Organización de datos, Curso Rodriguez
1er Cuatrimestre 2023

Docente: Rodriguez, Juan Manuel.

Estudiantes:
- Mendoza Hernandez, Sabrina Scarlet (108524)
- Valeriani, Matias Gabriel (108570)
- Jang, Lucas (109151)

Corrector: Bedoya, Gabriel.

Fecha de entrega: 29 de junio de 2023.

Informe del Trabajo Práctico 2: Críticas cinematográficas

Desarrollo

Nuestro primer objetivo en este trabajo práctico fue hacer el preprocesamiento del texto para que este pueda ser analizado por los distintos modelos. Para ellos usamos una representación vectorial de texto basada en el esquema de TF-IDF (Term Frequency-Inverse Document Frequency). Esto ajusta el modelo a los datos de texto y transforma los textos en una matriz TF-IDF. La matriz resultante es una representación numérica de los textos. Los modelos de clasificación que realizamos para el análisis de texto en lenguaje natural fueron: Bayes Naïve, Random Forest, XGBoost, un ensamble con 5 modelos y un modelo de red neuronal aplicando Keras y Tensor Flow.

Mostraremos la performance obtenida por cada uno de estos modelos en nuestro set de entrenamiento, luego publicaremos la calificación obtenida en Kaggle de cada uno de los mismos. Además, desarrollaremos los hiperparametros escogidos para cada modelo.

Clasificadores

- ### Bayes Naïve

  Si bien este clasificador no cuenta con muchos hiperparametros, realizamos un Grid Search para encontrar la mejor combinación de estos mismos. Utilizamos los parámetros alpha y fit_prior, dando como resultado alpha = 0.5 y fit_prior = False.

  Métricas obtenidas en el conjunto de entrenamiento:
  F1-score: 0.8623
  Precision: 0.87

Recall: 0.86

Puntaje obtenido en Kaggle: 0.718

- Random Forest

  Para el clasificador de Random Forest utilizamos randomized search mediante cross validation para probar diferentes combinaciones de hiperparametros. Encontramos que utilizando un árbol con un alto número de estimadores pero no tan grande la profundidad (max_depth) se obtenía un buen rendimiento. Sumado a esto el criterio de entropía, logramos obtener buenas métricas, El modelo más apto obtenido con randomized search fue:
  max_depth = 60, n_estimators= 160, min_sample_split = 8, min_samples_leaf = 7, criterion = entropy.

  Métricas obtenidas en el conjunto de entrenamiento:
  F1-score: 0.8476
  Precision: 0.83
  Recall: 0.87

  Puntaje obtenido en Kaggle: 0.728

- XGBoost

  Para el clasificador XGBoost nuevamente utilizamos randomized search mediante cross validation. Al igual que con random forest, encontramos como valores óptimos un número alto de estimadores, pero no tan alto de profundidad (para evitar el sobreajuste). Además, decidimos agregar un número pequeño de learning rate el cual hace que el modelo aprenda de manera más lenta y gradual, lo que puede ayudar a evitar el sobreajuste. El modelo óptimo para nuestro caso es:
  max_depth = 30, n_estimators= 160, learning_rate = 0.06, clsample_bytree= 0.8, eta = 0.5.

  Métricas obtenidas en el conjunto de entrenamiento:
  F1-score: 0.8457
  Precision: 0.8383
  Recall: 0.8533

  Puntaje obtenido en Kaggle: 0.713

- Ensamble de modelos

Para el ensamble de modelos decidimos utilizar el ensamble de tipo Voting y nos quedamos con 5 clasificadores tales como LinearSVC (Support Vector Classifier), SGDClassifier (Stochastic Gradient Descent Classifier), Regresion Logistica y los clasificadores de Random Forest y XGBoost utilizados anteriormente.
Para el caso de LinearSVC utilizamos randomized search para lograr hallar un modelo con los parametros: LinearSVC(C = 0.2373973042428288, dual = True, loss= 'squared_hinge', penalty = 'l2'). Luego para los clasificadores de Regresión Logística y SGD obtuvimos mejores predicciones dejando los parámetros por default que estableciéndose manualmente. Previamente a realizar las predicciones del ensamble, mostramos cómo se comporta cada uno de estos nuevos modelos de forma independiente, mostrando sus respectivas métricas.

Métricas obtenidas en el conjunto de entrenamiento:
F1-score: 0.89
Precision: 0.88
Recall: 0.91

Puntaje obtenido en Kaggle: 0.746

- Red neuronal

Para la red neuronal tuvimos que tokenizar los textos y convertirlos en secuencias numéricas. Luego determinamos la longitud máxima de una secuencia en el conjunto de entrenamiento y utilizamos la función pad_sequences para rellenar las secuencias y asegurarnos de que todas tengan la misma longitud.
Luego mostramos diferentes arquitecturas, las cuales fuimos mejorando progresivamente a medida que modificamos las capas (layers), las neuronas, los epochs y el tamaño del batch size. Con respecto a los epochs, nos quedamos en todos los casos con el valor 5 ya que por cuestiones de simplicidad (tiempo de ejecución) y rendimiento no encontramos grandes diferencias en escoger un valor entre 5 y 10.
Como modelo base empezamos con 3 capas (Embedding, LSTM y Dense), el cual nos dió un buen rendimiento. Luego entrenamos otro modelo agregando una capa LSTM pero reducimos el nivel de neuronas y el batch size, obteniendo métricas muy similares al primer modelo.
Posteriormente hicimos otros modelos en los que fuimos probando con 2 capas de Dense; o mismo el caso de 3 capas LSTM con el cual logramos obtener una muy alta calificación de F1 Score en red neuronal.

Finalmente como mejor modelo, entrenamos una red utilizando 2 capas Bidirectional la cual permite que una capa LSTM procese la secuencia de

entrada en ambas direcciones. En esta arquitectura bajamos nuevamente el batch_size y logramos obtener nuestro mejor puntaje público en Kaggle. Los parámetros que escogimos para este último y *mejor modelo* son:
Layers: Embedding (50 neuronas), 2 Bidirectional (25 neuronas), 2 Dense (16 y 1 neurona/s). Previamente habíamos tenido buenos resultados con el uso de las capas Embedding, 2 capas LSTM y 2 capas Dense.

Métricas obtenidas en el conjunto de entrenamiento:
F1-score: 0.859
Precision: 0.81
Recall: 0.92

Puntaje obtenido en Kaggle: 0.76972

Conclusiones

A lo largo de este trabajo práctico llevamos a cabo un análisis exhaustivo del conjunto de datos, realizando un preprocesamiento diferente al realizado en el trabajo práctico 1, ya que tuvimos que remover palabras como stopwords, símbolos de puntuación, convertir texto a minúscula. Este preprocesamiento ayudó a que los modelos se entrenen de manera más eficiente.
Entrenamos varios modelos predictivos y evaluamos su rendimiento, destacando el modelo de Voting y Red Neuronal los cuales nos permitieron obtener el mejor performance para determinar si una crítica es positiva o negativa. Mas precisamente es la red neuronal la que nos permitió obtener la mejor calificación en la plataforma de Kaggle.