TP2: Críticas Cinematográficas - Grupo 10

**Introducción**

El objetivo de ese TP es de predecir si una crítica cinematográfica tiene una emoción y un sentimiento globalmente positivo o negativo. Por eso, debemos construir diferentes modelos de Machine Learning para entrenarlos sobre un conjunto definido y que aprenden cómo clasificar una crítica cinematográfica correctamente.

Tenemos un primero dataset de entrenamiento de 50000 críticas con un id, un texto en español y el sentimiento de la crítica (positivo o negativo). El segundo dataset es de 8600 críticas son un id y un texto en español. Entonces, tenemos pocos datos para entrenar los modelos, lo que representa una alta dificultad para nosotros porque soló podemos basarnos en esos textos y no en otros datos como una nota del autor, sus gustos…

Para mejorar el dataset, intentemos algunas técnicas que no fueron todas realizadas o exitosas:

* Intentemos aumentar los datos con nuevas críticas, basadas en las del dataset de entrenamiento. Por eso, pudimos eliminar algunas palabras para generar nuevas frases, reemplazar palabras por sinónimos, eliminar palabras. Esa técnica no fue tan exitosa, así preferimos abandonarla.
* Intentemos lematizar el dataset de entrenamiento para simplificar las frases y ayudar el modelo a la comprehensión de una frase con menos palabras, que sean más sencillas a aprender.
* Pensemos en normalizar el dataset de entrenamiento, pero no lo hicimos porque el dataset “clásico” y su equivalente “lematizado” fueron suficientes.

Además, para preprocesar los datos, hicimos una transformación a través de algunos vectorizadores: CountVectorizer, HashingVectorizer, TFIDFVectorizer y TextVectorizer (ese fue para la red neuronal únicamente). Aunque vamos a detallar ellos en las siguientes partes, para cada modelo, hicimos comparaciones entre cada vectorizador mientras optimizándolos (con sus hiperparámetros) para ver cuál estaba el mejor para simplificar los datos y que se puede usar en los diferentes modelos.

**Cuadro de Resultados**

Basando en el F1-Test de Kaggle, obtuvimos por cada mejor versión de cada modelo, ese cuadro de resultados:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Modelo** | **F1-Test** | **Precision Test** | **Recall Test** | **Accuracy Test** | **Kaggle** |
| Bayes Naïve | 0.88978 | 0.87567 | 0.90435 | 0.8871 | 0.74840 |
| Random Forest |  |  |  |  | 0.72649 |
| XGBoost |  |  |  |  | 0.71680 |
| Red Neuronal |  |  |  |  | 0.76662 |
| Ensamble |  |  |  |  | 0.75518 |

**Descripción de Modelos**

Antes de describir cada modelo, hicimos algunas transformaciones que estaban útiles para todo el trabajo práctico. La primera estaba de dividir el conjunto de entrenamiento entre una parte de entrenamiento y de prueba según la ratio 80-20, para generar nuevos datasets que pudimos usar directamente para todos los modelos. Además, hemos generado dos datasets de entrenamiento y de prueba que fueron lematizados para poder usarlos después.

**1 – Bayes Naïve**

Para ese modelo, hicimos primero una búsqueda de los hiperparámetros optimizados para vectorizar las criticas cinematográficas. Por eso, usamos un antiguo modelo de Bayes Naïve, para hacer un cross\_val\_score sobre diferentes configuraciones de los parámetros min\_df y ngram\_range, entre TFIDF Vectorizer y Count Vectorizer, y entre la lematización o no.

Después de tener el mejor vectorizer posible, busquemos los hiperparámetros optimizados del modelo de Bayes Naïve, es decir el parámetro Alpha, con un Randomized Search, y después un Grid Search para afinar el parámetro al 0.05 más próximo.

Después, pudimos hacer una predicción del conjunto final con ese modelo optimizado y entrenado sobre los buenos conjuntos (lematizados o no, según la configuración del vectorizer, y con los mejores hiperparámetros para el conjunto de entrenamiento).

Generalmente, ese modulo estaba bastante rápido y con buenos resultados, pero no fue el mejor para nosotros. Quizás, debíamos probar otra técnica de “normalización” y vectorización de los datos para obtener mejor resultados.

**2 – Random Forest**

Para ese modelo, hicimos primero también una búsqueda de los hiperparámetros optimizados para vectorizar las criticas cinematográficas. Por eso, usamos un antiguo modelo de Random Forest, para hacer un cross\_val\_score sobre diferentes configuraciones de los parámetros min\_df y ngram\_range, entre TFIDF Vectorizer y Count Vectorizer, y entre la lematización o no. Sin embargo, esa búsqueda estaba muy lenta, así decidimos tomar soló 10% del conjunto de entrenamiento para optimizar esos hiperparámetros, lo que puede reducir la eficacia de nuestro modelo, pero permite obtener resultados en un tiempo razonable.

Después de tener el mejor vectorizer posible, busquemos los hiperparámetros optimizados del modelo de Random Forest, es decir los parámetros n\_estimators, max\_depth, cirterion, min\_samples\_split et min\_samples\_leaf, con un Randomized Search. Tuvimos los conjuntos enteros para ese entrenamiento, no reducidos, para obtener los mejores resultados posibles, pero el entrenamiento estaba bastante largo.

Después, pudimos hacer una predicción del conjunto final con ese modelo optimizado y entrenado sobre los buenos conjuntos (lematizados o no, según la configuración del vectorizer, y con los mejores hiperparámetros para el conjunto de entrenamiento).

Generalmente, ese modulo estaba bastante lento y resultados medios, entonces hemos rápidamente abandonado ese modelo para enfocar nosotros sobre un modelo mejor.

**3 – XGBoost**

Para ese modelo, no hicimos una búsqueda de los hiperparámetros optimizados para vectorizar las criticas cinematográficas, porque estaba tan lento y consumidor para nuestros ordenadores que estaba imposible tener resultados.

Entonces, hicimos dos versiones: una sin lematización y una con para analizar la diferencia entre ambos.

Por eso, para cada versión, busquemos los hiperparámetros optimizados del modelo de XGBoost, es decir los parámetros n\_estimators, max\_depth, learning\_rate, gamma, subsample et colsample\_bytree, con un Randomized Search. Tuvimos para ese entrenamiento un conjunto reducido para obtener resultados en un tiempo razonable (que fue todavía de algunas horas…).

Después, pudimos hacer una predicción del conjunto final con ese modelo optimizado y entrenado sobre los buenos conjuntos (lematizados o no, según la configuración del vectorizer, y con los mejores hiperparámetros para el conjunto de entrenamiento).

Generalmente, ese modulo estaba supra lento y con malos resultados, entonces hemos rápidamente abandonado ese modelo para enfocar nosotros sobre un modelo mejor.

**4 – Red Neuronal**

Para el modelo de Red Neuronal, antes de explicar todas las optimizaciones que hicimos, vamos a detallar la estructura que elegimos. En cada estructura, empezamos con una capa Input para acordarse al formato de los textos y una capa de vectorización que vamos a explicar después.

Hemos elegido las estructuras posibles siguientes:

* Una capa Dense de activación ReLU y de regularización L2 y una capa Dense de salida de activación sigmoidea y de regularización L2;
* Una sola capa Dense de salida de activación sigmoidea y de regularización L2;
* Un conjunto de capas Convolution, ReLU, Pooling y Dense;
* Una capa Dense de activación ReLU y de regularización L2, una capa Dropout y una capa Dense de salida de activación sigmoidea y de regularización L2.

El conjunto de capas fue imposible a entrenar, su tiempo de ejecución estaba tan largo y costoso que no pudimos hacer nada con eso. Así, preferimos entrenar un modelo más simple y menos costoso. Además, cuando miramos al F1-Score, la mejora estructura es la primera con una capa Dense de activación ReLU y de regularización L2 y una capa Dense de salida de activación sigmoidea y de regularización L2.

Entonces, decidimos optimizar algunos parámetros, como el optimizador y sus hiperparámetros (elegimos Adam, RMSProp y Adadelta), el vectorizer y sus hiperparámetros (elegimos TextVectorization con el output\_mode de TF-IDF), el nombre de epochs y el batch\_size para el fit del modelo y los hiperparámetros de las capas Dense (dense\_length). Además, para evitar el overfitting, usemos un Early Stopping y un Reduce Learning Rate On Plateau para mejorar el entrenamiento.

Para esa optimización, creemos nuestro propio Grid Search para ser adecuado a los modelos de Keras y buscar los mejores hiperparámetros. Después del fit con esos mejores valores, pudimos predecir los sentimientos de nuestro conjunto final.

Finalmente, el modelo de red neuronal fue el mejor, especialmente con la lematización, así que fue lo en el cuál hemos dedicado la mayoría de nuestro tiempo, para poder optimizarlo correctamente.

**5 – Ensemble**

Para el ensamble de modelos, hemos hecho cuatro modelos diferentes para poder ver si uno de ellos era mejor.

Empecemos con un Voting entre dos modelos de Bayes Naïve, dos de Random Forest y dos de XGBoost. Cada vez, elegimos el mejor modelo (sobre el F1-Score de Kaggle) y el más reciente. Cada modelo será ponderado por su resultado de F1-Score para dar más peso a los mejores modelos en la votación del VotingClassifier.

El segundo modelo es un Voting con redes neuronales únicamente porque necesita la creación de un clasificador especial que llamemos KerasVotingClassifier para cargar, entrenar y predecir desde nuestros modelos (ese clasificador genera dos predicciones, por promedio y por mayoría, para guardar lo mejor de los dos). Todos los modelos están también ponderados por sus resultados de F1-Score. Separemos también los modelos que son con y sin lematización para observar si tienen diferentes resultados.

El tercero modelo es un Stacking con dos modelos de Bayes Naïve, dos de Random Forest y dos de XGBoost. Cada vez, elegimos el mejor modelo (sobre el F1-Score de Kaggle) y el más reciente. Así, entrenemos cada modelo separamente antes de usar un modelo de LogisticRegression comó modelo de meta-aprendizaje.

El cuarto y último modelo es un Stacking con redes neuronales únicamente porque necesita la creación de un clasificador especial que llamemos KerasStackingClassifier para cargar, entrenar y predecir desde nuestros modelos y el modelo de meta-aprendizaje de LogisticRegression. Ese ultimo modelo está optimizado por un Grid Search sobre los hiperparámetros C, solver y penalty. Separemos también los modelos que son con y sin lematización para observar si tienen diferentes resultados.

Generalmente, el mejor modelo estaba él de Voting con redes neuronales con lematización, que nos ha dado buenos resultados, pero que no fueron suficientes para superar el score de la red neuronal clásica. Ese modelo fue implementado en los últimos días, pues quizás necesita todavía mejoras para estar perfecto.

**Conclusiones generales**

Aquí deben indicar las conclusiones generales del TP, qué aspectos destacan sobre los datos y sobre los resultados obtenidos. Comentar brevemente qué otras opciones hubiesen explorado y quedaron fuera del alcance de este trabajo

Las siguientes preguntas son una guía de ayuda para que puedan responder en el informe, pero no son las únicas que esperamos que respondan:

* ¿Fué útil realizar un análisis exploratorio de los datos?
* ¿Las tareas de preprocesamiento ayudaron a mejorar la performance de los modelos?
* ¿Cuál de todos los modelos obtuvo el mejor desempeño en TEST?
* ¿Cuál de todos los modelos obtuvo el mejor desempeño en Kaggle?
* ¿Cuál fue el modelo más sencillo de entrenar y más rápido? ¿Es útil en relación al desempeño obtenido?
* ¿Cree que es posible usar su mejor modelo de forma productiva?
* ¿Cómo podría mejorar los resultados?

**Tareas Realizadas**

|  |  |  |
| --- | --- | --- |
| Integrante | Principales Tareas Realizadas | Promedio Semanal (hs) |
| Alan Richmond |  |  |
| Flavian Ferre |  |  |