Críticas cinematográficas

Sebastian Makkos, Rodolfo Albornoz, Valeria Brzoza

Organización de datos, 1C23

Para este trabajo teníamos un dataset de training y uno de test para construir modelos que predigan si una crítica cinematográfica era positiva o negativa. Hicimos un pequeño análisis exploratorio, seguido del preprocesamiento. Para el mismo, eliminamos la puntuación, simplificamos la complejidad convirtiendo las mayúsculas en minúsculas, eliminamos las palabras que no tuvieran connotación. En algunos casos aplicamos la Lematización aunque esto no mejoraba significativamente los modelos, e incluso hubo casos donde aplicarlo empeoraba el modelo. Otro cambio que tuvimos que realizar a nuestros modelos, fue reemplazar las palabras “positivo” y “negativo” por 1 y 0 respectivamente.

El primer modelo que trabajamos fue Bayes Naive. Después de muchas pruebas conseguimos el mejor modelo de ese tipo aplicando Tfidf y MultiomialNB. También nos dimos cuenta que las métricas de este modelo mejoraba descartando únicamente algunos caracteres especiales, en vez de otros. Por otro lado le eliminamos stopwords tanto en español como inglés, y fue un modelo sin lematización. Las métricas de training de este modelo fueron:

```
Precision: 0.8888666666666667

Recall: 0.8894687541817209

f1 score: 0.8885769667802954
```

mientras que en Kaggle este modelo dio una puntuación pública de 0.73308

En cuanto al random Forest, probamos Tfidf y count, y Tfidf nos terminó devolviendo el mejor modelo. Este modelo en particular, nos dio la mejor predicción en Kaggle, con una puntuación de 0.74898. Sin embargo, las siguientes métricas de training, fueron corridas con un max_depth y min_depth reducidos, dado que el modelo original demoraba demasiadas horas en correrse. Suponemos que estas métricas de training se acercan a las que debería devolver el modelo en cuestión:

```
Precisión: 0.7628

Recall: 0.8132823823955725

f1 score: 0.7762545591749466
```

Realizamos el modelo de XGBoost, y nuevamente, nuestro mejor modelo fue el que fue aplicado con Tfidf. Sin embargo, al igual que con Random Forest, fue un modelo que tardó mucho en correr, por lo que tuvimos que bajar sus parámetros para sacar las siguientes métricas.

```
Precision: 0.7844666666666666
Recall: 0.8106469890631177
f1 score: 0.7919160713136384
```

La submisión en Kaggle de este modelo tuvo 0.7164 de puntaje.

Para el ensamble híbrido decidimos usar Voting, ya que al querer correr modelos de stacking teníamos diferentes errores y problemas. El modelo de voting que usamos tenía un bayes naive multinomial, un random forest y un KNN. Tuvo mejores resultados al aplicarle Tfidf. Sus métricas de train fueron:

```
Precision: 0.8568666666666667
Recall: 0.883383845038872
f1 score: 0.8619736419157827
```

Mientras que en Kaggle dio 0.70672

En cuanto a las redes neuronales, la arquitectura LSTM daba mejores métricas, aunque predecía todos los casos como “positivo”. Por esta misma razón, decidimos optar por analizar la arquitectura RNN. Sus métricas en training fueron:

```
Precision: 0.6067
Recall: 0.6378249652708871
f1 score: tf.Tensor(0.62040335, shape=(), dtype=float32)
```

Mientras que su predicción en Kaggle fue de 0.67745.

Tras terminar este trabajo práctico podemos concluir que en todos los modelos, Tfidf nos resultó en mejores modelos que cuando utilizabamos Count, sin embargo requerimos de ambos para hacer el bag of words. Por otro lado, realizando las últimas mediciones, pudimos probar que las ejecuciones en Jupyter eran más rápidas que las realizadas en collab por su velocidad de procesamiento. Por último, después de prueba y error, nos dimos cuenta que el preprocesamiento era mejor conservando algunos caracteres (Tildes, punto, coma, por ejemplo)