Martin Laurito, Lucas Franciulli, Dante Alejandro Finci

# Informe TP2

- ## Bayes Naive

  En este modelo utilizamos TfidfVectorizer.

  **F1 Score:** 0.8426381461675579 **Accuracy Score:** 0.8473192666897268

  **Recall Score:** 0.815934824634079 **Precision Score:** 0.8711484593837535

  Kaggle: **0.69645**

- ## XGBoost

  Para vectorizar palabras en este modelo utilizamos CountVectorizer.

  Primero probamos hacer un modelo sin ningún pre procesamiento de textos, solo buscando hiperparametros con Cross Validation. **En ese caso los resultados fueron:**

  **F1 Score: 0.7311505348291154 Accuracy Score: 0.7252**

  **Recall Score: 0.7473333333333333 Precision Score: 0.7156537282941777**

  **Kaggle: 0.68269**

  Luego aplicamos algunas técnicas de preprocesamiento de textos como transformar todas las mayúsculas de los textos en minúsculas para que no clasifique 2 palabras con el mismo significado como diferentes, borrar todo signo de puntuación porque no aportan información relevante, remover espacios vacíos y remover algunas de las stopwords más frecuentes que no aportan información (por ejemplo 'no' no se remueve por cambia el significado de 'no bueno' a 'bueno'). Además aplicamos stemming para palabras en español con el SnowballStemmer de NLTK y removimos las reseñas en inglés.

  Buscamos hiperparametros con Cross Validation, que dio los mismos resultados que sin preprocesamiento, y los resultados fueron:

  **F1 Score: 0.7818642558728853 Accuracy Score: 0.7755936024800958**

  **Recall Score: 0.8045102184637068 Precision Score: 0.7604583000266454**

  **Kaggle: 0.68928**

- ## Ensamble Tipo Voting

  En este modelo utilizamos CountVectorizer. Y utilizamos los modelos XGBoost, RandomForest y KNN.

  **F1 Score:** 0.7729715206878023 **Accuracy Score:** 0.7661708751297129

  **Recall Score:** 0.7975051975051975 **Precision Score:** 0.7499022546591946

  Kaggle: **0.66698**

- ## Random forest

  Para este modelo utilizamos el método de random search de cross validation para encontrar los hiperparametros que mejor se ajustan al modelo. Usamos una semilla de 42, 10 kfolds y unas 20 iteraciones. Los hiperparametros que ajustamos fueron: la cantidad de árboles de decisión (con un rango de 50 a 100), la profundidad (un rango de 10 a 50), min_samples_split (en un rango de 10 a 50) y min_samples_leaf (en un rango de 20 a 60). De esa forma obtuvimos: n_estimators = 60 min_samples_split = 32, min_samples_leaf= 49, max_depth = 37.

  **F1 Score:** 0.6963757650780551 **Accuracy Score:**0.6945693531649948

  **Recall Score:** 0.7017325017325017 **Precision Score:** 0.691100191100191

  **Mejor predict de kaggle:** ***0.69412***

- ## Red Neuronal

  La arquitectura de red neuronal que mejor encontramos fue con una capa de entrada de 90 neuronas con activación relu, una capa oculta con 87 neuronas con activación relu con un dropout de 0.2, otra capa oculta de 75 neuronas con activación relu con un dropout de 0.15 y lA capa de salida de una neurona de activación sigmoid. Para esa arquitectura usamos un learning rate de 0.003, 12 épocas y 36 como tamaño de lote.

  Elegimos esta arquitectura porque fue la que mejor varianza tuvo en las métricas en el transcurso de las épocas. Por eso, las métricas en la validación son estables.

  **Perdida:** 0.5651611685752869 **Accuracy:** 0.7207886576652527

  **Precision:** 0.7126421332359314 **Recall:** 0.738322913646698

  **f1_score:** 0.7252552756977535 **Predict en kaggle:** ***0.66388***