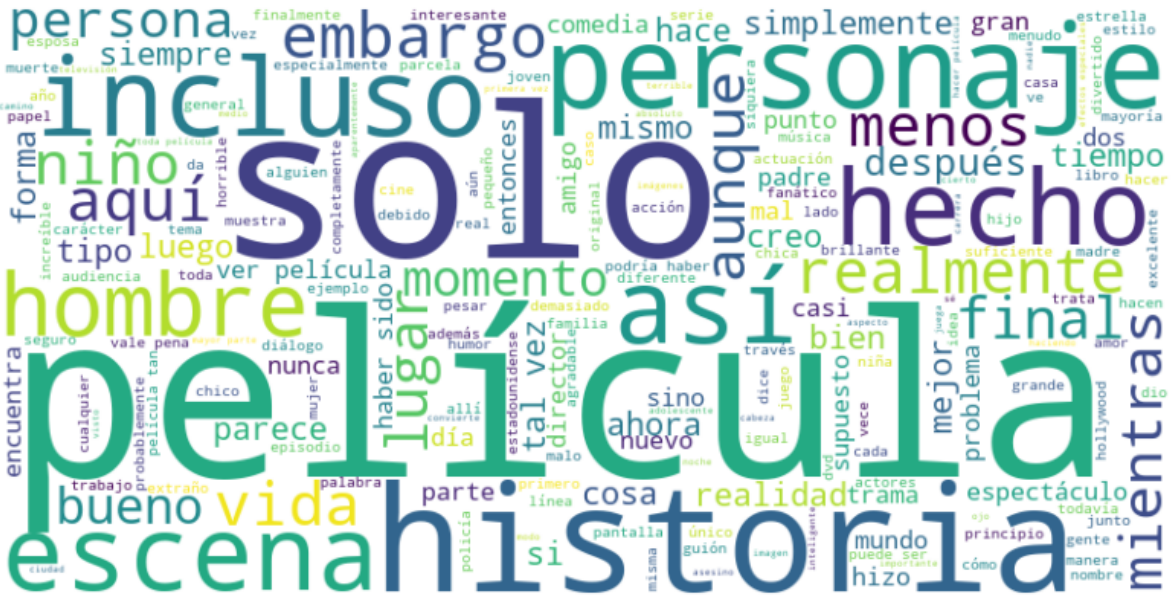
Materia: 75.06/95.58 - Organización de Datos
Cuatrimestre: 1° cuatrimestre 2023
Grupo: 27

Padrón	Apellido y Nombre
94727	Jarmolinski, Arian
108405	Porro, Joaquín
97538	Bordón Villavicencio, Fernando Nahuel

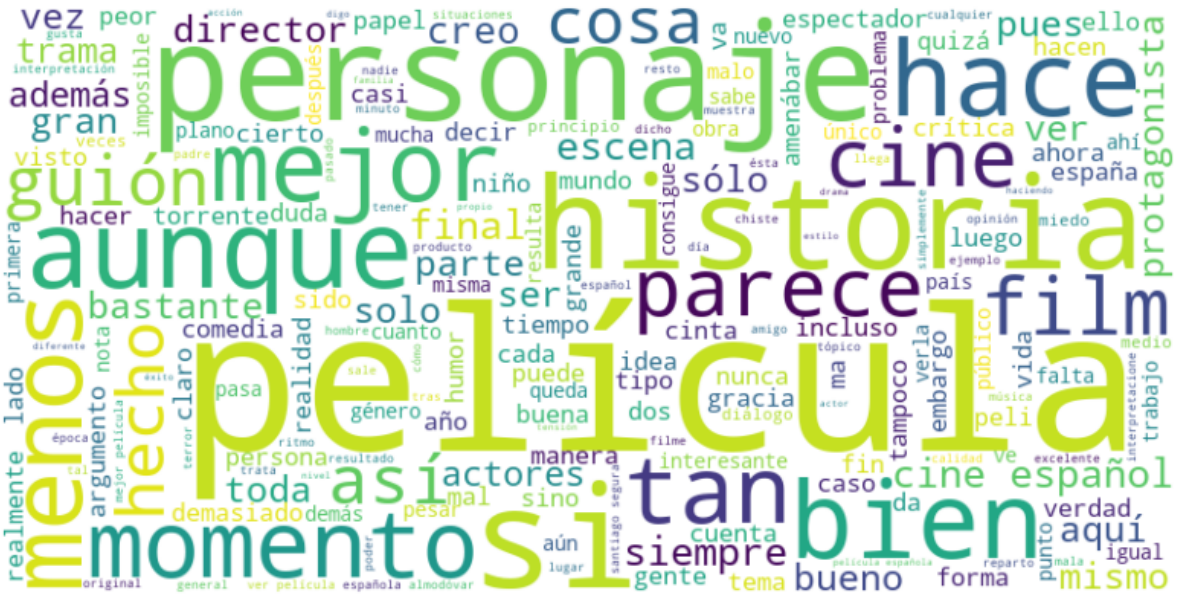
El siguiente trabajo consistió en predecir si una crítica es positiva o negativa según reviews de críticas cinematográficas en español.

Se muestran WordCloud del set de train y de test.

Word Cloud del set de train



Word Cloud del set de test



Para ello se utilizó bibliotecas de NLP para pre-procesar los datos:

- Se utiliza el vocabulario del set de train para entrenar los modelos. Si bien se podría haber utilizado también los del set de test (consiguiendo, tal vez, mejores resultados para la competencia), consideramos que en un caso real, no dispondremos de dicho set.
- Se detectaron algunas reviews que no estaban en español con la biblioteca langdetect y se las filtró.
- Se utilizó la biblioteca nltk para filtrar por stopwords y realizar stemming para reducir las palabras y llevarlas a su raíz idiomatica. También se filtró los signos de puntuación y se transformó el texto a minúscula.

Luego del filtrado se dividió el set de entrenamiento en train y test para entrenar los modelos, en una proporción de 75/25.

Entonces , se procedió a vectorizar el texto para que pueda ser aceptado por los modelos. Para ello se utilizó CountVectorizer, que vectoriza como Bag of Word, y luego, obteniendo mejores resultados, TfidfVectorizer, que es una mejora de Bag of Words, que le da peso inversamente proporcional a la frecuencia que aparece el término. Se probó cantidades de max_features de 10k y 25k.

Luego de tener las reviews vectorizadas, se procedió a reducir la dimensionalidad de las mismas utilizando PCA para poder agilizar los tiempos de entrenamiento y se busco una cantidad de dimensiones final que al menos represente más del 84% de explicabilidad total de los datos.

- Con 25k de max_features BOW alcanzo tener 3k de dimensiones luego de aplicar PCA
- Con 10k de max_features TFIDF necesito 4k de dimensiones luego de aplicar PCA

El target se lo convirtió de la variable categoría a binario mediante LabelEncoder.

Se procedió a realizar la búsqueda de los hiperparametros y entrenar los modelos con n_jobs=-1 para poder utilizar concurrencia y agilizar los tiempos de entrenamiento para ello se utilizó cross validation y una cantidad de fold igual a 5. Se midió el score con la métrica f1

A continuación se muestran los mejores resultados obtenidos con TFIDF de 25k y PCA con 4k de dimensión final en:

Bayes Naive

Se utilizó GridSearchCV y MultinomialNB. Se encontró:

‘alpha’ = 1

Random Forest

Se utilizó RandomForestClassifier con RandomizedSearchCV y n_iter entre 50 y 100, obteniendo:

'n_estimators': 160,

'min_samples_split': 4,

'min_samples_leaf': 1,

'max_features': 'sqrt',

'max_depth': 9,

'criterion': 'gini',

'ccp_alpha': 0.00125

XGboost

Se utilizó RandomizedSearchCV y XGBClassifier. Este es el modelo que más demoró en entrenarse, dependiendo de las dimensiones de entrada se tardo 9hs en la busqueda con n_iter entre 30 y 40. La búsqueda resultó en:

'subsample': 0.9,

'reg_lambda': 0.001,

'reg_alpha': 0.001,

'n_estimators': 300,

'max_depth': 4,

'learning_rate': 0.1,

'gamma': 0,

'colsample_bytree': 1

Red neuronal

Se utilizó GridSearchCV y KerasClassifier utilizando previamente para normalizar los datos StandardScaler.

Se probaron diferente arquitecturas, modificando la cantidad de neuronas por capa y agregando más capas intermedias, y la arquitectura final fue elegida por ser la que mejor resultados nos dio fue tal que, las funciones de activación de las capa de entrada e intermedias sean ReLu y la de salida sigmoidea ya que se trata de predecir un target binario. Además, se utilizó la

técnica de regularización con dropout=0.5 para apagar la mitad de las neuronas en cada capa de neuronas y evitar el sobreajuste del modelo

La arquitectura e hiperparametros encontrados son:

Capa densa de 64 neuronas relu y regularización del 50%

Capa densa de 32 neuronas relu y regularización del 50%

Capa densa de 1 neurona sigmoidea

Los hiperparametros encontrados fueron:

'batch_size': 16,

'epochs': 40

Ensambles

Luego de tener los modelos entrenados, se procedió a probar ensambles de 3 modelos. Entre los modelos basados en árboles de decisión, RandomForest y XGBoost, se eligió uno, para obtener mayor diversidad, resultando en que XGBoost obtuvo mejores mediciones.

Entonces, se realizó ensamble de tipo Stacking entre los modelos de XGBoost, bayes naive y red neuronal.

Para la combinación de modelos se probó con distintos estimadores finales: XGBClassifier, LogisticRegression, RandomForestClassifier, SVC y MLPClassifier.

Obtuvimos los mejores resultados con MLPClassifier que es un modelo de red neuronal, seguido por XGBClassifier.

A continuación se muestra una tabla con los mejores resultados obtenidos de los diferentes modelos y el resultado obtenido por Kaggle

Modelo	***Accuracy***	***Recall***	***Precisión***	***F1***	***F1-Kaggle***
Bayes	0.842	0.831	0.848	0.840	0.7263
RandomForest	0.776	0.777	0.778	0.778	0.70808
XGBoost	0.841	0.853	0.835	0.844	0.72029
Red Neuronal	0.854	0.846	0.862	0.854	0.72979
Stacking(XGBoost)	0.868	0.866	0.872	0.869	0.74568
Stacking(red)	0.872	0.883	0.866	0.874	0.75014

Como conclusión del trabajo, observamos que las métricas obtenidas por kaggle al evaluar el set de test difieren bastante para todos los modelos.

El modelo que mejor predijo el target fue el modelo de ensamble con red neuronal.

Observamos que se requiere cantidad de tiempo considerable al buscar hiperparametros en modelos como XGBoost y ensamble de modelos.

Como punto de mejora para el pre-procesamiento se puede utilizar la descomposición del texto en n-gramas y proceder a ver como performa como también utilizar embeddings que son algunas técnicas que mejoran el NLP