Universidad del País Vasco

Escuela de Ingeniería de Bilbao

Sentiment Analisys

Sistemas de Apoyo a la Decisión

Bosco Aranguren Joel Bra Diego Marta Vicente Ayarza

Mayo 2022

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
- **SMOTE**: Synthetic Minority Oversampling Technique

1. Datos: Análisis y Preproceso

1.1. División entre Train y Dev

Se ha separado de forma estratificado el fichero original obteniendo un Train de tamaño 79820 instancias y un Develop de 19956 instancias.

Conjunto De Datos	% de instancias	Num. de instancias
Train	80	79820
Dev	20	19956

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	25590	7773	46457
Dev	6398	2015	11543

1.2. Cuadro: Distribución Train y Dev

Estos resultados se han obtenido antes de aplicar un oversampling a las clases minoritarias que se usará para obtener el mejor resultado al estimar las clases sin optimizar ninguna clase en particular.

1.3. Descripción del preproceso

Se han empleado únicamente los campos summary y reviewText, ya que al hacer un análisis de sentimientos de las reviews solo nos interesan esos dos campos por ser los que contienen información introducida por usuarios. Los demás campos no almacenan información introducida por el usuario, aunque serían útiles para estimar la opinión que se le daría a un productos, por ejemplo usando el ranking o el precio.

Tras seleccionar los campos que más nos convenían limpiamos el texto eliminando las stopwords [1], signos de puntuación y pasando todo a minúsculas. Después lematizamos [2] el texto para obtener el lema, es decir pasar de una forma flexionada (plural, femenino, conjugada...) a una forma estándar (lema). Por ejemplo estudiaré sería estudiar. También hemos realizado stemming [3] sobre los datos para obtener la raíz de las palabras. En este caso dado estudiar obtendríamos la raíz estudi-.

Para vectorizar el texto nuestra primera opción fue el CountVectorizer (BoW) [4]. Después decidimos transformarlo en tf-idf con el TFIDFTransformer [5] y los resultados mejoraron pero no conseguimos los mejores resultados hasta vectorizarlo directamente con tf-idf usando TFDIFVectorizer [6].

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

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

- **XGBoost**:
  - Hiperparámetros: n_estimators, max_depth
  - Link: XGBoost Documentation
- **Logistic Regression**:
  - Hipeparámetros: C, penalty, solver
  - Link: Sklearn Logistic Regression
- **Multinomial Naive Bayes**:
  - Hiperparámetros: aplha
  - Link: Sklearn MultinomialNB

2.1. Resultados sobre el Development

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

2.1.1. Optimizando los resultados de la clase negativa

Algoritmo	Combinación hyperparámetros	Prec	Rec	F-sco
XGB sin SMOTE	max_depth=8 n_estimators=1000	0.81	0.88	0.84
XGB 2 clases SMOTE	max_depth=8 n_estimators=1000	0.83	0.84	0.84
XGB SMOTE	max_depth=8 n_estimators=1000	0.82	0.86	0.84
LR sin SMOTE	C=1 penalty=l2 solver=lbfgs	0.80	0.87	0.83
LR 2 clases SMOTE	C=1 penalty=l2 solver=lbfgs	0.78	0.89	0.83
LR 2 clases sin SMOTE	C=1 penalty=l2 solver=lbfgs	0.84	0.82	0.83
LR sin SMOTE	C=10 penalty=l2 solver=lbfgs	0.81	0.86	0.83
LR SMOTE (clase 0)	C=10 penalty=l2 solver=lbfgs	0.76	0.91	0.83
XGB sin SMOTE	max_depth=6 n_estimators=100	0.79	0.86	0.82
LR SMOTE	C=1 penalty=l2 solver=lbfgs	0.82	0.81	0.82

2.1. Cuadro: Resultados

2.1.2. Discusión

Para obtener los mejores resultados de la clase negativa hemos decidido variar más el preprocesado que los hiperparámetros. En algunos casos el no hacer oversampling mejora los resultados de la clase negativa al haber un número relativo de instancias de la clase negativa mayor. También hemos probado a unir las clases neutra y positiva en una, de forma que los datos quedaban distribuidos de forma binaria.

Finalmente los resultados obtenidos con XGB han sido los mejores y muy similares en casi todos los casos. Los resultados que se muestran son los de la clase negativa, llegando los globales a un weighted average de 0.89.

2.1.3. Sin optimizar ninguna clase en particular

Algoritmo	Combinación hyperparámetros	Prec	Rec	F-sco
XGB SMOTE	max_depth=6 n_estimators=1000	0.83	0.84	0.84
XGB SMOTE	max_depth=8 n_estimators=1000	0.83	0.84	0.84
LR SMOTE	C=1 penalty=l2 solver=lbfgs	0.83	0.84	0.82
LR SMOTE	C=10 penalty=l2 solver=saga	0.83	0.80	0.81
LR SMOTE	C=10 penalty=l2 solver=lbfgs	0.82	0.79	0.80
MNB SMOTE	alpha=1	0.83	0.75	0.78

2.2. Cuadro: Resultados

2.1.4. Discusión

La combinación max_depth=6 y n_estimators=1000 con un oversampling de las clases minoritarias usando SMOTE obtiene mejores resultados. El uso del oversampling mejora notablemente los resultados ya que iguala el número de instancias de las tres clases haciendo el train más equitativo.

Uno de los problemas que hemos tenido que afrontar ha sido el tiempo de entrenamiento del algoritmo XGB, tardando cerca de una hora en algunas combinaciones lo cual hacia tedioso el tuning de hiperparámetros. De todas formas hemos podido probar varias combinaciones y en la tabla se muestran las más significativas.

2.2. Conclusión

Para ambos casos hemos usado el algoritmo XGB, muy popular en competiciones de Inteligencia Artificial [7]. El algoritmo es similar al random forest pero en vez de crear los árboles de forma aleatoria y independientemente uno de otro crea los arboles de forma secuencial recibiendo información de las instancias peor clasificadas para intentar mejorar su resultado.

Consideramos que los buenos resultados se han obtenido en parte por el preprocesado. Al hacer el preprocesado a parte utilizando librerías especificas y controlando como se hace cada paso individualmente hemos tenido un mayor control sobre los datos. El oversampling hecho con SMOTE [8] nos ha dado muy buenos resultados y al crear nuevas instancias sintéticas hemos evitado el overfitting que puede darse con un oversampling que duplica instancias. En relación al overfitting hemos probado con diferentes random states y distribuciones de train y dev para comprobar que los resultados se mantenían.

Bibliografía

[1] Banjoko, J. (Oct 20, 2021). *Removing stop words with NLTK library in Python.* Medium. https://medium.com/analytics-vidhya/removing-stop-words-with-nltk-library-in-python-f33f53556cc1

[2] NLTK. *Documentación nltk.stem.wordnet* https://www.nltk.org/_modules/nltk/stem/wordnet.html

[3] NLTK. *Documentación nltk.stem.porter* https://www.nltk.org/api/nltk.stem.porter.html?highlight=stemmer#nltk.stem.porter.PorterStemmer

[4] SKLEARN. *Documentación sklearn.feature_extraction.text.CountVectorizer* https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[5] SKLEARN. *Documentación sklearn.feature_extraction.text.TfidfTransformer* https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html#sklearn.feature_extraction.text.TfidfTransformer

[6] SKLEARN. *Documentación sklearn.feature_extraction.text.TfidfVectorizer* http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer

[7] KAGGLE. *XGBoost.* https://www.kaggle.com/code/dansbecker/xgboost/notebook

[8] IMBALANCED LEARN. *Documentación SMOTE.* https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html