Trabajo Práctico 2

Críticas cinematográficas

Integrantes

Apellido	Nombre	Grupo
Lopez	Francisco	17
Corn	Franco	17
Queirolo Dominguez	Cristian Daniel	17

Informe

Introducción

En este trabajo práctico vamos a utilizar una colección de críticas cinematográficas en idioma español y vamos a tratar de identificarla como positiva o negativa.

Objetivos

En este trabajo práctico buscaremos construir los siguientes modelos para luego poder optimizar sus hiper parámetros:

- Bayes Naïve
- Random Forest
- XG Boost
- Un modelo de red neuronal aplicando Keras y Tensor Flow.
- Un ensamble de al menos 3 modelos elegidos por el grupo

Desarrollo

Preprocesamiento/Análisis de datos

Empezamos realizando un análisis en los datos de train, analizamos valores nulos y distribución de nuestros datos para observar que no hay nulos.

Para el preprocesamiento utilizamos “nltk” vamos a pasar a lowercase, tokenizar y extraer palabras conectoras del dataset. (esperando mejorar un poco el overfit), luego pasamos nuestra variable target a binaria para utilizarla con nuestros modelos predictores.

Predicciones y Modelos

Empezamos dividiendo nuestro dataset con una proporción de 70/30 para train y test para probar nuestros modelos y realizando una función de métricas para poder mostrar nuestras métricas.

Para poder tokenizar todas nuestras palabras en lugar de utilizar Bag-Words decidimos utilizar Tfidvectorizer ya que obtuvimos mejores resultados al momento de realizar nuestras predicciones.

Bayes naive

Primero creamos un modelo y lo probamos con nuestro se de train (sin optimizar hiper parámetros) obteniendo un f1: 0.85.

Para mejorar el modelo decidimos optimizar los hiper parámetro alpha (de nuestro modelo de bayes) y ngram range (Tfi) utilizando grid search logramos optimizar nuestros hiper parámetros y llegar a un f1: 0.873

Xg-Boost

Realizamos el mismo procedimiento que con bayes, creamos un modelo inicial sin realizar optimización de hiper parámetros y logramos un f1: 0.83

Para mejorar nuestro modelo decidimos optimizar los hiper parámetros: max_depth,n_estimators y learning_rate.

Utilizando grid search llegamos a obtener un f1 de 0.86, mejorando bastante nuestro modelo.

Random Forest

Volvemos a crear un modelo sin realizar una optimización de hiper parámetros logrando obtener una f1 de 0.84.

Para mejorar nuestro modelo elegimos los hiper parámetros: n_estimartors, max_depth y min_samples_split.

Utilizando grid search llegamos a obtener un f1 de 0.86

Voting

Utilizamos voting como ensamble de modelos, elegimos los modelos: Random forest, Xg-boost y Bayes naive.

Voting obtuvo un f1 de 0.88 en el conjunto de train y 0.749 en el conjunto de test siendo asi el mejor predictor que obtuvimos.

Redes neuronales

Nuestra red neuronal tiene una arquitectura de 5 capas. La primera es una capa de embedding que se encarga de convertir los índices enteros de las palabras en vectores de valores reales. La segunda es una capa Conv1D la cual permite que la red aprenda patrones y relaciones locales en los textos. La siguiente es una capa de GlobalMaxPooling1D que reduce la dimensionalidad de las características extraídas, manteniendo las más relevantes y nos ayuda a reducir el overfit. Las últimas dos son capas dense las cuales nos son útiles para que la red aprenda relaciones más complejas entre las características extraídas por las capas anteriores (activación relu) y luego comprima los resultados en un rango de 0 a 1 (activación sigmoid). Este resultado es la probabilidad de que sea positivo.

Mediante la búsqueda de hiperparametros y varias pruebas encontramos que los más adecuados son learning rate = 0,001, num_units = 32 y epochs = 1.

El máximo puntaje que logramos obtener en train es 0,89.

Resultados local y kaggle

Local

Bayes Naive: f1 = 0.881 - recall = 0.884 - precision = 0.878

Xg-Boost: f1 = 0.857 - recall = 0.864 - precision = 0.851

Random-Forest: f1 = 0.862 - recall = 0.876 - precision = 0.848

Redes-N: f1 = 0.890 - recall = 0.868 - precision = 0.909

Voting : f1 = 0.878 - recall = 0.886 - precision = 0.870

Kaggle

Bayes Naive: puntuación = 0.727

Xg-Boost: puntuación = 0.705

Random-Forest: puntuación = 0.738

Redes-N: puntuación = 0.54

Voting: puntuación = 0.749

Conclusiones

Problemas con el overfit conjunto local y kaggle

En primer lugar el overfit (entre otras cosas) puede producirse por el ruido en nuestros datasets, nosotros intentamos eliminar casi todo el ruido posible además de probar otras opciones para tratar de eliminar le overfit como:

Reducir el dataset y utilizar menos palabras: Con esta opción, quitando las stop words, palabras repetidas, partes del dataset que están en otro idioma, si bien se redujo el overfit al tener mucha menos información nuestras métricas bajan bastante (al parecer necesitamos la información que nos proporciona cierta parte del dataset que decidimos retirar). Por lo que decidimos no proceder con esto.

Creemos entonces, que no es un problema solo de overfit, si no que el dataset de test es diferente al de train (es un dataset distinto/contiene mas o menos palabras que no se encuentran en train) por lo que estos resultados no son solo por overfit si no por lo diferentes que son los dataset.

Mejor modelo y desempleo en general

En general, los modelos con el dataset de train funcionaron muy bien, nuestro mejor predic en el conjunto local fue las R-N. Sin embargo, fueron las peores a la hora de predecir en el conjunto de test (al parecer sufrieron mucho el “cambio” de datos al pasar de un dataset a otro).

Los modelos de Xg y Bayes obtuvieron scores bastante altos en train 0.88 y 0.86 pero sufrieron de nuevo cuando cambiamos los datos cayendo a 0.727 y 0.716, siendo bayes el mejor.

El mejor modelo (de manera individual) empleado fue R-M,si bien con el dataset de train no fue el mejor obteniendo 0.86, fue el que menos sintió el overfit y el que mejor funcionó prediciendo los datos de test con un 0.738 .

El voting fue el que mejor funcionó, logrando un predict de 0.7479 en test y 0.88 en train (era lo más esperado, ya que si daba un resultado más bajo que los modelos de forma individual estábamos en un problema)

Como conclusión final, todos los modelos sintieron el overfit que mencionamos arriba, la red neuronal tuvo un muy mal desempleo con los datos de test y voting fue el que mejor funcionó y menos sintió el overfit el dataset.