
  - Visualización de datos mediante *autoencoders*: **desde 06-10-2021 hasta 12-10-2021**
  - MP2: Resultados preliminares: **13-10-2021**
  - Preparar presentación e informe: **desde 13-10-2021 hasta 17-10-2021**
  - Presentación Intermedia: **desde 18-10-2021 hasta 29-10-2021**
  - Definir aspectos a mejorar: **desde 29-10-2021 hasta 31-10-2021**
  - Estudiar referencias para la Fase Final: **desde 29-10-2021 hasta 01-11-2021**
  - Receso Académico: **desde 01-11-2021 hasta 07-10-2021**
- **FASE FINAL:**
  - Clustering con modelo de mezcla de Gaussianas: **desde 08-11-2021 hasta 13-11-2021**
  - MP3: Resultados intermedios: **10-11-2021**
  - Visualización de las curvas asociadas a clusters y análisis: **desde 13-11-2021 hasta 21-11-2021**
  - Preparar presentación e informe: **desde 21-11-2021 hasta 24-11-2021**
  - MP4: Resultados finales: **24-11-2021**
  - Presentación final (grupos adelantados): **desde 29-11-2021 hasta 03-12-2021**
  - Entrega informe final: **desde 13-12-2021 hasta 23-12-2021**

Desde luego, las etapas más tempranas del proyecto - comprendidas entre el inicio de la Fase Intermedia y hasta la Presentación Intermedia - presentaron un importante retraso con respecto a lo planificado inicialmente, debido a las ya mencionadas dificultades que se presentaron al momento de extraer los datos. Sin embargo, gracias al apoyo de Monitor y Auxiliar y a la dedicación puesta por los autores en el proyecto, para la fecha de entrega del presente informe (que corresponde con el final de las presentaciones intermedias) se ha logrado ponerse al día y se pretende continuar el semestre respetando la carta Gantt sin mayores dificultades.

Referencias

[1] Förster, F., et al. (2020), *"The Automatic Learning for the Rapid Classification of Events (ALeRCE) Alert Broker."*, arXiv preprint arXiv:2008.03303.

[2] Zwicky Transient Facility (ZTF), https://www.ztf.caltech.edu/

[3] McInes, Leland, Healy & Melville: (Feb 2018), *"UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction."* arXiv e-prints, página arXiv:1802.03426

[4] K. P Murphy (2002), *"Deep auto-encoders" in Machine Learning: A Probabilistic Perspective."*

[5] B. Zong et al. (2018), *"Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection"*., ICLR.

[6] Datos de Alertas Astronómicas, https://docs.google.com/spreadsheets/d/18HZqS_VqPMDqjOi6q9G60ReGKC1D4NRUU08CvNtWiFU/edit?usp=sharing

[7] Repositorio Monitor, https://github.com/JavierM23/unsupervised_periodic_lc_clf/blob/main/labels_set.csv

[8] Datos del proyecto,https://drive.google.com/drive/folders/1CKlSU8ZU5yAGG5LgeorxjPdAt0HztVvS

[9] Repositorio del proyecto https://github.com/matiasvergaras/EL4106-8a

[10] Sánchez-Sáez, P., et al. (2020), *"-Alert Classification for the ALeRCE Broker System: The Light Curve Classifier."*, arXiv preprint arXiv:2008.03311.

[11] Repositorio de Turbo-FATS, https://github.com/alercebroker/turbo-fats

[12] B. T. H. Zang et al. (2019), *"Deep Neural Network Classifier for Variable Stars with Novelty Detection Capability"*.

[13] Bolshakova, n. (2003, 1 abril),*"Cluster validation techniques for genome expression data. ScienceDirect."* https://www.sciencedirect.com/science/article/abs/pii/S0165168402004759

[14] Pathak, M. (2020, 23 noviembre), *"Evaluation Metrics For Machine Learning For Data Scientists. Analytics Vidhya."* https://www.analyticsvidhya.com/blog/2020/10/quick-guide-to-evaluation-metrics-for-supervised-and-unsupervised-machine-learning/

[15] Reyes, I. (2021, mayo), *"lc_classifier/feature_extraction.ipynb at main · alercebroker/lc_classifier.GitHub."*, https://github.com/alercebroker/lc_classifier/blob/main/examples/feature_extraction.ipynb