**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 3

2 Argumentos a favor de la elección de los modelos 3

3 Tratamiento de los datos de entrada y preprocesado 4

4 Justificación de la función de pérdida usada 8

5 Selección del modelo lineal paramétrico y valoración de su idoneidad frente a otras alternativas 9

6 Aplicación de técnicas 9

7 Función de regularización 11

8 Valoración de los resultados 11

9 Justificación 11

# 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. Entre el resto de atributos nos encontramos 13 categóricos, los que son de la forma `<...>_is_<...>`, que se encuentran almacenados como 0 o 1.

Por tanto nuestro vector de características será real 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 (SVM)** y **RandomForest**. Como ya mencionamos anteriormente, nos enfrentamos a un problema de clasificación binaria.

Como nuestro problema es suficiente complejo, elegimos regresión logística como modelo lineal pues además está pensando para utilizarlo en problemas de clasificación, como es el caso. Además de este, elegimos otros dos modelos más complejos como son SVM y RandomForest, de los cuales sabemos que se adecuan bien a problemas de clasificación binaria como en el que nos encontramos.

# 3. Tratamiento de los datos de entrada y preprocesado

En primer lugar, tras eliminar los atributos no predicitivos (`url` y `timedelta`), comprobamos que no existan valores perdidos y ni nulos:

```
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 que nos indica en el archivo de información del conjunto de datos. 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.

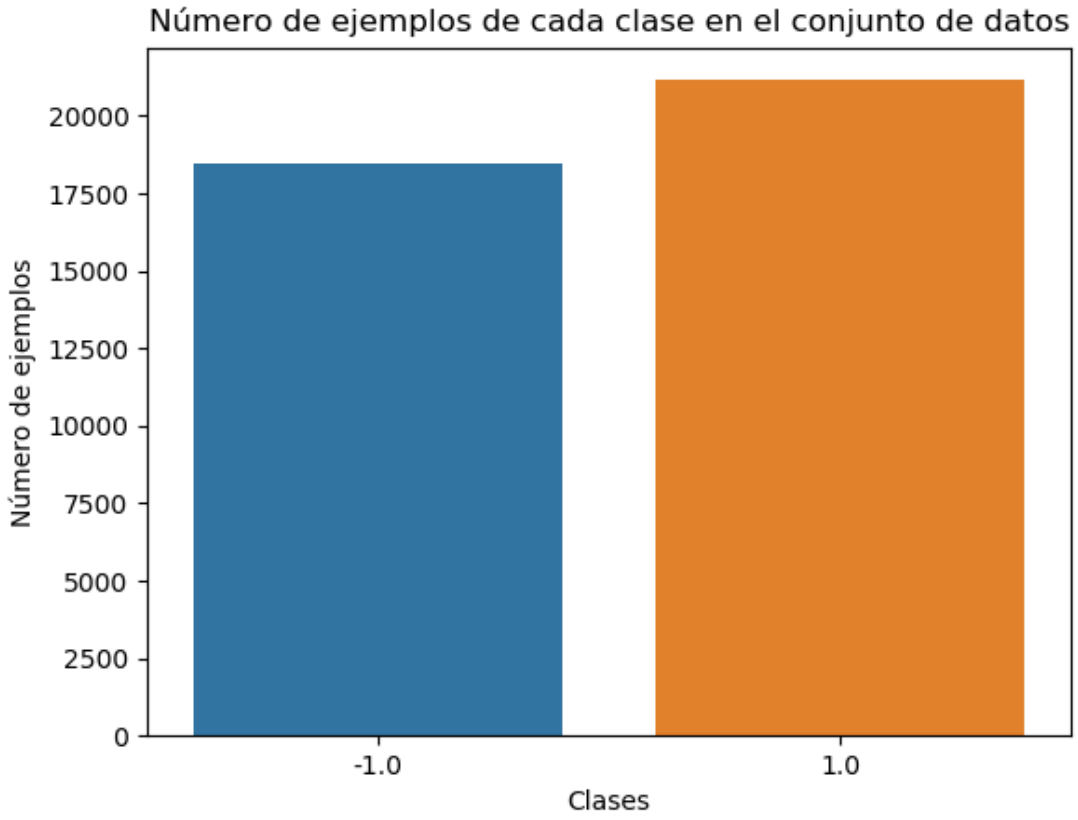


FIGURA 1: Gráfica que muestra el número de individuos de cada clase.

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. Aplicamos Análisis de Componentes Principales con el objetivo de reducir la dimensionalidad de las características. Debido a 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, al mismo tiempo que reducimos la correlación entre los atributos.

2. Tras esto, utilizamos la transformación `StandardScaler()` para reescalar los

atributos para evitar datos con distintas escalas. Tras este 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.

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

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

preprocesador = Pipeline(preprocesado)
```

Para analizar los logros obtenidos con el preprocesado de datos mostramos la matriz de correlaciones. En las siguientes imágenes podemos observar como se ha reduciendo a 35 las características y se han eliminado las correlaciones entre ellas, logrando los objetivos persiguidos.

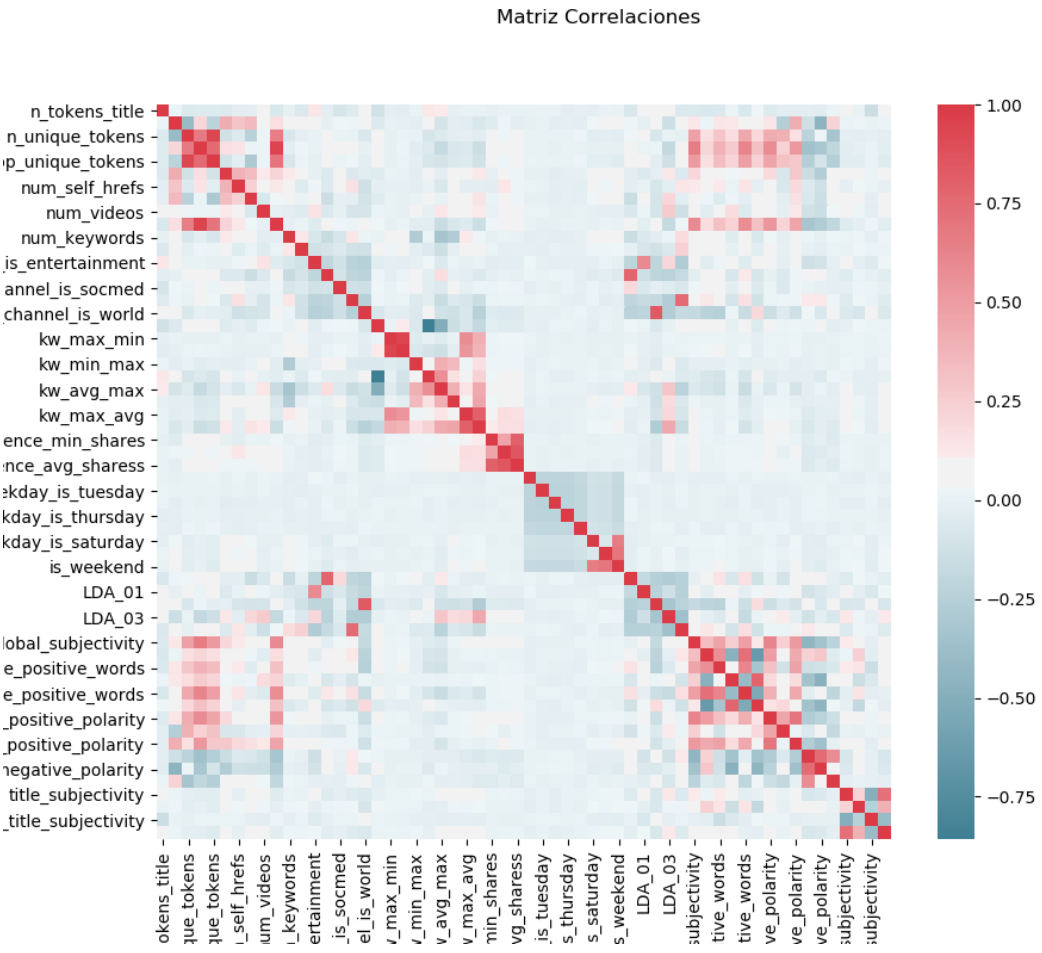


FIGURA 2: Matriz de correlaciones antes del preprocesador de datos.

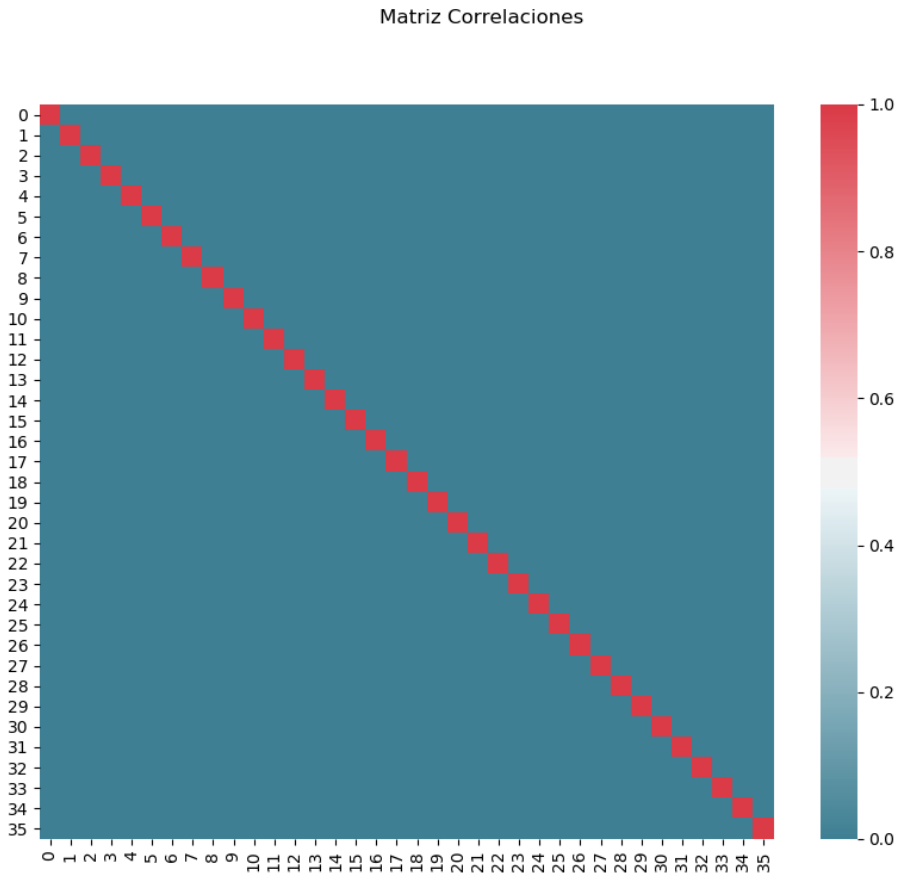


FIGURA 3: Matriz de correlaciones después del preprocesado de datos.

# 4. 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.

# 5. 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 ??.

# 6. Aplicación de técnicas

En primer lugar aclaramos que consideramos como modelos un estimador y un conjunto fijo de hiperparámetros, es decir cada modelos será un estimador y una combinación de sus hiperparámetros.

Los modelos elegidos están expresados en un diccionario, en el cual la parte `'clf'` hace referencia al estimador y `'clf__Parametro'` hace referencia a cada uno de los

hiperparámetros del estimador y los valores que toma para cada modelo. Los modelos elegidos son los siguientes:

- **Regresión logistica**. Elegimos este modelo porque suele obtener buenos resultados en problemas de clasificación multiclase. Elegimos como ya mencionamos anteriormente la regularización $L_2$, además indicamos que la estratégia de clasificación es la `'ovr'` (*one-vs-rest*) ya explicada en la clases de funciones que buscamos y por último elegimos un número máximo de iteraciones de 1000. Por otro lado hacemos variar el parámetro `C` entre los valores $\{2.0, 1.0, 0.1, 0.01, 0.001\}$, es decir finalmente habrán cinco modelos formados por el estimador y cada posible valor del parámetro `C`.

```
{'clf': [LogisticRegression(penalty='l2', # Regularización Ridge (L2)
               multi_class='ovr', # Indicamos que la regresión logística es
               solver = 'lbfgs', # Algoritmo a utilizar en el problema de
               # optimización
               max_iter = 1000)],
    'clf__C':[2.0, 1.0, 0.1, 0.01, 0.001]}
```

- **SVC**. Elegimos este modelo porque se adapta bien a los problemas de clasificación binaria. Fijamos el kernel `'rbf'` como se recomienda en el guión y establecemos que las clases están balanceadas. Por otro lado hacemos variar el parámetro `C` entre los valores $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1.0, 10, 10^2\}$, es decir estaremos considerando siete modelos formados por el estimador y cada posible valor del parámetro `C`.

```
{'clf': [SVC(kernel='rbf', # kernel gausiano
             class_weight="balanced", # clases balanceadas
             random_state=SEED)],
'clf__C': [10**a for a in range(-4, 2)]}
```

- **RandomForestClassifier**. Elegimos este modelo porque, al igual que el anterior, se adapta bien a los problemas de clasificación binaria. Hacemos variar dos parámetros `max_depth` y `n_estimators`. El parámetro `max_depth` varía entre los valores $\{10, 20, 30, 40, 50\}$ y el parámetro `n_estimators` entre los valores $\{50, 100, 150, 200\}$, por lo tanto estamos considerando 20 modelos formados por el estimador y cada posible combinación de los parámetros.

```
{'clf': [RandomForestClassifier(random_state=SEED,
```

```
                                    class_weight="balanced")],
'clf__max_depth': [10, 20, 30, 40, 50],
'clf__n_estimators': [50, 100, 150, 200]}
```

# 7. Función de regularización

Para evitar sobreajustes en alguno de los modelos 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):

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

# 8. Valoración de los resultados

# 9. Justificación