Taks 1: Sentiment Analysis at Tweet level

Jaime Ferrando Huertas

Mayo 2021

Contents

1 Introducción **2**

2 Procesado de datos **2**

3 Modelado **2**

4 Conclusiones **4**

1 Introducción

En esta tarea vamos a diseñar, evaluar e implementar un sistema de reconocimiento de hate speech. Hemos usado el dataset de tweets de la competicion TASS 2017[1], contando con 3 conjuntos de datos: train, development y test. Los datos consisten en el texto del tweet y una etiqueta entre P (positivo), N (negativo), neutro(NEU) o ninguno (NONE). En esta tarea hemos creado un modelo clasificador de estos tweets en función del contenido del texto y su correspondiente preproceso.

2 Procesado de datos

Los datos provenientes del TASS 2017 consisten en ficheros XML, estos ficheros contienen multitud de texto para darle formato, nosotros hemos decidido preprocesarlo y guardarlo como fichero separado por comas para facilitar su posterior uso en otras librerías que hemos usado para el modelado. Hemos extraído cada tweet/etiqueta y eliminado el texto inherente al formato XML.

Una vez extraídos los tweets se ha realizado un preprocesado del texto de los mismos para limpiar elementos que podrían añadir ruido a los modelos. Se ha tokenizado el texto con el tokenizador *TweetTokenizer* de la libreria nltk. Tambien se probo a eliminar stopwords junto expresiones regulares para reemplazar hashtags, urls, usuarios, emojis por un token único de la categoría pero nos dio peores resultados.

Una vez procesado el texto se ha vectorizado para su entrada en los modelos. Hemos usado la librería sklearn parar crear las matrices TF-IDF correspondientes a cada tweet. Las TF-IDF, Term Frequency(TF) and Inverse Document Frequency(IDF) tienen dos partes: Frecuencia de término (TF) y Frecuencia de documento inverso (IDF). La frecuencia de término indica la frecuencia de cada una de las palabras presentes en el tweet. IDF realmente nos dice qué tan importante es la palabra para el tweet. Esto se debe a que cuando calculamos TF, le damos la misma importancia a cada palabra. Si la palabra aparece en el conjunto de datos con más frecuencia, entonces su valor de frecuencia de término (TF) es alto aunque no es tan importante para el documento.

3 Modelado

Para el modelado de esta tarea hemos usado una serie de modelos definidos en la librería de sklearn[2].

- RFC, Random Forest Classifier
- XGB, XGBoostClassifier
- SVC, Support Vector Machines.
- LinearSVC, Linear Support Vectors.

- GaussianNB, Naive Bayes Classifier.
- GradientBoostingClassifier, Gradient Descent Classifier.
- SgdClassifier, Linear classifiers with gradient descent training
- KneighborsClassifier, K-nearest neighbors vote classifier.
- MLPClassifier, Multi layer perceptron classifier.

Para evaluar los modelos hemos optado por el método de validación cruzada con 5 cortes junto con la métrica de F1-macro. Usar validación cruzada nos permite juntar el conjunto de train y development a la hora de entrenar, aumentando los datos disponibles para entrenar. La métrica F1-macro se define como la media de las puntuaciones F1 por clase / etiqueta y varía entre 1 y 0, siendo 1 el mejor valor posible.

Primero hemos evaluado los modelos con los valores por defecto para cada parámetro, después se ha realizado una búsqueda a fuerza bruta de los mejores valores dentro de un set predefinido por nosotros. Los resultados para las dos variantes (default, optimized) son los siguientes:

Model_name	F1-Macro default	F1-Macro optimized
RandomForestClassifier	0.31	0.34
XGBoostClassifier	0.37	0.39
SVC	0.30	0.39
LinearSVC	**0.39**	0.39
GaussianNB	0.30	0.30
GradientBoostingClassifier	0.34	0.38
SGDClassifier	**0.39**	0.41
KNeighborsClassifier	0.35	0.36
MLPClassifier	0.37	**0.42**

Recordamos que estamos presentamos son obtenidos evaluando con validación cruzada con 5 cortes, no sobre el conjunto de development. Podemos ver que los mejores modelos sin optimizar parametros son LinearSVC y SGDClassifier con un F1-Macro de 0.39. Sin embargo MLPClassifier es el modelo que más mejoría encuentra al hacer una optimización por búsqueda profunda de parámetros. SGDClassifier también obtiene un resultado muy alto al optimizar sus parámetros por lo que pensamos que sería una opción a evaluar, ya que es la mejor solución a mitad camino entre parámetros default o optimizados.

A continuación podemos ver un análisis de clasificación de nuestro mejor modelo MLPClassifier. No se ha generado esta tabla para todos los modelos para mantener la memoria breve.

	precision	recall	f1-score	support
N	0.60	0.65	0.62	637
NEU	0.17	0.18	0.17	202
NONE	0.32	0.28	0.30	201
P	0.60	0.53	0.56	474
accuracy			0.50	1514
macro avg	0.42	0.41	0.42	1514
weighted avg	0.50	0.50	0.50	1514

Figure 1: MLPClassifier classification report

4 Conclusiones

Este ha sido nuestro primer contacto con tareas de hate speech, hemos aprendido a procesar los datos y crear representaciones TF-IDF de los tweets. El modelado ha sido más sencillo, ya que habíamos usado la librería de sklearn anteriormente, pensamos que el mejor modelo para esta tarea sería el MLPClassifier pues obtiene los mejores resultados una vez optimizado, pero si elegimos no optimizar los parámetros del modelo deberíamos ir con SGDClassifier. También mencionar que probamos a realizar ensembles de los modelos con F1-Macro mayor que 0.40 pero nos dio peores resultados.

Hemos adjuntado el código necesario para replicar todos los experimentos junto con la entrega de esta memoria.

References

[1] Eugenio Martínez-Cámara, Manuel Díaz-Galiano, Miguel García-Cumbreras, Manuel García-Vega, and Julio Villena-Román. Overview of tass 2017. 09 2017.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.