Índice general

Índice de figuras

Índice de cuadros

Acrónimos

- **LR**: Logistic Regression
- **XGB**: XGBoost
- **MNB**: Multinomial Naive Bayes
- **BoW**: Bag of Words
- **Tf-Idf**: Term frequency – Inverse document frequency

1. Datos: Análisis y Preproceso

Se puede encontrar el original en `https://es.overleaf.com/project/626b9937f5f5d5274f042ac7`

1.1. División entre Train y Dev

Conjunto De Datos	% de instancias	Num. de instancias
Train	80	9599
Dev	20	2400

1.1. Cuadro: División Train y Dev

1.2. Distribución de las clases en cada conjunto

Conjunto De Datos	Clase Neg	Clase Neutra	Clase Pos.
Train	5567	2304	1728
Dev	1392	576	432

1.2. Cuadro: Distribución Train y Dev

1.3. Descripción del preproceso

La aplicación de las técnicas que vamos a implementar es importante para el análisis de sentimiento. Estas nos permiten identificar patrones y tendencias en las opiniones de los clientes.

En primer lugar, hemos aplicado preprocesos como convertir los valores a unicode (útil cuando hay caracteres especiales o acentos), convertir las fechas en valores de tiempo UNIX, mapeado de valores, e imputación de valores faltantes.

En segundo lugar, hemos hecho un preprocesado del lenguaje natural para reducir la complejidad del texto y mejorar la eficacia del algoritmo. Primeramente, se ha realizado una traducción de los emoticonos a texto real. Esto se debe a que hemos pensado que los emoticonos podrían aportar información para valorar los tuits, aunque existen configuraciones en las que hemos decidido que los emoticonos se eliminen para ver si se incrementaban las figuras de mérito. Seguidamente, se han eliminado signos de puntuación y se han cambiado las mayusculas por minúsculas (es decir, se ha normalizado el texto) para que el algoritmo no distinga entre palabras iguales con mayúsculas diferentes. Al eliminar palabras como los stop words, se reduce la dimensión del conjunto de datos y se mejora la precisión del análisis. Por último, hemos lematizado las palabras para reducir la complejidad del texto, es decir, hemos reducido las palabras para quedarnos con sus raíces.

Finalmente, hemos utilizado dos técnicas de preproceso de lenguaje natural para poder evaluar los tuits. Por un lado, hemos utilizado la técnica de bag of words que se utiliza para convertir el texto en un array de frecuencia de palabras. Esta es importante para convertir el texto en formato que puede ser

procesado por los algoritmos. Por otro lado, hemos utilizado TF-IDF para reducir el peso de las palabras frecuentes, y aumentar el peso de las palabras que son más relevantes para el análisis de sentimientos como 'bueno' y 'malo'.

1.4. Primeros resultados

En las primeras versiones del modelo no coordinabamos los vectores de tf-idf entre crear y probar el modelo, por lo tanto los resultados que obteníamos eran bastante mejorables, además solo se utilizaba una versión de Naive-Bayes y no existía la opción de hacer BoW ni de borrar los emoticonos, solo traducirlos. Empleando esta información el f-score obtenido rondaba entre 0.44 y 0.49.

1.5. Descripción del Proceso de Submuestreo o Sobremuestreo

Inicialmente nuestro algoritmo tenía problemas cuando la muestra era demasiado pequeña, por lo que a la hora de determinar si el modelo va a emplear over o under sampling este observa la cantidad de instancias. Para más de 10000 instancias se utilizará Oversampling, y para menos Undersampling, para evitar situaciones como la ocurrida con las instancias dadas para la preparación de este proyecto, con las cuales el test solo contenía una única instancia de clase neutral por lo que no era capaz de generar un modelo.

1.6. Instrucciones

- Ejecutar installer.sh para instalar todos los paquetes necesarios para el script.
- El fichero README.md contiene las llamadas a realizar para generar o probar el modelo respectivamente.

2. Algoritmos, link a la documentación y nombre de los hiperparámetros empleados

2.1. Algoritmos empleados: Breve Descripción

Hemos empleado los siguientes algoritmos con los siguientes hiper-parámetros:

- **Gaussian Naive Bayes**:
  - Hiperparámetros: priors, var smoothing
  - Link: Sklearn GaussianNB

  En el caso de Gaussian Naive Bayes, se asume que los datos utilizan una distribución normal (o Gaussiana). Por ello, es importante discretizar los valores, porque algunos valores tienen una gran cantidad de decimales y no están uniformemente distribuidos, lo que puede resultar en un sesgo hacia unos valores en concreto.

- **Mixed Naive Bayes**:
  - Hiperparámetros: alpha, fit prior, class prior
  - Link: mixed-naive-bayes MixedNB

  En el caso de Mixed Naive Bayes, se permiten diferentes distribuciones para diferentes características, por lo que se pueden utilizar diferentes funciones de densidad de probabilidad. Por lo tanto, no es necesario discretizar los datos, ya que el modelo de Mixed Naive Bayes puede manejar características continuas y discretas sin problemas.

- **Multinomial Naive Bayes**:
  - Hiperparámetros: alpha, fit prior, class prior
  - Link: Sklearn MultinomialNB

  Este modelo se usa para la clasificación de texto y tiene en cuenta la frecuencia de las palabras. Intenta predecir la clase en función de esa frencuencia.

- **Bernoulli Naive Bayes**:
  - Hiperparámetros: alpha, binarize, fit prior, class prior
  - Link: Sklearn BernoulliNB

  Este modelo considera la presencia o ausencia de cada característica.

- **Complement Naive Bayes**:
  - Hiperparámetros: alpha, fit prior, class prior, norm

- Link: Sklearn ComplementNB

Es una variante del NB Multinomial y se ha creado para manejar los conjuntos de datos desequilibrados.

- **Logistic Regression**:
  - Hiperparámetros: penalty, random state
  - Link: Sklearn LogisticRegression

La Regresión Logística utiliza una función de activación para transformar la salida del modelo en una probabilidad. Es una curva que separa las instancias entre sí como una frontera, para determinar si una instancia pertence a una clase u otra.

Hemos observado durante las pruebas que Logistic Regression tiende a clasificar más instancias como positivas que los modelos basados en Naive-Bayes. Y que de entre las distintas versiones de Naive-Bayes, la que consistentemente obtenía mejores resultados ha sido **Bernoulli Naive Bayes**.

2.2. Resultados sobre el Development

En esta sección se presentan los resultados obtenidos para el development.

2.2.1. Optimizando los resultados de la clase negativa

Algoritmo	Combinación hyperparámetros	Prec	Rec	F-sco
Logistic Regression	tf-idf y Traducir Emojis/Emoticonos	0.73	0.72	0.73

2.1. Cuadro: Resultados peso de clase negativa

2.2.2. Discusión

El f-score correspondiente a los comentarios negativos ha sido de 0.75. Esta composición ha sido la que ha obtenido mejores resultados de manera consistente entre las pruebas de los distintos miembros del grupo. Es la versión en la que originalmente más confiábamos ya que es la que más cantidad de información tiene en cuenta a la hora de determinar las clases.

2.2.3. Sin optimizar ninguna clase en particular

Algoritmo	Combinación hyperparámetros	Prec	Rec	F-sco
Logistic Regression	tf-idf y Traducir Emojis/Emoticonos	0.73	0.72	0.73

2.2. Cuadro: Resultados equilibrados

2.2.4. Discusión

Esta composición es igual a la mencionada en el apartado anterior por lo que no se considera necesario volver a desarrollarla. Cabe destacar que en una de las pruebas realizadas por los integrantes, la composición de hiperparámetros Logistic Regression, BoW y Borrar Emojis/Emoticonos, ha conseguido los mismos valores para Precision, Recall y F1-Score. De esto deducimos que algunos de los modelos que se mencionarán en el Anexo 2 podrían teoricamente obtener valores superiores a los que aparecen aquí.

2.3. Conclusión

El mismo modelo obtiene los mejores resultados en ambos apartados de este punto, pero eso no significa que siempre vaya a ser el mejor a la hora de distinguir las clases. Por esto además de entregar este modelo enviaremos también aquel que empleando Naive-Bayes obtiene el mejor resultado (Naive-Bayes, tf-idf y Traducir Emojis/Emoticonos). Aparentemente el preproceso tiene más influencia que el algoritmo a la hora de obtener resultados, aunque Logistic Regression ha sido consitentemente superior a Naive-Bayes a lo largo de todas las combinaciones.

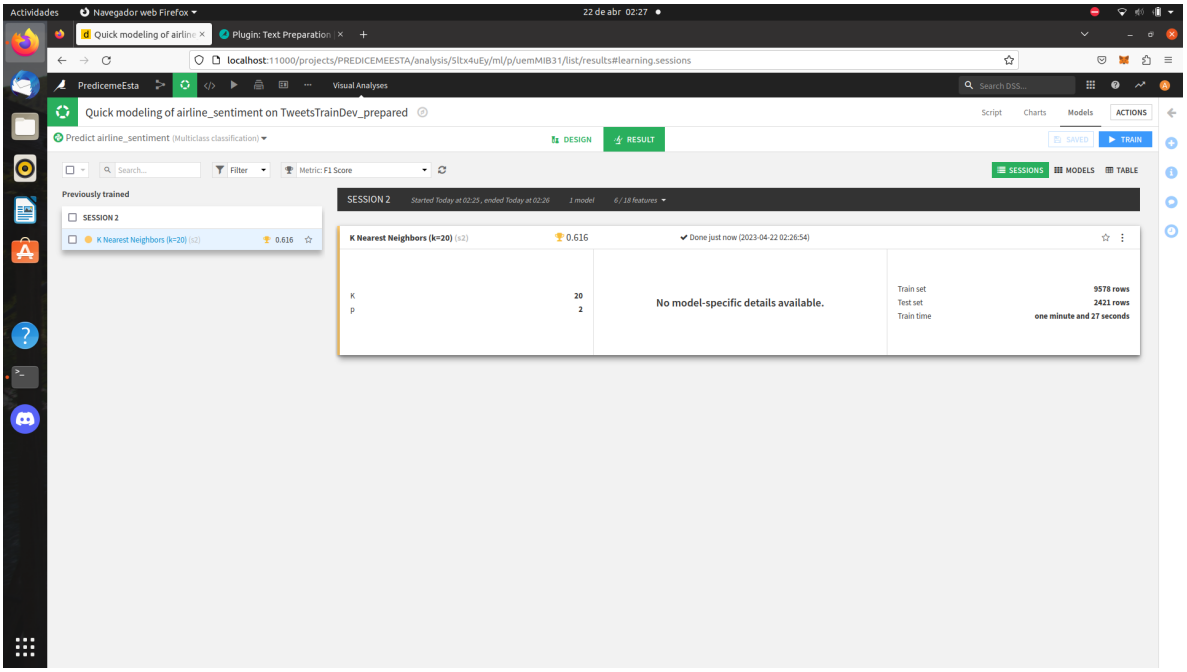
3. Anexos

3.1. Anexo 1: Baseline

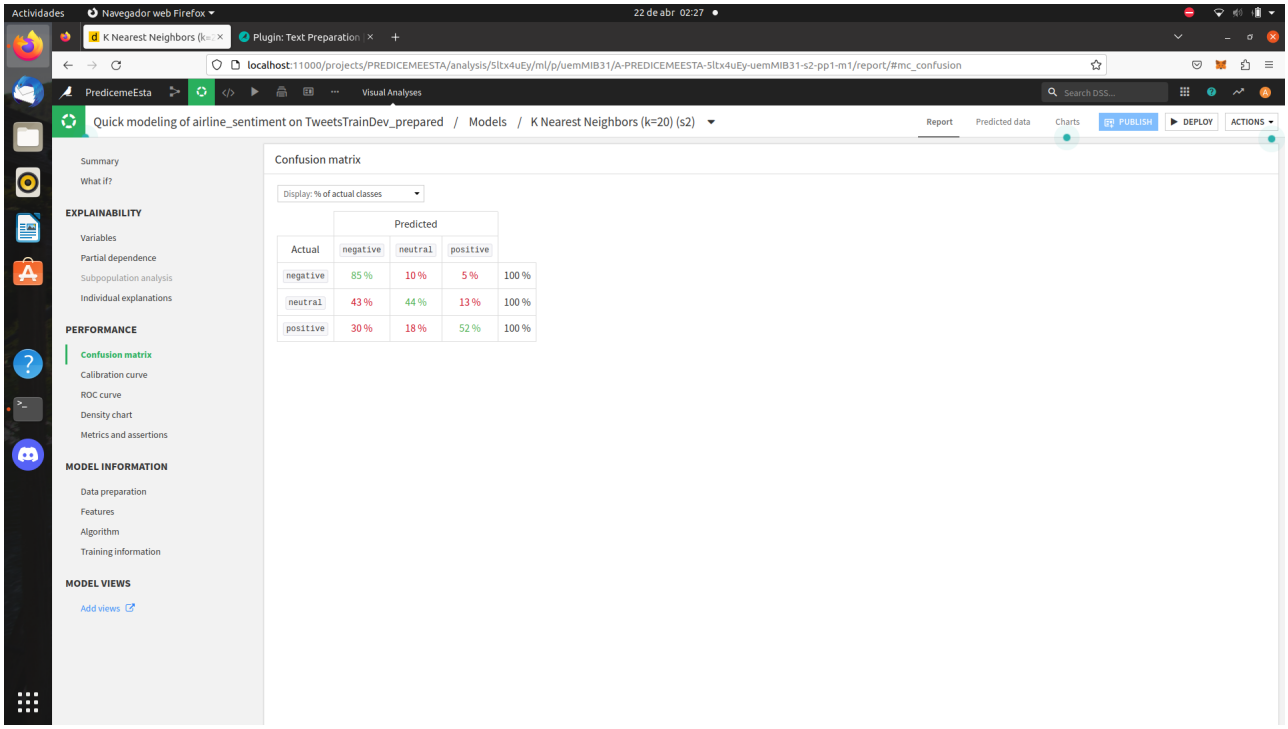
Para comprobar la efectividad de nuestro modelo frente a otros, hemos generado un Baseline empleando un modelo knn generado mediante la aplicación de Dataiku.

3.1.1. Resultados obtenidos

Empleando un modelo básico KNN hemos obtenido un F1-Score del 62 % aproximandamente (Fig 3.1).



3.1. Figura: Resultado KNN

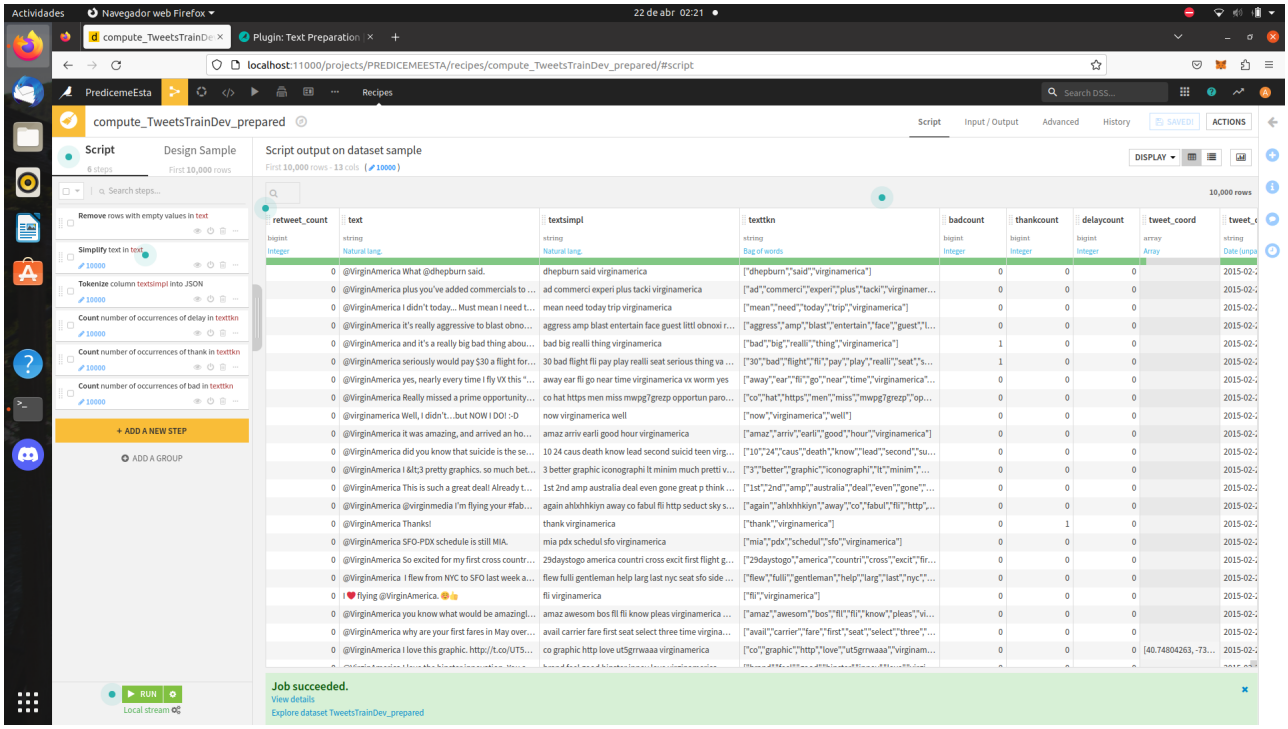


3.2. Figura: Matriz de confusión del Modelo

3.1.2. Preproceso e hiperparámetros

Dado que la tarea no trata de optimizar el modelo KNN, el preproceso utilizado no ha sido muy riguroso, y se ha limitado a características básicas para permitir generar un BoW (Fig 3.3):

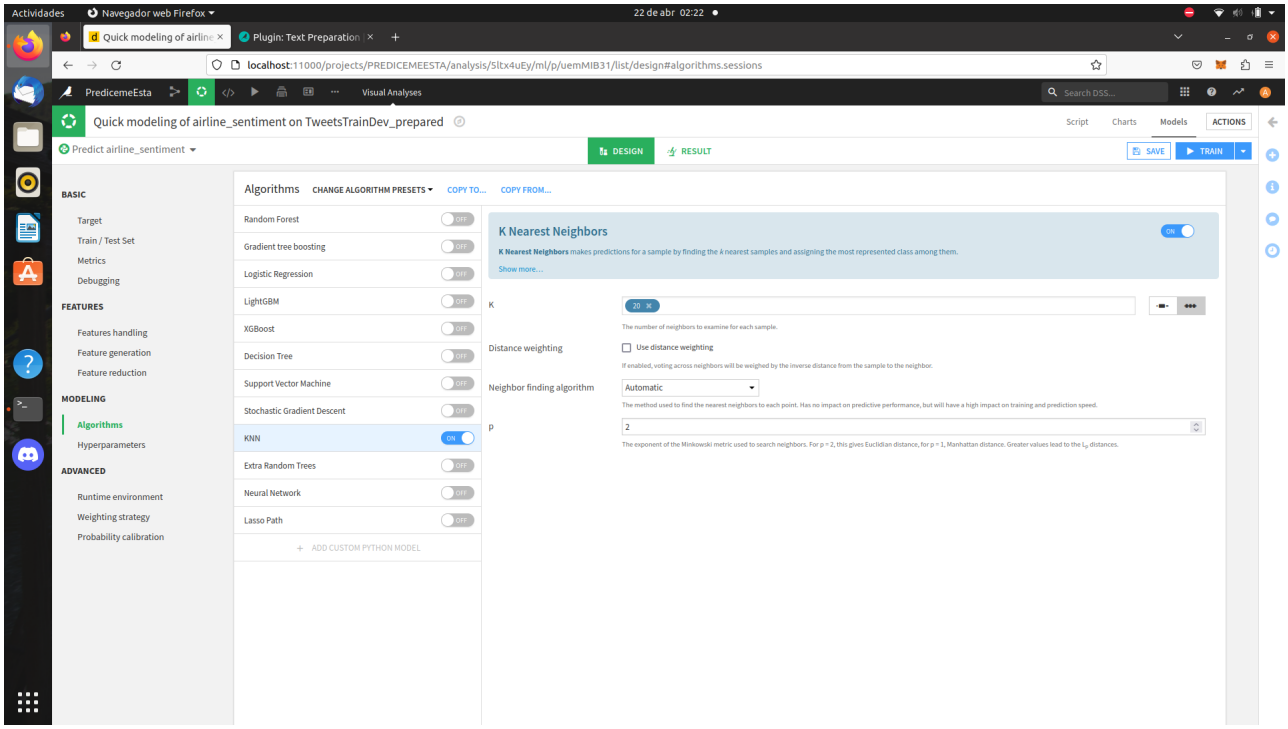
- Se simplifica el texto eliminando símbolos de puntuación.
- Se eliminan las 'stopwords' que no aportan información relevante.
- Se tokenizan las palabras y se reducen a su raíz.
- Seleccionamos algunas palabras que pueden denotar emociones positivas o negativas y las contamos.



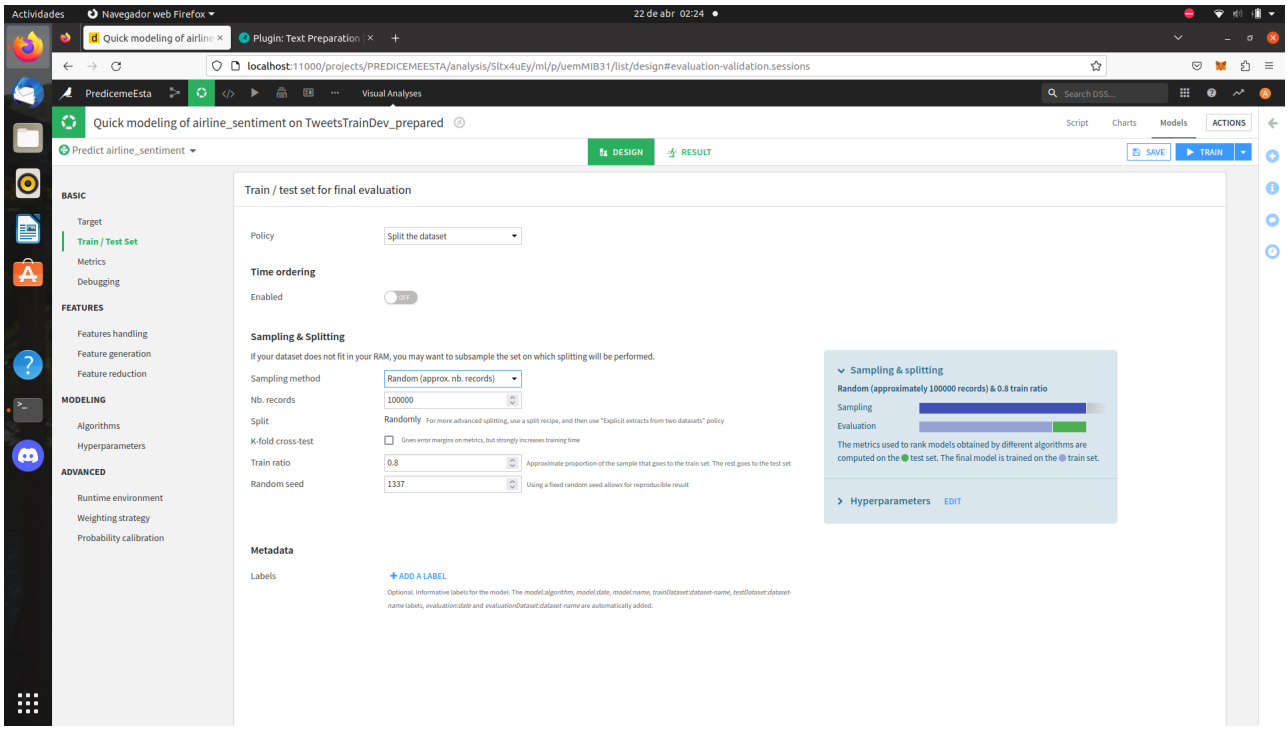
3.3. Figura: Preproceso de los datos

Además de esto la configuración del método a utilizar ha sido la siguiente:

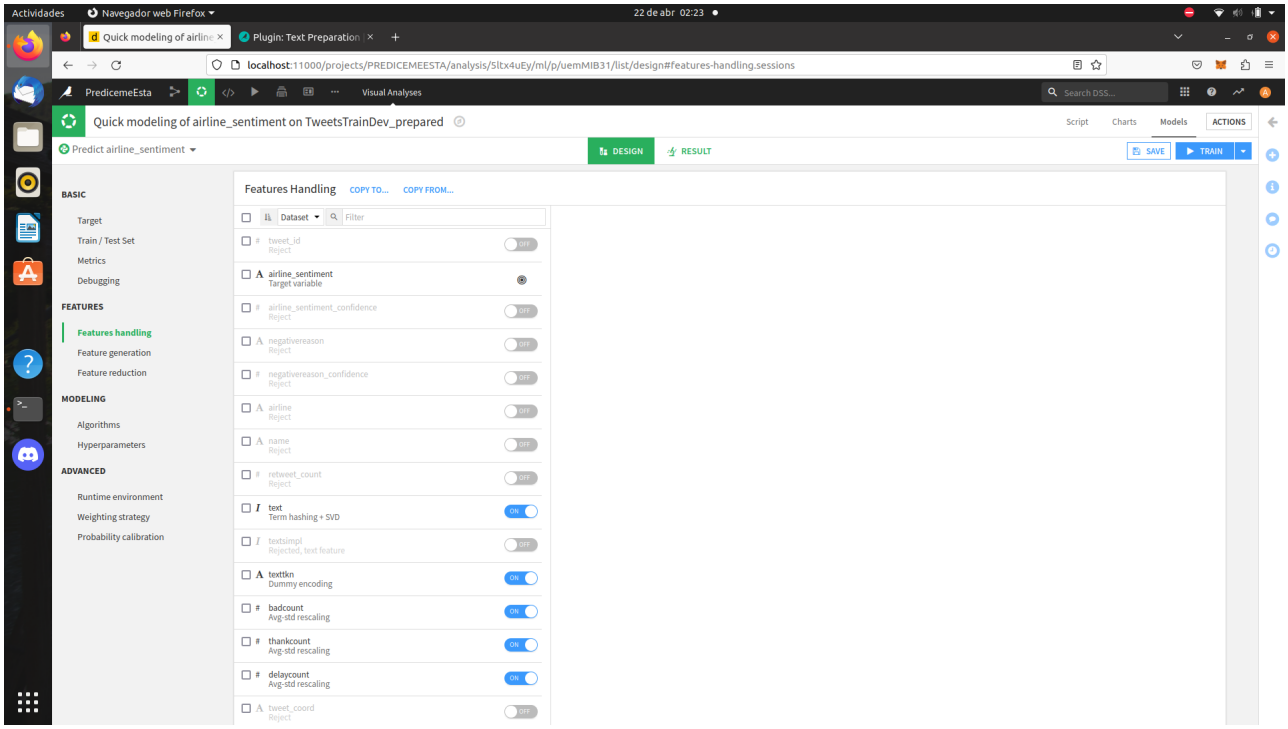
- Se han tenído en cuenta los 20 vecinos más cercanos para determinar la clase (Fig 3.4).
- La división train/test ha sido un 80/20, y se han seleccionado las filas al azar (Fig 3.5).
- Las columnas que se han tenído en cuenta han sido el texto, las palabras tokenizadas y los contadores de palabras (Fig 3.6).
- Se ha empleado el F1-Score para determinar la mejor aproximación (Fig 3.7).



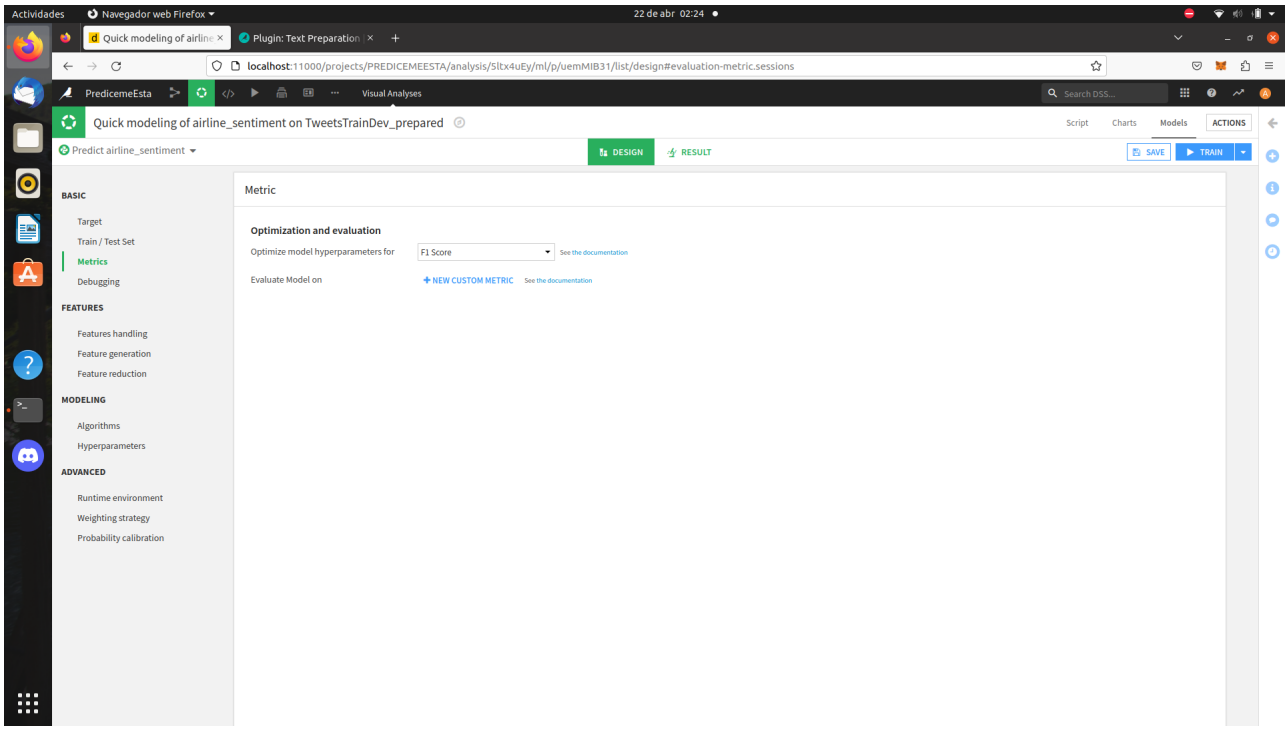
3.4. Figura: Modelo seleccionado



3.5. Figura: División Train/Test



3.6. Figura: Columnas estudiadas



3.7. Figura: Determinador del mejor modelo

3.2. Anexo 2: Otros modelos probados

A lo largo de la práctica hemos probado distintas variaciones en el algoritmo empleado y en el preproceso de los datos en busca de un mayor f-score. Los dos algoritmos empleados no tenían hiperparámetros personalizables. Adicionalmente, el script que genera los modelos **filtra automáticamente entre todas las versiones de Naive-Bayes** para elegir la que ofrezca un mayor f-score, por lo tanto la cantidad de tests ha sido menor que la de otros grupos.

Las combinaciones que hemos probado han sido las siguientes:

Algoritmo	Características preproceso	Prec	Rec	F-sco
Naive-Bayes	BoW, Traducir Emojis/Emotes	0,6773	0,6707	0,6618
Naive-Bayes	BoW, Borrar Emojis/Emotes	0,6949	0,6919	0,6827
Naive-Bayes	tf-idf, Traducir Emojis/Emotes	0,7017	0,6959	0,6880
Naive-Bayes	tf-idf, Borrar Emojis/Emotes	0,6982	0,6894	0,6817
Logistic Regression	BoW, Traducir Emojis/Emotes	0,7266	0,7211	0,7225
Logistic Regression	BoW, Borrar Emojis/Emotes	0,7039	0,6959	0,7225
Logistic Regression	tf-idf, Borrar Emojis/Emotes	0,7107	0,6968	0,6992

3.1. Cuadro: Combinaciones probadas

Además de estas pruebas realizadas, había una versión del script que realizaba un barrido de parámetros para Logistic Regression, pero el proceso de generar un modelo tardaba una hora aproximadamente y el resultado obtenido era muy similar al que obteníamos sin realizarlo, por lo que no hemos considerado que merezca la pena seguir adelante con esa versión.