TP2: Críticas Cinematográficas - Grupo 16

Introducción

El presente trabajo radica en la necesidad de poder realizar predicciones de "sentimiento" sobre un dataset (corpus) donde las observaciones (documentos) son textos escritos en lenguaje natural. La tarea a resolver, consiste en la generación de modelos predictores, que categoricen para este caso, en forma dicotómica ("positiva" o "negativa") un conjunto de nuevas reseñas.
Disponemos de dos datasets. El primero el dataset utilizado para el entrenamiento y validación, y otro dataset correspondiente a los registros sobre los cuales deben realizarse las predicciones que posteriormente conformarán los submits de la competencia de kaggle.

- *Dataset train:* Se compone de tres columnas, la primera correspondiente al ID, la segunda identificada como "review_es" y corresponde a las críticas en sí mismas, escritas en lenguaje natural, mientras que la última columna corresponde la categoría (sentimiento) de cada crítica. Posee un total de 50.000 registros, sin valores nulos, y se encuentra balanceado, es decir el 50% de las críticas categorizadas como positivas y el otro 50% como negativas.
- *Dataset test:* posee sólo 2 columnas, una ID y la segunda identificada como "review_es" la corresponde a las críticas en sí mismas, escritas en lenguaje natural al igual que en el dataset de train. Este dataset posee 8600 registros dentro de los cuales tampoco se observan documentos nulos.

Se realizaron las mismas técnicas de limpieza de datos en ambos datasets. En principio se renombra la columna "review_es" como "review" y se realizó un label encoding de la columna "sentimiento" a fin de disponer de una categorización numérica y ya no strings "positivo" y "negativo". Se procedió a uniformizar el texto convirtiéndolo en minúsculas a menos que toda la palabra estuviera en mayúsculas. Se continuó aplicando a los registros una función que quite los signos de puntuación y los espacios. De igual forma, mediante otra función se eliminaron los valores numéricos dado que entendimos que no aportan información significativa para determinar la categoría del registro. Se procedió a eliminar de los registros los espacios dobles.
Por último, se probaron técnicas de stemming (utilizando los tres algoritmos más conocidos Porter / Porter2 (Snowball) / Lancaster), así como también técnicas de lematización, esto mediante las librerías de Spacy y NLTK.

Modelos

Se llevaron a cabo una multiplicidad de modelos producto de la combinación de diferentes predictores, optimizaciones de sus hiperparámetros y diversos tratamientos del corpus.
La optimización de hiperparámetros en los modelos y en los vectorizers, se llevó a cabo mediante la librería Optuna la cual permite maximizar una función objetivo en forma automatizada, mediante iteraciones donde se ejecutan iterativamente diversas configuraciones de hiperparámetros de manera bayesiana. Si bien la librería permite la optimización multiobjetivo (de varias métricas), para este tp se contempló solo f1 Score.

Como se puede observar en las notebooks que acompañan el presente informe algunos de los modelos y sus modificaciones estudiados son los siguientes:

- Un modelo Naive Bayes default con count vectorizer.
- Un modelo Naive Bayes con optimización de hiperparámetros bayesiana del modelo y del vectorizer (count vectorizer + tfidf transformer) con variantes:
    - sólo unigramas
    - unigramas y bigramas
    - sólo bigramas
    - bigramas y trigramas
    - trigramas
- Un modelo Random Forest con optimización de hiperparámetros bayesiana del modelo y del vectorizer (count vectorizer + tfidf transformer).
- Un modelo XGBoost con optimización de hiperparámetros bayesiana del modelo y del vectorizer (count vectorizer + tfidf transformer).
- Un modelo KNN con optimización de hiperparámetros bayesiana del modelo y del vectorizer (count vectorizer + tfidf transformer).
- Modelo Red Neuronal - Perceptron Multicapa utilizando:
    - sólo unigramas con arquitectura 1
    - sólo bigramas con arquitectura 1
    - unigramas y bigramas
        - con arquitectura 1
        - con arquitectura 1 + capas de dropout intermedias + regularización L2 en capas densas

Arquitectura 1 : 1 capa de entrada con tantas neuronas como inputs + una capa densa con 32 neuronas con activación tanh + una capa densa de 16 neuronas con activación ReLu + una capa de salida con una neurona. La capa de entrada toma los datos vectorizados por un vectorizer (count vectorizer más tfidf transformer)

- Modelo Red Neuronal - Deep Learning con arquitectura 2

Arquitectura 2 : 1 capa de entrada con tantas neuronas como inputs + 4 capas densas intermedias + una capa de salida con una neurona. La capa de entrada toma los datos vectorizados por un vectorizer (count vectorizer más tfidf transformer)

- Ensamble tipo Stacking con meta-modelo de regresión logística y aplicando nuestros mejores modelos (RandomForest , Naive Bayes y el perceptrón multicapa)

Cuadro de Resultados

Presentamos un cuadro de resultados donde exponemos las métricas F1, Precision y Recall que obtuvimos de nuestros modelos en test y la métrica F1 obtenida de las predicciones subidas a Kaggle.

Modelo	F1-Test	Precision Test	Recall Test	Kaggle
Naive Bayes	0.8844	0.8768	0.8921	0.7546
Random Forest	0.8727	0.8620	0.8836	0.7449
XGBoost	0.8820	0.8708	0.8935	0.7301
Red Neuronal	0.8958	0.8844	0.9076	0.77398
Ensamble Stacking	0.9792	0.9798	0.9787	0.7633
KNN	0.8384	0.8123	0.8663	0.6797

Descripción de Modelos

Descripción de modelos:

- **Naive Bayes:** Se creó un pipeline utilizando countVectorizer y TfidfTransformer junto con un predictor Naive Bayes Multinomial y se procedió a la optimización de sus hiperparametros mediante la librería optuna. El principal hiperparámetro optimizado del modelo fue el alpha, o sea, el coeficiente de suavizado, resultando en un valor óptimo del ~0.0047. El count vectorizer para el ejercicio con f1 score más alto tomaba en cuenta unigramas y bigramas.
- **Random Forest:** Se creó un pipeline utilizando countVectorizer y TfidfTransformer junto con un predictor random forest y se procedió a la optimización de sus hiperparametros mediante la librería optuna. Los parámetros optimizados del

árbol de decisión han sido: n_estimator, criterion, min_samples_split y min_samples_leaf. El resultado es modelo predictor con 840 arboles, el criterio para la división de nodos es por entropía, y la mínima cantidad de muestras por nodo para dividirlo es de 23 y tres muestras por hoja. Con este modelo logramos un score f1 local de 0,8727.

- **XGBoost:** Al igual que el modelo anterior se procedió a generar un pipeline utilizando countVectorizer, TfidfTransformer y nuestro modelo seleccionado XGboost. Se llevó a cabo la optimización de hiperparametros mediante la librería optuna con una cantidad de 40 iteraciones , puntualizando en este caso los parametros max_depth, learning_rate, n_estimators, min_child_weight, gamma, subsample, reg_alpha, reg_lambda, eval_metrics, dando como resultado un modelo que posee una profundidad máxima para los árboles de 6, una tasa de aprendizaje 0,158 (buscando prevenir el overfitting), una cantidad de árboles máxima de 870 como características sobresalientes. Mediante este modelo conseguimos un f1 score local de 0,8820 superando a RF. Si bien el modelo consiguió una buena métrica la optimización de todos los hiperparametros del pipeline, tomó un total aproximado de 11.3h.
- **Red Neuronal:** Para converger a este modelo se comenzó con un perceptrón simple con 1 capa de entrada una capa densa y una capa de salida, sin optimización de parámetros. Luego buscando bibliografía pudimos leer varios papers y apalancarnos particularmente en una tesis de la Universidad Politécnica de valencia (se cita en las fuentes), donde se proponía una estructura de red neuronal multicapa con una capa de entrada con tantas neuronas como input tengamos, una capa densa con 32 neuronas con función de activación “tanh”, una tercera capa densa de 16 neuronas con activación ReLu, para terminar en una capa de output de 1 neurona.Este modelo, mostró una performance bastante buena 0,749. Sin embargo, considerando que la métrica F1-Score de validación era considerablemente más elevada, se buscó mejorar el modelo suponiendo que el problema radicaba en el overfitting. Por tal motivo se agregaron dos capas de dropout así como una regularización del tipo “ridge o L2” a fin de penalizar los pesos grandes en la función de pérdida. Estas modificaciones sumadas a la utilización de la librería optuna (especialmente aplicada a la optimización de hiperparametros de la vectorización y del TfidfTransformer en estudios de modelos previos) nos dejaron nuestro mejor modelo.
- **Ensamble:**
  Nuestro mejor ensamble, se consiguió mediante la utilización de la técnica de stacking, y se utilizaron los mejores tres modelos anteriores, es decir, el mejor RF, la mejor red neuronal y el mejor Naive Bayes, con un meta-modelo de regresión logística y el passthrough desactivado, este ejercicio terminó obteniendo un f1

score elevado 0.979, sin embargo su desempeño en kaggle se detrimentó a 0.76332.

- **Modelo Extra KNN:**
  De forma extra a los algoritmos solicitados, y guiados por dos papers, es que realizamos algunas pruebas utilizando un modelo predictor KNN. De igual forma que en los otros modelos, utilizamos la librería optuna para la optimización de hiperparámetros. Los resultados óptimos obtenidos han sido 19 vecinos y ponderación (*peso*) por distancia (*distance*). Se observaron algunas particularidades respecto a este modelo. En principio la performance f1 score en las pruebas de validación no ha sido de las más elevadas pero tampoco extremadamente baja (0,8384), sin embargo podemos apreciar el fuerte descenso con el dataset de test, dado que en Kaggle ha sido 0,679 lo cual nos permite inferir serios problemas de generalización, es decir overfitting. El segundo punto particular es que con el modelo ya entrenado, el tiempo de predicción fue considerablemente más elevado que en otros modelos, tardando aproximadamente 20 min en realizar una predicción sobre el dataset Test.

Conclusiones generales

Al igual que en el trabajo anterior, consideramos que el análisis exploratorio no sólo es importante sino que puede ser decisivo en los resultados que se obtengan, e incluso condicionar el uso de determinadas herramientas o bien determinar cierto procesamiento previo que deba realizarse en el dataset.
Ejemplo de esto fue conocer qué tipo de datos teníamos, textos con caracteres alfabéticos, alfanuméricos, signos de puntuación u otro tipo de caracteres (esto nos permite conocer que preprocesamiento realizar para sacar todas aquellas palabras que no aportaran información relevante). De igual manera es fundamental conocer el idioma del dataset, dado que esto condiciona las palabras vacías o “stop words” que se contemplarán. Incluso al utilizar modelos pre entrenados podríamos por error pretender utilizar un modelo como BERT (en su versión pre entrenado en inglés) para predecir algo relacionado a un documento en español. Otro ejemplo de un análisis exploratorio nos permite identificar el tipo de datos no solo del corpus sino también, por ejemplo, de la columna objetivo y comprender si su tipo de datos es compatible con el modelo que deseamos, por ejemplo, no podríamos utilizar una categoría textual para entrenar un modelo de red neuronal, lo que nos obliga a realizar una codificación, como se realizó en el TP.
De igual forma nos permite tener idea del tamaño de nuestro dataset, así como su balance, a fin de tomar las acciones que consideremos necesarias.

Sin duda las tareas de preprocesamiento han sido fundamentales para la mejora en la performance de los modelos.

Tal como se indicó, luego de realizar un análisis exploratorio se puede proceder a procesar el dataset. Teniendo presente que para el PNL normalmente se realiza un procedimiento de tokenización, es fundamental no desperdiciar recursos dejando palabras / signos de puntuación, por ejemplo, que no aportan información y si los dejáramos generarían matrices de dimensiones muy superiores aumentando el costo computacional del entrenamiento, e incluso pudiendo empeorar la performance de las predicciones.
Ejemplos son el cambio de mayúscula a minúscula, la eliminación de signos de puntuación y espacios dobles o el uso de librerías como NLTK permitiéndonos importar stop words para excluirlas en el uso de los modelos.
Otro caso donde se aprecia de forma clara las bondades y afectación de la performance del modelo es al momento de dar tratamiento al texto como la selección de unigramas / bigramas. Se aprecia la diferencia de performance (f1 score) en los distintos modelos de redes neuronales, donde la misma red unigramas ofrece 0,8725 mientras que la misma red pero con el documento tratado como unigramas y bigramas nos ofrece un f1 score de 0,899.

De todos los modelos, el que mejor desempeño tuvo en TEST, ha sido el ensamble de tipo stacking que hemos realizado, si bien su rendimiento en Kaggle posee una diferencia considerable. Entendemos que la diferencia significativa responde al overfitting del modelo.

El modelo que mejor desempeño tuvo en kaggle, ha sido el modelo de red neuronal (perceptrón) multicapa donde la red se compone de una capa input, dos capas densas de 32 y 16 neuronas, una capa de salida y capas de dropout + detrimento de pesos mediante “L2” en las capas densas escondidas. Creemos que la cercanía entre los valores de test y de kagle justamente se deben a las capas de dropout y L2, lo que nos permite mejorar la capacidad de generalizar, es decir, combatir el overfitting.

Considerando los valores obtenidos en el presente trabajo, el modelo más rápido para entrenar en relación con su buen desempeño ha sido el modelo Naive Bayes, consiguiendo una performance óptima.
La red neuronal, también ha tenido tiempos de entrenamiento relativamente bajos considerando que ha sido el modelo más performante (con GPU y aumentando el tamaño del batch adecuadamente).

Consideramos que sería posible utilizar nuestro modelo predictor de forma productiva, si bien entendemos que el corpus sobre el que se realice la predicción debe tener características similares al utilizado para el entrenamiento a fin de obtener predicciones reales.

Esta consideración se corresponde con lo observado en los modelos donde hemos notado que el principal problema sería el sobreajuste, lo cual no reduce la capacidad de lograr mejores generalizaciones provocando grandes pérdidas de performance en dataset que sean muy disímiles.

Como lo hemos indicado anteriormente consideramos que el problema preponderante es el overfitting. A diferencia del trabajo práctico N°1, aquí el sobre ajuste aumenta de forma significativa con pequeños cambios en los hiperparámetros de los modelos. Notamos esto debido a la gran diferencia entre los resultados obtenidos en las pruebas de validación y los resultados obtenidos en Kaggle.
Por tal motivo, hemos previsto que para mejorar los resultados deberíamos concentrar nuestros esfuerzos en reducir el overfitting. Esto lo logramos modificando hiperparametros y/o realizando algún preprocesamiento sobre el dataset que nos permita limpiar aún más el contenido sin información.
Dentro de lo analizado, dada la diferencia importante entre validación y kaggle que obtuvimos, buscaríamos reducir el overfitting espacialmente en el modelo de ensamble (stacking).

Bibliografía:

- Aplicación para el análisis de sentimientos y tendencias en redes sociales.

- ANÁLISIS DE SENTIMIENTO EN TIEMPO REAL DE MENSAJES DE TWITCH CON ALGORITMOS DE CLASIFICACIÓN

Tareas Realizadas

Integrante	Promedio Semanal (hs)
DIEM, Walter Gabriel	15
MAIOLO, Alejandro Cristian	12