.UBAfiuba
FACULTAD DE INGENIERÍA

75.06 /95.58 Organización de Datos
Dr. Ing. Rodríguez - 2°C 2023

TP2: Críticas Cinematográficas

Grupo 01 - Los defensores de Pearson

Introducción

El objetivo de este informe es detallar el desarrollo del trabajo realizado donde dado un listado de reseñas cinematográficas se solicitó entrenar un modelo capaz de interpretar críticas en español y determinar para cada una si es positiva o negativa. Este modelo fué puesto a prueba en una competición de Kaggle donde compitió contra los modelos de los otros grupos.

Dataset

Se provee un dataset completo en el archivo **train.csv**. Este cuenta con 50000 registros y tres columnas:

Columna	Tipo	Descripción
ID	entero	ID de la reseña. No aporta valor al análisis.
review_es	texto	La reseña escrita en español. No hay reseñas vacías ni nulas.
sentimiento	texto	Toma los valores “positivo” o “negativo”. Es la columna objetivo. No presenta ningún valor distinto a los mencionados.

Como particularidad, encontramos que el dataset está perfectamente balanceado, encontrando 25000 reseñas positivas y otras 25000 negativas.

También se provee otro dataset en el archivo **test.csv**. Este último no cuenta con la columna *sentimiento*, ya que es la que el modelo debe predecir.

Hipótesis y supuestos

Al inicio y durante el desarrollo del trabajo planteamos ciertas hipótesis a poner a prueba para mejorar los rendimientos.
Entre ellas:

- El inglés es un lenguaje con semántica más sencilla y las librerías están más adecuadas al mismo.
- Las presencia de mayúsculas y signos de puntuación pueden afectar negativamente al rendimiento de los modelos.
- Las palabras con mucha presencia que no puedan clasificarse como positivas o negativas pueden entorpecer el ajuste del modelo.
- El stemming y el uso de stopwords probablemente no ayude.

Preprocesamiento y pruebas

Búsqueda de reviews en inglés

Utilizando palabras clave en inglés identificamos las más de 1500 reseñas que las contienen y las eliminamos del dataset de entrenamiento. En el dataset de entrega solo una reseña está en inglés así que no la tuvimos en cuenta.

Los resultados tras eliminar estas filas fueron positivos, aumentando la precisión del modelo, por lo cual las sacamos.

Traducción de las reviews

Sabiendo que las librerías están más preparadas para trabajar con texto en inglés, sumado a que la semántica del idioma es más sencilla, consideramos la opción de traducir el data frame entero usando la librería de google.

Terminamos descartando la opción ya que no solo la traducción demora mucho, sino que entendemos que no es un camino válido.

Stopwords

Las stopwords son una lista de palabras las cuales puede considerarse que no tienen peso en la valoración de sentimientos. Eliminarlas aumenta la densidad de las palabras persistentes y con relevancia por lo que permite que el modelo se ajuste de mejor manera.

Las stopwords pueden venir dadas por librerías, pero también el usuario puede determinar por su cuenta las palabras a omitir analizando los datos.

- **Librerías:** Utilizando las stopwords en español provistas por nltk realizamos pruebas y notamos que el rendimiento de los modelos decaía, por lo que las descartamos como opción.
- **Propias:** Analizando las palabras con mayor presencia hicimos pruebas eliminando algunas y duplicando la presencia de otras:
    - película: Eliminarla mejoró el rendimiento
    - películas: Eliminarla redujo el rendimiento
    - no: Duplicarla no cambió el rendimiento
    - bien: Duplicarla no cambió el rendimiento

Normalización de texto y limpieza de caracteres

Eliminando mayúsculas, espacios en blanco, signos de puntuación y otros caracteres especiales los modelos podrían performar mejor.

En el caso de NB las mayúsculas y vocales con tilde no afectaron al rendimiento. Los otros cambios empeoraron el rendimiento.

Para los demás modelos, el pasaje de texto a minúsculas significó una leve mejoría en el rendimiento. No así la eliminación de signos de puntuación y demás.

Stemming

La técnica de stemming consiste en la reducción de las palabras a su raíz. De esa forma una familia de palabras se reduce a un único término lo que aumenta su densidad en el conjunto. Por otro lado, se pierde el valor gramatical que aporta la variedad de palabras dentro de una familia.

Aplicar esta técnica redujo considerablemente el rendimiento de todos los modelos con los que se probó, por lo que la descartamos.

Tokenización

La tokenización consiste en la división de un texto en unidades más pequeñas (tokens). Esto es un paso fundamental para el uso de los modelos de procesamiento de lenguaje natural. Los tokens pueden ser palabras, letras, frases, entre otros.
En nuestro caso realizamos una tokenización por palabras utilizando TFIDFVectorizer y CountVectorizer. El segundo nos ofreció mejores resultados localmente y en la competición.

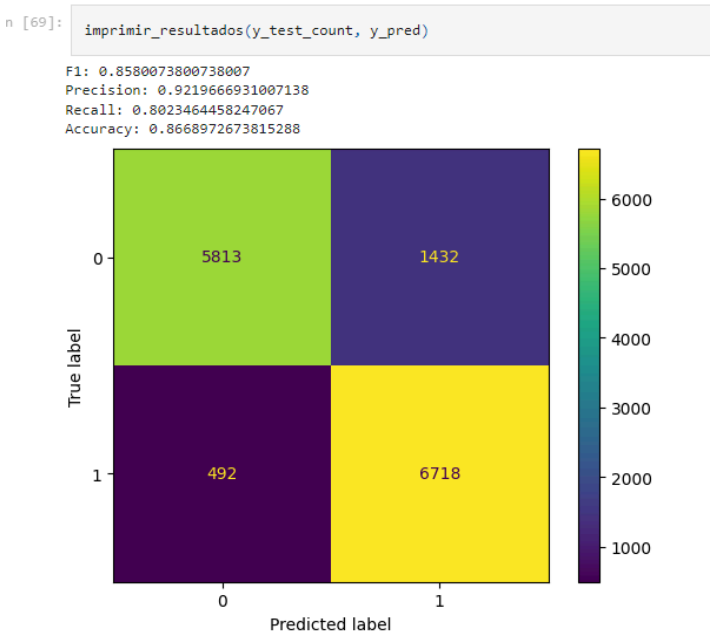
Otras pruebas

Como un extra, también probamos hacer un submit en Kaggle hardcodeando todas las reseñas como “Positivo”. Esto terminó dando un puntaje de 0.7. Con esto sabemos que de ese 60% de datos con el que se evalúa en el leaderboard, 70% son positivos.

O sea, mínimo el 42% de las reseñas son positivas y mínimo el 18% son negativas.

Esto nos fue útil para “boostear” el puntaje en el leaderboard público centrándonos un poco más en acertar las predicciones positivas sin tener tan presente las negativas.

Este es un ejemplo de un modelo que entrenamos específicamente para escalar en la tabla pública.



Criterio de evaluación de los modelos

Se nos otorga un dataframe de train y otro de test. Este data frame de test vamos a llamarlo **entrega** para simplificar la explicación.

1. Separar el dataframe de train en train y test.
2. Entrenar el modelo utilizando train y evaluarlo utilizando test.
3. Si los números mejoran respecto al intento anterior, entrenar el modelo con el dataframe de train entero (train + test). La métrica que más nos interesa es el accuracy, ya que es la misma con la que evalúa Kaggle.
4. Evaluar el dataframe de entrega y exportar los resultados.

Cuadro de Resultados

Medidas de rendimiento en el conjunto de TEST:

Modelo	F1-Test	Precision Test	Recall Test	Accuracy	Kaggle
Bayes Naive	0.853	0.839	0.868	0.851	0.766
Random Forest	0.842	0.824	0.859	0.839	0.762
XGBoost	0.864	0.856	0.871	0.863	0.722
Red Neuronal	0.861	0.864	0.857	0.861	0.731
Ensamble	0.863	0.878	0.849	0.865	0.752

Descripción de Modelos

Bayes Naive (Bernoulli)

El modelo consiste en el teorema de Bayes, el cual lo calcula como

$$p(C|F_1,\dots,F_n) = \frac{p(C)\,p(F_1,\dots,F_n|C)}{p(F_1,\dots,F_n)}.$$

siendo C la variable objetivo (sentimiento), y las Fi las variables independientes.

De los distintos predictores que otorga Naive Bayes, terminamos utilizando el de Bernoulli, ya que la variable a predecir es binaria, como los posibles resultados de dicha variable aleatoria.

Hiperparámetros

Se buscaron los siguientes hiperparámetros:

- alpha: es el factor de suavizado de las probabilidades calculadas por el modelo. En otras palabras, evita que alguna de ellas sea 0, o que sea 1, en el segundo caso para evitar el overfitting.
- class_prior: especifica las probabilidades a priori de las clases en lugar de calcularlas automáticamente a partir de los datos de entrenamiento.

Los hiperparámetros hallados son: { alpha: 0.65, class_prior: [0.1, 0.5] }

Random Forest

El modelo Random Forest, como ya se ha explicado en el TP1, consiste en combinar distintos árboles de decisión, y utilizando bagging, decidir cuál es el valor de la variable sentimiento.

Hiperparámetros

Se buscaron los siguientes hiperparámetros:

- **Criterion:** es la función que se utiliza para medir la calidad de la división. Los valores utilizados fueron {gini, entropy, log_loss}
- **Min_samples_leaf:** mínimo de muestras para estar en una hoja. Se probó con 1, 5, 10 y 20.
- **Min_samples_split:** mínimo de muestras necesarias para que un nodo se pueda dividir Los valores a elegir fueron 2, 4, 10, 12, 16 y 20.
- **N_estimators:** número de árboles del forest. Se probó con 10, 20, 30, 40, 50, 60, 70, 80, 90 y 100.
- **Max_depth:** determina la profundidad máxima del árbol y se probó con el rango 15 a 50 inclusive.

Los hiperparámetros hallados son:
{
'n_estimators': 90,
'min_samples_split': 10,

'min_samples_leaf': 5,
'max_depth': 39,
'criterion': 'gini'
}

XGBoost

El algoritmo consiste en un árbol jerárquico en el cual cada nodo hijo aprende del error del padre para no cometer sus mismos errores.

Hiperparámetros

Se investigaron los siguientes parámetros:

- **"learning_rate":** [0.01, 0.1, 0.2],
- **"max_depth":** [3, 5, 7],
- **"n_estimators":** [100, 200, 300]

Red neuronal

Una red neuronal es un modelo conformado por capas de neuronas conectadas entre sí total o parcialmente con sus capas vecinas.

Arquitectura

Tras probar distintas combinaciones de capas y cantidades de neuronas llegamos a un modelo de 4 capas:

- Densa de 650 a la que se le elimina el 10% de las conexiones
- Densa de 250 a la que se le elimina el 20% de las conexiones
- Densa de 35
- Densa de 1

A las primeras tres capas se les aplica una regularización con el objetivo de reducir el overfitting penando las conexiones neuronales con mayor peso.

Hicimos la prueba poniendo primero una capa más chica, luego una más grande y luego otra mas chica, pero terminó funcionando peor. Hicimos muchas pruebas retocando la eliminación de conexiones y la cantidad de neuronas hasta llegar a este resultado.

Ensamble

Para realizar el ensamble, decidimos hacer uno del tipo voting, utilizando los tres clasificadores previamente entrenados (Naive Bayes, Random Forest y XGBoost).

Para ello, aprovechando que ya teníamos las predicciones de cada modelo, decidimos realizar a mano una función que dada una entrada, decida cuál es el sentimiento que gana la misma.

Conclusiones generales

Tras el análisis, desarrollo y puesta en marcha de nuestros modelos, consideramos haber logrado un buen trabajo. Los modelos alcanzan un accuracy que creemos aceptable y que cumple con los objetivos planteados.

Nuestro modelo ganador fue la red neuronal, que fue con la que alcanzamos los mejores resultados tanto localmente como en el leaderboard público de Kaggle.

En la competencia seleccionamos el mejor submit de ensamble y el mejor submit local de red. El de ensamble dentro de los de su tipo fue el mejor tanto localmente como en Kaggle, pero el de red no. La mejor red en Kaggle, que nos dió un puntaje de 0.787 no la elegimos ya que su accuracy está 0.2 puntos por debajo de la red que seleccionamos. En la tabla pública ese modelo dió ese número tan alto ya que predice muy bien los positivos, pero es regular para los negativos. Por lo mencionado anteriormente sabemos que en la tabla pública clasificar bien los positivos está mucho mejor compensado.

Respecto a la selección y armado del ensamble, como se había visto en clase, un conjunto de malos predictores pueden dar un buen predictor, y es lo que ha sucedido con el modelo de ensamble. Al hacer el voting con los modelos que mejor score habían obtenido en Kaggle, nos dio en Kaggle mismo un score de 0.71; en cambio, cuando utilizamos los modelos que quedaron finalmente en la notebook de la entrega, los cuales habíamos buscado hiperparametros pero el resultado en Kaggle no era el mayor, el score en Kaggle fue superador, confirmando lo que observó Galton en su experimento con las vacas.

Análisis exploratorio

Realizar un análisis exploratorio siempre es fundamental para comprender los datos con los que contamos y en base a eso planificar una estrategia para alcanzar los objetivos planteados.

Dada la poca complejidad del dataset trabajado, el análisis no se basa tanto en la composición del mismo y la relación entre sus columnas, sino en el contenido de las reseñas.

La columna objetivo (sentimiento) puede optar únicamente por dos valores (positivo/negativo). Ninguna fila la tiene en nulo ni por fuera de los valores esperados. Además verificamos que el target se encuentra balanceado (la mitad de las filas tienen un sentimiento *positivo* y la otra mitad *negativo*).

No existen reseñas nulas tampoco, aunque gracias a este análisis detectamos la presencia de reseñas en inglés siendo que la inmensa mayoría están en español (lo cual además es coherente con el nombre de la columna; **review_es**).

Resultados del preprocesamiento

El preprocesamiento junto a la selección de parámetros y arquitectura de los modelos son la parte fundamental que distingue el rendimiento de los modelos. Tras poner a prueba distintas técnicas de preprocesado, terminamos seleccionando las que mejor rendimientos nos ofrecieron:

- Convertir el texto a minúsculas
- Eliminar la palabra “película”
- Vectorizar por palabra

Mientras que las siguientes técnicas resultaron en un empeoramiento del rendimiento y por lo tanto fueron descartadas:

- Stemming
- Stopwords de librerías
- Eliminación de caracteres especiales y acentos

Mejor rendimiento en TEST

El modelo que mejor rendimiento otorgó (basándonos en la métrica de accuracy) sobre el dataframe de test es el modelo del ensamble, el cual juntó los resultados de Naive Bayes, XGBoost y Random Forest.

Mejor rendimiento en Kaggle

El modelo que más puntaje F1 otorgó sobre el dataframe que utiliza Kaggle, es el modelo de Naive Bayes sorprendentemente, y como mencionamos aquí debajo, es el que menos problemas ocasionó para entrenarlo.

Entrenamiento más sencillo

El modelo más sencillo de entrenar y a su vez el más rápido fue Naive Bayes. El entrenamiento consistió simplemente en jugar con los pocos hiperparámetros, ajustandolos buscando un mejor rendimiento.

Con la grán variedad de combinaciones que probamos, tres de los cinco hiperparámetros siempre tomaron el valor por defecto (force_alpha true, binarize 0.0 y fit_prior true). Esto redujo aún más la complejidad.

Este modelo fue no solo el más sencillo de entrenar, sino con el que encabezamos el leaderboard de Kaggle durante más de la mitad de la competencia, por lo que fue sumamente útil.

Entrenamiento más complejo

En contraparte al punto anterior, la red neuronal fue la más compleja de desarrollar. La grán flexibilidad que ofrece poder determinar la arquitectura de la misma, junto a la necesidad de gestionar los recursos (como la memoria), jugar con la taza de aprendizaje, el uso de librerías que requieren actualización manual, entre otros factores fue lo que complejizó su implementación.
Si bien fue la más compleja, también fue la que mejor resultado terminó ofreciendo mejores resultados tanto localmente como en Kaggle.

Modelo productivo

Consideramos que dado el puntaje obtenido no sería una buena idea aplicar el modelo productivamente. Creemos que sería necesario un accuracy superior a 0.9 en test para que el modelo pueda considerarse fiable.

Posibles mejoras

Por limitaciones temporales no pudimos realizar todas las pruebas que nos hubiera gustado. Algunas de ellas son:

- Definir stopwords y palabras/frases relevantes acorde al negocio. En este caso, comparaciones con el cine de Tarantino por ejemplo podrían aportar un sentimiento mucho más positivo del que nuestro modelo está siendo capaz de interpretar. Y como esta existen otro tipo de referencias propias del negocio que podríamos detectar e incrementar su peso.

- Utilizar redes de aprendizaje profundo. Si bien pudimos dar un mínimo vistazo, no llegamos a probar entrenar modelos de esta forma. Creemos que un modelo de este tipo podría posicionarse entre los mejores.

Competencia en Kaggle

Como grupo desde el día 1 nos planteamos competir buscando el podio en Kaggle. Siendo conscientes de que el leaderboard público no suele coincidir con el privado, muchos de nuestros submits se centraron en mejorar la performance en este primero. Esto nos permitió encabezar la lista durante más de la mitad de la competencia.

Esto aportó un extra de emoción al trabajo ya que siempre es divertido competir, e incentivaba a mejorarnos cada vez que éramos alcanzados.

No obstante, así como competimos por el leaderboard, también fuimos avanzando nuestros modelos a conciencia de que el leaderboard era engañoso y a sabiendas de que los mejores puntajes no necesariamente reflejaban a los mejores modelos, por lo que nuestros mejores modelos los determinamos en base al accuracy obtenido localmente.

Problemáticas durante el desarrollo

- Las limitaciones de recursos por parte de Collab fue un limitante en cuanto a la escala de las pruebas a realizar. De igual forma, somos conscientes de que uno no puede utilizar recursos infinitos y que la idea es balancear rendimientos y costos.
- El idioma español. Las librerías no suelen estar tan preparadas para el español como para otros idiomas. Sumado a esto, ciertas tareas del preprocesamiento, como el reemplazo de caracteres especiales deben considerar caracteres como las vocales con tilde que en inglés pueden limpiarse. Librerías como Stanza (https://stanfordnlp.github.io/stanza/v130performance.html) cuentan

con herramientas interesantes para aplicar, pero que lamentablemente no están desarrolladas para el lenguaje en cuestión.
- El fin de cuatrimestre nos limitó bastante en el tiempo que pudimos destinar a hacer más comprobaciones, experimentos y pruebas. Esto nos obligó a cerrar varias ventanas de posibilidades para centrarnos en la obtención de un trabajo en condiciones pero no tan ambiciosos como nos hubiera gustado.

Tareas Realizadas

Integrante	Principales Tareas Realizadas	Promedio Semanal (hs)
Fernandez Ignacio	Entrenamiento Naive Bayes Entrenamiento Red Neuronal Elaboración Informe	11
Molina Taiel	Entrenamiento Random Forest Preprocesamiento de datos Elaboración Informe	10
Vera Sebastian	Entrenamiento XGBoost Entrenamiento modelo de ensamble Elaboración de informe	10