UNIVERSIDADE
DE VIGO

Análisis de Hate Speech en comentarios de Steam

MIGUEL GMEZ GARCÍA

Índice de contenidos:

Indice

Indice ilustraciones

1. Relevancia y objetivos de análisis

Las grandes comunidades en internet están formadas por una gran variedad de personas que deben convivir y comunicarse. Está en el interés de las compañías cuyo modelo de negocio se basa en estas interacciones (o en las que estas interacciones son un componente importante) que todo el mundo pueda sentirse cómodo y respetado.

En el caso de Steam, es interesante porque un gran componente de su modelo de negocio está en los comentarios/reviews de los videojuegos que se encuentran en la tienda. Cada uno de ellos tiene una sección en la que los usuarios pueden comentar sus experiencias y hablar con otros usuarios para tomar una decisión de compra.

Pues está en el interés de Valve (empresa matriz de Steam) mantener estas conversaciones bajo un cierto control para evitar insultos, discriminación…. Cosas que espantarían tanto a los usuarios como a las empresas que deciden poner su juego a la venta en esta tienda.

Este control no es viable que se realice solamente por humanos ya que Steam tiene cientos de miles de videojuegos y cientos de miles de usuario comentando diariamente. Por tanto, es interesante valorar formas automatizadas de controlar esto, y una de las más interesantes es la utilización del machine learning.

Por lo tanto, el objetivo principal de este proyecto es el siguiente. A través de la utilización de un dataset de entrenamiento formado por una serie de tweets etiquetados en función de si contienen o no Hate Speech, se comprobará el desempeño de una serie de modelos basados en distintos algoritmos de machine learning.

El punto final será la aplicación de el/los algoritmos que mejor funcionen a un dataset con una serie de reviews de Steam.

2. Materiales y métodos

Para la elaboración de este trabajo se han utilizado dos dataset:

- El dataset de entrenamiento, que contienen una serie de tweets etiquetados en función de si tienen o no hate speech, en este tenemos 31.962 filas.
- En el caso del dataset de reviews de steam que contiene 6.417.106 y varias columnas que no nos son útiles y por ello son eliminadas antes de tratar el dataset.

Posteriormente, y en ambos casos, se realiza una limpieza y adaptación de los textos. Para ello transformamos todo el texto a minúsculas, eliminamos los enlaces que pueda haber en los tweets o comentarios, eliminamos caracteres especiales que no nos aportan nada, como @. Además, y visto que la parabra 'User' tenía demasiada importancia (esto se puede ver gracias a los wordcloud) también la eliminamos de los tweets.

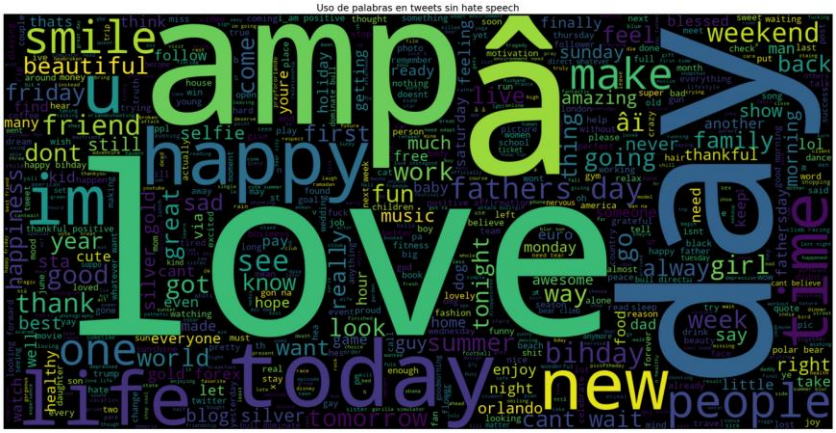

Ilustración 1 Palabras sin hate

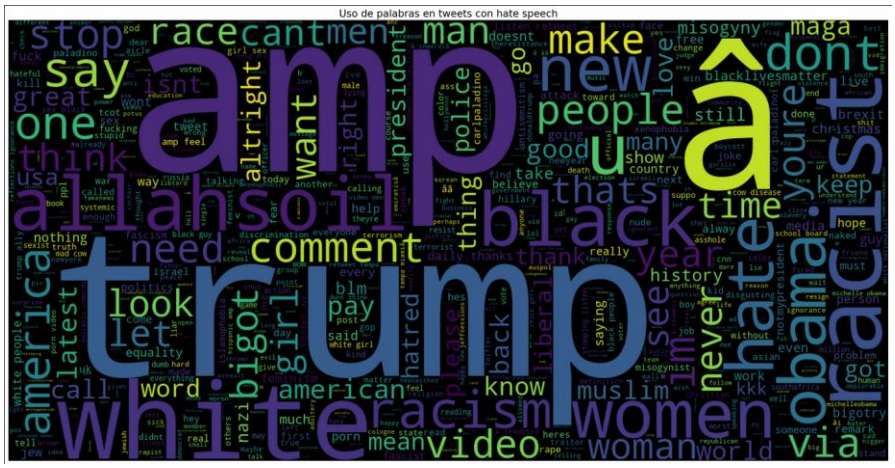

Ilustración 2 Palabras con hate

Luego se realiza una tokenización de las palabras y, finalmente, eliminamos las stopwords (en inglés).

Ahora se realiza una comprobación de la proporción entre tweets con hate speech (1) y sin el (0)

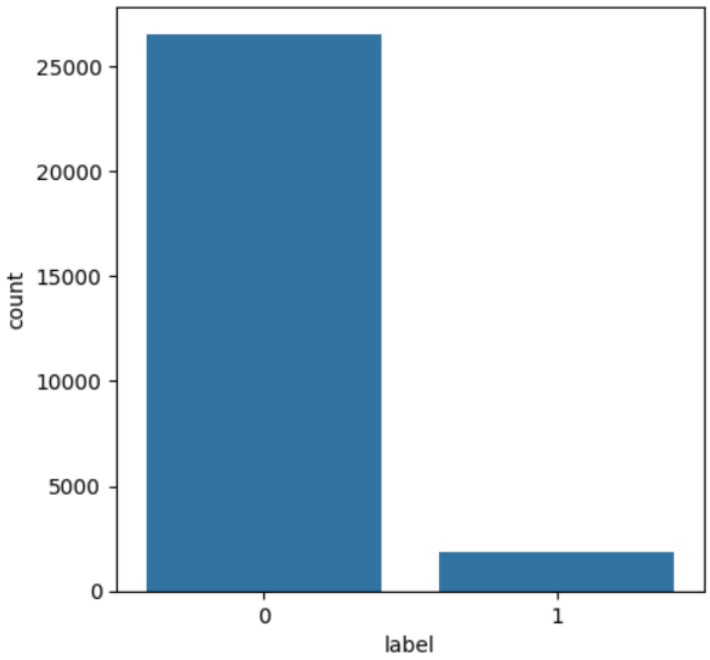

Ilustración 3 Balance inicial

Como se puede apreciar, hay un desbalanceo entre ellos. Como queremos utilizar este conjunto de datos como fuente de entrenamiento, tendremos que balancear el dataset para evitar problemas con las predicciones del futuro modelo que se entrene con sus datos.

La solución a esto es un algoritmo denominado SMOTE.

Una vez hecho esto, se procede a realizar la división del conjunto de datos en los de test y los de entrenamiento. La proporción utilizada ha sido un 30% se utilizará para test y el resto para train.

Ahora se procede a la realización de diversas pruebas con distintos modelos, para ello se han probado distintos algoritmos:

2.1. KNeighborsClassifier

Para la selección del mejor algoritmo se ha hiperparametrizado de la siguiente forma:

- 'n_neighbors':np.arange(1, 50),
- 'weights':['uniform', 'distance'],
- 'algorithm':['auto', 'ball_tree', 'kd_tree', 'brute']}

Dando como resultado ideal {'algorithm': 'auto', 'n_neighbors': 8, 'weights': 'distance'} con una nota de 0.8

2.2. LogisticRegression

Para la selección del mejor algoritmo se ha hiperparametrizado de la siguiente forma:

- 'penalty': ['l2','l1','elasticnet'],
- 'solver':['sag','lbfgs','newton_cg'],
- 'class_weight':['dict','balanced'],
- 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
- 'solver':['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'],
- 'multi_class':['auto', 'ovr', 'multinomial']

Dando como resultado ideal {'C': 0.1, 'class_weight': 'balanced', 'multi_class': 'auto', 'penalty': 'l1', 'solver': 'liblinear'} con una nota de 0.51

2.3. Stochastic Gradient Descent (SGD)

Para la selección del mejor algoritmo se ha hiperparametrizado de la siguiente forma:

- 'penalty': ['l2','l1','elasticnet'],
- 'l1_ratio': [0, 0.05, 0.1, 0.2, 0.5, 0.8, 0.9, 0.95, 1],
- 'epsilon': [0.1,0.2,0.3,0.4,0.15],
- 'alpha': [10 ** x for x in range(-6, 1)]

Dando como resultado ideal {'alpha': 0.1, 'epsilon': 0.1, 'l1_ratio': 0.9, 'penalty': 'elasticnet'} con una nota de 0.66

2.4. Support Vector Machines

Para la selección del mejor algoritmo se ha hiperparametrizado de la siguiente forma:

- 'C': [0.1, 1, 10, 100, 1000],
- 'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
- 'kernel': ['linear', 'poly', 'rbf', 'sigmoid']

2.5. Random Forest

Para la selección del mejor algoritmo se ha hiperparametrizado de la siguiente forma:

- 'max_depth': [80, 90, 100, 110],
- 'max_features': [2, 3],
- 'min_samples_leaf': [3, 4, 5],
- 'min_samples_split': [8, 10, 12],
- 'n_estimators': [100, 200, 300]

2.6. Bagged Decision Trees

Para la selección del mejor algoritmo se ha hiperparametrizado de la siguiente forma:

- 'n_estimators':[1500,400,500],
- 'random_state':[42,52],
- 'max_samples' :[1.0,1.5,2.0]

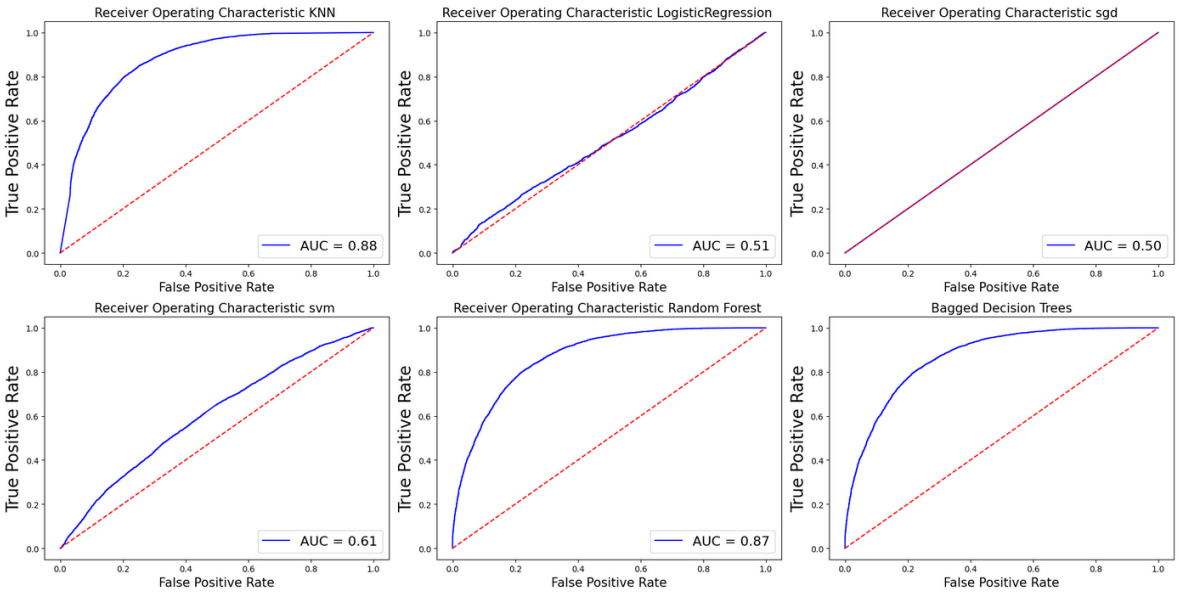

Ilustración 4 Curvas ROC

Aquí Podemos ver una representación del desempeño de cada uno de los algoritmos que se presentaron anteriormente, a través de la utilización de curvas ROC.

En la curva roc se representa en el eje y la tasa de verdaderos positivos, la división entre positivos verdaderos/(verdaderos positivos + falsos negativos). En el eje y tenemos la tasa de falsos positivos, la división entre falsos positivos/(falsos positivos + verdaderos negativos).

Por tanto, cuanto más alta sea la curva, mejor es el método de clasificación que se está evaluando. Y cuanto más alto sea el valor de AUC (area under the curve) mejor.

En esta última gráfica podemos ver que los modelos ideales son Random forest y KNC

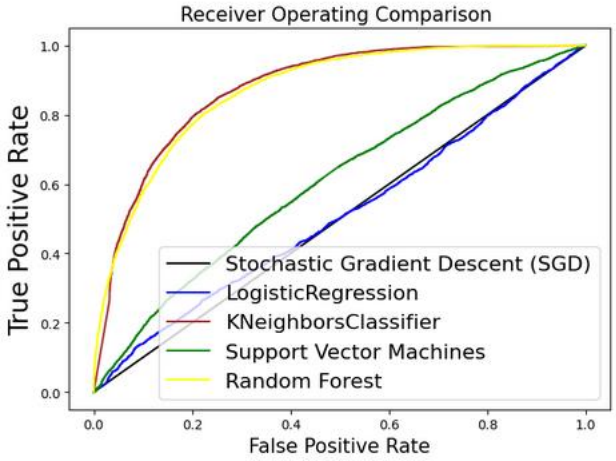

Ilustración 5 Comparativa ROC

Análisis de sentimientos

Como bonus en este proyecto, se ha implementado también un algoritmo que permite evaluar como de positivos o negativos son los sentimientos que se transmite en cada uno de los tweets.

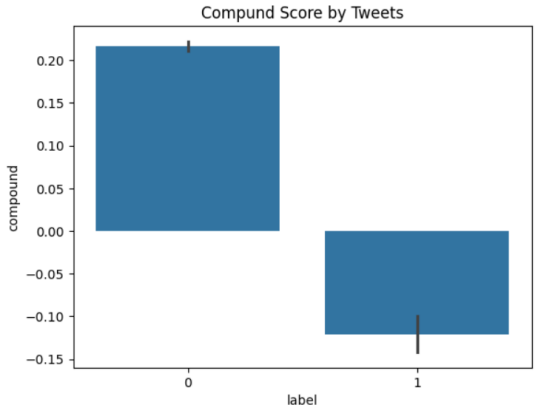

Ilustración 6 Análisis de sentimientos

Como se puede apreciar, son los tweets catalogados como hate speech los que contienen un lenguaje más negativo.

3. Resultados y conclusión

3.1. Matrices de confusión

Las matrices de confusión son una representación que nos permite observar varias cosas. En el eje x se representa la predicción del algoritmo (0 = sin hate, 1 = con hate) y en el eje y cuál es el valor verdadero.

Esto nos permite hacer una distribución de las predicciones con las que podemos conocer los falsos positivos, falsos negativos, verdaderos positivos y verdaderos negativos entre otras métricas. La más importante, y la que se utiliza durante todo el proyecto para calcular la nota de cada algoritmo, es la f1-score.

3.2. KNeighborsClassifier:

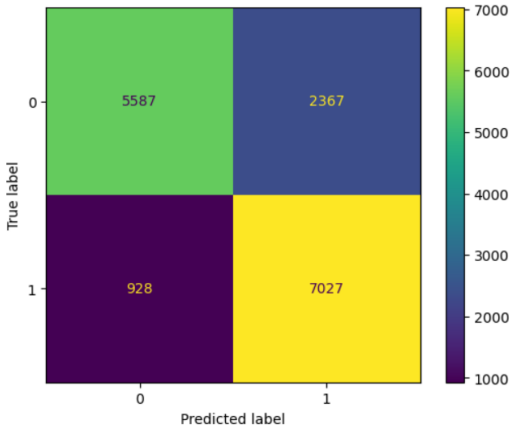

Ilustración 7 Matriz confusión KNC

En la diagonal con colores vivos vemos los casos positivos que se predijeron positivos, y los negativos que se predijeron negativos.

En violeta tenemos unos pocos casos en los que la label verdadera era 1 (tiene hate) pero el algoritmo detectó que no. Esta sería la peor situación, que hay que evitar al máximo, porque estaríamos dejando pasar mensajes con odio.

En la casilla azul se muestra el número de casos que no tenían hate pero se etiquetaron como que si lo tenían. Aunque sean errores, no se consideran igual de graves ya que es preferible eliminar de más que dejar pasar odio.
Así, el f1-score es 0.79

3.3. Stochastic Gradient Descent

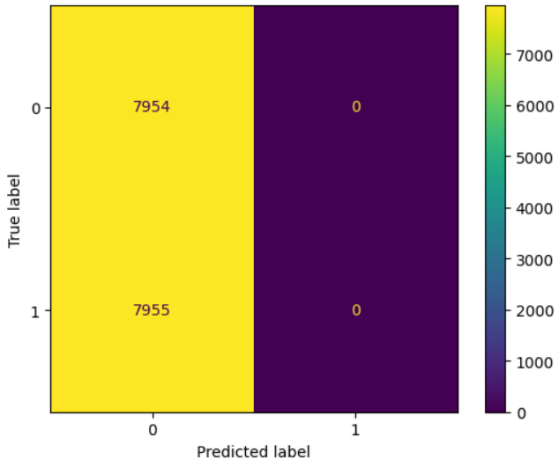

Ilustración 8 Matriz confusión SGD

En este caso el resultado vemos un modelo que para nada nos sirve para nuestro caso ya que, a pesar de que no ha clasificado ningún tweet como que si tenía hate sin tenerlo, si ha clasificado mal una gran cantidad de tweets diciendo que no tenían hate pero si lo tenían, lo cual es el peor error posible.

Su nota es 0.66 y queda completamente descartado.

3.4. LogisticRegression

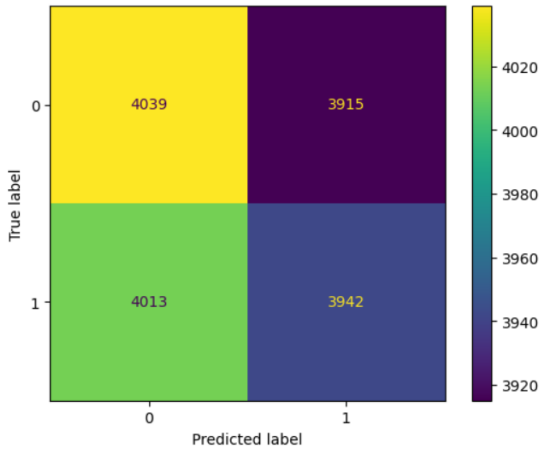

Ilustración 9 Matriz confusión LogisticRegression

Este modelo, a pesar de no ser bueno y tener peor nota que el SGD, es mejor que él ya que a pesar de que hay cierto número de tweets clasificados como que tienen hate sin tenerlo, el número de ellos que se clasificaron erróneamente como sin hate es mucho menor.

La nota de este algoritmo es 0.51

3.5. Support Vector Machines

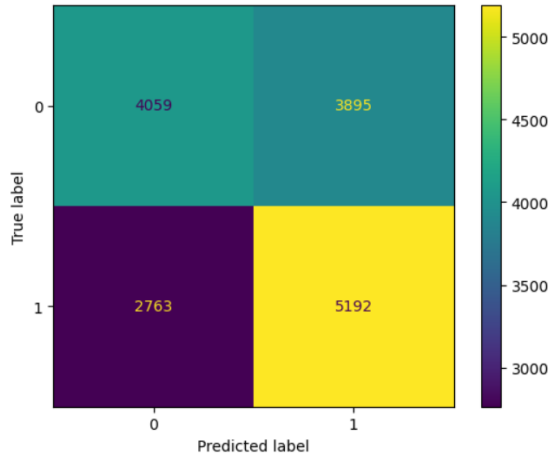

Ilustración 10 Matriz confusión SVC

Caso similar al anterior, mejora en que hay algo menos de casos en los que la predicción falla diciendo 0 siendo 1. Su nota es de 0.57

3.6. Random Forest

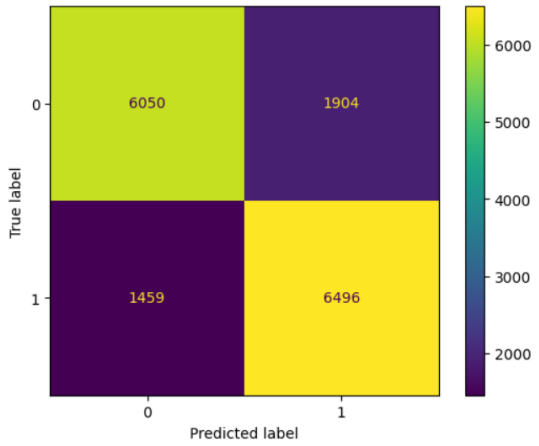

Ilustración 11 Matriz Random Forest

Este caso, siendo mejor que los anteriores, no consigue superar al primer caso (KNC) y se queda en una nota de 0.78. No hay discusión posible, ya que empeora en la métrica más importante que es en los tweets con hate que marca como que no lo tienen.

3.7. Bagged Decision Trees

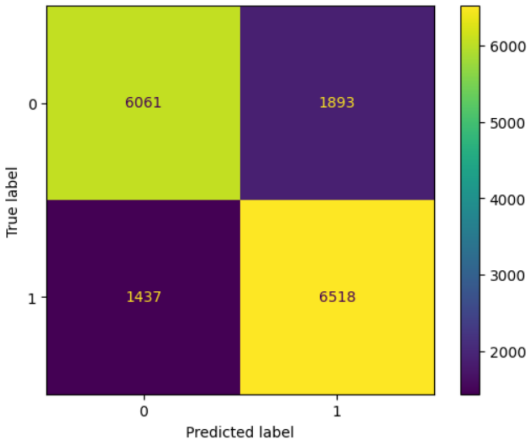

Ilustración 12 Matriz Bagged decision trees

El resultado es muy similar al del primer modelo, su nota también lo es (0.79)

4. Experimento: Intento de aplicación de algoritmos de regresión linear:

Como pequeña ampliación se ha intentado aplicar a nuestro caso de estudio una serie de algoritmos de regresión linear para ver cómo sería su desempeño. Para ellos hemos evaluado

LinearRegression, Ridge , DecisionTreeRegressor , KNeighborsRegressor y se concluye que solamente DecissionTreeRegressor y KNeighborsRegressor podrían ser mínimamente válidos para nuestro caso.

Esto lo podemos ver claramente en la siguiente gráfica en la que se representa cada modelo a través de la distribución de su MSE:

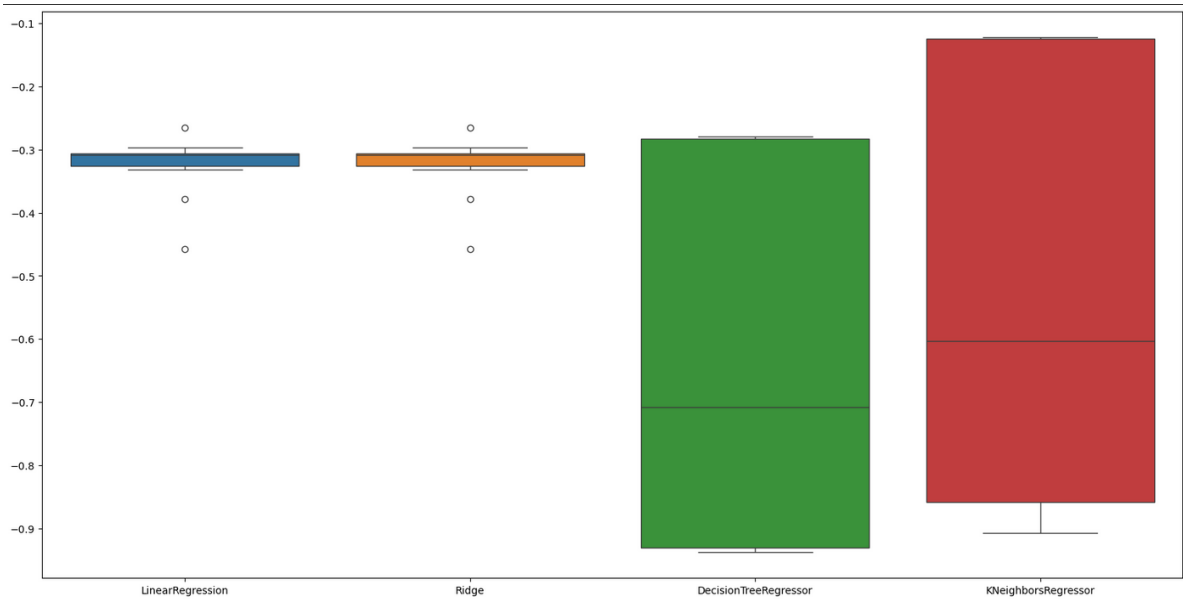

Ilustración 13 Comparación Regresión linear

5. Aplicaciones al dataset de Steam

Se procede a la a aplicación de los mejores modelos anteriores a dataset de steam.

En este primer caso vemos la proporción obtenida al aplica KNeighborsClassifier a este caso (su nota fue 0.7986):

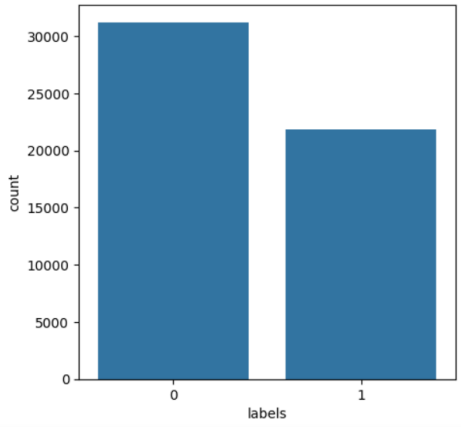

Ilustración 14 Resultados finales KNC

En este segundo, utilizamos BaggingClassifier con un DecisionTreeClassifier como estimador su nota fue de 0.790.

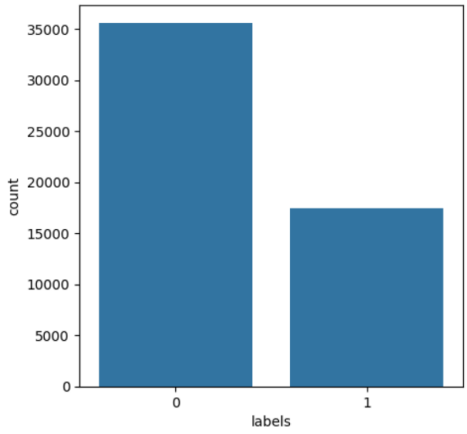

Ilustración 15 Resultados finales BaggingClassifier

En este caso hay una menor cantidad de tweets categorizados como hate. Si recordamos las matrices de estos modelos veremos como el más fiable de los dos sería el primero.

Finalmente, nuestra última implementación se ha realizado con un modelo basado en un RandomForestClassifier, que es el otro que nos dio unos buenos resultados antes. Aquí lo vemos:

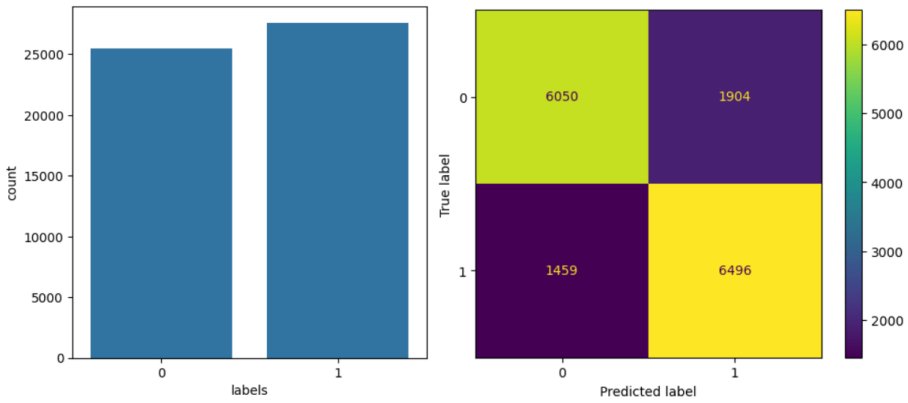

Ilustración 16 Resultados finales RandomForest

Aquí se pone la matriz al lado de la gráfica de resultados para que se aprecie como este modelo tiende desde el principio a sobredimensionar los mensajes en los que dice detectar odio. Esto se translada en que es el único en el que, tras analizar el conjunto de datos de Steam, concluye que hay más mensajes con odio que sin él.

Conclusiones

Tras la utilización de algoritmos de machine learning muy distintos vemos como no todos tienen los mismos fuertes, en este caso han sido BaggingClassifier y KNeighborsClassifier los que mejor nos han funcionado. Todo depende de los datos de entrada, imágenes, números, palabras…

Así, analizando nuestra casuística, nuestros algoritmos estrella son kNN que puede considerarse un sistema de votación, donde la etiqueta de clase mayoritaria determina la etiqueta de clase de un nuevo punto de datos entre sus vecinos 'k' (donde k es un número entero) más cercanos en el espacio.

Por otro lado, nuestra otra mejor opción es la utilización de un BaggingClassifier con 1500 estimators, que serán DecisionTreeClassifier. Este es un algoritmo completamente distinto que se basa en un árbol de decisión que es una estructura similar a un diagrama de flujo donde un nodo intermedio representa una característica (o atributo), la rama representa una regla de decisión y cada nodo hoja representa el resultado.

Son forma de trabajar muy distintas que, sin embargo, son ambas eficaces para nosotros. Con lo cual podemos deducir que, dentro de la gran variedad de alternativas, nunca nos podemos quedar con una concreta sin probar las demás en cuanto a machine learning se refiere.

También concluimos que, viendo las gráficas de análisis de sentimientos, está claro que la negatividad reina en los mensajes con hate.

Bibliografía

"What is Accuracy, Precision, Recall and F1 Score?," *www.labelf.ai*. https://www.labelf.ai/blog/what-is-accuracy-precision-recall-and-f1-score

scikit-learn, "scikit-learn: machine learning in Python," *Scikit-learn.org*, 2019. https://scikit-learn.org/stable/

A. Shafi, "KNN Classification Tutorial using Sklearn Python," *www.datacamp.com*, Feb. 2023. https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn