Clasificación de Tweets

Sistemas de Apoyo a la Decisión
2022-2023
...
Titulación:
Grado en Informática de Gestión y Sistemas de Información
...
3º Curso(2º Cuatrimestre)

Oier Arribas Adrián Cuadrón Danel Alonso Adrián López

26 de abril de 2023

Índice general

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

Hemos decidido usar una división de 80 % train y 20 % dev. Creemos que es una buena relación para que se pueda entrenar con suficientes datos y realice una correcta evaluación de los datos de dev. Al realizar esta división, nos dimos cuenta que al modificar el random_state(seed) de la aleatoriedad, había cambios en los resultados. Probando diferentes números (40,41,50,52...) nos hemos dado cuenta de que a pesar de la bonanza del sistema puede mejorar o empeorar al cambiar la distribución de los datos, no es suficiente para que eso influya mucho en el resultado. Para las pruebas hemos decidido coger random_state=50.

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	5827	2131	1641
Dev	1457	533	410

1.2. Cuadro: Distribución Train y Dev

1.3. Descripción del preproceso

La primera idea del equipo ha sido realizar el preproceso clásico de los textos. Esto lo realizamos con la librería que tiene el objeto de TfidfVectorizer(stop_words='english') añadiéndole varios parámetros para que lo haga automáticamente dentro del método de realizar el tf-idf.

La segunda idea fue hacer el preproceso con BOW (Bag of words), con la librería CountVectorizer que contiene el objeto de CountVectorizer(stop_words='english') añadiéndole varios parámetros para que lo haga automáticamente dentro del método de realizar el BOW.

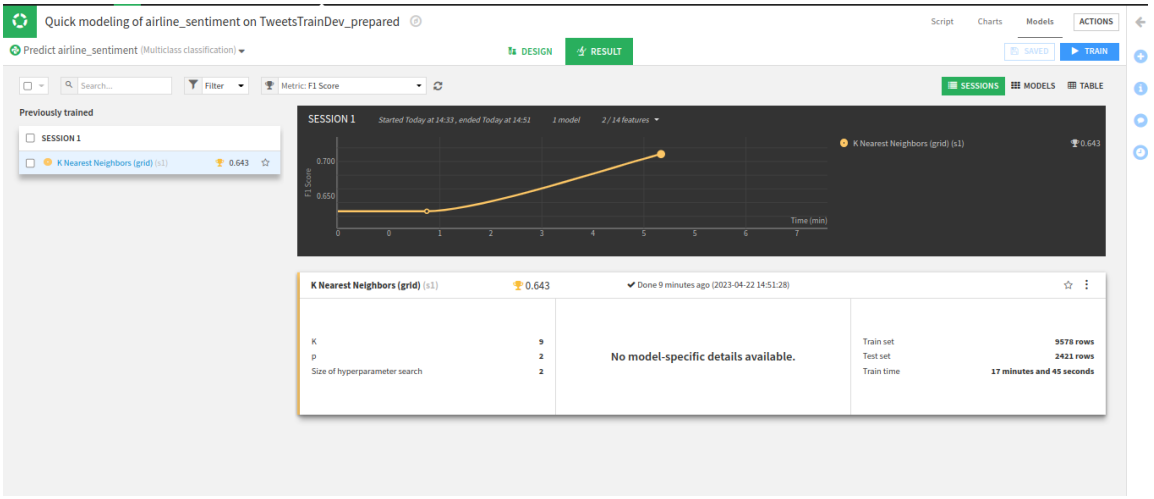
Por otro lado, hemos probado a separar la parte de preprocesar el texto con realizar el tf-idf o BOW. En este preproceso incluimos lematizar el texto(quitar plurales y coger la raíz de las palabras), pasar de mayúsculas a minúsculas, quitar las stop words, quitar caracteres especiales y tratar los emoticonos. Aparte de esto, también quitamos cualquier formato de URL("HTTP"por ejemplo).

Dentro de esta ultima forma de realizarlo, también hemos probado a quitar el tratamiento de los emoticonos, para ver si cambia algo en la bonanza del sistema, ya que puede haber gente que utilice los emoticonos de manera que pueda confundir al sistema.

Finalmente al entrenar los modelos con los distintos preprocesos que hemos mencionado anteriormente,como algunas clases de la muestra estaban desbalanceadas, entonces hicimos uso del oversample y undersample para balancear la muestra y comprobar si mejoraba la evaluación.

1.4. Primeros resultados

Para establecer una baseLine como objetivo mínimo para el entrenamiento del modelo, hemos utilizado un modelo de knn con el preprocesado básico con Dataiku y hemos establecido su resultado como baseLine. Este preprocesado consiste en una normalización del texto de los tweets(simplify Text)



1.5. Descripción del Proceso de Submuestreo o Sobremuestreo

Respecto al ajuste de muestras del conjunto del train y el dev tras haber probado diferentes configuraciones; utilizar la muestra sin alterarla,hacer un submuestreo ajustando las demás clases a la clase minoritaria y finalmente hacer un sobremuestreo ajustando las clases minoritarias a la clase mayoritaria. La configuración que mejores resultados nos ha dado ha sido esta ultima(oversampling).

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.

- **Multinomial Naive Bayes**:
  El algoritmo Naive Bayes Funciona calculando la probabilidad de que un texto pertenezca a una determinada categoría o clase en función de las frecuencias de las palabras en el documento respecto a la frecuencia en la que aparecen en otros. La variante multinomial se especializa para muestras con datos distribuidos multinomialmente.
  - Hiperparámetros: aplha
  - Link: Sklearn MultinomialNB
- **Bernoulli Naive Bayes**:
  Es parecido al MultinomialNB pero MultinomialNB trabaja con recuentos de ocurrencias, y BernoulliNB está diseñado para características binarias o booleanas.
  - Hiperparámetros: aplha
  - Link: Mixed Naive Bayes
- **Gaussian Naive Bayes**:
  Especialización del algoritmo de Naive Bayes para muestras con una distribución gausiana.
  - Hiperparámetros:
  - Link: Sklearn Gaussian Naive Bayes
- **Decision Tree Classifier**:
  Es un algoritmo que dado un conjunto de datos, genera un arbol de decisiones para generar predicciones de items nuevos segun sus caracteristicas.
  - Hiperparámetros: max_depth, min_samples_split,min_samples_leaf
  - Link: Sklearn Decision Tree Classifier

2.2. Resultados sobre el Development

En esta sección se presentan los resultados obtenidos para el development aplicando los diferentes algoritmos y formas de preproceso. Para maximizar los algoritmos hemos utilizado como metrica para la evaluacion el F-score weighted. A parte de esto, en cada prueba se hace un barrido del parámetro de alpha entre 1 y 5 para el algoritmo Naive Bayes. Para el algoritmo de Decision Tree hemos usado el barrido de hiperparametros de maxDepth del 3 al 15.

2.2.1. Multinomial Naive Bayes

- Aplicando BOW
  - No sample → 0.7216
  - Undersample (Not minority) → 0.702
  - Oversample (Not majority) → 0.73
- Aplicando preproceso con expresiones regulares + BOW
  - No sample → 0.7248
  - Undersample (Not minority) → 0.693
  - Oversample (Not majority) → 0.743
- Aplicando TF-IDF (preproceso automático)
  - No sample → 0.56
  - Undersample (Not minority) → 0.70
  - Oversample (Not majority) → 0.58
- Aplicando preproceso con expresiones regulares + TF-IDF
  - No sample → 0.59
  - Undersample (Not minority) → 0.71
  - Oversample (Not majority) → 0.60
  - quitando los emoticonos + undersample → 0.69

2.2.2. Bernoulli Naive Bayes

- Aplicando BOW
  - No sample → 0.7297
  - Undersample (Not minority) → 0.7189
  - Oversample (Not majority) → 0.72
- Aplicando TF-IDF
  - No sample → 0.7297
  - Undersample (Not minority) → 0.6957
  - Oversample (Not majority) → 0.739
- Aplicando preproceso con expresiones regulares + BOW
  - No sample → 0.6784
  - Undersample (Not minority) → 0.7046
  - Oversample (Not majority) → 0.6917
- Aplicando preproceso con expresiones regulares + TF-IDF
  - No sample → 0.6784
  - Undersample (Not minority) → 0.7184
  - Oversample (Not majority) → 0.6957

2.2.3. Gaussian Naive Bayes

- Aplicando BOW
  - No sample → 0.5242
  - Undersample (Not minority) → 0.474
  - Oversample (Not majority) → 0.551
- Aplicando TF-IDF
  - No sample → 0.5243
  - Undersample (Not minority) → 0.47
  - Oversample (Not majority) → 0.54
- Aplicando preproceso con expresiones regulares + BOW
  - No sample → 0.4974
  - Undersample (Not minority) → 0.47
  - Oversample (Not majority) → 0.52
- Aplicando preproceso con expresiones regulares + TF-IDF
  - No sample → 0.4966
  - Undersample (Not minority) → 0.48
  - Oversample (Not majority) → 0.53

2.2.4. Decision Tree

- Aplicando BOW
  - No sample → 0.6562
  - Undersample (Not minority) → 0.5964
  - Oversample (Not majority) → 0.6826
- Aplicando TF-IDF
  - No sample → 0.6164
  - Undersample (Not minority) → 0.5865
  - Oversample (Not majority) → 0.6080
- Aplicando preproceso con expresiones regulares + BOW
  - No sample → 0.6530
  - Undersample (Not minority) → 0.5909
  - Oversample (Not majority) → 0.6695
- Aplicando preproceso con expresiones regulares + TF-IDF
  - No sample → 0.6342
  - Undersample (Not minority) → 0.6029
  - Oversample (Not majority) → 0.6317

2.2.5. Optimizando los resultados de la clase negativa

Algoritmo	Combinación hyperparámetros	Prec	Rec	F-sco
Naive Bayes Multinomial BOW	alpha=1	0.78	0.96	0.86

2.1. Cuadro: Resultados

2.2.6. Discusión

Hemos comprobado que para optimizar la clase negativa, la mejor opción es hacer oversample o no hacer rebalanceo. Esto se debe a que es la clase mayoritaria, por lo tanto, el entrenamiento va a ser más eficaz, ya que va a tener muchas más muestras diferentes con respecto a las demás clases.

En cambio, al realizar undersample, eliminamos muchas muestras y el f-score en la clase negativa disminuye considerablemente, ya que el número de muestras que se obtienen es mucho menor.

2.2.7. Sin optimizar ninguna clase en particular

Algoritmo	Combinación hyperparámetros	Prec	Rec	F-sco
Naive Bayes Multinomial BOW	alpha=1	0.77	0.76	0.74

2.2. Cuadro: Resultados

Algoritmo	Combinación hyperparámetros	Prec	Rec	F-sco
Naive Bayes Bernoulli TF-IDF	alpha=1	0.76	0.76	0.739

2.3. Cuadro: Resultados

2.2.8. Discusión

En este caso, haciendo undersample, se consigue una mejor distribución de los datos. Además, cada clase en particular obtiene mejores resultados y estos están más equilibrados. Por lo tanto, el macro-fscore no va a tener influencia sobre la clase mayoritaria.

En cambio, al no balancear, en el macro-fscore tienen la misma influencia todas las clases por igual, con lo que las clases con peores resultados hacen que este disminuya demasiado.
Por este motivo, hemos utilizado como métrica de evaluación el weighted-fscore, que pondera las tres medidas de f-score en función de su peso en la muestra.

La combinación naive bayes multinomial con preproceso bow y oversampling obtiene mejores resultados para weighted average pero peores para macro average. La razón puede ser que hemos hecho oversampling y tiene en cuenta el peso de cada clase para el calculo.

2.3. Conclusión

Para la optimización de los resultados sobre la clase negativa finalmente se ha seleccionado Multinomial Naive Bayes con preproceso de expresiones regulares + BOW haciendo Oversample. Este modelo, es el que más f-score ha obtenido para clasificar las instancias negativas, con un f-score de 0.86.

Para la optimización de los resultados generales finalmente se ha seleccionado Multinomial Naive Bayes con preproceso de expresiones regulares + BOW por que es el que mejor weighted f-score ha obtenido.

Además, nos hemos dado cuenta que a la hora de visualizar el texto de los Tweets preprocesados, hay muchas menciones (denotadas con un @), por lo que el símbolo va a tener mucho peso a la hora de obtener la matriz de palabras. Esto no tiene sentido, ya que puede ser decisivo a la hora de predecir la

clase de una instancia, cuando en realidad no tiene ningún peso, por lo que sería conveniente preprocesar estas menciones para que no se tengan en cuenta en el entrenamiento.

2.4. Llamadas y requisitos

- LLamada al mejor mejores modelo optimizando clase negativa
  - Naive Bayes Multinomial expresiones regulares + BOW → $python plantillaProyecto.py -a naivebayes --tipo multinomial --vectorizar bow --preproceso 1 --oversample 1 -f TweetsTrainDev.csv -o resumenTweets.csv
- LLamadas a los dos mejores modelos sin optimizar ninguna clase
  - Naive Bayes Multinomial expresiones regulares + BOW → $python plantillaProyecto.py -a naivebayes --tipo multinomial --vectorizar bow --preproceso 1 --oversample 1 -f TweetsTrainDev.csv -o resumenTweets.csv

    Llamada para preprocesar el test → $ python plantillaTest.py -m naiveMultinomial.sav --preproceso 1 --vectorizar bow --oversample 1 --train TweetsTrainDev.csv -f TweetsTestSubSample.csv
  - Naive Bayes Bernoulli TF-IDF → $python plantillaProyecto.py -a naivebayes --tipo bernoulli --vectorizar tfidf --preproceso 0 --oversample 1 -f TweetsTrainDev.csv -o resumenTweets.csv

    Llamada para preprocesar el test → $ python plantillaTest.py -m naiveBernoulli.sav --preproceso 0 --vectorizar tfidf --oversample 1 --train TweetsTrainDev.csv -f TweetsTestSubSample.csv
- Librerias necesarias
  - pip install numpy
  - pip install pandas
  - pip install imblearn
  - pip install nltk. Además, hay que descargar los datos necesarios para la utilización de esta librería:
    1. nltk.download('stopwords')
    2. nltk.download('wordnet')

3. Problemas

Uno de los problemas que nos encontramos fue al realizar la transformación del tf-idf, ya que lo estábamos haciendo directamente sobre la matriz del pandas. Entonces al hacer el entrenamiento nos decía que había diferencias en la longitud de la parte de los datos(X) y la parte del target(Y). Con lo cual tuvimos que investigar cual era el causante del problema. Al final nos dimos cuenta de que el error se encontraba cuando hacíamos la transformación del tf-idf, que estábamos cogiendo el objeto de la matriz del pandas en si, en vez del contenido del atributo "text". Hicimos el cambio en el código (tfidf_vectorizer.fit_transform(trainX['text'])) y en ese momento ese problema quedó solucionado.

Después de arreglar el anterior, el entrenamiento del modelo (método fit()) también nos daba problemas al probar con el modelo de GaussianNB() para valores continuos. Aquí el problema era que lo que teníamos que pasar el vector del tf-idf a un "dense numpy array". Esto lo arreglamos añadiendo al train y al dev el método de .toarray().

Finalmente, a la hora de evaluar nuestro modelo con el test proporcionado, cuando preprocesábamos el test, nos daba el siguiente error (ValueError: X has 54 features, but MultinomialNB is expecting 11622 features as input). Este error ocurre porque la matriz X tiene 54 características, mientras que el modelo entrenado espera 11622 características. Entonces nos hemos dado cuenta de que las matriz utilizada para entrenar y predecir tienen que tener el mismo numero de características, ya que los modelos entrenados solo pueden hacer predicciones sobre características que hayan sido incluidas en la matriz de características de entrenamiento. Para arreglar el problema, hemos tenido que pasar la matriz que hemos utilizado para entrenar el modelo, para que el test se ajuste a esa matriz para las nuevas instancias.