TP2: Críticas Cinematográficas - Grupo 07

Introducción

En el presente proyecto se planteó la creación de 5 modelos de procesamiento de lenguaje natural con el objetivo de predecir si una opinión de una película es positiva o negativa. Para esto se crearon modelos de tipo Bayes Naive, XG Boost, Random Forest, una red neuronal y un ensamble de tipo voting, y luego fueron optimizados sus hiperparametros. Los detalles de estos modelos serán definidos más adelante en el presente informe.

Una vez entrenado cada modelo, se lo evaluará y se someterá su predicción a una competencia en Kaggle.

Con el objetivo de entrenar y posteriormente testear los modelos creados, la cátedra nos facilitó dos conjuntos de datos, uno train y otro test:

- Por un lado, el conjunto train consiste de 3 columnas, una de ID de la muestra (*ID*), otra de correspondiente a la opinión sobre la película (*review_es*) y una última correspondiente a la variable *target* que indica el sentimiento, positivo o negativo (*sentimiento*).
  Consta de un total de 50001 muestras.

- Por otro lado, tenemos el conjunto test, que consta únicamente de las columnas *ID* y *review_es*. Lógicamente, al ser el conjunto test, este set de datos no cuenta con la columna *target*.
  La cantidad de muestras de este set es 8598.

Respecto al preprocesamiento de la data, se probaron diferentes métodos:

1. Ningún preprocesamiento: Curiosamente, fue el que mejores resultados dio. En este caso, lo máximo que se hizo es mapear los valores posibles de la columna “sentimiento” a valores 0 y 1 para modelos como XG Boost que requieren de ello (one hot encoding)
2. Preprocesamiento con stop-words y manejo de tildes: Para empezar, consideramos que las palabras con tildes podían llegar a ser un problema. Lo primero que se hizo fue reemplazar las letras tildadas, por la misma letra pero sin tildar.

   Luego, se consideró la opción de que aparecieran palabras en inglés. Es por esta razón que se definieron stop-words tanto en inglés como en español para omitir las palabras más comunes que poca información pueden aportar al análisis de la muestra.

   Por último, consideramos otras maneras de evaluar el “peso” de una palabra más allá de su simple significado, como es el caso de las palabras en mayúsculas que transmiten un gran énfasis. Sin embargo, en el caso de los emojis, haciendo un análisis a mano con VS Code + Ctrl F, se encontró

que en el conjunto train hay un total de 7 emojis de tipo ":D", 40 de tipo ":(" y 380 de tipo ":)", lo que en conjunto representa un porcentaje menor al 1% de las muestras. Es por esta razón que no se tomaron en cuenta, aunque si se tuviera un set de datos con mayor cantidad de emojis presentes, seguro serían un factor clave a analizar para resolver el problema planteado.

También intentó pasar todas las letras a minúsculas. A pesar de todo lo mencionado, el preprocesamiento solamente empeoró los resultados de predicción de nuestros modelos.

Por lo mencionado anteriormente, se tomó la decisión de únicamente mantener el preprocesamiento de los stopwords y quitado de tildes.

Cuadro de Resultados

Modelo	F1-Test	Presicion Test	Recall Test	Kaggle
Bayes Naive (OPTIMO)	0.86	0.86	0.86	0.74
Bayes Naive (Ensamble)	0.86	0.85	0.86	0.70
Random Forest	0.84	0.83	0.84	0.69
XgBoost	0.84	0.84	0.84	0.70
Red Neuronal	0.84	0.84	0.84	0.739
Ensamble	0.87	0.87	0.86	0.72

Descripción de Modelos

1. Bayes Naive MB:
    - Fue el modelo que mejor rendimiento de predicción ofreció, aunque muy lento de entrenar.
    - Para vectorizar las palabras, se utilizó un Tfidf Vectorizer, aunque también se intentó utilizar Count Vectorizer, ofreciendo este último peores resultados.
    - La única manera de conseguir hacerlo predecir correctamente fue mediante el uso de un pipeline. Al intentar vectorizar las palabras primero y posteriormente realizar la predicción sobre esa data procesada, los resultados eran muy inferiores a lo esperado, ni siquiera superando .60.
    - Se optimizó su hiperparametro "*multinomial alpha*" encontrando que su valor más efectivo era 0.1. Se intentó optimizar otros valores, sin embargo se recibía error en el

proceso, o incluso en ocasiones donde no se recibe error, el rendimiento del modelo era inferior.
- Respecto al Tfidf Vectorizer, se optimizó el parámetro "*tfidfvectorizer__ngram_range*" el cual se definió su mejor valor como (1, 2). También, a pesar de que esto bajaba ampliamente el rendimiento del modelo, en diversos entrenamientos se tuvo que limitar el parámetro "*max_features*" debido a que si esto no se hacía, el colab fallaba por consumo excesivo de RAM. Sin embargo, con el pipeline esto no fue necesario.
- Todas las optimizaciones se hicieron sobre el *pipeline* con *grid search* con *cv* = 5 y *n_jobs* = -1.
- Este modelo se consiguió sin aplicar preprocesamiento sobre la data.

2. XG Boost:
   - Ofreció un rendimiento normal. Ni el mejor, ni el peor.
   - Para vectorizar las palabras, se utilizó un Tfidf Vectorizer, aunque también se intentó utilizar Count Vectorizer, ofreciendo este último peores resultados.
   - Se encontró el mismo problema que con el Bayes respecto al uso de vectorizador y pipeline. Al intentar hacerlo predecir sobre un set de datos vectorizado sin pipeline, terminó realizando todas las predicciones a la clase "0".

     Es por esto que se decidió utilizar un pipeline de Tfidf Vectorizer y el modelo. Pero al utilizar este pipeline, no era posible optimizar los hiperparametros sin recibir errores. La solución fue, entrenar un modelo sin pipeline, optimizar los hiperparametros de este y una vez obtenidos los mejores hiperparametros, crear un modelo en el pipeline, directamente instanciado con los mejores hiperparametros obtenidos previamente.

     Los hiperparametros optimizados con *grid search* y sus mejores valores fueron:
     1. *Random_state* = 0
     2. *Subsample* = 0.8
     3. *N_estimators* = 300
     4. *Min_child_weight* = 1
     5. *Max_depth* = 4
     6. *Learning_rate* = 0.2
     7. *Colsample_bytree* = 1.0
   - No fue necesario limitar los features del Tfidf Vectorizer.
   - Fue muy lento de entrenar y optimizar.

3. Redes Neuronales:
   - Primeros realizamos una Red Simple, en la misma creamos una capa con 500 neuronas de activación ReLu, con la intención de observar cómo se comportan las redes para PLN. La capa de predicción posee una activación sigmoidea, la cual da resultados entre 0 y 1, nosotros utilizamos un umbral de 0.4

- Luego de esta primera impresión decidimos hacer una red Grid Search con el objetivo de conseguir una óptima cantidad de batch_size, para ello utilizamos los siguientes parámetros:

```
param_grid = {
        'epochs': [20]
        'batch_size' : [16,32,64]
        'callbacks' : [tf.keras.callbacks.EarlyStopping("accuracy",patience=5)]
}
modelo_cv = KerasClassifier(build_fn=create_model)
grid = GridSearchCV(estimator=modelo_cv, param_grid=param_grid,n_jobs=-1,cv=4)
```

  Esta red es similar a la red anterior solo que la cantidad de neuronas estaba determinada por la cantidad de features de la vectorización dividido 2(EL mejor caso nos dio con 1000 features) El mejor batch_size fue de 64 y algo importante que notamos es que a partir de 10 epochs el conjunto estaba overfiteando
- Después de probar varias arquitecturas con más capas de ReLu y capas Dropout para evetira el overfitting, la mejor red fue igual a la de Grid Search:

```
d_in = x_train.shape[1]
modelo= keras.Sequential([ keras.layers.Dense((d_in/2), activation='relu'),
keras.layers.Dense(1, activation='sigmoid')])

modelo.compile(optimizer="adam",loss='binary_crossentropy',metrics=['accuracy'])

red=modelo.fit(x_train,y_train,epochs=5)
```

- Obtuvimos en testing local un accuracy es: 0.84, f1 de: 0.847, un recall de: 0.83 y la precisión es: 0.87
-

4. Random Forest
   - Primero creamos un modelo base los parámetros min samples leafs (5), min samples splits (5) y n estimators (30), rando_state (7). Se probó utilizando Tfidf Vectorizer con y sin stopwords dando mejores resultados utilizando el vectorizer del preprocesamiento. Usamos este último para la búsqueda de hiperparametros con Grid Search se optimizaron los siguientes hiperparametros:
   - *"criterion"* : ["gini", "entropy"],
   - *"min_samples_leaf"* : [1, 2, 5],
   - *"min_samples_split"* : [2, 4, 10],
   - *"n_estimators"*: [30, 40, 50] }

   Los hiperparametros recomendados por Grid search (5 folds):
   - *'criterion'*: 'gini',

- *min_samples_leaf*': 2,
- *min_samples_split*': 2,
- *'n_estimators*': 40

El modelo optimizado, rindió un poco peor que el base, podemos suponer un sobre ajuste por cv, sin embargo usamos este último para el armado del ensamble porque creemos que al estar un poco más "especializado" ayudaría a generalizar .

En general, los resultados obtenidos de este modelo fueron peores que XG Boost y Naives Bayes (con stopwords).

5. Voting

Para el ensamble optamos por armar un Voting que utiliza alguno de los modelos anteriormente entrenados (Naive Bayes, XG Boost y un Random Forest).

Probamos con y sin validación cruzada de 5 folds, tuvimos peores resultados haciendo validación cruzada el modelo tendió a sobreajuste, por lo tanto descartamos este último tratamiento y nos quedamos con el original (sin cross validation).

Conclusiones generales

A modo de conclusión, creemos que si bien no obtuvimos los resultados esperados por nosotros en kaggle, sobre todo en comparación del trabajo realizado en la primera parte de la materia, intentamos aún más maneras de obtener los mejores resultados. Desde preprocesamiento hasta optimización de hiperparametros.

Sin embargo, también creemos que la clave no estaba en el preprocesamiento de la data, si no en la obtención de los mejores parámetros.

Nos sorprendió la tardanza en los entrenamientos de los modelos, rondando entre 1 y 3 horas cada uno, a excepción del ensamble voting que duró entre 10 y 15 minutos.

Claramente el mejor modelo fue Bayes Naive, ofreciendo un rendimiento en Kaggle superior al 0.74, pero también fue uno de los más lentos de entrenar.
Sin embargo, creemos que nuestro modelo está lejos de ser un modelo usable en el día a día, ya que consideramos que su rendimiento es pobre, sobre todo comparándolo con lo hecho en el trabajo práctico 1

Para obtener mejores resultados creemos que se deben explorar más a fondo los hiperparametros de bayes naive, que entendemos que es el modelo con más potencial para realizar predicciones certeras en procesamiento de lenguaje natural.
Por otro lado, la red neuronal también puede ser muy prometedora, ya que si bien no obtuvo el mejor resultado, se podría decir que es una alternativa muy buena, en cuestion de costo, ya que la que mejor rendimiento tuvo por debajo de Bayes Naive, y además la búsqueda de hiperparametros no es muy costosa computacionalmente hablando

Tareas Realizadas

Integrante	Promedio Semanal (hs)
Guido Menendez	10
Mateo Riat Sapulia	10
Rafael Wu	10