Universidad del País Vasco
Euskal Herriko Unibertsitatea

Universidad del Pais Vasco - Euskal Herriko Unibertsitatea
Facultad De Ingeniería
Año 2023 - 2do Cuatrimestre

Sistemas de Apoyo a la Decisión

Estudio de Tweets: Valoraciones de aerolíneas estadounidenses.

Andreea Emilia Vasilica	avasilica001@ikasle.ehu.eus
Joel Moisés García Escribano	jgarcia424@ikasle.ehu.eus
Francisco González Curiel	fgonzalez020@ikasle.ehu.eus
Mario Almandoz Latierro	malmandoz005@ikasle.ehu.eus

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

Los datos proporcionados para este proyecto vienen dados por un archivo csv, llamado Tweets-TrainDev, el cual está compuesto de 13 columnas, que componen los distintos atributos de las instancias con información concreta de los tweets: tweet_id, airline_sentiment, airline_sentiment_confidence, negativereason, negativereason_confidence, airline, name, retweet_count, text, tweet_coord, tweet_created, tweet_location y user_timezone. Y a su vez, posee 11.999 instancias, donde cada una contendrá la información de un tweet.

1.1. División entre Train y Dev

Conjunto De Datos	% de instancias	Num. de instancias
Train	0,8	8399
Dev	0,2	3600

1.1. Cuadro: División Train y Dev

A partir de las 11.999 instancias que posee el archivo, como se puede ver en la tabla, hemos decidido dividirlas de forma que el 80 % de los datos se utilicen para el entrenamiento, siendo 8399 instancias, y el restante 20 % se utilice para las pruebas de dicho entrenamiento, siendo 3600 instancias. Esta división se ha realizado mediante el siguiente código:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2, random_state= 26)
```

1.1. Figura: Línea de código para la división entre train/dev.

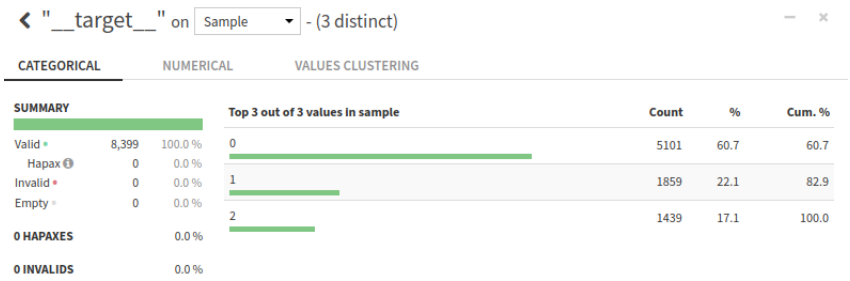
1.2. Distribución de las clases en cada conjunto

Tras la división de los datos para Train y Dev, hemos realizado con Dataiku un análisis de las dos muestras obtenidas, a partir de las que hemos obtenido los siguientes resultados:

Conjunto De Datos	Clase Neg	Clase Neutra	Clase Pos.
Train	5101	1859	1439
Dev	2183	805	612

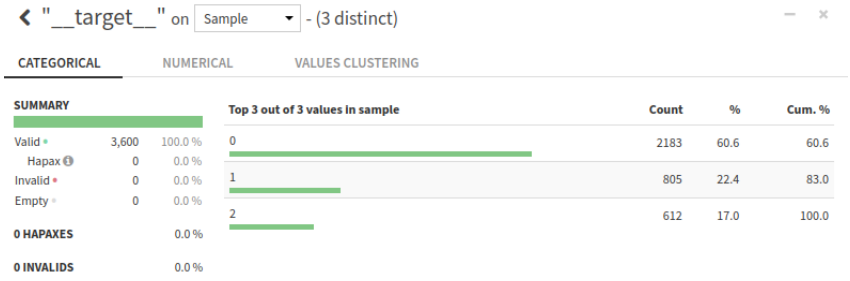
1.2. Cuadro: Distribución Train y Dev

Como se puede ver en la figura inferior, la muestra de Train posee un 60.7 % de elementos negativos, un 22.1 % de elementos neutros y un 17.1 % de elementos positivos.



1.2. Figura: Análisis de Dataiku para las instancias utilizadas en train.

También observando la figura inferior, se puede observar que la muestra de Dev posee un 60.6 % de elementos negativos, un 22.4 % de elementos neutros y un 17 % de elementos positivos.

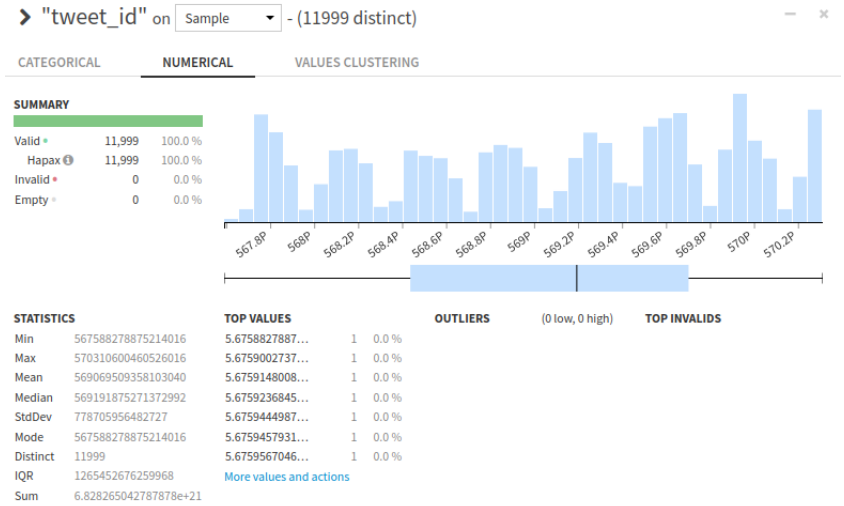


1.3. Figura: Análisis de Dataiku para las instancias utilizadas en dev.

Por lo tanto, podemos concluir que independientemente de las muestras que hemos obtenido tanto para entrenar como para probar el entrenamiento, poseemos un desbalance entre las clases, ya que la clase negativa se encuentra mucho más representada que las clases neutras o positivas.

1.3. Análisis descriptivo de los datos

- **tweet_id**: Este dato es el identificador de cada instancia de tweet, el cual será único. El tipo de dato es numérico continuo, y cuantitativo.

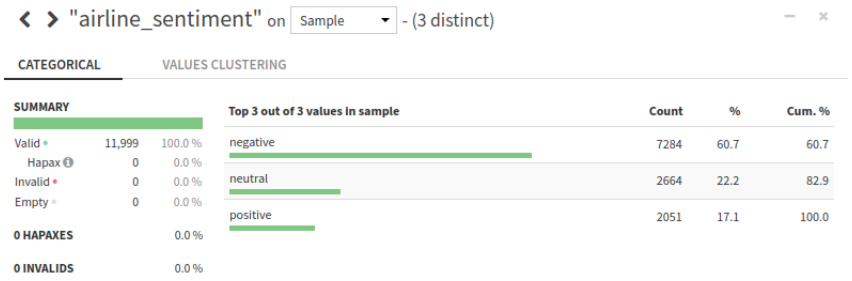


1.4. Figura: Análisis de Dataiku para la columna tweet_id.

A partir del análisis podemos observar que este atributo no es faltante para ninguna de las instan-

cias, coincide con la característica de ser único, y sigue un parecido entre toda su información, por lo que no hará falta preprocesarlo más adelante.

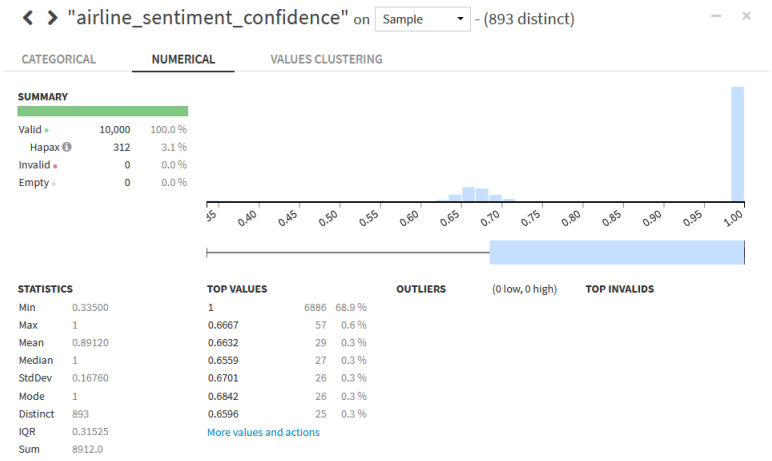
- **airline_sentiment**: Esto es el indicador del sentimiento que produce el tweet, el cual conforma las clases que pueden tomar las instancias. Las clases serán, en este caso, tres: Negativo, Neutral y Positivo. El tipo de dato es categórico ordinal, y cualitativo.



1.5. Figura: Análisis de Dataiku para la columna airline_sentiment.

A partir del análisis podemos observar que el atributo tiene un formato correcto, y que tampoco es faltante en ninguna de las instancias. Al tratarse del atributo que define las clases, este no necesitará de preproceso, si no que, en su caso, se eliminará la columna correspondiente para ser traspasada la información a la nueva columna __target__, la cual será utilizada para el entrenamiento.

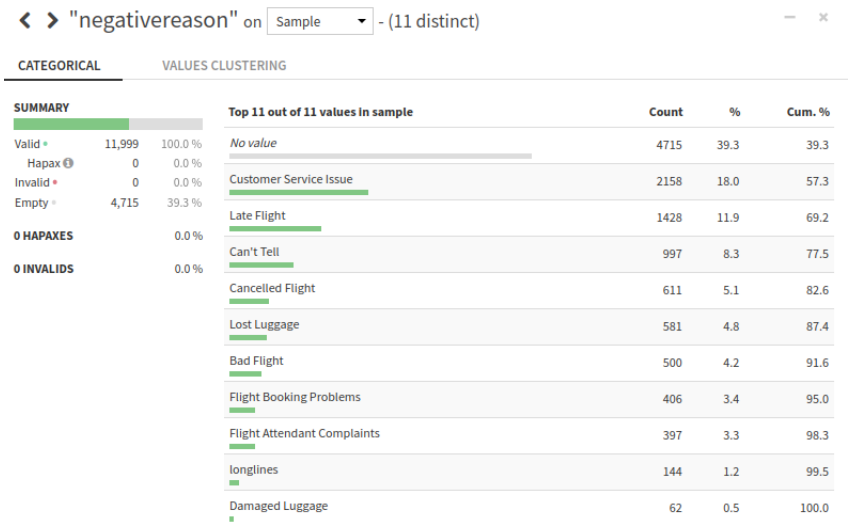
- **airline_sentiment_confidence**: Es un valor del 0 al 1 referente a la probabilidad, que nos demuestra la confianza con la que se ha realizado la clasificación en en el atributo airline_sentiment. El tipo de dato es numérico continuo, y cuantitativo.



1.6. Figura: Análisis de Dataiku para la columna airline_sentiment_confidence.

A partir del análisis podemos, hemos visto que el atributo no es faltante en ninguno de los casos, y también que se encuentra siempre en un rango entre 0 y 1, por lo tanto, es un atributo que no necesita de preprocesado.

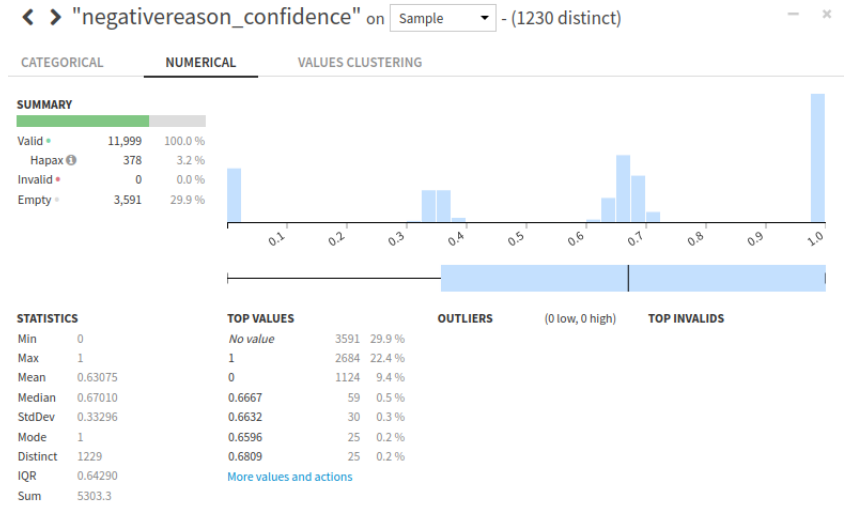
- **negativereason**: Este atributo da la razón asociada a la opinión (y clase) negativa. El tipo de dato es categórico nominal, y cualitativo.



1.7. Figura: Análisis de Dataiku para la columna negative_reason.

A partir del análisis de los datos, podemos observar que muchos de los datos se encuentran vacíos. La solución a este problema deberá realizarse utilizando clusters, los cuales realizaremos en la siguiente entrega. Para esta entrega, se ha hecho un preproceso de los datos que nos ha ayudado a trabajar mejor con la información a la hora de entrenar.

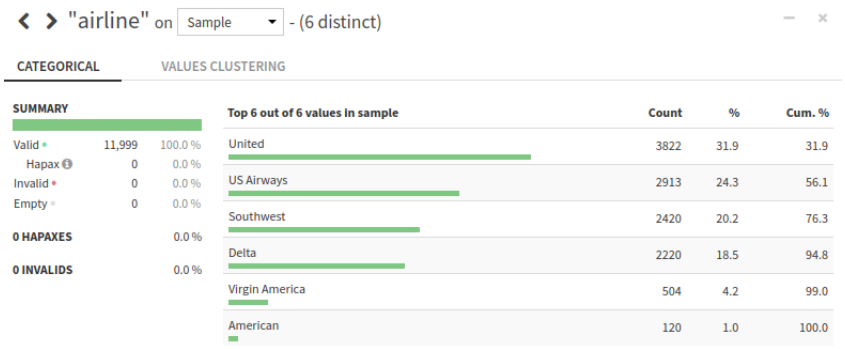
- **negativereason_confidence**: Es un valor del 0 al 1, referente a la probabilidad que nos demuestra con cuánta confianza se ha definido la información en el atributo negativereason. El tipo de dato es numérico continuo, y cuantitativo.



1.8. Figura: Análisis de Dataiku para la columna negative_reason_confidence.

A partir del análisis de datos podemos observar que la información de los datos se encuentra entre 0 y 1, con lo cual no resulta un problema a la hora de tratar la información, pero sí el hecho de que una parte de los datos se encuentre vacía. Por este motivo, este atributo será preprocesado.

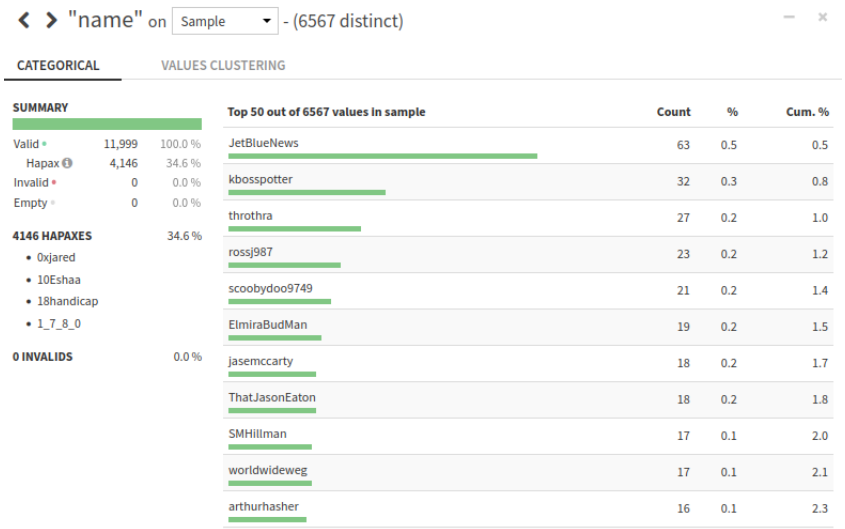
- **airline**: Este atributo indica cuál es el nombre de la aerolínea. El tipo de dato es categórico nominal.



1.9. Figura: Análisis de Dataiku para la columna airline.

A partir del análisis, podemos observar que ningún atributo se encuentra vacío, y que las aerolíneas son 5: United, Southwest, Delta, US Airways y Virgin America. Para este atributo no se realizará ningún preprocesado, ya que los datos son correctos.

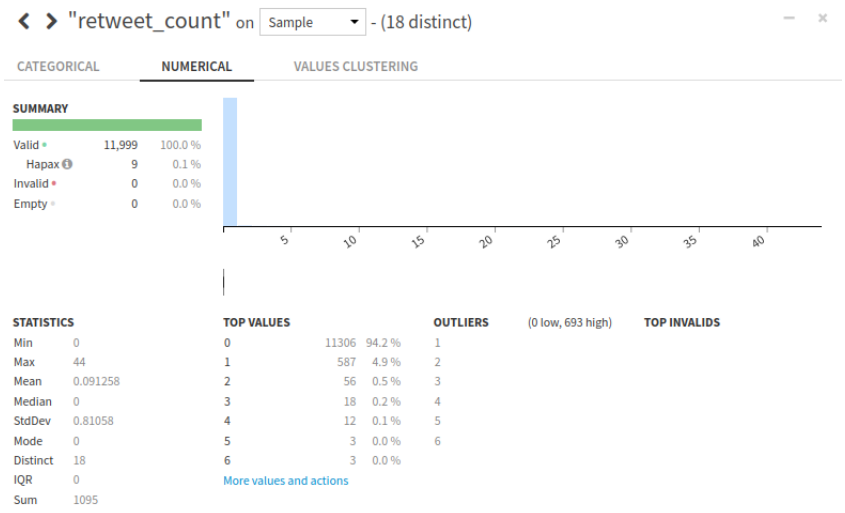
- **name**: El atributo name es el nombre del usuario de twitter que ha escrito el tweet. Este tipo de dato es categórico nominal.



1.10. Figura: Análisis de Dataiku para la columna name.

A partir del análisis observamos que ninguna instancia tiene este atributo faltante, y que los datos son correctos para todos los nombres de usuario, con lo cual, este atributo no requerirá de preprocesado.

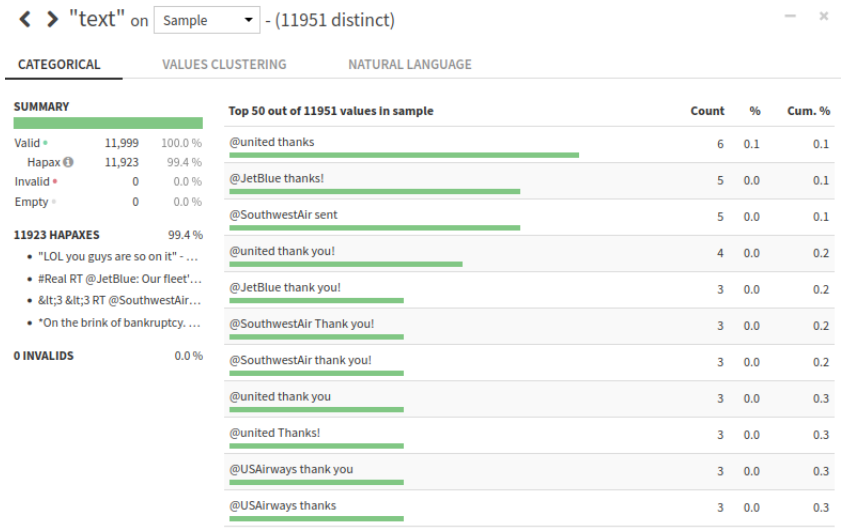
- **retweet_count**: Este atributo contiene el número de retweets de un tweet. E tipo de dato es numérico discreto, y cuantitativo.



1.11. Figura: Análisis de Dataiku para la columna retweet_count.

A partir del análisis encontramos que los valores oscilan entre 0, siendo el mínimo, y 31, siendo el máximo, con lo cual se requerirá de un preprocesado para escalar el atributo.

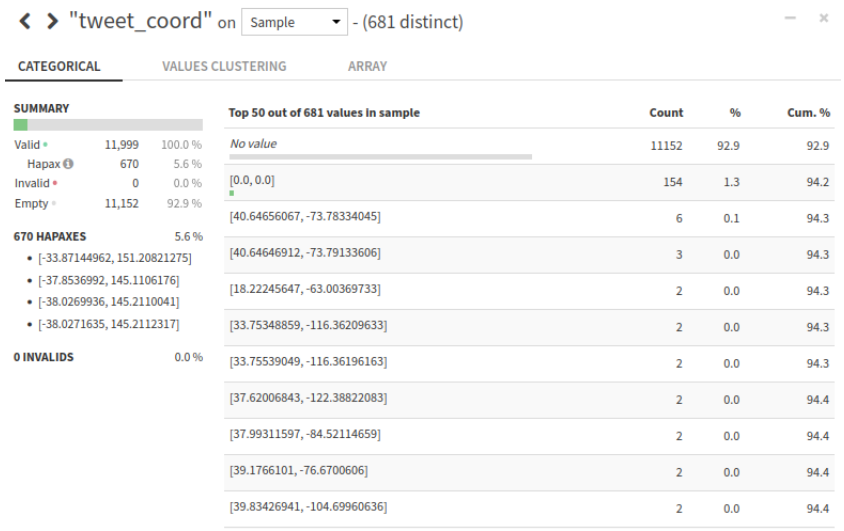
- **text**: Este atributo es el contenido de texto del propio tweet. El tipo de dato es categorial nominal.



1.12. Figura: Análisis de Dataiku para la columna text.

A partir del análisis podemos observar que no se encuentra ningún atributo faltante, y en principio no se encuentran problemas, pero sabiendo que se trata de un texto, se hará el preprocesado necesario para lenguaje natural.

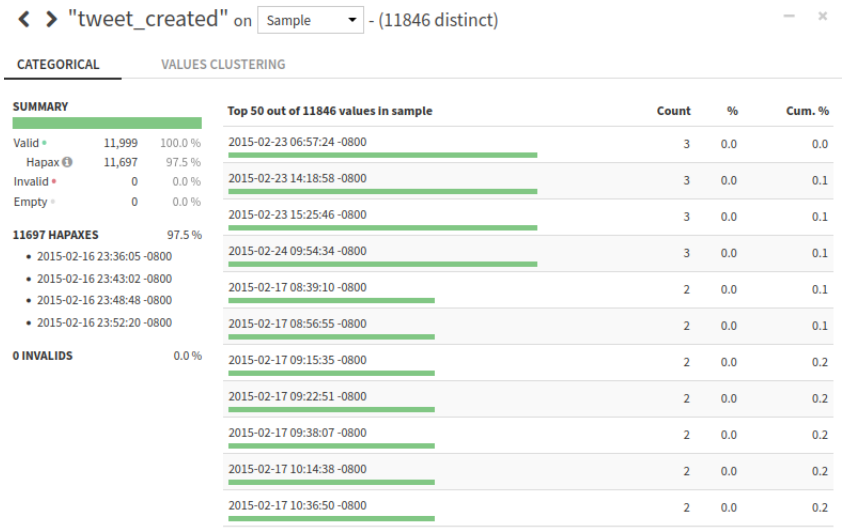
- **tweet_coord**: Este atributo contiene la información de coordenadas en latitud y longitud, del lugar donde se ha publicado el tweet. El tipo de dato es numérico continuo, y cuantitativo.



1.13. Figura: Análisis de Dataiku para la columna tweet_coord.

A partir del análisis de las coordenadas, podemos observar que una gran mayoría de los datos se encuentran vacíos, o con las coordenadas "[0.0, 0.0]", con lo cual se requerirá de un preprocesado para mejorar la información.

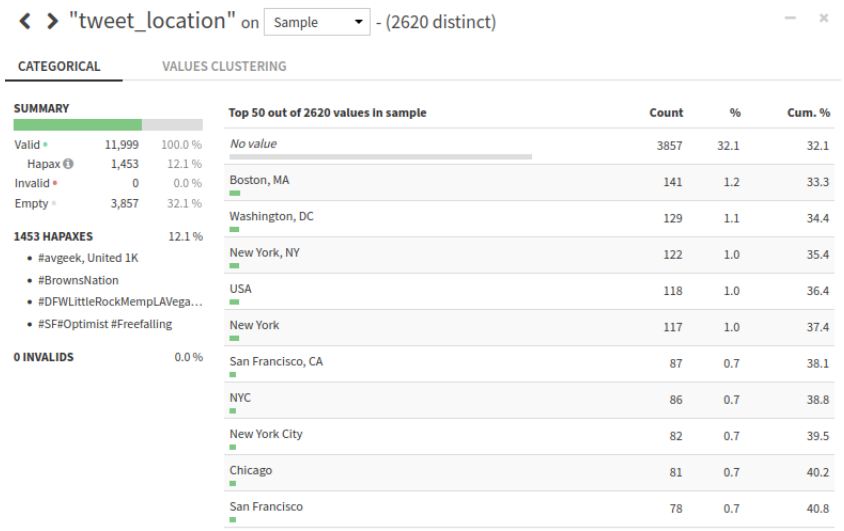
- **tweet_created**: Es el atributo que contiene la fecha, hora y zona horario del momento en el que se ha publicado el tweet. El tipo de dato es numérico continuo, y cuantitativo.



1.14. Figura: Análisis de Dataiku para la columna tweet_created.

A partir del análisis de la fecha, podemos observar que para ninguna instancia el atributo es inválido, y el formato es correcto, con lo cual, no requerirá de un preprocesado.

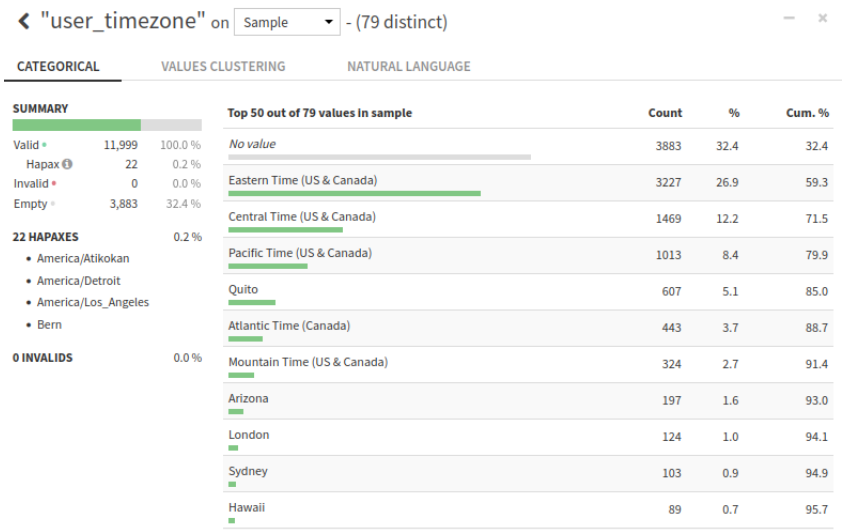
- **tweet_location**: Este atributo indica la ciudad desde donde se ha sido publicado el tweet. El tipo de dato es categórico nominal.



1.15. Figura: Análisis de Dataiku para la columna tweet_location.

A partir del análisis de datos, podemos observar que una parte de los datos se encuentran vacíos, e incluso algunos de ellos que no lo están, necesitan de un preprocesado de texto para eliminar símbolos, emotes, etc. Para ello se aplicará un preprocesado.

- **user_timezone**: Este atributo indica la zona horaria de la ciudad dese donde se ha publicado el tweet. El tipo de dato es categórico nominal.



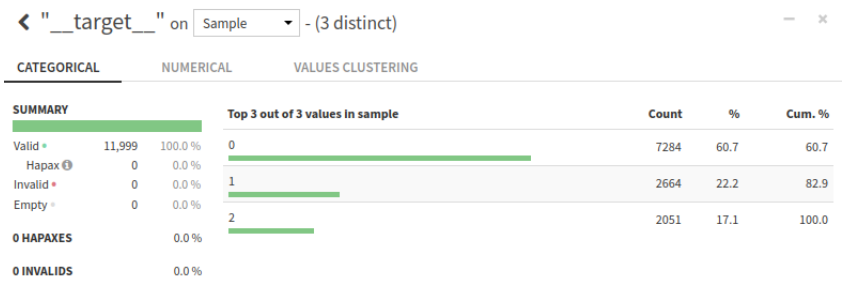
1.16. Figura: Análisis de Dataiku para la columna user_timezone.

A partir del análisis de los datos podemos encontrar que una parte de los datos posee información faltante para este atributo, a pesar de ello, no se realiza ningún preprocesado, ya que no será necesario su preproceso en el entrenamiento.

1.4. Descripción del preproceso

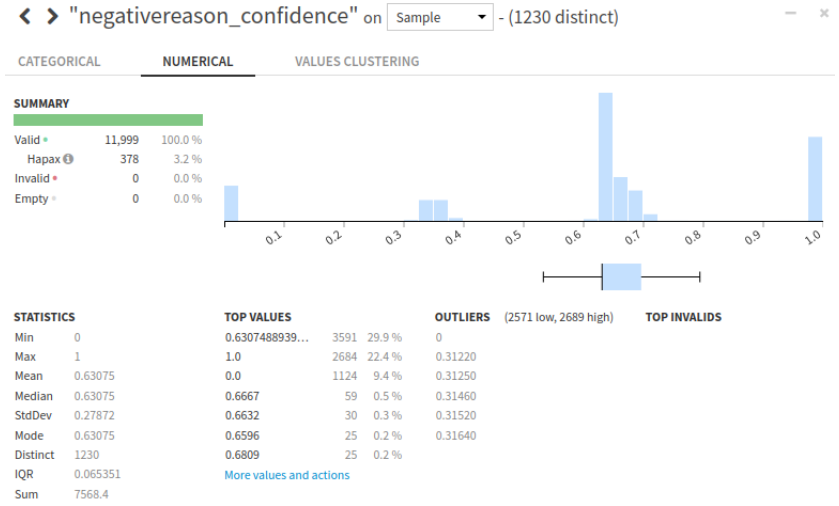
- **Atributo airline_sentiment**
  - Se han convertido los valores de la clase airline_sentiment de categóricos a numéricos: 0 es negativo, 1 neutral y 2 positivo.

- A la hora de ejecutar los modelos de Naive Bayes y Logistic Regression se realiza tf-idf sobre el texto para poder utilizarlo.



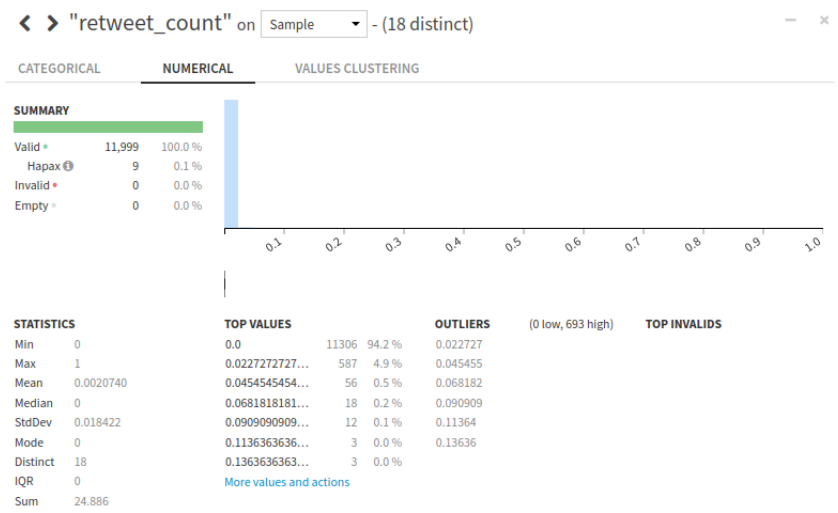
1.17. Figura: Análisis de Dataiku para la columna __target__ tras el preprocesado.

- **Atributo negative_reason_confidence**
  - Para tratar los valores faltantes en este atributo se ha optado por rellenarlos con la media del resto de valores existentes, que es 0.61 exactamente.



1.18. Figura: Análisis de Dataiku para la columna negativereason_confidence tras el preprocesado.

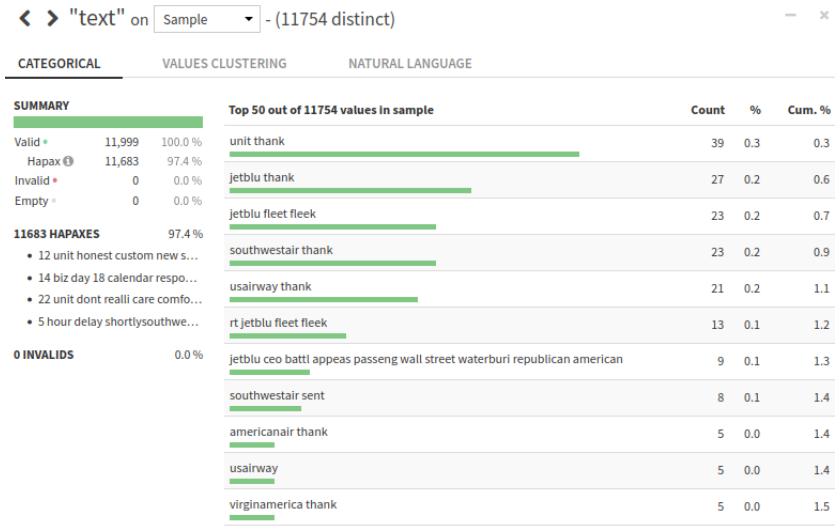
- **Atributo retweet_counts**
  - Para escalar retweet_counts se utiliza el algoritmo MinMax, que en este caso normaliza los valores en el rango de 0 a 0.7.
  - Aún así, quedan muchos valores cercanos a 0, por lo que puede que no sea un preprocesado idóneo para utilizar luego este atributo en aprendizaje.



1.19. Figura: Análisis de Dataiku para la columna retweet_count tras el preprocesado.

- **Atributo text**

  - Al atributo **text** se le realizan una variedad de preprocesos, entre ellos se encuentran pasar todo a minúsculas, eliminar caracteres especiales y signos de puntuación, eliminar los stopwords y lematizar las palabras. Estos cambios se realizan para limpiar y normalizar el texto para facilitar su procesamiento y análisis.



1.20. Figura: Análisis de Dataiku para la columna text tras el preprocesado.

- **Atributo tweet_coord**

  - A la hora de tratar este atributo hemos detectado valores "[0,0]" y valores faltantes. Ambos tipos de valores los hemos sustituido por nuevas coordenadas de la siguiente forma: primero tratamos de conseguir la ciudad mediante la zona horaria, y si no se puede, se utiliza la ciudad en la que se encuentra la sede de la aerolínea.
  - Según la zona horaria:

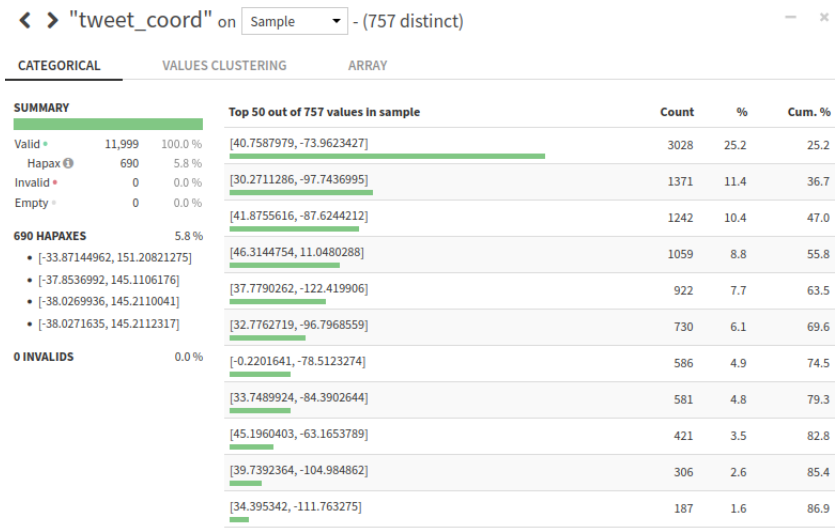
Zona horaria	Ciudad
Eastern Time (US Canada)	New York City, New York
Central Time (US Canada)	Austin, Texas
Mountain Time (US & Canada)	Denver, Colorado
Pacific Time (US & Canada)	San Francisco, California
Atlantic Time (Canada)	Nova Scotia, Canada

1.3. Cuadro: Zonas horarias y Ciudades

- Según la sede de la aerolínea:

Sede	Ciudad
United	Chicago, Illinois
Southwest	Dallas, Texas
Delta	Atlanta, Georgia
US airways	Alexandria, Virginia
Virgin America	Burlingame, California

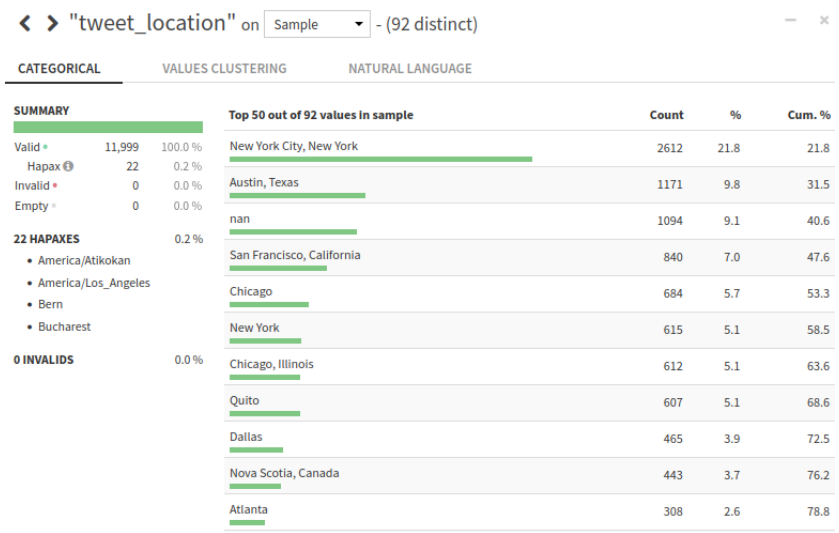
1.4. Cuadro: Aerolíneas y sus sedes



1.21. Figura: Análisis de Dataiku para la columna tweet_coord tras el preprocesado.

- **Atributo tweet_location**

  - Si falta este valor en una instancia, se intenta obtener la ciudad de la misma forma que se ha intentado en las coordenadas; primero mirando el huso horario y si no se puede, con la sede de la aerolínea.
  - Si no falta el valor, como en este atributo hay muchos errores ortográficos, primero se ha procesado como texto pasando todo a minúsculas y eliminando los caracteres especiales y los signos de puntuación.
  - Luego se utiliza la librería de python **Geotext** para extraer la ciudad del texto, ya que puede ser que el usuario haya su ubicación con una oración
  - Si no se reconoce la ciudad de esta forma, se intuye de la misma forma que se ha intuido para los valores faltantes



1.22. Figura: Análisis de Dataiku para la columna tweet_location tras el preprocesado.

1.5. Descripción del Proceso de Submuestreo o Sobremuestreo

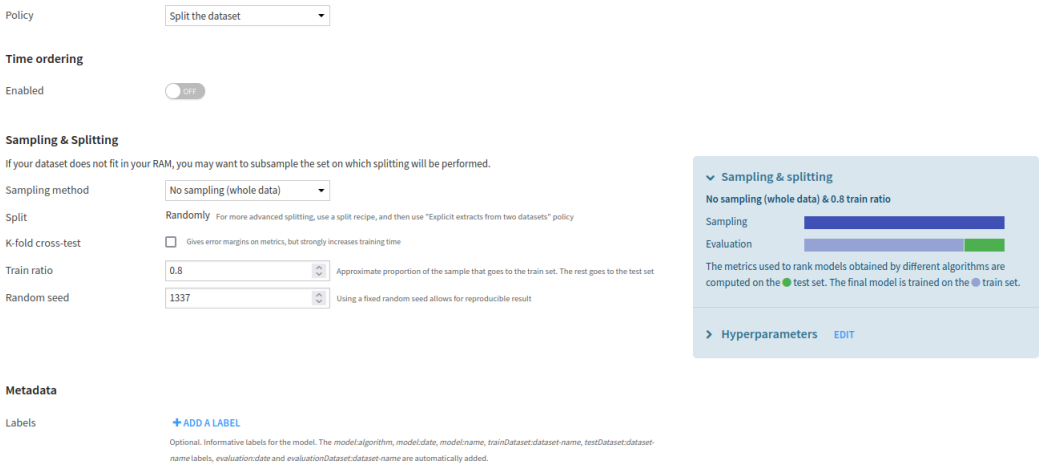
Para tratar el desbalance observado en el dataset, se ha utilizado la librería SMOTE, ya que, tras el análisis de los datos, se han encontrado un gran número de instancias pertenecientes a la clase "negativo". La librería SMOTE ha sido de ayuda para realizar un oversampling de los datos, de forma que se han creado nuevas instancias de las clases encontradas con menos frecuencia ("neutral z "positivo"), para así diversificar la información del dataset y ajustar este desbalance inicial. Este proceso nos ha sido de gran utilidad para obtener una Accuracy mucho más alta tras las pruebas realizadas, en las que hemos podido observar una mejora de 0.2, que no habríamos podido obtener sin haber realizado con anterioridad este ajuste.

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

2.1. Algoritmo KNN con Dataiku

Los datos utilizados para el modelo KNN han sido ya preprocesados previamente con nuestro algoritmo de python, para permitir a Dataiku una igualdad de condiciones y asegurarnos que el preprocesado de Dataiku no es distinto al nuestro.

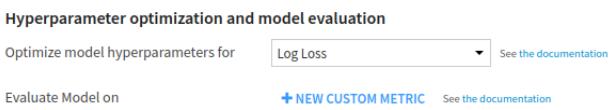
A partir de los datos preprocesados, se ha elegido el modelo multiclase donde el atributo target ha sido __target__. Para la elección de división entre train/test se ha seguido los siguientes pasos:



2.1. Figura: Elecciones de Train/Test para el algoritmo KNN semejante a los propios con Dataiku.

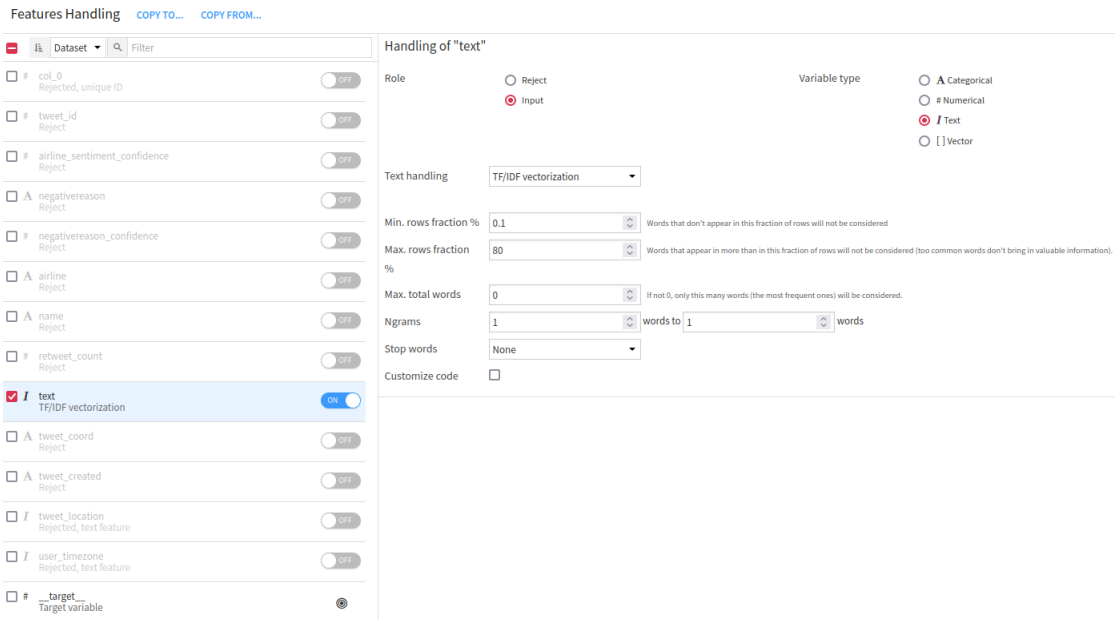
En este caso, se ha tomado el dataset entero para la división entre Train y Test. Los porcentajes serán 80 % para el entrenamiento y 20 % para el test, así como la random seed 1337.

Para el apartado de métricas, se ha optimizado el modelo de hiperparámetros con Log Loss:



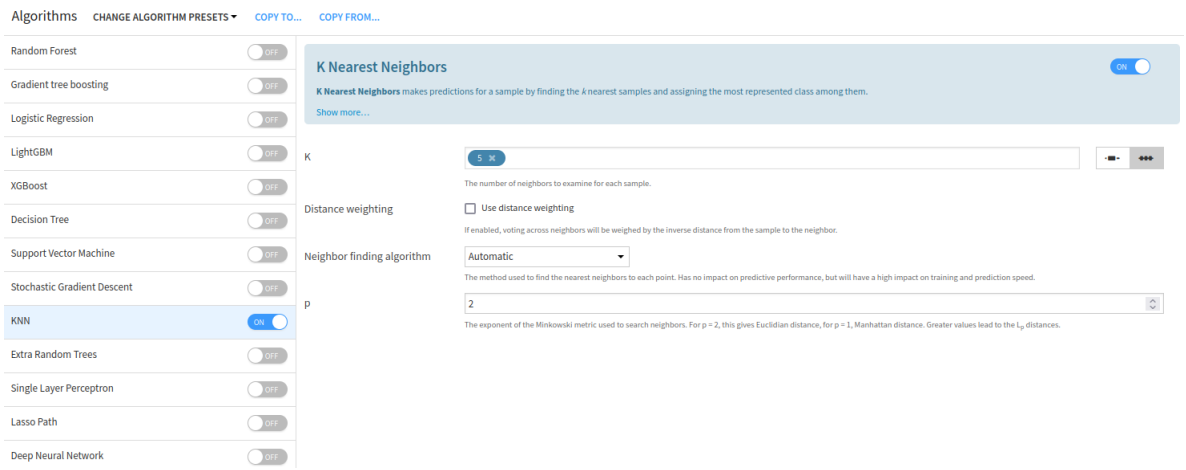
2.2. Figura: Elecciones de Métrica para el algoritmo KNN semejante a los propios con Dataiku.

Dentro de las Features, solo se ha tomado el atributo text, ya que es el único que hemos utilizado para los demás algoritmos de python, y así como en esos algoritmos, hemos procesado el texto mediante TF/IDF.



2.3. Figura: Elecciones de Features para el algoritmo KNN semejante a los propios con Dataiku.

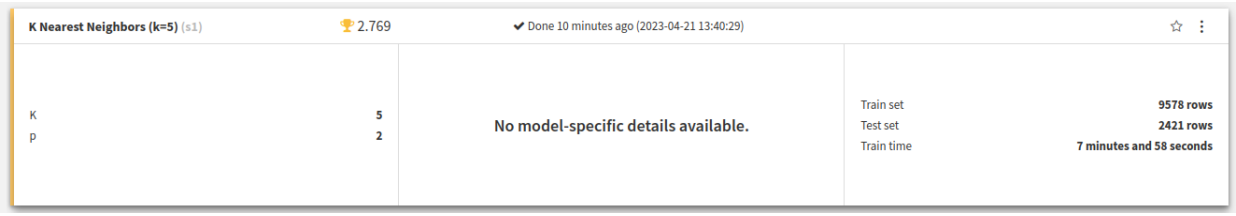
El algoritmo elegido ha sido KNN, con los parámetros k=5 y p=2, y distancia uniforme.



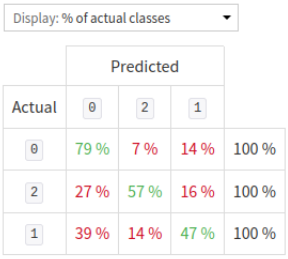
2.4. Figura: Elecciones del algoritmo KNN semejante a los propios con Dataiku.

Los apartados del modelo que no se han mencionado no han sido modificados, y por lo tanto poseen los parámetros por defecto de Dataiku.

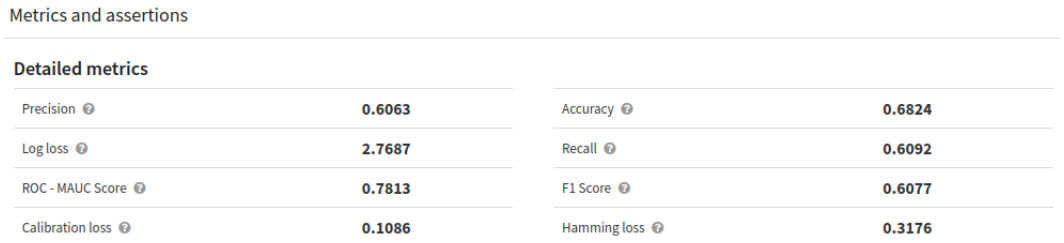
Los resultados del modelo han sido los siguientes:



2.5. Figura: Resultado resumido del algoritmo KNN semejante a los propios con Dataiku.

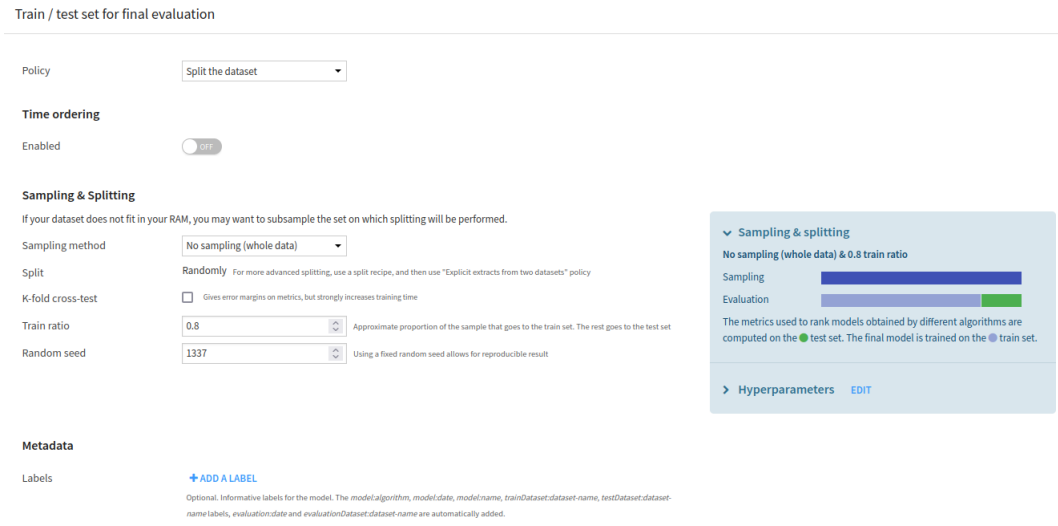


2.6. Figura: Resultado de la matriz de confusión del algoritmo KNN semejante a los propios con Dataiku.

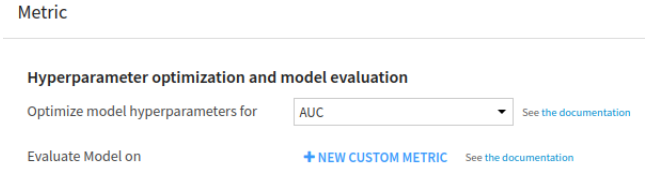


2.7. Figura: Resultado de la métrica del algoritmo KNN semejante a los propios con Dataiku.

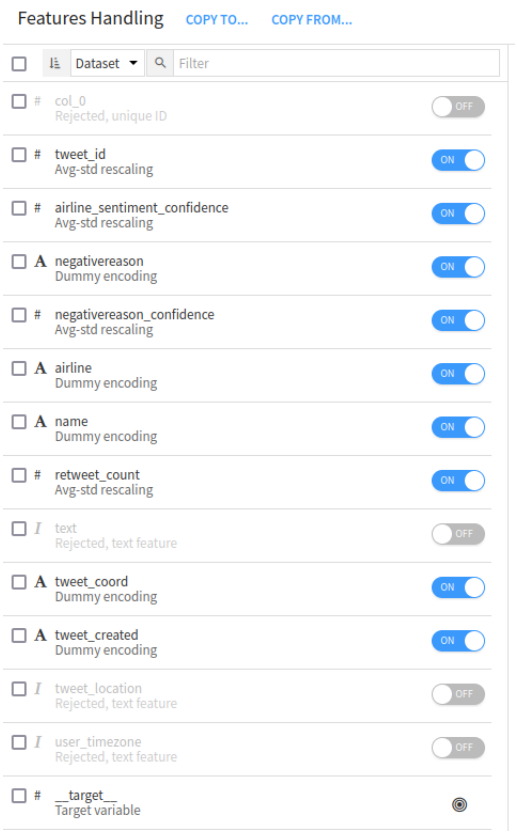
El resultado obtenido mediante este modelo nos ha parecido demasiado bajo, así que hemos probado con una combinación de elementos distinta, en este caso, la que nos proporciona Dataiku por defecto sobre toda la muestra.



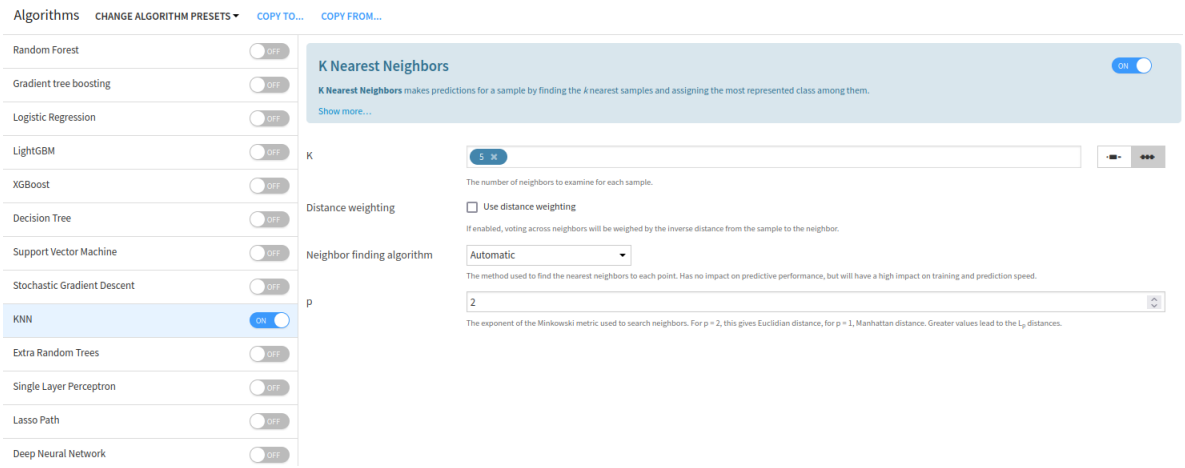
2.8. Figura: Elecciones de Train/Test para el algoritmo KNN por defecto con Dataiku.



2.9. Figura: Elecciones de Métrica para el algoritmo KNN por defecto con Dataiku.



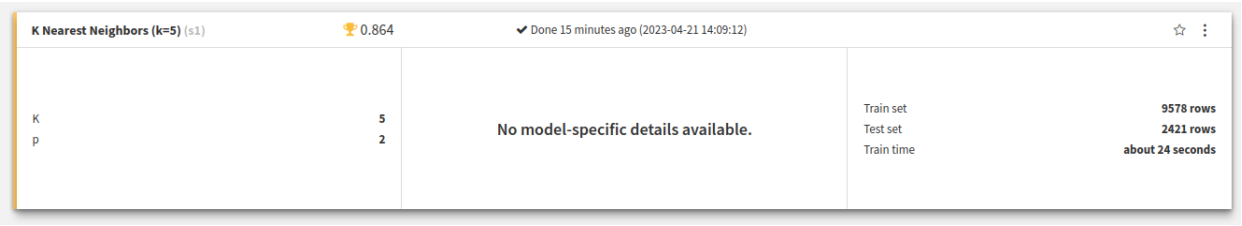
2.10. Figura: Elecciones de Features para el algoritmo KNN por defecto con Dataiku.



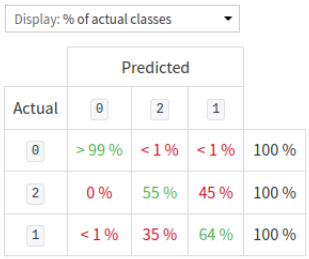
2.11. Figura: Elecciones del algoritmo KNN por defecto con Dataiku.

Los apartados del modelo que no se han mencionado no han sido modificados, y por lo tanto poseen los parámetros por defecto de Dataiku.

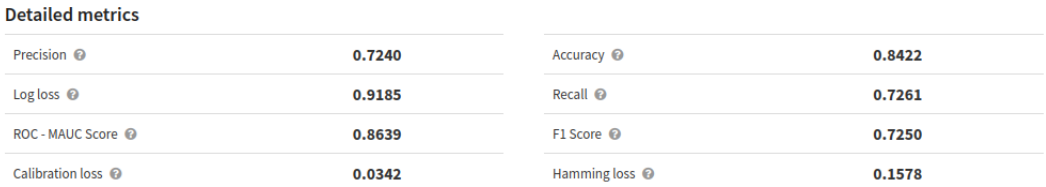
Los resultados del modelo han sido los siguientes:



2.12. Figura: Resultado resumido del algoritmo KNN por defecto con Dataiku.



2.13. Figura: Resultado de la matriz de confusión del algoritmo KNN por defecto con Dataiku.



2.14. Figura: Resultado de la métrica del algoritmo KNN por defecto con Dataiku.

El resultado obtenido mediante este modelo por defecto nos ha parecido mejor así que hemos utilizado estas métricas como umbral para mejorar nuestros propios algoritmos con python.

2.2. Algoritmos empleados: Breve Descripción

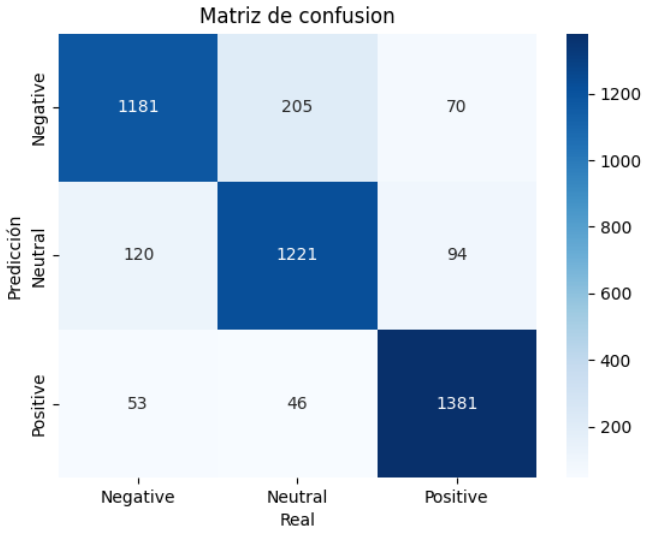
A la hora de determinar los mejores hiperparámetros para los algoritmos de Logistic Regression y Naive Bayes, hemos implementado dos iteradores, uno para cada algoritmo en particular.

El iterador de Logistic Regression evalúa el hiperparámetro C en un rango que parte de 0 y llega hasta 100, en intervalos de 2'5. El parámetro C es el encargado de controlar la regularización, un método para evitar el sobreajuste.

Por otro lado, el iterador de Naive Bayes itera sobre los valores de alpha y fit_prior. El valor Fit_prior puede ser Verdadero o Falso, y determina si se deben calcular las probabilidades a priori utilizando el conjunto de datos o si se deben calcular de manera uniforme, sin tener en cuenta el conjunto de datos. Alpha es el parámetro encargado de controlar el suavizado cuando hay valores cercanos a cero.

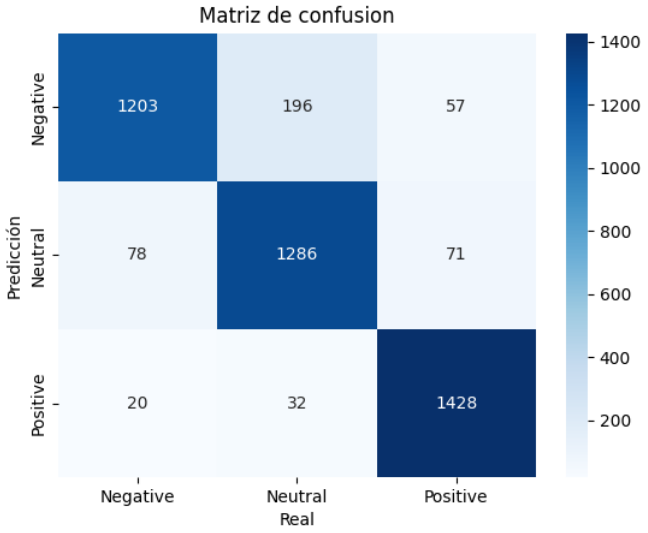
Posteriormente, y tras seleccionar las mejores puntuaciones obtenidas de los f-scores en ambos algoritmos, hemos guardado los hiperparámetros correspondientes a las mejores puntuaciones:

- **Multinomial Naive Bayes**:
  - Hiperparámetros: alpha = 0, fit_prior = False
  - f-Score Weighted: 0.87
  - Link: Sklearn MultinomialNB
  - Se ha obtenido la siguiente matriz de confusión:



2.15. Figura: Matriz de confusión para Naive Bayes.

- **Logistic Regression**:
  - Hiperparámetros: C = 32.5
  - f-Score Weighted: 0.901
  - Se ha obtenido la siguiente matriz de confusión:



2.16. Figura: Matriz de confusión para Logistic Regression.

2.3. Resultados

En esta sección se presentan los mejores resultados obtenidos para ambos algoritmos que nos han sido propuestos. En el caso de Naive Bayes, a la hora de ejecutar el iterador de hiperparametros, se observan ocasiones en las que el resultado más óptimo se realiza con fit-prior=False y en otras ocasiones con fit-prior=True. Esto sucede debido a que a la hora de separar los datos en train/test se realiza de forma aleatoria y no siempre se obtienen los mismo resultados. A pesar de ello, el resultado con el mejor F-Score obtenido es el que se puede observar en la siguiente tabla.

Algoritmo	Parametros	F-Score
Logistic Regresion	C=32,5	0,901
Naive Bayes	a=0,fit-prior=False	0,87

2.1. Cuadro: Mejores resultados de los algoritmos

Tras observar los resultados obtenidos, y teniendo en cuenta el mejor modelo que se ha generado utilizando Dataiku para el algoritmo de KNN propuesto (k=5 y p=2, descrito anteriormente), podemos observar que nuestros algoritmos tanto Logistic Regression como Naive Bayes, han sido más exitosos, ya que el f-score obtenido con Dataiku no superó en ninguna de nuestras pruebas 0'7250, mientras que nuestros algoritmos han llegado hasta 0'901 y 0'87 respectivamente.

2.3.1. Pruebas

- Se nos proporcionaba una API (OpenCageGeocode) a partir de la cual podíamos conseguir las coordenadas de la ubicación de los usuarios, pero para no tener que depender de una API externa que posee un máximo de requests diarias, hemos utilizado una librería (Geopy.geocoders) que realiza el mismo trabajo y que nos ha dado muy buenos resultados.
- Se ha intentado utilizar los sentimientos contenidos en los *emojis* encontrados en el textos para lograr mejores resultados en análisis de sentimiento. Sin embargo, empeoraron bastante las predicciones en ambos algoritmos, por lo que hemos prescindido de los *emojis* y los hemos eliminado de los textos.
- Se ha intentado borrar los arrobas (por ejemplo, @USAirlines) de los textos, pero no hemos conseguido realizar la transformación de forma correcta y algunos textos empeoraban, por lo que nos hemos desecho de esa funcionalidad.
- Se ha probado a utilizar el atributo que guarda el número de retweets para mejorar el modelo, pero los resultados no han mejorado. Creemos que es porque la gran mayoría de los valores del atributo son muy cercanos a cero.
- Las primeras pruebas se han realizado probando de forma manual a introducir hiperparámetros, pero tras ver que estos pueden variar considerablemente, hemos concluido que la mejor solución sería crear iteradores que encuentren los mejores hiperparámetros para ambos algoritmos. Por lo tanto, hemos adoptado esta técnica para todas las pruebas siguientes.

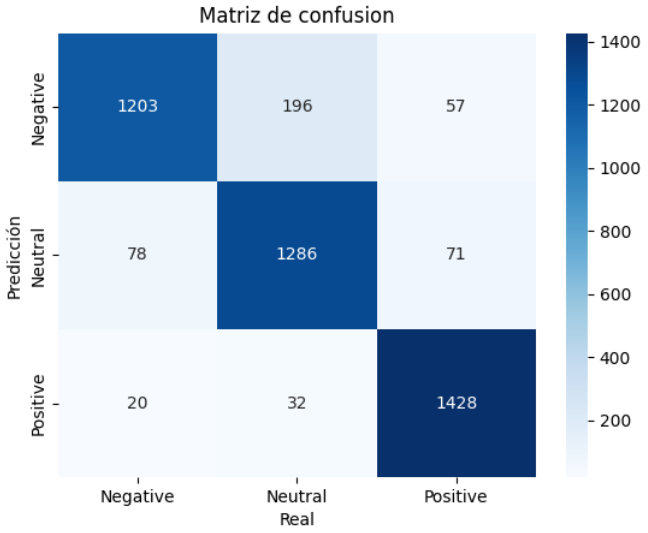
2.3.2. Discusión

En secciones anteriores, hemos evaluado los resultados obtenidos por los modelos de predicción utilizando diferentes técnicas de balanceo de datos. Por lo que es importante destacar que en los problemas de clasificación donde existe un desequilibrio entre clases, como en nuestro caso con TweetsTrainDev, las técnicas de balanceo pueden mejorar el rendimiento y resultados considerablemente.

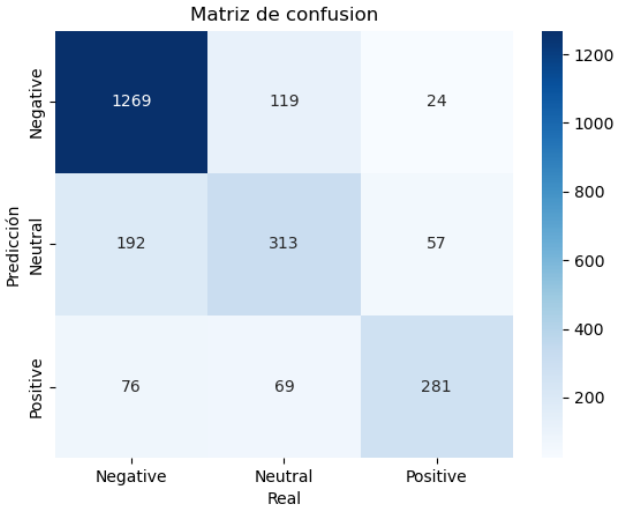
En nuestro análisis, hemos comparado los resultados obtenidos con y sin técnicas de balanceo de datos, y hemos observado que estos resultados mejoran considerablemente cuando se aplican este tipo de técnicas. Como la que hemos utilizado nosotros, SMOTE (Oversampling), que ha mejorado la precisión en un 0.2 en comparación con el modelo sin balanceo.

También hemos utilizado el F-score para evaluar la capacidad de predicción del modelo en ambas situaciones, y hemos observamos que, sin aplicar estas técnicas, el modelo presentaba un desempeño menor en las clases con menor representación, lo que resultó en un F-score bajo para dichas clases. Sin embargo, utilizando SMOTE, el F-score para la clases menos representadas mejoró notablemente, lo que indica que tras tratar el desajuste se puede predecir correctamente la clase minoritaria.

Por ello, podemos concluir que el uso de técnicas de balanceo de datos, como SMOTE, puede mejorar las métricas y el desempeño del modelo en situaciones de desequilibrio entre clases. Pero es importante tener en cuenta que, aunque estas técnicas pueden mejorar el rendimiento del modelo, siempre debemos evaluar su efecto en los datos de prueba antes de implementarlas.



2.17. Figura: Matriz de confusión para Logistic Regression usando SMOTE.



2.18. Figura: Matriz de confusión para Logistic Regression sin usar técnica de balanceo.

Nuestros análisis han demostrado que Logistic Regression ha sido el algoritmo que mejor se ha adaptado a nuestro dataset, obteniendo un F-Score superior al de Naive Bayes. Sin embargo, es importante tener en cuenta que estos resultados se han obtenido después de haber realizado una limpieza exhaustiva de los datos, por lo que es posible que con otros métodos de preprocesamiento, Naive Bayes supere a Logistic Regression en rendimiento.

2.4. Conclusión

Después de la realización de la primera parte del proyecto, podemos concluir que el preprocesado de los datos es crucial para obtener buenos resultados en modelos de aprendizaje automático. Esto se debe a que en la mayoría de ocasiones los datos no se encuentran en el formato óptimo para poder obtener los resultados deseados, y por tanto, es necesario aplicar técnicas de limpieza de datos para prepararlos adecuadamente para el modelo.

Además, la selección apropiada de hiperparámetros es también clave para construir modelos precisos y efectivos. La elección de los parámetros de ajuste adecuados ayuda a mejorar significativamente el rendimiento del modelo y a garantizar que esté optimizado para solucionar los problemas deseados.

Por lo tanto, podemos concluir que tanto el preprocesado de los datos como la selección adecuada de hiperparámetros son esenciales para construir modelos precisos y efectivos. Un enfoque cuidadoso y riguroso en estas etapas iniciales puede marcar una gran diferencia en la calidad de los resultados finales obtenidos.

3. Anexo

3.1. Librerías utilizadas

- **geopy**: https://geopy.readthedocs.io/en/stable/
- **geotext**: https://pypi.org/project/geotext/
- **nltk**: https://www.nltk.org/

3.2. Ficheros

- **requirements.txt**: Contiene las dependencias requeridas
- **instaladorDependencias.sh**: Automatiza la instalación de las dependencias requeridas
- **TweetsTrainDev.csv**: Dataset de los tweets sin preprocesar
- **datosProcesados.csv**: Dataset de los tweets ya procesados
- **preproceso.py**: Toma el dataset sin preprocesar y le aplica los preprocesos, generando así el archivo *datosProcesados.csv*
  - Llamada: python preprocesado.py TweetsTrainDev.csv
- **iteradorNaiveBayes.py**: Ejecuta el algoritmo de Naive Bayes de forma iterativa para hallar la mejor combinación de hiperparámetros.
- **iteradorLogisticRegression.py**: Itera el algoritmo de Logistic Regression para conseguir la mejor combinación de hiperparámetros.
- **logisticRegression.py**: Genera el modelo de Logistic Regression y lo guarda en un .sav
  - Llamada: python logisticRegression.py datosProcesados.csv
- **naiveBayes.py**: Genera el modelo de Naive Bayes y lo guarda en un .sav
  - Llamada: python naiveBayes.py datosProcesados.csv
- **nInstanciasTestyDev.py**: Guarda el dataset ya preprocesado y dividido en train y dev en dos archivos .csv