Aprendizaje automático

Proyecto final

Johanna Capote Robayna
Guillermo Galindo Ortuño
5 del Doble Grado en Informática y Matemáticas
Grupo A

UNIVERSIDAD
DE GRANADA

Índice

1. Definición del problema a resolver y enfoque elegido

El problema que inicialmente se nos plantea es del estimar la popularidad de un artículo (medido como número de veces que este es compartido) basándonos en una serie de características de este, como por ejemplo la longitud o si trata de temas como tecnología, estilo de vida, etc.

Aunque lo natural sería haberlo plantearlo como un problema de regresión, en nuestro caso hemos decidido enfocarlo como un problema de clasificación binario. Esto lo hemos hecho para poder utilizar y analizar modelos de clasficación tal y como hemos estudiado, que creemos que será más interesante. Siguiendo las recomendaciones de los creadores de la base de datos, trataremos este problema como un problema de clasificación binaria, considerando todos aquellos valores del atributo objetivo menores o iguales que un umbral (1400 en particular) como una clase y los mayores como la otra. Esto podemos interpretarlo como que queremos conocer si un artículo será popular o no (supera o no el umbral de *shares*).

El *dataset* consta de 61 atributos, siendo dos de ellos no predictivos (*url* y *timedelta*) y otro distinto el objetivo. Por tanto nuestro vector de características tendrá será de tamaño 58. Formalmente:

- Nuestro espacio muestral será $\mathcal{X} = \mathbb{R}^{58}$.
- El espacio de etiquetas será $\mathcal{Y} : \{-1, 1\}$.
- Nuestro objetivo será encontrar $f : X \to Y$ que estime si un artículo será popular o no (1 ó -1).

2. Argumentos a favor de la elección de los modelos

Los modelos que estudiaremos en esta práctica son **Regresión Logística**, **Maquinas de Vectores de Soporte**, **RandomForest**. Como ya mencionamos anteriormente, nos enfrentamos a un problema de clasificación binaria.

Elegimos

3. Codificación de los datos de entrada par hacerlo útiles a los algoritmos

En priner lugar, tras quitar los atributos no predicitivos (`url` y `timedelta`),comprobamos que no hay valores perdidos y que ninguno sea *null*:

```
datos_perdidos = datos.columns[datos.isnull().any()]
datos_perdidos = datos.columns[datos.isna().any()]
```

A continuación dividimos el *dataset* en el conjunto de características y el conjunto de etiquetas. Y transformamos las etiquetas asignándole el valor -1 si la etiqueta tiene un valor menor que 1400 y asignándole el valor 1 en el otro caso.

```
datos_perdidos = datos.columns[datos.isnull().any()]
datos_perdidos = datos.columns[datos.isna().any()]
y = y.apply(lambda x: -1.0 if x < 1400 else 1.0)
```

Por último antes de pasar al preprocesado de los datos comprobamos que los valores se encuentran dentro del rango. El valor mínimo es $-1{,}0$ y el valor máximo es $843300{,}0$, por lo que no hay valores fuera de rango. Además comprobamos que las clases están balanceadas.

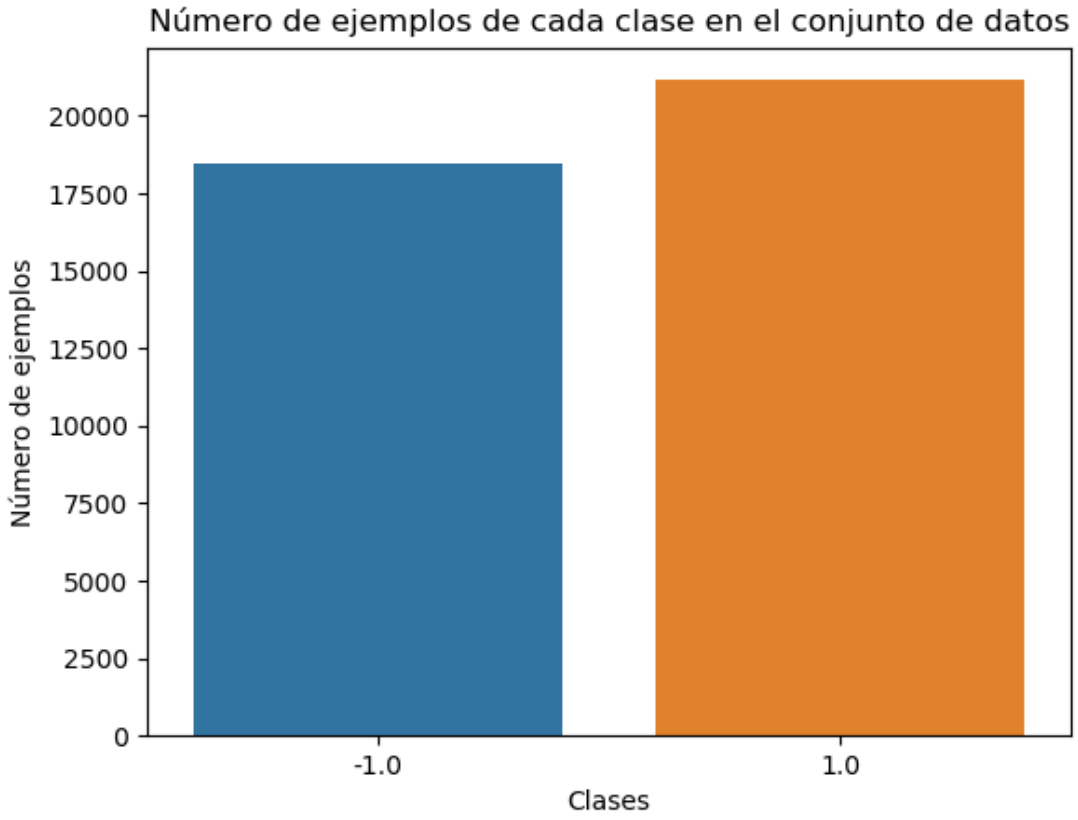

FIGURA 1: Gráfica que muestra el número de individuos de cada clase.

Dividimos el conjunto de datos en el conjunto de entrenamiento y el conjunto de test, para ello utilizamos la función `train_test_split()` de la librería *sklearn*. Elegimos que el conjunto de test tenga un tamaño del 20 %, medida estándar.

Para preprocesar los datos utilizamos una estructura `Pipeline` de sklearn para agrupar todas las transformaciones. Realizamos dos transformaciones de los datos:

1. En primer lugar utilizamos la transformación `StandardScaler()` para reescalar los atributos para evitar datos con distintas escalas. Tras estre reescalado los atributos tienen media 0 y varianza 1. Realizamos esta transformación ya que es altamente recomendable que se realice antes de entrenar los modelos que hemos elegido.

2. Además aplicamos el algoritmo PCA, con el conseguimos reducir la dimensiona-

lidad de las características. Se ha fijado que seleccione el número de componentes de modo que la cantidad de varianza que deba explicarse sea mayor del 95 %. Elegimos aplicar esta transformación puesto que la cantidad de atributos es considerablemente grande, con ella buscamos mejorar la eficiencia de los modelos y encontrar una base de coordenadas que sea más representativa (eliminando las correlaciones entre los atributos), hallando aquellas características con mayor relevancia.

Por lo que el Pipeline del preprocesador quedaría de la siguiente forma:

```
preprocesado = [("escalado", StandardScaler()),
                ("PCA", PCA(n_components=0.95))]

preprocesador = Pipeline(preprocesado)
```

Para analizar los logros obtenidos con el preprocesado de datos utilizamos una matriz de correlaciones con la que podemos observar que el preprocesador de datos ha mejorado la correlación de las características. En las siguientes imágenes podemos observar como se ha reduciendo a 35 las características y se han eliminado las correlaciones entre ellas.

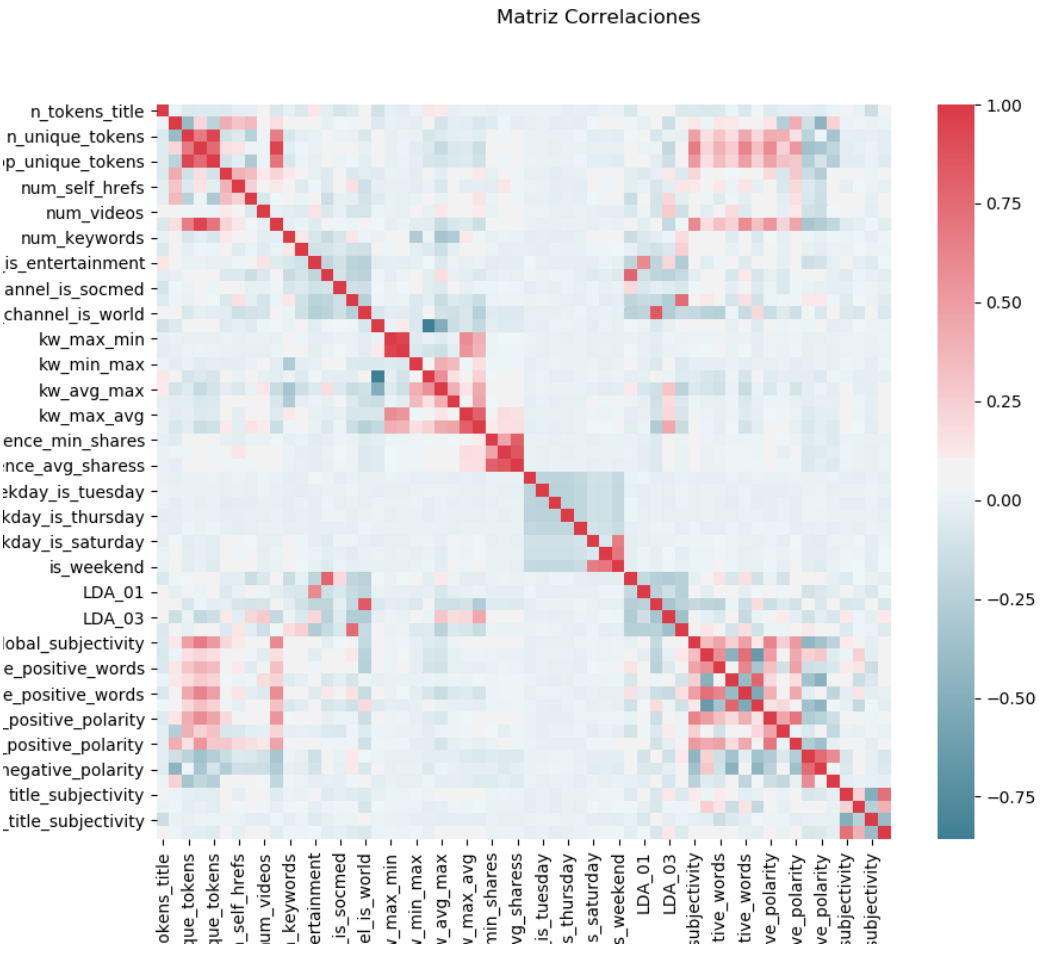

FIGURA 2: Matriz de correlaciones antes del preprocesador de datos.

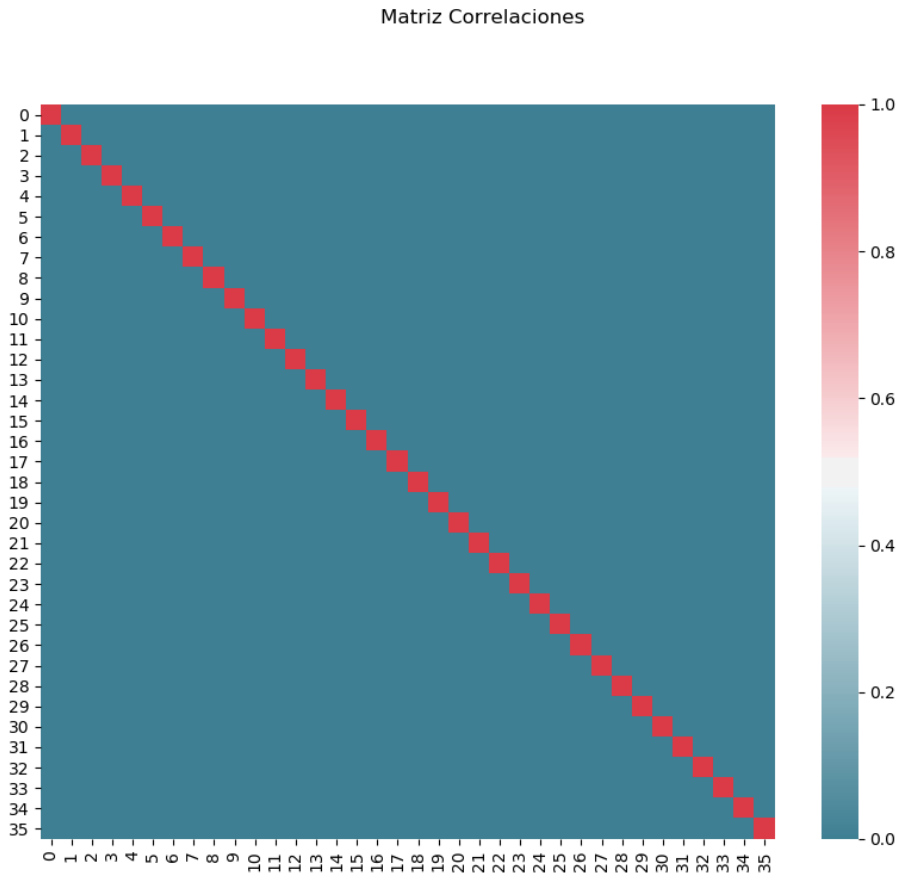

FIGURA 3: Matriz de correlaciones después del preprocesado de datos.

4. Valoración del interés de las variables para el problema y selección de un subconjunto

5. Normalización de las variables

6. Justificación de la función de pérdida usada

Como métrica de error utilizaremos el *accuracy*, la usual en este tipo de problemas. Esta medida expresa el error como un valor entre 0 y 1, siendo 0 cuando todos los

puntos están bien clasificados y 1 cuando están todos mal clasificados. Para calcularla, dado un $h \in H$ el error viene dado por:

$$E_{in}(h) = \frac{1}{N} \sum_{x_n \in X} [[h(x) \neq y_n]]$$

Para visualizar y analizar el error utilizamos la matriz de confusión, aunque está no es una medida métrica, la mayoría de métricas se basan en esta matriz. Esta matriz es un método visual en el que podemos ver el rendimiento de un modelo supervisado. Esta matriz muestra los falsos positivos y los verdaderos positivos.

7. Selección del modelo lineal paramétrico y valoración de su idoneidad frente a otras alternativas

Para seleccionar el mejor modelo utilizamos la función `GridSearchCV` la cual utiliza la técnica de *cross-validation* para entrenar y validar los distintos modelos. Esta función elabora un grid con todas las posibles combinaciones de los diccionarios sin mezclar entre ellos (cada estimador con sus parámetros),por lo que le pasamos el preprocesador y la lista con todos los modelos a probar. A continuación entrenamos el *grid* con la función `fit` y elegimos como nuestro clasificador final el mejor estimador, el cual será el que tenga mejor *accuracy*. Este estimador que nos devuelve el `GridSearchCV` ya está entrenado en todo el conjunto de entrenamiento por lo que no es necesario volverlo a entrenar.

```
grid = GridSearchCV(preprocesador, modelos, scoring='accuracy', cv=5,
                    n_jobs = -1)
grid.fit(X, y)
clasificador = grid.best_estimator_
```

Tras ejecutar esto con nuestra lista de modelos, obtenemos como resultado que el mejor clasificador es la combinación del estimador ?? y el parámetro ??.

8. Aplicación de técnicas

9. Función de regularización

Para evitar sobreajustes en el modelo debido a la alta dimensión de nuestro conjunto de datos introducimos técnicas de regularización, las cuales reducen la complejidad de modelo introduciendo un término en la función de coste. Es decir, la regularización reduce la varianza del modelo sin incrementar considerablemente el sesgo de este.

Dentro de todos los métodos de regularización, elegimos la Regularización de Ridge(L_2) ya que proporciona mejores resultados cuando la mayoría de los atributos son relevantes, como es nuestro caso.

Este método añade una penalización cuadrática en los pesos a la función de pérdida (L):

$$L_{(L_2)}(w) = L(w) + \lambda \|w\|_2^2$$

10. Valoración de los resultados

11. Justificación