Universidad de La Laguna

SECCIÓN DE MATEMÁTICAS | FACULTAD DE CIENCIAS

Lára Kristjánsdóttir

Exploración de grandes estudios espectroscópicos con métodos de aprendizaje automático

Exploring large spectroscopic surveys using machine learning methods

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, Septiembre de 2018

Dirigido por

Thomas Masseron

Thomas Masseron
Departamento de Astrofísica
Universidad de La Laguna
38271 La Laguna, Tenerife

Agradecimientos

En primer lugar, me gustaría dar la gracias a mi tutor Thomas Masseron por su paciencia y dedicación para realizar este trabajo fin de grado.

En segundo lugar, gracias a mi mejor amigo y pareja Pedro Suárez Miranda que, sin tus consejos, ayuda, ánimo y sobre todo tu paciencia durante todos estos años no hubiera podido acabar este proyecto.

En tercer lugar, me gustaría dar las gracias a Sara, Herr Batista y Frau Cruz por la ayuda, el enorme ánimo y todos los quesos este año.

Quisiera agradecer también a mi amiga Freydís la ayuda con las traducciones al inglés.

Por último, me gustaría dar las gracias a Pedro, Quirina, mis padres, hermanos y baby. Gracias por apoyarme siempre con todo lo que decido hacer. Estos años en matemáticas no hubieran sido lo mismo sin ustedes.

Resumen · Abstract

Resumen

En esta memoria presentamos y aplicamos dos algoritmos de aprendizaje automático. Uno de clasificación y otro de reducción dimensional. Aplicamos estos algoritmos a una muestra de gran tamaño de espectros estelares del estudio APOGEE con el objetivo de identificar valores atípicos. Con el clasificador construimos una matriz de similaridades entre cada par de estrellas. Vemos la gran utilidad de la matriz de similaridades y nos damos cuenta que contiene bastante información sobre la muestra y su estructura. Utilizamos la reducción dimensional para visualizar y estudiar la matriz de distancias, que es la matriz inversa de la de similaridades. Discutimos los problemas y las limitaciones a la hora de computar una matriz de similaridades en una muestra muy grande. Como resultado se proponen futuros trabajos de mejora y avance. Este trabajo esta basado en los siguientes trabajos: [14] y [15].

Palabras clave: *aprendizaje automático – detección de valores atípicos – reducción dimensional – espectros estelares – estrellas extrañas.*

Abstract

In this work we will study and apply machine learning methods for detecting outliers and visualizing structure in a large dataset. We will first apply a classification algorithm to stellar spectra from the APOGEE survey and then use it to build a matrix containing a pair-wise similarity measure between every two objects in the data. Using dimesionality reduction techniques to explore the matrix (or its inverse, the distance matrix) will give us an inside of the importance of similarity measures. After applying the algorithms, we will discuss the issues and limitations when calculating a similarity matrix in a large dataset. Lastly, we will propose ideas on future work, mostly on how to continue and improve the project. This project is based on the following papers: [14] y [15].

Keywords: *machine learning – outlier detection – dimensional reduction – stellar spectra – peculiar stars.*

Contenido

Introducción

El aprendizaje automático (o *machine learning* en inglés), nombre introducido por Arthur Samuel en 1959 [1], es una rama de la inteligencia artificial, que a su vez forma parte de las ciencias de la computación, cuyo objetivo es utilizar técnicas estadísticas para dar a un sistema la capacidad de "aprender" a partir de datos sin ser programado. Es decir, trata de crear algoritmos capaces de generalizar comportamientos a partir de una información sumistrada en forma de observaciones del mundo real.

De esta manera, el aprendizaje automático es una manera de llevar las matemáticas al mundo práctico. En este trabajo veremos sobre todo la conexión con la estadística fundamental y con la teoría de la probabilidad. Asimismo, los conocimientos de optimización nos permitirán entender la eficiencia y la escalabilidad de nuestros algoritmos y la estructura de los datos de nuestras muestras.

Un gran número de problemas prácticos están siendo resueltos, optimizados o estudiados utilizando aprendizaje automático. La detección de "spam" en el e-mail y el reconocimiento de caras en una fotografía de Facebook son ejemplos que vemos cada día.

Por otro lado, la cantidad de datos provenientes de estudios científicos que estamos recogiendo, analizando y manipulando es cada vez mayor. Al aumentar la cantidad y la complejidad de los datos es necesario desarrollar nuevos métodos de modelos predictivos. En astronomía cada vez se están utilizando más las nuevas técnicas de aprendizaje automático, ya sea para detectar o clasificar objetos específicos o para buscar correlaciones o clústeres en datos de alta dimensión.

El objetivo de este trabajo es intentar entender el comportamiento y estructura de conjuntos de datos con un gran número de muestras. Para ello exploramos un gran conjunto de datos espectroscópicos, provenientes de un estudio científico, utilizando métodos de aprendizaje automático. Analizamos el funcio-

namiento de los métodos y los resultados de aplicarlos a las observaciones del estudio.

La memoria está estructurada en tres capítulos, esta misma introducción y un apéndice. El capítulo 1 comienza con una presentación de los fundamentos teóricos necesarios para después realizar la exploración del conjunto de datos. Trataremos sobre valores atípicos y métodos de detección, en concreto, el método llamado Random Forest 1.3.2. Se explica cómo funciona el método y cómo se puede utilizar para la detección de valores atípicos. A continuación se presenta el concepto de reducción dimensional y, finalmente, el método t-SNE 1.4, que es un ejemplo de reducción dimensional.

El capítulo 2 empieza con una introducción al conjunto de datos sobre los que aplicaremos las técnicas presentadas anteriormente, APOGEE. A continuación se presenta el repositorio con el código desarrollado para el proyecto, y se explica cómo instalarlo y configurarlo. Finalmente se detalla el funcionamiento del algoritmo que hemos desarrollado.

En el capítulo 3 se presentan los resultados obtenidos al ejecutar el programa, para lo cual utilizaremos una serie de tablas y visualizaciones. A continución realizamos una valoración de los resultados y del propio proyecto en las conclusiones. Finalmente, se plantean posibles ampliaciones al proyecto.

El anexo contiene el código del programa, escrito en python.

1

Fundamentos teóricos

En este capítulo se definen la terminología y los conceptos que se usarán a lo largo del proyecto. Empezamos con los valores atípicos, su detección y el método Random Forest no supervisado, que se usa para construir la matriz de distancias entre las muestras del conjunto de datos. Luego pasamos a la reducción dimensional, vemos los distintos algoritmos con los que podemos realizarla y finalmente hablamos del algoritmo t-SNE, que usamos para hacer la reducción dimensional de la matriz de distancias y así poder detectar los valores atípicos de manera más sencilla.

1.1. Valores atípicos

Un valor atípico es una observación que es distante al resto de observaciones de una muestra. Puede haber muchas explicaciones para la aparición de un valor atípico, como por ejemplo errores (humanos, de medida, al extraer o procesar los datos, etc.), errores intencionales (para valorar el método detección) o verdaderamente que la observación es muy diferente al resto de muestras (no hay error).

Hay dos tipos de valores atípicos: univariantes y multivariantes. Los multivariantes, como los que vamos a trabajar en este proyecto, se pueden encontrar en espacios de grandes dimensiones y eso nos puede dificultar explorarlos. La solución a este problema es la utilización de un modelo de aprendizaje automático, que nos hace posible analizar los valores atípicos.

Como no siempre es la misma característica la que hace a un valor atípico, no existe ningún método matemático estándar. Dependiendo del análisis de la muestra, hay distintos métodos para explorar estos valores. Por ejemplo, hay métodos para detectar valores extremos en estadística (análisis de valores extremos), para crear modelos de probabilidad de algo no probable (modelos

probabilísticos), para proyectar una muestra a una dimensión inferior linealmente (modelo linear) o para detectar valores alejados de la zona ocupada por un clúster (modelo basado en proximidad).

1.2. Aprendizaje supervisado y no supervisado

Típicamente, los diferentes algoritmos de aprendizaje automático se suelen clasificar de dos maneras: aprendizaje supervisado y aprendizaje no supervisado.

- El aprendizaje supervisado es un tipo de aprendizaje automático que permite deducir una función a partir de datos de entrenamiento. Se parte de un conjunto de datos etiquetados, es decir, datos para los cuales se conoce el valor de una variable objetivo, y a partir de ellos se genera un modelo que permite predecir el valor de esta variable para datos no etiquetados.
- El aprendizaje no supervisado es un método de aprendizaje automático que permite descubrir la estructura oculta en los datos. Estos algoritmos no reciben datos etiquetados, es decir, no hay un conocimiento a priori. Aunque la aplicación del método no supervisado normalmente se utiliza para la estimación de densidad en estadística, en este proyecto lo vamos a aplicar para descubrir patrones ocultos en los datos.

1.3. Random Forest

En este trabajo vamos a utilizar un método llamado Random Forest no supervisado.

1.3.1. Árboles de decisión

Un árbol de decisión es un modelo o grafo, de tipo árbol, de decisiones y sus posibles consecuencias. Cada árbol tiene nodos que dividen la muestra según un valor. Empezando desde la raíz, con toda la muestra, el árbol se divide en ramas de manera recursiva hasta llegar a un nodo terminal de clasificación. A este nodo de llegada se le llama hoja. Cuando el árbol está construido, se puede clasificar una observación desconocida recorriendo el árbol desde la raíz.

Un árbol de clasificación s nos permite predecir la clase de una observación x dada. Empezando en la raíz de árbol (la parte más alta), recorremos sus ramas respondiendo preguntas sobre x hasta llegar a una de sus hojas, y, donde y es un valor numérico si estamos tratando un problema de regresión o una clase si estamos tratando un problema de clasificación. Un árbol de este tipo es una aplicación que nos hace posible predecir la clase y para cada x dada. A la aplicación $s : X \to Y$ la llamamos clasificador, donde X es el conjunto de las

observaciones e Y son las etiquetas de las clases (o los valores numéricos en un problema de regresión). 1.1

Normalmente los árboles de clasificación son deterministas, es decir, que una misma muestra de entrenamiento siempre producirá el mismo árbol. Los árboles creados para Random Forest no tienen esta característica, ya que el criterio para elegir las variables según las que dividir no es el de mejor división (*best split*). Lo que se hace es que se toma un subconjunto aleatorio de todas las variables en cada nodo y por tanto puede ser que no sea el mismo en dos ejecuciones sucesivas.

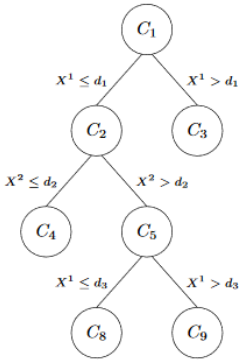

Figura 1.1. Árbol de clasificación

1.3.2. Random Forest

Random forest es un método de aprendizaje automático principalmente usado para problemas de clasificación y de regresión. Funciona de tal manera que construye un bosque de árboles de decisión según va entrenando el modelo. Para generar una predicción utiliza la moda, o valor más frecuente, de las clases predecidas los áboles individuales (en problemas de regresión, la media de los valores predecidos). El primer método de random forest fue introducido por Ho[16] en 1995.

En 2001, Breiman[6] y Adele Cutler, hicieron una extensión del algoritmo (lo llamaron Random Forest, con mayúsculas) uniendo el concepto de las características aleatorias de Ho y la idea de la agregación de bootstrap de Breiman[17].

La agregación de bootstrap, también conocida como empaquetado, es un algoritmo diseñado para mejorar la estabilidad y precisión de los algoritmos de clasificación estadística y regresión. A continuación vamos a describir la técnica.

Sea D un conjunto de entrenamiento estándar de tamaño n. La agregación de bootstrap genera m nuevos conjuntos de entrenamiento D_i, cada uno de un tamaño n', mediante muestreo uniforme y con reemplazo de D. Como los nuevos conjuntos se generan con reemplazo, algunas de las observaciones D_i se deben repetir. Si $n' = n$ entonces para un n grande, se esperan $1 - \frac{1}{e}$ elementos de D, que no estén duplicados, en el conjunto D_i. Este tipo de muestra se conoce como bootstrap. Para la aproximación de los m modelos se utilizan m muestras boostrap y se combinan con la moda (clasificación) o el promedio (regresión).

El conjunto de aprendizaje para cada árbol en un Random Forest está construido eligiendo n ejemplos de manera aleatoria y reemplazándolos con los n ejemplos libres en la muestra. Como resultado cerca de un tercio de los n ejemplos no están presentes en el conjunto de aprendizaje de cada árbol. Luego para estos ejemplos se puede predecir una etiqueta. Al agregar las predicciones de todas estas observaciones, las que no están presentes en el conjunto de aprendizaje, sobre todos los árboles se puede determinar una estimación interna del error de generalización del método Random Forest1.2.

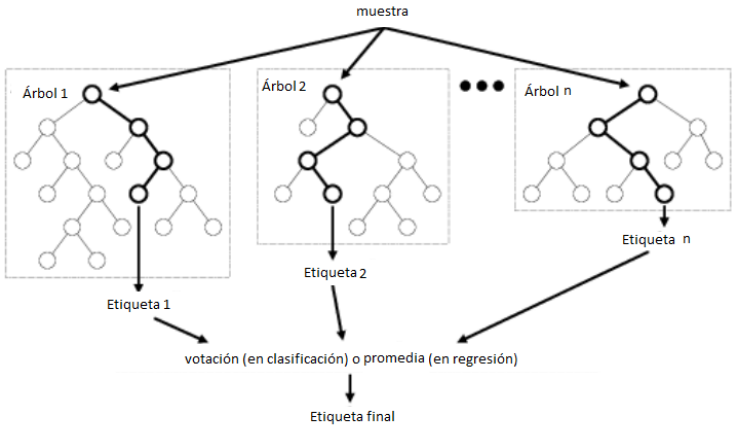

Figura 1.2. Estructura de un modelo de un método a base de árboles de clasificación

1.3.3. Matriz de distancias y detección de valores atípicos

Utilizando Random Forest podemos conseguir medidas de la similaridad entre pares de observaciones de la muestra. Cada una de las n observaciones las representamos con un vector de características. Las similaridades se inicializan a cero y si las observaciones i y j, al bajar por el árbol de clasificación, terminan en el mismo nodo entonces la similaridad s_{ij} se incrementa en uno. La medida de

similaridad de un par de observaciones se normaliza según el numero de árboles en el bosque. Por lo tanto las similaridades forman una matriz de $n \times n$ donde los elementos son las similaridades s_{ij} normalizados. Entonces como los valores de la matriz de similaridades son probabilidades, podemos construir la matriz de distancias correspondiente, que es la matriz donde los elementos son $d_{ij} = 1 - s_{ij}$.

Una observación x se considera como un valor atípico si el valor de la distancia media, respecto de todas las demás observaciones de la misma clase, es alto. Este valor de ahora en adelante lo llamaremos *weirdness score*.

1.4. Reducción dimensional y t-SNE

En esta sección hablamos sobre la reducción dimensional y sus fundamentos. Luego se presenta un algoritmo llamado t-SNE, que es el que utilizaremos en el proyecto.

1.4.1. Reducción dimensional

Muchas veces los problemas de aprendizaje automático tienen que ver con un gran número de características para cada instante del aprendizaje. Esto puede hacer que el aprendizaje sea muy lento y tienda a ser sobreajustado [*]. A este fenómeno, a veces conocido como "the curse of dimensionality", se asocian muchos problemas y para resolverlo es necesario reducir bastante el numero de dimensiones para mejorar la función del modelo y ayudarnos llegar una solución óptima. La resolución de este tipo de problemas se llama reducción dimensional y funciona manipulando la muestra de aprendizaje, eliminando observaciones que no son importantes o uniendo observaciones muy parecidas, y reduciéndolas de manera que se pierda la mínima cantidad de información sobre los datos. Haciendo eso, podemos mejorar mucho la rapidez del aprendizaje. Otro efecto deseable de la reducción dimensional es que permite la visualización en 2 o 3 dimensiones de datos con dimensión original alta, lo que permite entender mejor el conjunto de datos.

Las técnicas de reducción dimensional están divididas en dos grupos: selección de características (o *feature selection* en inglés) y extracción de características (o *feature extraction* en inglés).

- selección de características: son técnicas que buscan subconjuntos más pequeños de un conjunto multidimesional de datos para crear un modelo.
- extracción de características: transforma datos de alta dimensión a espacios de menos dimensiones. Existen muchos métodos para hacer eso, por ejemplo el método llamado t-SNE cuyo utilizamos en este trabajo.

[*] Sobreajuste (u *overfitting* en inglés) es el efecto de sobreentrenar un algoritmo de aprendizaje con unos ciertos datos para los que se conoce el resultado deseado[27]

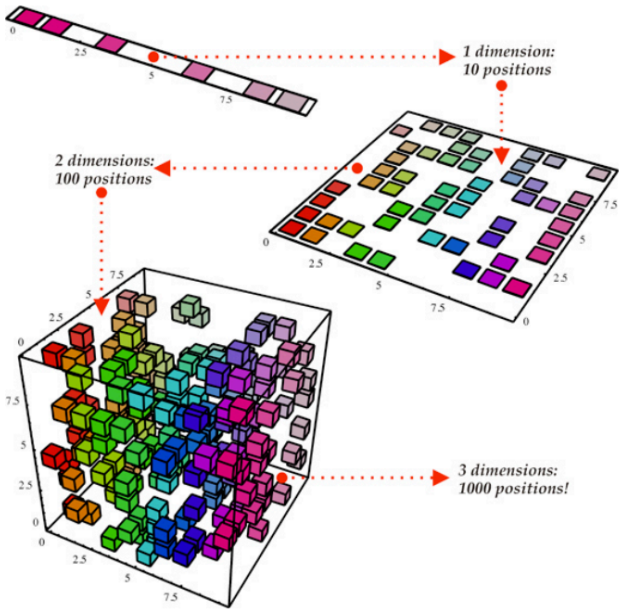

Figura 1.3. Explicación simple de la reducción dimensional [11]

1.4.2. t-SNE

t-SNE o *t-distributed stochastic neighbor embedding*, desarrollado por Laurens van der Maaten y Geoffrey Hinton [12], es un método de reducción dimensional no lineal apropiado para la visualización de datos de alta dimensión en un espacio de dos o tres dimensiones.Es una técnica relacionada con el método *Stochastic Neighbor Embedding* de Hinton y Roweis [13].

SNE o *Stochastic Neighbor Embedding* empieza convirtiendo un espacio euclídeo de distancias entre puntos de datos de alta dimensión , X, a probabilidades condicionadas que representan sus similaridades. Es decir, la similaridad entre x_i y x_j, en X, es la probabilidad condicionada, $p_{i|j}$, de que x_j elegiría a x_i como vecino. Luego para el espacio Y de la dimensión a la que queremos reducir, es posible computar unas probabilidades condicionadas, $q_{i|j}$, similares para los puntos y_i y y_j de Y a partir de los puntos x_i y x_j de X. Para que el modelo esté bien hecho, la probabilidad condicionada de los puntos x_i y x_j tiene que ser la misma que la de y_i y y_j. El objetivo de SNE es encontrar una representación de dimensión baja que minimiza la diferencia entre $p_{i|j}$ y $q_{i|j}$.

La diferencia entre t-SNE y SNE es que el método t-SNE es mucho más fácil de optimizar y produce una visualización mucho mejor por reducir la posi-

bilidad de que todas la observaciones se junten en el centro del mapa de visualización.

Como SNE, t-SNE empieza midiendo las similaridades entre pares de puntos u observaciones de alta dimensión, mirando solamente las similaridades entre puntos cercanos, de la manera siguiente:

Sea X un espacio de alta dimensión y sea x_i una observación de X. Tomamos el punto x_i, y la gaussiana centrada en él. Luego medimos la densidad para todos los puntos dentro del área encerrada por la gaussiana 1.4. A continuación normalizamos, como vemos en el denominador de la ecuación 1.1. Si $x_j \in X$, entonces la similaridad entre x_i y x_j es la probabilidad

$$p_{ij} = \frac{exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})}{\sum_k \sum_{l \neq k} exp(-\frac{\|x_k - x_l\|^2}{2\sigma^2})} \tag{1.1}$$

Es una distribución de probabilidad de un par de puntos donde la probabilidad de elegir un par de puntos es proporcional a la similaridad. Para puntos cercanos esa probabilidad es relativamente grande, mientras que para puntos muy lejanos puede ser ínfima.

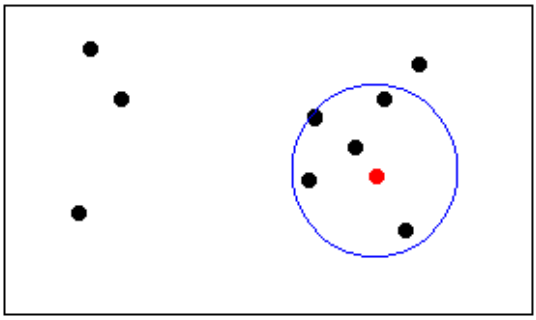

Figura 1.4. Espacio de dimensión alta donde x_i es el punto rojo y el área cerrada por la gaussiana está representada por el azul.

En la práctica, se mide la probabilidad de manera un poco diferente para poder controlar el número de puntos que entran al área cercana. Para ello calculamos la distribución condicional

$$p_{i|j} = \frac{exp(-\frac{\|x_i - x_j\|^2}{2\sigma_i^2})}{\sum_{j' \neq i} exp(-\frac{\|x_i - x_{j'}\|^2}{2\sigma_i^2})} \tag{1.2}$$

Como vemos en la ecuación 1.2, solamente se normalizan los puntos relacionados a $x_i \in X$. Eso nos deja poner el ancho de banda de σ_i de manera que

la perplejidad de la distribución condicional sea fija. Muchas veces hay diferente partes de un espacio con diferentes densidades y calculando la probabilidad de esta manera podemos adaptarnos a esas densidades.

Luego tenemos la similaridad final en el espacio X:

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N} \tag{1.3}$$

Ahora nos fijamos en el espacio de dimensión baja Y, que es al que queremos llegar (normalmente de dimensión dos o tres). Empezamos poniendo puntos en el mapa representando cada punto de la alta dimensión X como un punto en Y. Aquí hacemos más o menos lo mismo que en el espacio de alta dimensión. Fijamos $y_i \in Y$ correspondiente a $x_i \in X$ y tomamos un área, usando la distribución t-student con un grado de libertad, con y_i como centro1.5. Utilizamos la t-student, no la gaussiana como en SNE, para que dos puntos muy lejanos en el espacio de alta dimensión están mucho más alejados en el de baja dimensión para conseguir una visualización mejor. Ahora medimos la densidad entre y_i y todos los puntos del espacio. Normalizamos dividiendo entre todos los pares de puntos. Entonces tenemos que la similaridad entre los puntos y_i e y_j en Y es:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|y_k - y_l\|^2)^{-1}} \tag{1.4}$$

Queremos que $q_{i|j}$ represente a $p_{i|j}$ de la mejor manera. Si $q_{i|j}$ y $p_{i|j}$ son similares, la estructura aparece en el mapa de dimensión baja y en el de dimensión alta de manera similar. Ahora medimos la diferencia entre $q_{i|j}$ y $p_{i|j}$ utilizando la divergencia de Kullback-Leibler, que es un método estándar para medir la distancia entre distribuciones de probabilidades.

$$KL(P \mid\mid Q) = \sum_i \sum_{j \neq i} p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{1.5}$$

t-SNE ordena los puntos minimizando $KL(P \mid\mid Q)$ de manera que los valores de p_{ij} sean similares a q_{ij}.

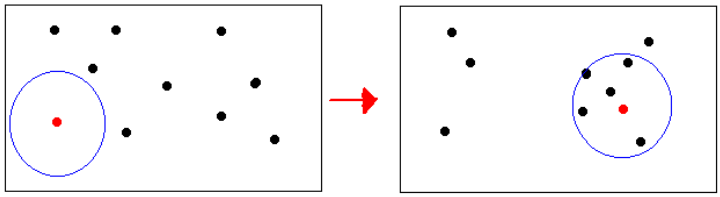

Figura 1.5. Espacio de dimensión baja donde x_i es el punto rojo y el área cerrada por la t-student está representada en azul.

2

Aplicación de Random Forest y t-SNE

En este capítulo empezamos introduciendo el estudio APOGEE y los datos con los que vamos a trabajar. Luego pasamos a hablar sobre la metodología que hemos utilizado, en particular, sobre la implementación de nuestro algoritmo.

2.1. El estudio APOGEE y Los datos

APOGEE o *The Apache Point Observatory Galactic Evolution Experiment*[18] es un proyecto que recoge espectroscopía infrarroja [*] para realizar un estudio sistemático de todas las partes de nuestra galaxia. Con el instrumental de APOGEE se pueden captar hasta 300 espectros a la vez. El estudio recoge un gran número de regiones distintas del cielo y en la mayoría de ellas se hacen múltiples observaciones en épocas diferentes para identificar las estrellas binarias [*] a través de la detección de la variación de su velocidad radial [*]. El proyecto APOGEE tiene como objetivo principal el estudio de las estrellas frías, sobre todo de las gigantes rojas, en las distintas componentes de la Vía Láctea: disco fino, disco grueso, halo y bulbo galáctico.

La alta resolución de espectros de APOGEE nos da información sobre las propiedades de las estrellas individuales con mucho detalle. Los *APOGEE Stellar Parameters and Chemical Abundances Pipeline* (ASPCAP) determinan los parámetros de las estrellas y 15 abundancias elementales a través del análisis automático de los espectros. Para cada objeto a considerar en este trabajo tenemos un fichero que contiene información tipo ASPCAP sobre el mismo en formato `.fits`.

[*] la parte infrarroja (H-band) del espectro electromagnético[26].

[*] estrella binaria es un sistema estelar compuesto de dos estrellas que giran mutuamente alrededor de un centro de masas común[25]

[*] Velocidad radial de un objeto respecto a un punto dado es el ritmo de cambio de distancias entre el objeto y el punto[24].

2.2. Metodología

En esta sección detallamos los pasos que hemos dado para realizar el proyecto. En particular, explicamos con detalle cómo funciona nuestro algoritmo. Esta sección también sirve como guía para instalar el proyecto en una nueva máquina y ejecutar el programa.

2.2.1. Instalación

Para poder ejecutar el programa que implementa nuestro algoritmo es necesario haber instalado el software de control de versiones git[19] y el entorno de desarrollo para python Anaconda, que, en particular incluye el intérprete de python. Debe usarse la versión 3 de este lenguaje de programación. El programa puede ejecutarse tanto en un sistema operativo Windows como en Linux.

Para instalar el programa, debemos ejecutar el siguiente comando desde la consola de Linux o desde el programa "git bash"de Windows:

```
git clone https://LaraKristjansdottir@bitbucket.org/LaraKristjansdottir/tfg.git
```

Se nos creará una carpeta llamada `tfg` dentro del directorio donde hayamos ejecutado el comando.

2.2.2. Estructura del repositorio

En la carpeta raíz del directorio podemos encontrar una serie de ficheros `.py` que contienen el código desarrollado para el proyecto. Además, hay una serie de subdirectorios que describimos a continuación:

- `img`, que contiene imágenes que hemos ido generando.
- `log`, que contiene un fichero de log donde se van registrando una serie de eventos que tienen lugar durante las sucesivas ejecuciones del programa, junto con la fecha y la hora en la que se terminaron.
- `other_data`, que contiene ficheros de datos generados por nosotros y que son utilizados desde el código del proyecto. Estos ficheros son:
  - `corrupt_fits.txt`, que contiene un listado de los ficheros `.fits` corruptos.
  - `targets.txt`, que contiene un listado con ficheros `.fits` de especial interés para los investigadores.
  - `targets.csv`, que contiene una tabla con datos adicionales sobre los ficheros de especial interés.
- `output`, que contiene los ficheros de datos que va generando el programa.
- `papers`, que contiene un cuaderno de iPython[22] utilizado por los autores de [14] , a modo de referencia.

- `tableau`, que contiene los ficheros de Tableau[23] utilizados para la visualización de los datos.
- `trial_apogee_spectra`, que contiene una pequeña selección de ficheros `.fits` pertenecientes a APOGEE 2.1 que pueden servir para realizar pruebas sobre el algoritmo sin tener acceso al propio APOGEE ni gastar mucho tiempo de ejecución.
- `utils`, que contiene dos scripts de python que hemos utilizado para generar los datos del directorio `other_data`:
  - `fits_audit.py`, que permite detectar qué ficheros `.fits` de APOGEE están corruptos y hemos de ignorar en nuestro programa. Con los datos generados por este script hemos rellenado el fichero `other_data/corrupt_fits.txt`
  - `targets_audit.py`, que permite comprobar si los ficheros de especial interés están presentes en el conjunto APOGEE y no están corruptos. Con los datos generados hemos rellanado el fichero `other_data/targets.txt`

2.2.3. Configuración

Antes de ejecutar el programa, es necesario indicar en qué directorio se encuentran los ficheros `.fits`. Esto lo hacemos modificando el fichero de configuración situado en `config/config.json`, de manera que la ruta del directorio fuente esté asociada a la clave `fits_directory`. En el repositorio se incluye este fichero con datos de ejemplo.

2.2.4. Ejecución

En el repositorio que se descarga con git está incluido el fichero

`execute_outlier_detection.py`

que contiene un script de python que permite ejecutar el programa. Para ejecutarlo, podemos simplemente escribir el siguiente comando desde la terminal de Linux:

```
python3 execute_outlier_detection.py
```

Si estamos en Windows, desde dentro del entorno de desarrollo Spyder, que viene instalado con Anaconda, simplemente abrimos el fichero y pulsamos ejecutar (F5).

Sin embargo, si estamos trabajando de manera remota utilizando un cliente de SSH, como es el caso cuando trabajamos con el computador Divan, es preferible ejecutar el comando de la siguiente forma:

```
nohup python3 execute_outlier_detection.py > f.out 2> f.err < /dev/null &
```

De esta manera nos aseguramos de que la ejecución del programa no se interrumpe si nos desconectamos de la sesión SSH, lo cual es muy útil cuando el programa tarda mucho tiempo en terminar, como es nuestro caso.

2.2.5. Parámetros del algoritmo

Dentro del fichero `execute_outlier_detection.py`, se encuentra una única instrucción con una llamada a la función `outlier_detection()`, que es la que verdaderamente hace el trabajo. Los parámetros de esta función son:

- El número de muestras, `n_samples`.
- El número de procesadores, `n_jobs`.
- El número de estimadores para construir el modelo Random Forest, `n_estimators`.

De esta manera los parámetros se pueden ajustar a las necesidades del usuario. Si queremos ejecutar el programa para todo el conjunto APOGEE podemos pasar como número de muestras la variable `all_samples`, que contiene el número total de ficheros válidos de APOGEE. El número de procesadores interviene en la paralelización de la generación del modelo Random Forest, de manera que un número más elevado supone un menor tiempo de ejecución. El valor máximo de este parámetro está limitado por el hardware donde se ejecute el programa.

2.2.6. Descripción del algoritmo

El código al completo del proyecto se puede consultar en 3.3 En esta sección detallaremos paso a paso lo que hace la función `outlier_detection()`, en adelante "la función principal"del fichero `outlier_detection.py`

Lectura del fichero de configuración

La ruta con los ficheros `.fits` se obtiene, como hemos dicho, de un fichero de configuración `.json` situado en el subdirectorio `config`. Para leer este fichero se utiliza la función `generate_config()` del fichero `config_manager.py`. Esta función simplemente importa la librería `json` de python y la utiliza para el el fichero de configuración. Posteriormente, los datos de configuración son accedidos haciendo `config[<clave>]` donde `clave` es la clave definida en el fichero `json`. Esto lo hemos hecho así porque de esta manera es muy sencillo crear y acceder a nuevas clases mientras se mantiene legible el fichero de configuración.

Carga de la matriz de características

En primer lugar, generamos la matriz de características a partir de los ficheros `.fits` del conjunto APOGEE. Esto lo hacemos con la función `load_feature_matrix_random()` del fichero `common.py`. La matriz de características, `X`, tiene tantas filas como número de muestras hayamos pasado como parámetro a la función principal, y un número fijo de columnas, 8575, que describen el espectro electromagnético de cada muestra.

La función `load_feature_matrix_random()` primero carga 119 ficheros que se corresponden con estrellas de especial interés para los investigadores del IAC. Es por este motivo que si el número de muestras que se pasa a la función principal es menor que este valor, el programa notificará un error y terminará. A continuación rellena el resto de muestras, `n_samples - 119`, con una selección aleatoria de ficheros `.fits` válidos. Por ficheros válidos entendemos aquellos ficheros `.fits` que no dan error al leerlos con la función `fits.getdata()`, que pertenece a la librería `astropy` y se usa para obtener el vector con el espectro electromagnético a partir del fichero `.fits` asociado a una estrella. De esta manera, nos aseguramos de que la carga no falla. Hay 39 ficheros no válidos, que se pueden consultar en el fichero `other_data/corrupt_fits.txt`. Las 119 estrellas de especial interés se pueden consultar en el fichero `other_data/targets.txt`.

Además de generar la matriz de características, la función de carga también devuelve una lista con los identificadores o índices de cada muestra de la matriz. Estos identificadores nos permitirán posteriormente etiquetar cada estrella en la visualización que generemos. Un identificador se obtiene a partir del nombre del fichero `.fits` eliminando el prefijo `aspcapStar-r8-l31c.1-` y la extensión `.fits`.

Generación de datos sintéticos

La generación de datos sintéticos, o muestras *bootstrap*, se hace en la función `get_synthetic_data()` del fichero `synthetic_data.py`. Esta función recibe como parámetro la matriz de características, `X`, y calcula otra matriz, `X_syn`, con las mismas dimensiones que `X` y que contiene los datos sintéticos.

Para generar las muestras *bootstrap* se recorre la matriz de características por columnas (cada columna representa una característica diferente). Para la columna i-ésima se genera una combinación con repetición de sus elementos, obteniendo una nueva columna que será a su vez la columna i-ésima de la matriz de datos sintéticos. Esto lo hacemos con las instrucciones

```
obs_vec = X[:,i]
syn_vec = np.random.choice(obs_vec, len(obs_vec))
X_syn[:,i] += syn_vec
```

Combinación de datos reales y sintéticos y etiquetado de las muestras

A nuestro clasificador Random Forest queremos proporcionarle todos los datos que tenemos, tanto los reales como los sintéticos que acabamos de generar. También queremos etiquetar estos datos, puesto que Random Forest es originalmente un algoritmo de aprendizaje supervisado y necesita datos etiquetados para funcionar. Etiquetamos las muestras reales como pertenecientes a la clase 1 y las

muestras sintéticas como pertenecientes a la clase 2. Luego concatenamos las dos matrices de características, `X` y `X_syn`, para formar `X_total`, y los dos vectores de variable objetivo (que contienen las clases), `Y` e `Y_syn`, para formar `Y_total`. Este trabajo lo hace la función `merge_data_and_synthetic_samples()` del fichero `merge_and_assign_class.py`

Ajuste del modelo Random Forest

A continuación nuestro programa ajusta un modelo Random Forest a los datos combinados de muestras reales y sintéticas etiquetadas en el paso anterior. Esto se hace desde la función principal utilizando el objeto `RandomForestClassifier` de la librería `sklearn`, que es la librería de referencia de python para aprendizaje automático. En primer lugar hay que instanciar un objeto `RandomForestClassifier`, especificando el número de estimadores (árboles) con los que queremos trabajar en el parámetro `n_estimators` y el número de procesadores que queremos destinar al cálculo en el parámetro `n_jobs`.

```
rand_f = RandomForestClassifier(n_estimators=n_estimators, n_jobs = n_jobs)
```

Una vez hecho esto, se ajusta el modelo con la instrucción

```
rand_f.fit(X_total, Y_total)
```

Cálculo de la matriz de distancias y de la *weirdness score*

Para obtener la matriz de distancias primero es necesario calcular la matriz de similaridades asociada. El cálculo de la matriz de similaridades tiene lugar en la función `build_similarity_matrix()` del fichero `similarity_matrix.py`. Esta función recibe como parámetros el modelo previamente ajustado de Random Forest, `rand_f` y la matriz de características, `X`.

La similaridad entre dos muestras A y B, en base al modelo Random Forest, se define como el número de árboles que clasifican en la misma clase a A y a B donde además A y B son predecidos como pertenecientes a la clase real y terminan en la misma hoja del árbol. Este número es dividido entre el número de predicciones reales realizado para obtener la similaridad normalizada. Por ejemplo, si las muestras x_0 y x_1 son clasificadas dentro de la clase 1 por el árbol y_0, y además x_0 y x_1 acaban en la misma rama del árbol, contaremos un punto de similaridad de cara al resultado final. Si por el contrario, las muestras x_0 y x_1 terminan en hojas diferentes, o si una de ellas es clasificada como sintética, no contaremos ningún punto.

La matriz de similaridades consiste en una matriz simétrica con unos en la diagonal, donde la posición (i, j) contiene la similaridad entre la muestra i y la muestra j.

Así pues, la función `build_similarity_matrix()` lo primero que hace es aplicar el modelo `rand_f` a cada muestra de la matriz de características, de manera que obtenemos otra matriz, `apply_mat`, de dimensiones

```
n_samples X n_estimators
```

con el índice de la hoja en la que terminó cada muestra para cada árbol. Las hojas que predijeron las muestras como sintéticas se marcan con el valor -1 para indicar que no son válidas.

El cálculo de la matriz de similaridades sin normalizar se realiza en la función `compute_numerator_numpy()` del mismo fichero. Esta función itera sobre el número de muestras y, para cada una, rellena la fila de la matriz de similaridades correspondiente. Para ello, se vale de las operaciones matriciales de `numpy`, la librería de python para cálculo numérico. En concreto, la instrucción

```
numerator[i] = np.sum((apply_mat == apply_mat[i]) \
                  & (apply_mat != exclusions), axis=1)
```

lo que hace es primero, con `apply_mat == apply_mat[i]`, generar una matriz booleana de tamaño `n_samples X n_estimators` que en la posición (j, k) indica si el árbol k termina en misma hoja para la muestra i que para la muestra j. En particular, cuando i y j valen lo mismo, es decir, cuando se compara una muestra contra sí misma, todas las posiciones valen cierto. La segunda comprobación, `apply_mat != exclusions`, es similar a la primera y permite conocer qué posiciones de `apply_mat` son diferentes de -1, es decir, son posiciones válidas. Lo que sigue es una operación lógica `AND` entre las dos matrices posición a posición y finalmente una suma de los elementos de cada fila. En una suma de booleanos, se interpreta el valor `cierto` como 1 y `falso` como 0. De esta manera se genera un vector de tamaño `n_samples` que se asigna a la fila i de la matriz de similaridades y representa la similaridad, sin normalizar, entre la muestra i y todas las demás.

Hemos incluido en el código una versión anterior de esta función, llamada `compute_numerator()` y que realiza exactamente el mismo cálculo con la diferencia de que los valores de la matriz de similaridades se calculan posición a posición, en lugar de fila a fila. Esta función es más sencilla de entender, ya que no usa las operaciones matriciales de `numpy` pero al mismo tiempo es mucho menos eficiente. Ejecutando con los mismos parámetros, la primera versión resultó ser un 42 % más eficiente*.

También nos parece adecuado mencionar que la misma operación en [CITA] se hace de una manera todavía más compacta, de tal manera que con sólo una instrucción se consigue generar entera la matriz de similaridades sin normalizar. La instrucción en cuestión es

* 5 horas y 20 minutos frente a 2 horas y 17 minutos, con 20000 muestras, 44 procesadores y 2000 estimadores

```
numpy.sum((apply_mat[:, None] == apply_mat[None, :]) \
    & (apply_mat[:, None] != -1) & (apply_mat[None, :] != -1), axis=2)
```

Al hacer `apply_mat[:, None] == apply_mat[None, :]`, lo que se obtiene es una matriz tridimensional que contiene el resultado de comparar cada fila de `apply_mat` con toda la matriz `apply_mat`, es decir, un cubo de

```
n_samples X n_estimators X n_samples
```

elementos. Luego al hacer la suma, se elimina esta dimensión extra, quedando como resultado final la matriz de similaridades sin normalizar. Este método, aparte de poder realizar el cálculo con una sola instrucción, probablemente es más eficiente, ya que explota al máximo las operaciones matriciales de `numpy`. Sin embargo, encontramos que el hecho de generar una matriz con una dimensión extra consume muchísima memoria, hasta el punto que se agota fácilmente la memoria de la máquina donde se ejecuta el programa. Por ejemplo, con 16GB de memoria de nuestro equipo local no pudimos pasar de las 4500 muestras, y con los 4TB de memoria del computador Divan, no pudimos pasar de las 10000 muestras. Por este motivo recomendamos utilizar la versión propuesta en `compute_numerator_numpy()`.

Como último paso de esta función la matriz sin normalizar se normaliza dividiendo todos los elementos de la columna i-ésima entre el número de elementos válidos de la fila i-ésima. De esta manera obtenemos unos en la diagonal y valores entre cero y uno en el resto de elementos. Esto es así porque, precisamente, el elemento (i, i) de la matriz sin normalizar contiene la suma de elementos no nulos de la fila i.

Una vez fuera de la función `build_similarity_matrix()`, obtenemos la matriz de distancias restando de 1 cada elemento de la matriz de similaridades. Esto hace que, en particular, obtengamos ceros en la diagonal, lo que quiere decir que la distancia de una muestra a sí misma es cero. En notación matricial de `numpy`:

```
dis_mat = 1 - sim_mat
```

Finalmente, la *weirdness score* de cada muestra se calcula sumando los elementos de cada fila de la matriz de distancias y dividiendo por el número de muestras. Este valor se almacena en la variable `sum_vec`. Un mayor valor de la *weirdness score* se debe interpretar como una estrella que está "más alejada de todas las demás"que el resto.

Reducción dimensional con t-SNE

Con la matriz de distancias y las *weirdness score* calculadas queremos finalizar nuestro programa generando datos que permitan una sencilla visualización e identificación de las muestras atípicas. Como la visualización de la matriz de

distancias es algo complicado, aplicamos la técnica de reducción dimensional t-SNE a los datos de la matriz de distancias, de manera que podamos representar en dos dimensiones datos que originalmente tienen `n_samples` dimensiones. Esto lo hacemos desde la función principal utilizando el objeto `t-SNE` de la librería `sklearn`, de la que ya hablamos cuando tratamos el ajuste del modelo Random Forest. Nuevamente hay que seguir dos pasos. En un primer paso se instancia el objeto especificando los parámetros más importantes, en este caso, el número de componentes, `n_components`, la tasa de aprendizaje, `learning_rate` y la perplejidad, `perplexity`.

```
t-SNE = t-SNE(n_components=2, verbose=1, learning_rate=1000, perplexity=2000)
```

En nuestro caso, el número de componentes debe ser 2, ya que queremos poder representar los datos de la matriz de distancias en dos dimensiones. Los valores de `learning_rate` y de `perplexity` afectan a cómo de apretujados quedan los datos después de haber realizado la reducción dimensional. Los valores propuestos se han validado con los resultados obtenidos a través de un proceso de ensayo y error. `verbose=1` simplemente genera mensajes de log sobre la evolución del algoritmo.

En un segundo paso, se ejecuta la reducción dimensional con la instrucción

```
t-SNE_results = t-SNE.fit_transform(dis_mat)
```

Volcado a disco de los resultados

Para terminar, guardamos en el disco duro los resultados generados por el programa. Para esto utilizamos un `DataFrame` de `pandas`. Un `DataFrame` es esencialmente una tabla indexada por filas y columnas. La librería de python `pandas`, basada en `numpy`, permite trabajar con estos objetos de manera potente y sencilla. Hemos elegido este formato porque más adelante es muy fácil recuperar la información y combinarla con la de otros `DataFrame` ya existentes. Nuestro `DataFrame` contiene dos columnas con las componentes X e Y generadas por t-SNE (indexadas como `'0'` y `'1'`) y una tercera columna con la *weirdness score* de cada muestra, indexada como `'W_S'`. Para indexar las filas utilizamos los identificadores que generamos en la etapa de carga de datos. La escritura a disco en sí se realiza en la función `save_results()` del fichero `file_manager.py`. Además de escribir a disco, esta función crea el directorio de salida `output` si no existe y compone el nombre del fichero de salida, indicando un prefijo, en nuestro caso `'t-SNE'`, el número de muestras y la fecha y hora de creación del fichero. De esta manera podemos distinguir los diferentes ficheros generados por su nombre.

Sistema de *log*

La función principal cuenta con un sistema de *log* que va dejando un registro de las acciones que tienen lugar durante la ejecución del programa. Este sistema usa la librería de python `logging` y escribe sus mensajes en el fichero `log/outlier_detection_log.txt`. Junto a cada mensaje se guarda la fecha y hora del evento que se registra. El sistema permite monitorizar la ejecución del programa, lo cual es útil en un programa que tarda mucho tiempo en terminar, como es nuestro caso. También permite analizar los tiempos de ejecución de cada parte parte del algoritmo así como comparar distintas ejecuciones entre sí. Los eventos que se registran son:

- El inicio de la ejecución.
- La carga de la matriz de características.
- La generación de datos sintéticos.
- El ajuste del modelo Random Forest.
- El cálculo de la matriz de distancias y del *weirdness score*.
  - Cada 100 filas calculadas de la matriz de similaridades se crea un registro.
- El cálculo del t-SNE.
- El final de la ejecución.

2.2.7. *Hardware* utilizado

Para la realización de este proyecto se ha trabajo en un equipo local y en el computador Divan del IAC. Las especificaciones del equipo local fueron:

- Procesador AMD Ryzen 5 1600, con 6 núcleos y 12 hilos a 3.20 GHz.
- 16 GB de memoria RAM
- Sistema operativo Windows 10 de 64 bits.

Las especificaciones del computador fueron:

- 44 procesadores Intel Xeon CPU E5-2699 v4 a 2.20GHz
- 4 TB de memoria RAM
- Sistema operativo Linux Fedora 26

2.2.8. *Software* utilizado

El software empleado en las distintas etapas fue:

- Anaconda, distribución de python para ciencia de datos y aprendizaje automático. Que incluye:
  - El intérprete del lenguaje python en su versión 3.6
  - El entorno de desarrollo integrado (IDE) Spyder, con el que se ha desarrollado el código.

  - El servicio Jupyter Notebook[21], con el que se ha podido probar el código incluido en distintos papers en formato `.ipynb`
- `git`, software de control de versiones.
- Tableau Desktop, una plataforma de análisis de datos que hemos utilizado para generar las visualizaciones.
- Para la redacción de la memoria:
  - TeXstudio, editor para el lenguaje LaTeX.
  - MikTex, gestor de paquetes para el lenguaje LaTeX.
- Para la interacción con el computador Divan:
  - El comando `ssh`, para conectar remotamente.
  - El comando `scp`, para enviar y descargar ficheros del computador.
  - El comando `nohup`, para ejecutar nuestro programa en segundo plano y que no se interrumpa si se cierra la sesión de `ssh`

3

Resultados y conclusiones

3.1. Resultados

Una vez generados los datos de t-SNE y de *weirdness score* vamos a visualizarlos de distintas maneras para ver si podemos entenderlos mejor y extraer alguna conclusión de ellos. Para las visualizaciones hemos utilizado el software Tableau, que nos permite generar todo tipo de gráficas interactivas a partir de nuestros datos. Todas las gráficas parten de dos ejecuciones del programa con 20000 muestras, la primera incluyendo las estrellas de especiel interés y la segunda excluyéndolas.

3.1.1. Estrellas con mayor *weirdness score*

A continuación podemos ver una tabla (3.1) con las 10 estrellas de mayor *weirdness score*. Podemos ver que el programa genera valores muy altos (cercanos a 1.0) para algunas estrellas.

3.1.2. *Weirdness score* de las estrellas de especial interés

En cuanto a las estrellas de especial interés para los investigadores del IAC, podemos ver que, al menos algunas de ellas, tienen *weirdness score* muy altos, en el top 100 de estrellas con mayor *weirdness score*. Esto es muy interesante, ya que confirma la sospecha por parte de los investigadores que estas estrellas en particular son diferentes al resto en alguna medida. Mostramos las 20 estrellas con mayor *weirdness score* de entre el conjunto de estrellas de especial interés, junto con su ID, posición en la lista ordenada de estrellas con mayor *weirdness score* y origen (3.2). Por origen entendemos el fichero en el que se ubicó la estrella originalmente, que de alguna manera las clasifica. Si hay elementos repetidos es porque algunas estrellas tenían más de un fichero de origen.

ID	W_S
2M18424477+4357001	0.998136
AP17440158-2451474	0.997948
2M23001597+5824024	0.997902
AP00431542+4111250	0.997861
2M23144932+6406504	0.997847
AP17452860-2440518	0.997825
2M22573253+5926004	0.997760
2M12482669+1102447	0.997699
2M21284509+1330424	0.997607
2M12560123+5816138	0.997460

Tabla 3.1. Estrellas con mayor *weirdness score* de una muestra de 20000 estrellas.

3.1.3. Distribución de la *weirdness score*

Para hacernos una mejor idea de cómo se distribuye la *weirdness score*, decidimos construir un histograma de esta variable. Podemos ver que, aunque el grueso de los datos se sitúa entre 0,38 y 0,64 (de hecho la media es 0,51 y la desviación típica 0,13), hay una cantidad nada despreciable de muestras con una *weirdness score* muy alta.

En amarillo, sobre las baras del histograma 3.1, podemos ver la proporción de estrellas de especial interés en relación al total para un cierto intervalo de *weirdness score*. Se confirma lo que ya sospechábamos al visualizar los datos tabulados: las estrellas de especial interés tienen una *weirdness score* alta.

3.1.4. Datos de reducción dimensional frente a *weirdness score*

El siguiente paso que nos interesa dar es cruzar los datos de la reducción dimensional generados con t-SNE con los valores de la *weirdness score*. Para esto hemos diseñado una serie de diagramas de dispersión en Tableau, en los que el eje horizontal lo ocupa la componente X calculada por t-SNE, el eje vertical lo ocupa la componente Y y el color de cada punto representa su *weirdness score*. Cuanto más oscuro el punto, mayor es este valor (ver figura 3.2).

Podemos ver que la *weirdness score* es mayor según nos vamos desplazando a la izquierda en el eje de las X y lejos del origen en el eje de las Y. Además, vemos cuatro grupos de puntos que están situados lejos de la masa principal y tienen *weirdness score* alta. Estos resultados son consistentes con nuestra definición de *weirdness score*: es normal que los puntos con *weirdness score* más alta estén en los extremos de la masa principal (aunque no todos los extremos tengan *weirdness score* alta), o separados de ella, y cercanos unos de otros.

ID	W_S	ranking	origin
2M17443011-2901143	0.993723	65	bulge
2M17443011-2901143	0.993723	65	lindqvist
2M17443011-2901143	0.993723	66	bulge
2M17443011-2901143	0.993723	66	lindqvist
2M17450714-2850268	0.993300	70	bulge
2M17463122-2839486	0.991571	82	bulge
2M17463122-2839486	0.991571	81	lindqvist
2M17463122-2839486	0.991571	81	bulge
2M17463122-2839486	0.991571	82	lindqvist
2M17454533-2853460	0.990857	88	bulge
2M17445959-2911152	0.989664	96	bulge
2M03244820+6300289	0.989250	97	carbon
2M17460160-2850006	0.989099	98	lindqvist
2M17460160-2850006	0.989099	100	bulge
2M17460160-2850006	0.989099	98	bulge
2M17460160-2850006	0.989099	100	lindqvist
2M17460160-2850006	0.989099	99	lindqvist
2M17460160-2850006	0.989099	99	bulge
2M00261363+5742167	0.988861	104	carbon
2M17445683-2913255	0.988853	105	lindqvist

Tabla 3.2. Las 20 estrellas de especial interés con *weirdness score* más alta considerando una muestra de 20000 estrellas.

Decidimos filtrar sobre la gráfica de dispersión los puntos que se corresponden con estrellas de especial interés. Podemos ver el resultado a continuación (ver figura 3.3).

Podemos observar que todos los grupos de puntos separados de la masa principal se corresponden con estrellas de especial interés y que dentro de la masa principal, la mayoría de estrellas de especial interés se sitúa en la región con mayor *weirdness score*.

El resto de las visualizaciones de diagramas de dispersión las vamos a hacer sin tener en cuenta las estrellas de especial interés, de manera que la gráfica no sea tan estrecha en el eje horizontal. Para esto fue necesario realizar una segunda ejecución del programa con 20000 muestras. Es por este motivo que la forma de la nube de puntos ha variado: son muestras diferentes y hay una cierta componente aleatoria en el cálculo del Random Forest y de t-SNE que hace que los resultados varíen.

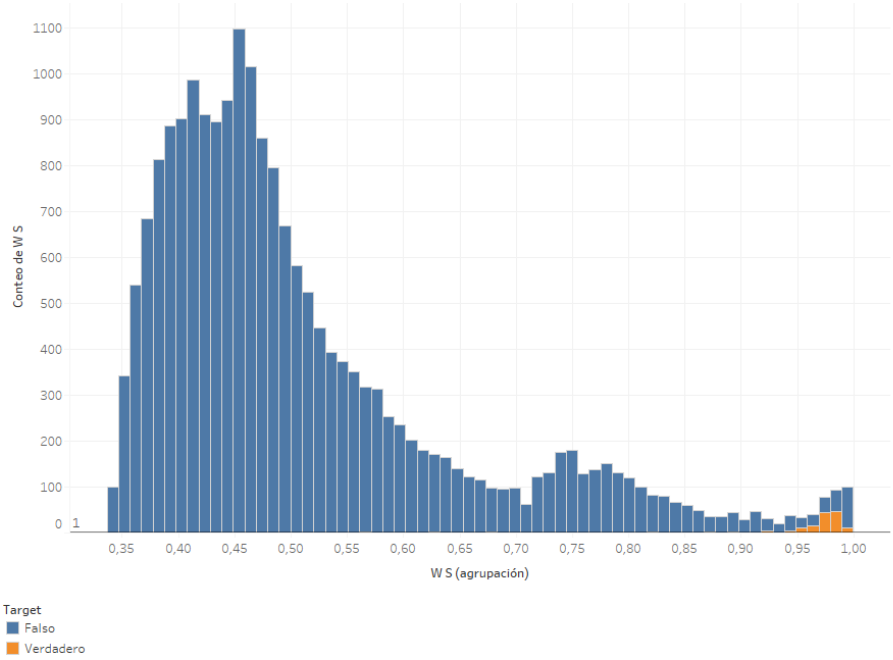

Figura 3.1. Histograma de la distribución de *weirdness score* sobre la muestra.

3.1.5. Visualización interactiva con Tableau

Para demostrar la potencia de Tableau como herramienta de visualización y análisis interactiva, queremos incluir dos diagramas de dispersión adicionales. Los dos están basados en los mismos datos que la imagen anterior.

En el primero (3.5), podemos ver que al pasar el cursor sobre un punto de la gráfica se despliega un listado con el detalle de los valores del punto: componentes X e Y, *weirdness score* y, sobre todo, su identificador. De esta manera los investigadores del IAC que usen la herramienta pueden identificar al momento los puntos con mayor *weirdness score*.

En el segundo(3.6) , podemos ver la gráfica ampliada en la región de mayor *weirdness score*. Además, hemos añadido a la visualización etiquetas con el identificador de cada punto.

3.1.6. Datos de reducción dimensional frente a otras magnitudes

Para muchos ficheros de APOGEE hay toda una serie de datos relativos a cada estrella que ya están calculados y que son de libre acceso. Estos datos son producto de otras investigaciones. Hemos decidido generar visualizaciones de

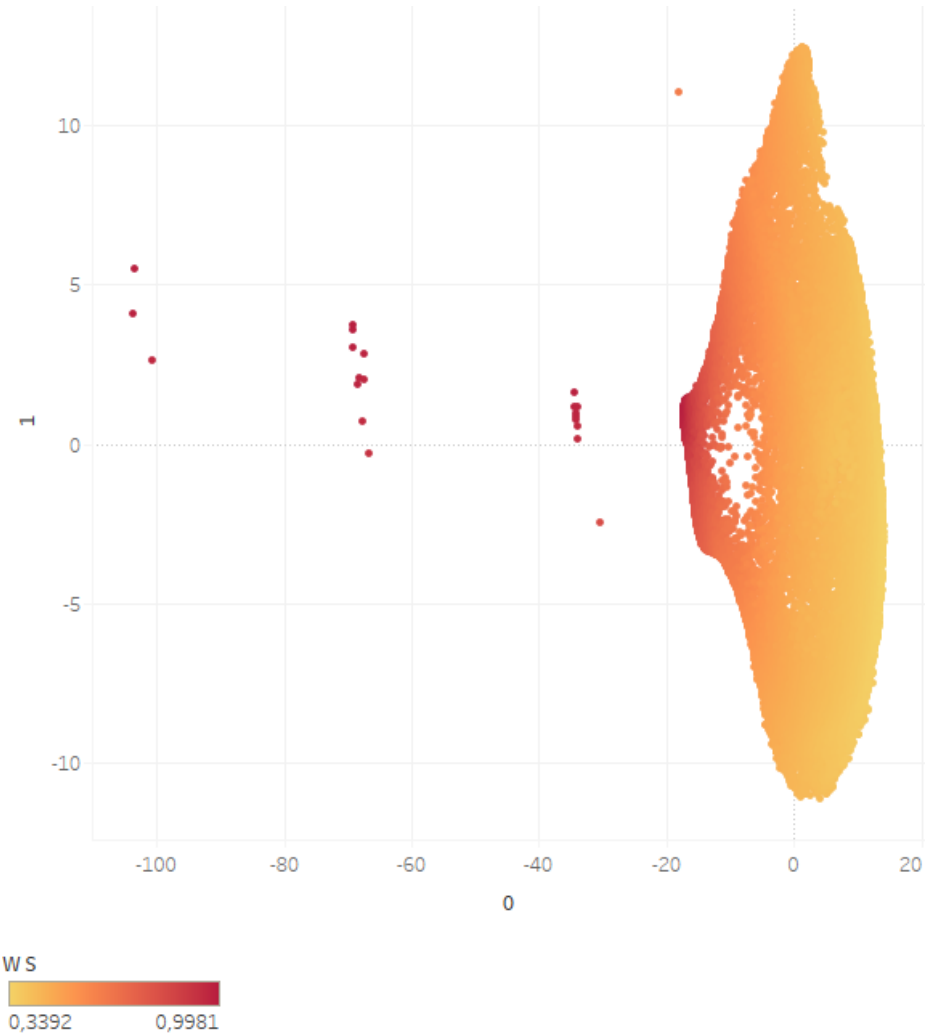

Figura 3.2. Datos de la reducción dimensional generados con t-SNE con los valores de la *weirdness score* considerando las estrellas de especial interés

estas magnitudes con Tableau de la misma manera que lo hicimos con la *weirdness score*: como diagramas de dispersión a partir de la reducción dimensional calculada por t-SNE. Nuestro propósito es encontrar relaciones entre ellas y la *weirdness score*. No nos corresponde en este proyecto intepretar las magnitudes en términos astrofísicos.

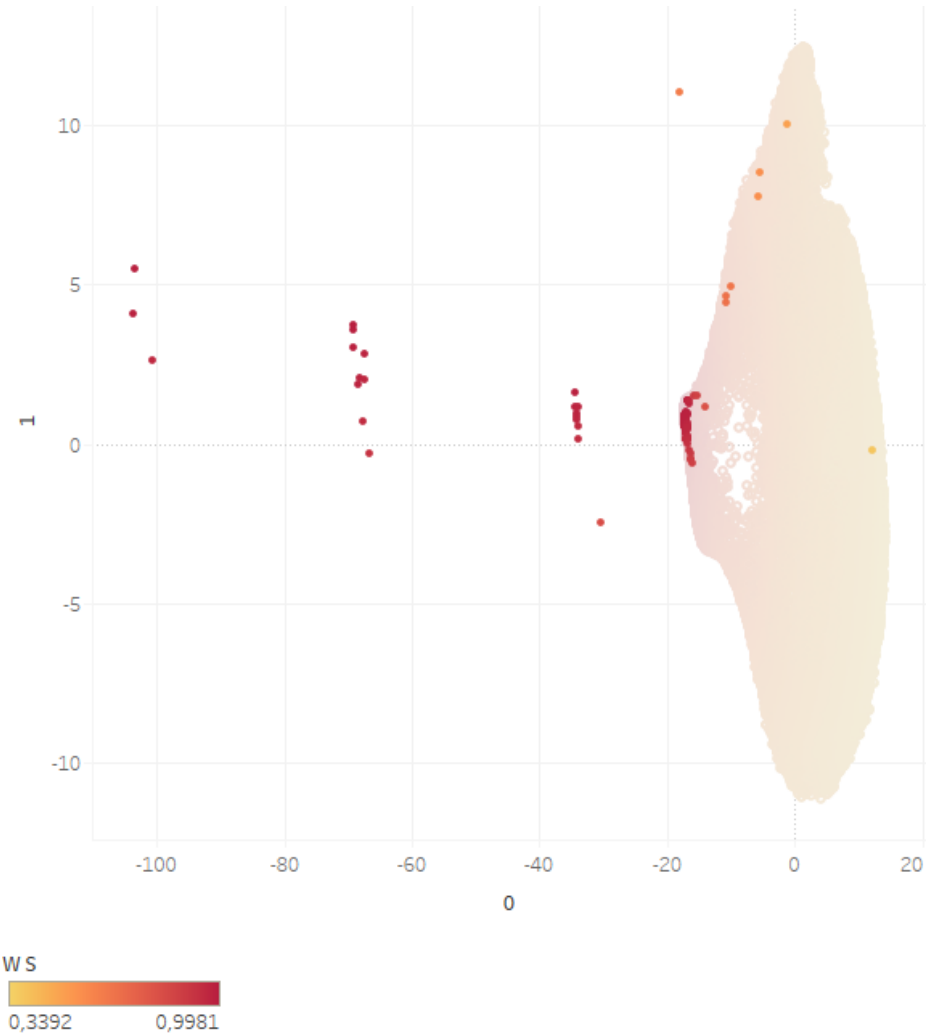

Figura 3.3. Gráfica de dispersión con los puntos que se corresponden con estrellas de especial interés.

Podemos ver que algunos puntos situados en los extremos de la figura 3.7, en particular en las regiones superior e inferior izquierda presentan valores extremos de la magnitud visualizada. Esto coincide con la localización de los puntos con mayor *weirdness score*. En este sentido podría haber algún tipo de correlación entre estas magnitudes y la *weirdness score*, lo que podría ser de interés en un futuro estudio.

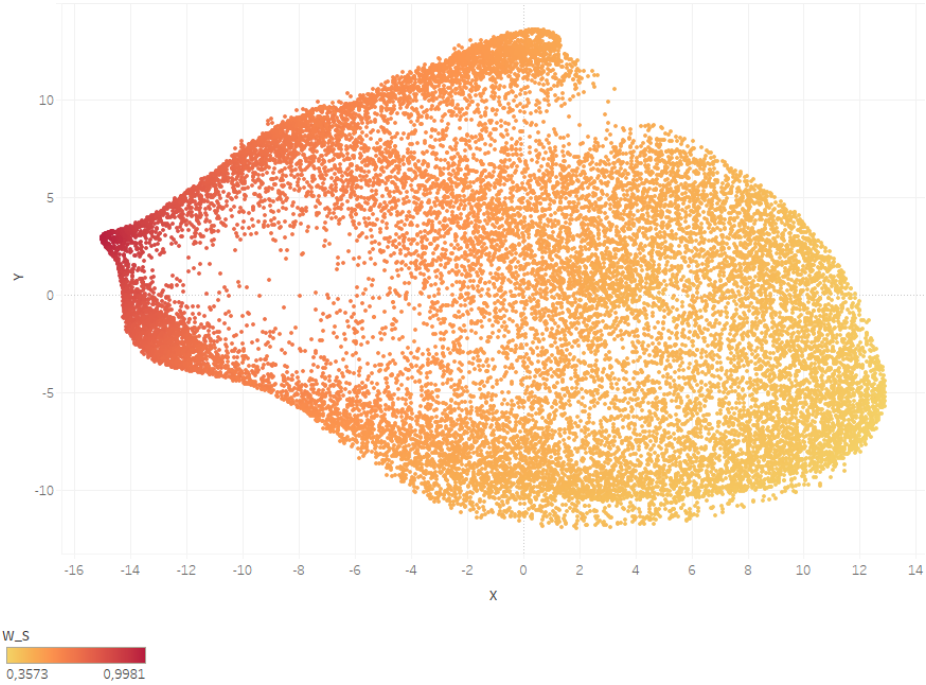

Figura 3.4. Un segundo diagrama de dispersión con 20000 muestras sin tener en cuenta las estrellas de especial interés.

3.2. Conclusiones

Este trabajo valida los resultados presentados en el artículo [14] de mayo de 2018 aplicando las técnicas propuestas a un nuevo conjunto de datos. También mejora el código del artículo [15] al resolver un problema con la memoria RAM que impedía la ejecución con grandes conjuntos de datos.

Por otro lado, se ha desarrollado una herramienta basada en aprendizaje automático que permitirá a los investigadores del IAC detectar valores atípicos en datos de espectros estelares, no necesariamente pertenecientes al conjunto APOGEE. También se ha validado la hipótesis de que una serie de estrellas de especial interés dentro de este conjunto debían presentar valores altos de *weirdness score*.

Las técnicas desarrolladas son aplicables a otras disciplinas científicas y otros conjuntos de datos, siempre y cuando se nos presente un problema de detección de valores atípicos en un conjunto de datos con muchas características, en donde sea interesante aplicar una reducción dimensional.

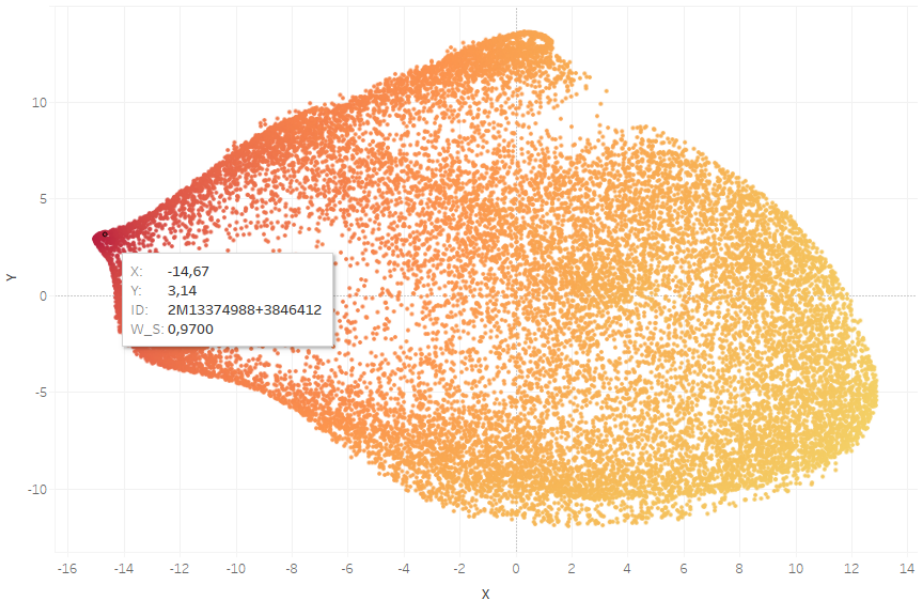

Figura 3.5. Al pasar el cursor sobre un punto de la gráfica se despliega un listado con el detalle de los valores del punto: componentes X e Y, *weirdness score* y, sobre todo, su identificador.

3.3. Futuros trabajos

El trabajo se puede ampliar en distintos sentidos:

- Calculando el t-SNE para todo el conjunto de datos APOGEE, de 249179 elementos. En nuestro caso, las pruebas con número de muestras más grandes que hemos podido realizar contaban con 20000 espectros. Intentamos una ejecución con 50000 muestras y a las 32 horas todavía no había acabado, aunque sí había calculado la matriz de distancias.
- Usando otros métodos de aprendizaje no supervisado para generar la matriz de distancias, posiblemente más eficientes o que arrojen mejores resultados.
- En el mismo sentido, usando otras técnicas de reducción dimensional diferentes a t-SNE. La validación de qué técnica es mejor vendrá dada a posteriori según cuál permita identificar mejor los valores atípicos.
- Corrigiendo posibles ineficiencias en el cálculo de la matriz de similaridades. Tal vez sea posible mejorar el rendimiento de esta parte del algoritmo haciendo un mejor uso de las operaciones matriciales de `numpy`. También es posible que se pueda mejorar el rendimiento utilizando paralelización.

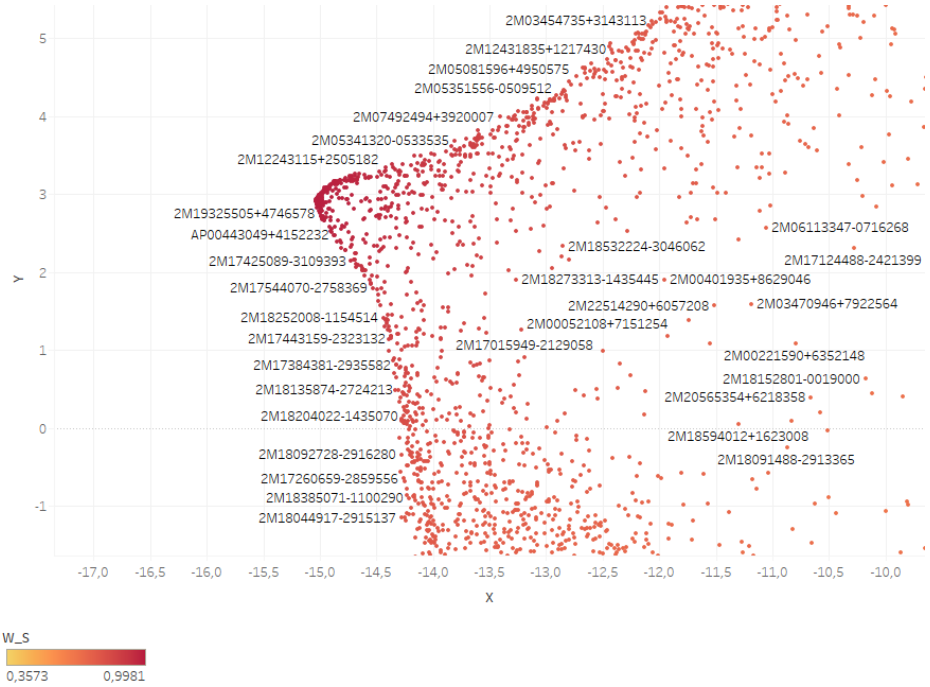

Figura 3.6. Gráfica ampliada en la región de mayor *weirdness score* y etiquetas con el identificador de cada punto.

- Explorando las posibles correlaciones entre la *weirdness score* calculada en este proyecto y las otras magnitudes ya disponibles para el conjunto APOGEE.

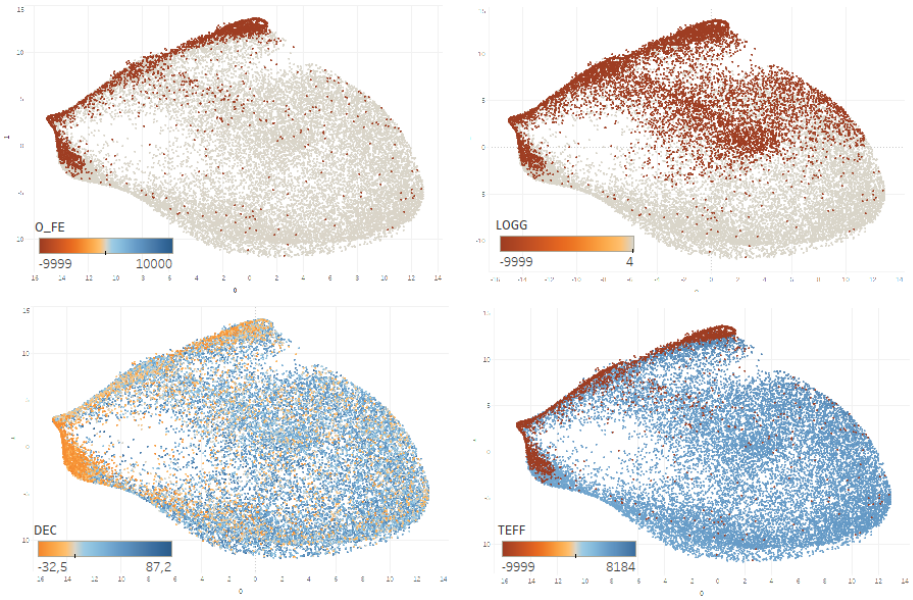

Figura 3.7. Otras magnitudes.

A

Código del proyecto

```
"""
execute_outlier_detection.py
"""
import outlier_detection

all_samples = 249179
outlier_detection.outlier_detection(20000, 44, 2000)
```

```
"""
outlier_detection.py
"""
import logging
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from   sklearn.manifold import TSNE

#Project modules
import config_manager
import file_manager
import common
import synthetic_data
import merge_and_assign_class
import similarity_matrix

def outlier_detection(n_samples, n_jobs, n_estimators):

    if n_samples < 119:
        raise Exception('El numero de muestras no puede ser menor que 119')

    logging.basicConfig(filename='log/outlier_detection_log.txt', \
        level=logging.INFO, \
        format='%(asctime)s %(levelname)s:%(message)s')

    # Setup logging and config
    logging.info('-------------------------------------------------------')
    config = config_manager.generate_config()
```

```
30 |     logging.info('A new execution has begun, working with %d samples and %d jobs',\
31 |         n_samples, n_jobs)
32 |
33 |     # Load data
34 |     (X, ids) = common.load_feature_matrix_random(config['fits_directory'],\
35 |         n_samples)
36 |     logging.info('Feature matrix has been loaded')
37 |
38 |     # Generating synthetic data based in the marginal distribution
39 |     X_syn = synthetic_data.get_synthetic_data(X)
40 |     # Merge X and X_syn into one sample with their assigned classes
41 |     X_total, Y_total = merge_and_assign_class.merge_data_and_synthetic_samples(X, X_syn)
42 |     logging.info('Synthetic data generated')
43 |
44 |     # Random forrest training (RF)
45 |     rand_f = RandomForestClassifier(n_estimators=n_estimators, n_jobs = n_jobs)
46 |     rand_f.fit(X_total, Y_total)
47 |     logging.info('Random Forest model was fitted')
48 |
49 |     # Building similarity matrix and building distances matrix
50 |     sim_mat = similarity_matrix.build_similarity_matrix(rand_f, X)
51 |     dis_mat = 1 - sim_mat
52 |     # Sum matrix to get outlier score (weirdness score)
53 |     sum_vec = np.sum(dis_mat, axis=1)
54 |     sum_vec /= float(len(sum_vec))
55 |     logging.info('Similarity matrix and weirdness score computed')
56 |
57 |     # Dimentional reduction with t-Sne
58 |     tsne = TSNE(n_components=2, verbose=1, learning_rate=1000, perplexity=2000)
59 |     tsne_results = tsne.fit_transform(dis_mat)
60 |     logging.info('TSN2 computed')
61 |
62 |     # Save results
63 |     df_tsne = pd.DataFrame(tsne_results, index=ids)
64 |     df_tsne['W_S'] = pd.Series(sum_vec, index=ids)
65 |     file_manager.save_results(df_tsne, n_samples, \
66 |         'tsne_estimators_' + str(n_estimators) + '_')
67 |
68 |     logging.info('The program has finished')
```

```
1 |"""
2 |config_manager.py
3 |"""
4 |import json
5 |
6 |def generate_config():
7 |    with open('config/config.json') as f:
8 |        return json.load(f)
```

```
1 |"""
2 |synthetic_data.py
3 |"""
4 |import numpy as np
5 |
6 |def get_synthetic_data(X):
```

```
    features = X.shape[1]
    X_syn = np.zeros(X.shape)

    for i in range(features):
        obs_vec = X[:,i]
        syn_vec = np.random.choice(obs_vec, len(obs_vec))
        X_syn[:,i] += syn_vec

    return X_syn
```

```
"""
merge_and_assign_class.py
"""
import numpy as np

def merge_data_and_synthetic_samples(X, X_syn):

    # build the labels vector
    Y = np.ones(len(X))
    Y_syn = np.ones(len(X_syn)) * 2

    Y_total = np.concatenate((Y, Y_syn))
    X_total = np.concatenate((X, X_syn))

    return X_total, Y_total
```

```
"""
similarity_matrix.py
"""
import numpy as np
import logging

# Builds the similarity matrix based on the feature matrix X
# based on already trained random forest.
def build_similarity_matrix(rand_f, X):
    # apply to get the leaf indices
    apply_mat = rand_f.apply(X)
    # find the predictions of the sample
    is_good_matrix = np.zeros(apply_mat.shape)

    for i, est in enumerate(rand_f.estimators_):
        d = est.predict_proba(X)[:, 0] == 1
        is_good_matrix[:, i] = d

    # mark leaves that make the wrong prediction as -1,
    # in order to remove them from the distance measurement
    apply_mat[is_good_matrix == False] = -1

    # now calculate the similarity matrix
    numerator = compute_numerator_numpy(apply_mat)
    denominator = np.asfarray(np.sum([apply_mat != -1], axis=2), dtype='float')
    sim_mat = numerator / denominator
    return sim_mat

def compute_numerator(apply_mat):
```

```
30 |     n_samples = apply_mat.shape[0]
31 |     numerator = np.zeros([n_samples, n_samples])
32 |     for i in range(n_samples):
33 |         for j in range(i+1):
34 |             numerator[i,j] = count_same_prediction(apply_mat[i], apply_mat[j])
35 |             numerator[j,i] = numerator[i,j]
36 |     return numerator
37 |
38 | def count_same_prediction(row_a, row_b):
39 |     return np.sum((row_a != -1) & (row_b != -1) & (row_a == row_b))
40 |
41 | def compute_numerator_numpy(apply_mat):
42 |     n_samples = apply_mat.shape[0]
43 |     numerator = np.zeros([n_samples, n_samples])
44 |     exclusions = np.ones(apply_mat.shape[1]) * -1
45 |     for i in range(n_samples):
46 |         numerator[i] = np.sum((apply_mat == apply_mat[i])
47 |             & (apply_mat != exclusions), axis=1)
48 |         if i % 100 == 0:
49 |             logging.info('Row ' + str(i) + ' computed')
50 |     return numerator
```

```
1 | """
2 | file_manager.py
3 | """
4 | import os
5 | import time
6 |
7 | def save_results(df, n_samples, prefix):
8 |     timestr = time.strftime("%Y%m%d_%H%M%S")
9 |     output_dir = 'output'
10 |     if not os.path.exists(output_dir):
11 |         os.mkdir(output_dir)
12 |     df.to_csv(os.path.join(output_dir, \
13 |         prefix + '_' + str(n_samples) + '_samples_' + timestr + '.csv'),\
14 |         sep=';')
```

Bibliografía

[1] Samuel A. L. (1959). Some Studies in Machine Learning Using The Game of Checkers. *IBM Journal of Research and Development. 3(3), 210–229.* Recuperado de http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5392560&isnumber=5392559

[2] Akinfaderin W. (24 de marzo, 2017). The Mathematics of Machine Learning. *Towards Data Science.* recuperado de https://towardsdatascience.com/the-mathematics-of-machine-learning-894f046c568

[3] Santoyo S. (12 de septiembre, 2017). A Brief Overview of Outlier Detection Techniques. *Towards Data Science.* recuperado de https://towardsdatascience.com/a-brief-overview-of-outlier-detection-techniques-1e0b2c19e561

[4] Aggarwal C. C. (2016). *Outlier Analysis (segunda edición).* Nueva York,Estados Unidos: Springer International Publishing.

[5] González A. (30 de julio, 2014). Conceptos básicos de Machine Learning. *Clever Data.* recuperado de https://cleverdata.io/conceptos-basicos-machine-learning/

[6] Breiman L. (2001). Random Forests. *Machine Learning. 45(1), 5.* Recuperado de https://link.springer.com/article/10.1023/A:1010933404324

[7] Mitchell L., Sloan T.M, Mewissen M., Ghazal P., Forster T., Piotrowski M., Trew A. (25 de marzo, 2014). Parallel classification and feature selection in microarray data using SPRINT. *PMC.* Recuperado de https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4038771/

[8] Louppe G. (28 de julio, 2014). Understanding Random Forests: From Theory to Practice. *Cornell University Library.* Recuperado de

https://arxiv.org/abs/1407.7502

[9] Gray G. R., Aljabar P., Heckemann R. A., Hammers A., Rueckert D. (15 de enero, 2014). Random forest-based similarity measures for multi-modal classification of Alzheimer?s disease. *PMC.* Recuperado de https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3516432/

[10] Ray S. (28 de julio, 2015). Beginners Guide To Learn Dimension Reduction Techniques. *Analytics Vidhya.* Recuperado de https://www.analyticsvidhya.com/blog/2015/07/dimension-reduction-methods/

[11] Reid S. (october, 2014). *The Curse of Dimensionality [Figura].* Recuperado de http://www.turingfinance.com/artificial-intelligence-and-statistics-principal-component-analysis

[12] van der Maaten L.J.P.; Hinton G.E. (2008). Visualizing Data Using t-SNE. *Journal of Machine Learning Research. 9: 2579–2605.* Recuperado de http://www.jmlr.org/papers/volume9/vandermaaten08a

[13] Hinton G.E., Roweis S.T. (2003). *Advances in Neural Information Processing Systems. 15: 833–840* Cambridge, Massachusetts, Estados Unidos: MIT press.

[14] Reis I., Poznanski D., Baron D., Zasowski G., Shahaf A. (31 de ocutubre, 2017. Ultima revisión 28 de mayo 2018). Detecting outliers and learning complex structures with large spectroscopic surveys - a case study with APOGEE stars. *Cornell University Library.* Recuperado de https://arxiv.org/abs/1711.00022

[15] Poznanski D., Baron D. (22 de noviembre, 2016). The weirdest SDSS galaxies: results from an outlier detection algorithm. *Cornell University Library.* Recuperado de https://arxiv.org/abs/1611.07526

[16] Ho T.K. (1995). Random Decision Forest. *Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal. 14?16. 278?282.* Recuperado de https://web.archive.org/web/20160417030218/http://ect.bell-labs.com/who/tkh/publications/papers/odt.pdf

[17] Breiman L. (1996). Bagging predictors. *Machine Learning. 24(2), 123–140.* Recuperado de https://doi.org/10.1007/BF00058655

[18] Apache Point Observatory Galactic Evolution Experiment. APOGEE Survey and Instruments Overview. *SDSS.* Recuperado de https://www.sdss.org/dr14/irspec/

[19] Git. *Gast version control.*Recuperado de https://git-scm.com/

[20] Anaconda. *Anaconda cloud.*Recuperado de https://anaconda.org/

[21] Jupiter. *Jupiter Notebook*.Recuperado de http://jupyter.org/
[22] ipython. *ipython*.Recuperado de https://ipython.org/
[23] Tableau. *Tableu*.Recuperado de https://www.tableau.com/
[24] Astropedia. *Velocidad radial*.Recuperado de http://astronomia.wikia.com/wiki/Velocidad_radial
[25] Astropedia. *Estrella binaria*.Recuperado de http://astronomia.wikia.com/wiki/Estrella_binaria
[26] Astropedia. *espectroescopia infrarroja*.Recuperado de http://astronomia.wikia.com/wiki/
[27] Wikipedia. *Overfitting*.Recuperado de https://en.wikipedia.org/wiki/Overfitting
http://jupyter.org/

Lista de Figuras

Exploring large spectroscopic surveys using machine learning methods

Lára Kristjánsdóttir

Facultad de Ciencias · Sección de Matemáticas
Universidad de La Laguna
alu0100758795@ull.edu.es

Abstract

In this work we will study and apply machine learning methods for detecting outliers and visualizing structure in a large dataset. We will first apply an algorithm on stellar spectra from the APOGEE survey and then use it to build a matrix containing a pair-wise similarity measure between every two objects in the data. Using dimesionality reduction techniques to explore the matrix (or its inverse, the distance matrix) will give us an inside of the importance of similarity measures. After applying the algorithms, we will discuss the issues and limitations when calculating a similarity matrix in a large set of data. Lastly, we will propose ideas on future work, mostly on how to continue and improve the project. This project is based on the following papers: [1] y [2].

1. Introduction

Machine learning, a name introduced by Arthur Samuel[3] in 1959, is a field of computer science that uses statistical techniques to give a system the ability to "learn" with data and without being explicity programmed. Machine learning is a way to bring mathematics to practice and in this work we will mostly see connections with fundamental statistics and probability theory. Having basic knowledge in optimization will also help us understand the efficiency and scalability of the algorithms. The work consists of an introduction, three chapters and an appendix. In the first chapter, we will go over the theoratical studies of the algorithms that we will later use to explore a given dataset. In the second chapter we will introduce the dataset that is going to be used during the project and then go on to describe installation and necessary tools. After this we will go over how the project works, step by step. In chapter three we will discuss the results with text and visuals. Next we will present conclusions and suggest future work for improving the project. Lastly, in the appendix we will have all the code we used for this work.

2. Theoratical studies

In this chapter we start by defining what is an outlier and then we go on to describe a method, called Random Forest, that we use to detect those outliers. Random Forest is a machine learning algorithm used mostly for classification and regression. We will describe how it works and how we use it to build a similarity matrix for detecting outliers.

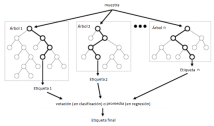

Figure 1: *Architecture of the random forest model*

Next, we discuss the problems with high dimensional data in how dimensional reduction can be of help. Dimensionality reduction is a technique used to reduce the dimensions losing the minimum amount of information. Now, we introduce t-SNE, which is a dimensonal reduction method, and see how it can help us visualize the data.

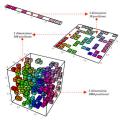

Figure 2: *Explaining dimensional reduction in a simple way. [5]*

3. Applying Random Forest and t-SNE

In this chapter we start by describing the dataset on which the we apply the algorithms. The dataset we use is stellar spectra collected by the *Apache Point Observatory Galactic Evolution Experiment*[6] survey. Next we go over the installation and tools used and need for completing the project. We finish this chapter by explaning with detail every step of the code.

4. Discussing results

In this third and last chapter, we describe the results using graphs and maps. We will discuss stills of interactive graphs realized in Tableau[7] where we explore the similarity matrix and see the outliers. In the end we will have a finished tool ready for use an we finish of the chapter by suggesting improvements.

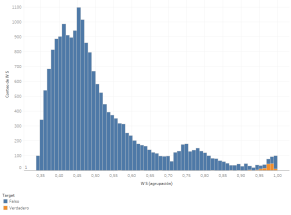

Figure 3: *Histograma on distribution on wierdness score.*

References

[1] Reis I., Poznanski D., Baron D., Zasowski G., Shahaf A. (31 de ocutubre, 2017. Ultima revisiń 28 de mayo 2018). Detecting outliers and learning complex structures with large spectroscopic surveys – a case study with APOGEE stars. *Cornell University Library.*

[2] Poznanski D., Baron D. (22 de noviembre, 2016). The weirdest SDSS galaxies: results from an outlier detection algorithm. *Cornell University Library.*

[3] Samuel A. L. (1959). Some Studies in Machine Learning Using The Game of Checkers. *IBM Journal of Research and Development. 3(3), 210–229.*

[4] Breiman L. (2001). Random Forests. *Machine Learning. 45(1), 5.*

[5] Reid S. (october, 2014). *The Curse of Dimensionality [Figura].*

[6] Apache Point Observatory Galactic Evolution Experiment. APOGEE Survey and Instruments Overview. *SDSS.*

[7] Tableau. *Tableu.*