UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Gabriel Gros

Memoria DETEST

Detection and classification of racial stereotypes in Spanish

Lo siento por adelante para la calidad de expresión de esta memoria dado que el Castellano no es mi primera lengua. Por otra parte, he hecho este trabajo solo, entonces, no tengo número ni nombre de equipo, aparte de mi propio nombre.

Natural Language Processing es un dominio en *Machine Learning* que da a un ordenador la oportunidad entender e interpretar el lenguaje humano. Los algoritmos de este tipo se han utilizado mucho en los últimos años como en recuperación de la información, traducción automática, análisis de opiniones o también en reconocimiento de voz.

En particular, la tarea DETEST manda detectar y clasificar los estereotipos en las frases de unos comentarios publicados en español, en respuesta a diferentes artículos de noticias en línea relacionados con la inmigración. Por así decirlo, es necesario desarrollar un modelo basado sobre la teoría del *Natural Language Processing* para después predecir si un texto totalmente desconocido del algoritmo contenga o no estereotipos.

Obtención del conjunto de datos

Primero, he construido el *dataset* que será al centro del entrenamiento del modelo. Para ser más preciso, es necesario recuperar los datos de entrenamiento (un vector cuyo cada coordenado contiene un comentario) y su clase (otro vector con 0 y 1, dependiendo de si el comentario contiene estereotipos o no). Después, he concatenado estos dos vectores del fichero *csv* para obtener un *dataframe*.

Preprocesamiento

Es fundamental preprocesar los datos para que el algoritmo pueda entenderlos. Esto elimina las fuentes de ruido (es decir, las informaciones en las que no nos interesamos) y, de esta manera, obtener mejores resultados. Esto consiste en varias etapas que he realizado gracias por la librería *nltk*.

Primero, tenemos que normalizar los datos. Para hacer esto, he eliminado los espacios y he pasado todas las letras por minúsculas. Después, he utilizado *Tokenization* que es un proceso para transformar una frase en un vector de palabras o de grupos de palabras (n-gramas) que se llaman *tokens*. Yo he elegido hacer vectores de palabras solas.

Por fin, he aplicado *Lemmatizing* que he elegido en lugar del *Stemming* porque es más preciso. De esta manera, las palabras que tienen una base común son consideras las mismas. Además, he hecho un *Stopping* para cada elemento para que las palabras las más comunas de

la lengua española (como determinantes, por ejemplo) sean eliminadas. Para las palabras que no son *stop words*, he utilizado *POS Tagging* que es un modelo estadístico capaz de entender la clase gramatical de una palabra. Mas precisamente, he filtrado todas las palabras que no estaban adjetivos, verbos o nombres.

	sentence	stereotype
0	La solución es desarrollar el pensamiento crít...	0
1	Hay que enseñar que la magia no existe.	0
2	Que todos los avances de la humanidad siempre ...	0
3	Enseñar en las escuelas la historia de las rel...	0
4	Desde las religiones de la edad de piedra hast...	0

	text_final
0	['solución', 'desarrollar', 'pensamiento', 'cr...
1	['enseñar', 'magia', 'existe']
2	['avances', 'humanidad', 'siempre', 'sido', 'p...
3	['enseñar', 'escuelas', 'historia', 'religione...
4	['religiones', 'edad', 'piedra', 'iglesia', 'c...

Comparación de los datos antes y después las etapas de preprocesamiento

Representación de textos

Una vez completado el preprocesamiento de los textos, he calculado la TF-IDF (*Term Frequency – Inverted Document Frequency*) de la librería *scikit-learn* de cada comentario para mesurar la importancia de las palabras en los comentarios y entre comentarios. Después, permite asignar un identificado único por cada palabra y obtener una matriz de *features* para entrenar y validar el algoritmo de *Machine Learning*.

En otros términos, esta etapa permite pasar un vector de vectores de palabras en una matriz dispersa de *features*. Tenía también que definir un número máximo de *features* y, por lo que he visto en mi investigación en Internet, he elegido $feature\_\max = 3000$.

Estrategia de clasificación

Una vez los *features* obtenidos para entrenar el modelo, he extractado los *labels* de los comentarios, es decir, su clase que estaban representados en la segunda columna del *dataframe*. La última etapa antes de empezar la codificación del clasificador es de hacer un *Split* de las observaciones: 70% de los datos son utilizados para el entrenamiento (y validación) del modelo y 30% para el test.

La librería *scikit-learn* es muy practica para modelizar clasificadores. En mi caso, he empleado seis clasificadores simples diferentes: SVM (*Support Vector Machine*), Regresión Logística, Arboles de Decisión y Red Neuronal (más precisamente, el *Perceptron* MLP de *scikit-learn*), Bosque Aleatorio y Adaboost. Además, he trabajado sobre la búsqueda de hiperparámetros óptimos gracias por el módulo de *scikit-learn GridSearch* para esperar obtener mejor escores. Por otra parte, no he empleado BERT, el modelo de Google, porque me daba errores.

Tenía también que elegir medidas de evaluación de los modelos. Dado que la proporción de individuos no es la misma entre las dos clases (solo 24% de los comentarios de entrenamiento contienen estereotipos), la precisión no es la mejora medida. Para abordar este problema, he

elegido emplear el *F1 score* que explica mejor la eficacidad de un modelo con clases desequilibradas. He también calculado la varianza de los modelos para evitar *over fitting*.

Para hacer la validación cruzada de los modelos, he empleado el *Kfold* standard con K=3 porque son los más empleados generalmente. Esta etapa es crucial para hacer aprendizaje supervisado porque permite cambiar los conjuntos de datos, lo que permite reducir la varianza del modelo para que sea lo más general posible.

Después de esto, he utilizado *Bagging* que es un método que construye varias instancias de un clasificador con subconjuntos aleatorios del conjunto de entrenamiento original. Después, combine las predicciones del clasificador sobre los conjuntos, lo que permite reducir la varianza del modelo.

Por fin, después de estas primeras comparaciones, he empleado combinaciones de modelos utilizando *Stacking* para ver si aprovechaban las adventajas de los diferentes modelos.

Comparación de los resultados

No he aplicado el *Bagging* y el *GridSearch* para el Red Neuronal porque mi ordenador no podía hacerlo.

		F1 score	Varianza
SVM	Standard	0.955	0.833
	GridSearch	0.869	0.544
	Bagging	0.522	0.030
	Bagging & GridSearch	0.605	0.065
Regresión Logistica	Standard	0.913	0.697
	GridSearch	0.913	0.697
	Bagging	0.640	0.000
	Bagging & GridSearch	0.651	0.000
Árbol de Decisión	Standard	0.998	0.991
	GridSearch	0.745	0.315
	Bagging	0.664	0.000
	Bagging & GridSearch	0.632	0.000
Red Neuronal	Standard	0.998	0.991
	GridSearch	0.998	0.991
	Bagging	0.659	0.000
Bosque Aleatorio	Standard	0.439	0.000
	GridSearch	0.000	0.000
	Bagging	0.428	0.000
	Bagging & GridSearch	0.428	0.000
Adaboost	Standard	0.622	0.000
	GridSearch	0.617	0.064
	Bagging	0.669	0.000
	Bagging & GridSearch	0.672	0.000

Considerando los *F1 scores* de todos los modelos conservados, los mejores modelos son el Árbol de Decisión y el Red Neuronal sin *Bagging* o *GridSearch*. Sin embargo, la varianza de estos modelos es mucho mayor que la de los clasificadores que utilizan estas operaciones (lo que es lógico dado que el *Bagging* combina las predicciones de un clasificador). Es por eso que he elegido conservar también el *GridSearch SVM* (porque tiene un buen *F1 score* y una varianza media) y el *Adaboost* con *GridSearch* y *Bagging* (porque tiene el mejor *F1 score* de los modelos empleando *Bagging* y una varianza muy baja).

Luego, quería implementar *Stacking* para combinar los cuatros modelos seleccionados. Sin embargo, solo he obtenido resultados para el *Stacking* con el *GridSearch SVM* y el Árbol de Decisión Standard (nunca he obtenido algo con las otras combinaciones).

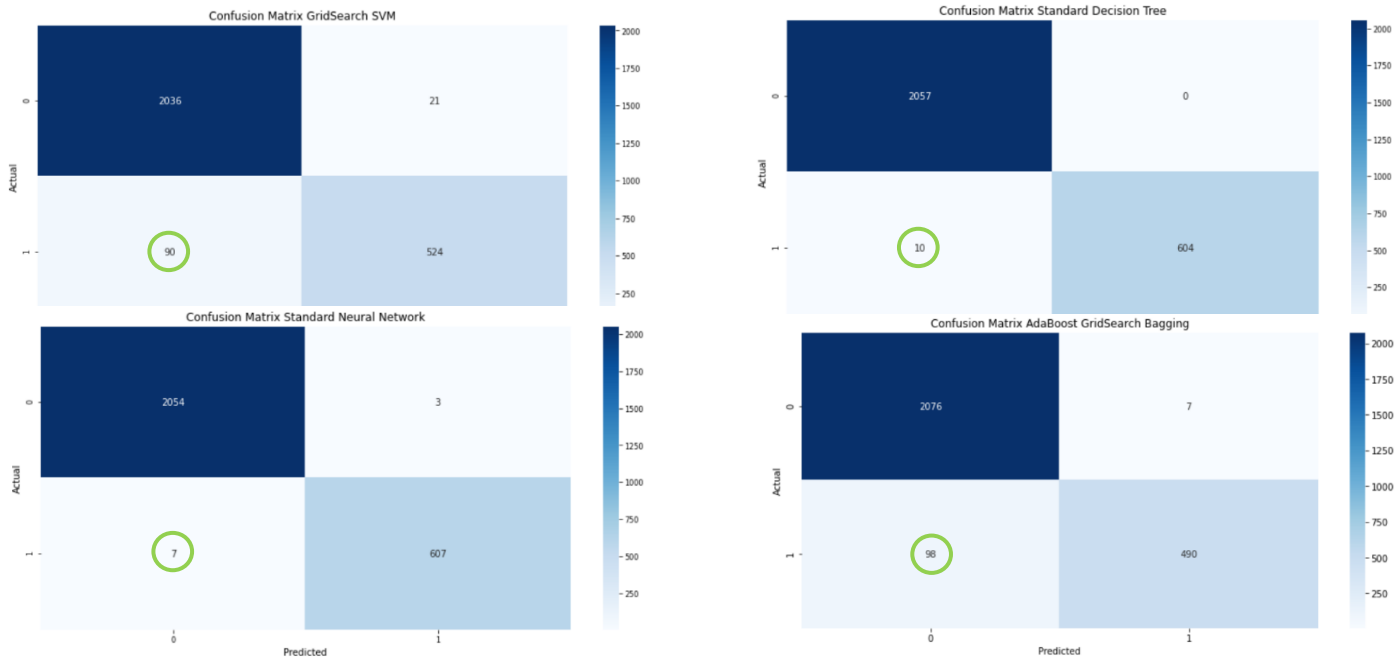
Stacking	F1 score	Varianza
GridSVM & Árbol de Decisión Standard	0.671	0.008

Al fin y al cabo, este clasificador creado por *Stacking* es menos interesante que el *AdaBoost* con *GridSearch* y *Bagging*. Es por eso que solo he continuado con los cuatros otros clasificadores conservados antes para el análisis de errores.

Análisis de errores

Esta parte es muy útil porque permite entender dónde vienen los errores de clasificación. Quería ver algunos ejemplos de comentarios que no se han clasificado correctamente y extractar características de estos comentarios de manera empírica.

Para ello, he elegido mostrar la matriz de confusión de cada modelo. Esto muestra cuántos elementos se han clasificado correctamente y distingue entre errores de falsa alarma (FP) y de negligencia (FN).



Se puede constatar que hay más errores de tipo FP que de tipo FN. Por así decirlo, entre los errores, se señalan más comentarios estereotipados que no estereotipados. Entonces, he decidido mirar los comentarios comunes de que plantean problemas FP a los clasificadores por separado y sacar conclusiones para entender mejor las fuentes de problemas. Entre las frases observadas, las causas de los problemas que parecen ser recurrentes son:

- Las secuencias de palabras escritas en mayúsculas, lo que es extraño porque he implementado una etapa de normalización de los textos en el preprocesamiento.

- Las frases largas con mucha puntuación. El problema puede ser que he procesado las palabras individualmente y no como n-gramas, lo que hace que la clasificación sea más confusa en las frases con más palabras.

- Pero también las frases muy cortas. Por ejemplo, la palabra "Hay que pasar esto" fue mal clasificada por todos los clasificadores. En este caso, por el contrario, es probablemente el reducido número de palabras que ha influido en la clasificación errónea.

- Por fin, el tema de la religión también ha sido calificado erróneamente como estereotipo por los clasificadores, quizás porque es un campo léxico que se ha asociado a estereotipos durante el entrenamiento.

Entonces, una idea de mejora potencial sería trabajar con n-gramas y jugar con el valor de n para encontrar uno que sea óptimo (tanto para que las palabras por sí solas conserven su importancia, como para que los grupos de palabras puedan ser útiles).

Predicción y conclusión

Por fin, tenía que elegir uno de los cuatros modelos haciendo una predicción. Dado que he separado los datos originales entre 70% para el entrenamiento (y la validación), siempre tengo 30% de datos de test que son etiquetados. En efecto, podría utilizar el *csv* test del concurso *DETEST*, pero no tendría una idea de la eficacidad del modelo. Al hacerlo, se consiguen estos resultados:

Clasificador	F1 score de la predicción correspondiente
SVM con GridSearch	0.674
Arbol de Decision Standard	0.683
Red Neuronal Standard	0.652
Adaboost con Bagging y GridSearch	0.685

Al fin y al cabo, el *Adaboost* con *GridSearch* y *Bagging* es el clasificador pareciendo el más eficiente (había enviado mi *run* con Árbol de Decisión para el concurso que presente muy buenos escores también). Más ampliamente, este proyecto me permitió ganar competencia en el procesamiento automático del lenguaje, pero más ampliamente, dar un paso atrás en los métodos de *Machine Learning* y cómo investigar para encontrar soluciones a un problema.