Tarea 3
Modelos de clasificación

Kin Nevárez
2 de junio de 2022
Maestría en Ciencia de Datos
Facultad de Ciencias Físico Matemáticas

[1]

***Resumen*—El código a continuación es una ejemplificación de diferentes algoritmos de aprendizaje supervisado que se pueden utilizar para la clasificación de tres tipos de sentimiento en comentarios de redes sociales: Positivo, Negativo y Neutral. Cabe destacar que el código presente en el apartado de Referencias también contiene funciones de pre-procesamiento de texto, sin embargo, no es el objetivo principal de la asignación. Este reporte es una continuación del Reporte Tarea 2 que también se encuentra en las Referencias.**

Palabras clave: *análisis de sentimiento, clasificación, KNN, SVM, Ridge, Random Forest, Naive Bayes.*

I. Introducción

El análisis de sentimiento es una técnica analítica que utiliza la estadística, el procesamiento del lenguaje natural y el aprendizaje automático para determinar el significado emocional de las comunicaciones.

Se puede usar en muchos ámbitos como:

- *Negocios:* En el campo del marketing, las empresas lo utilizan para desarrollar sus estrategias, comprender los sentimientos de los clientes hacia los productos o la marca, cómo las personas responden a sus campañas o lanzamientos de productos y por qué los consumidores no compran algunos productos.
- *Política:* En el ámbito político, se utiliza para realizar un seguimiento de la visión política, para detectar consistencia e inconsistencia entre declaraciones y acciones a nivel de gobierno. ¡También se puede usar para predecir los resultados de las elecciones!
- *Acciones públicas:* El análisis de sentimientos también se utiliza para monitorear y analizar los fenómenos sociales, para detectar situaciones potencialmente peligrosas y determinar el estado de ánimo general de la blogósfera.

[1]

El objetivo del presente análisis es encontrar el algoritmo que logre clasificar de mejor manera los sentimientos de los comentarios de redes sociales.

II. Datos

La base de datos utilizada tiene el nombre "First GOP Debate Twitter Sentiment" y fue obtenida del repositorio de Kaggle. Esta base de datos contiene más de 10,000 tweets sobre el primer debate presidencial de 2016 celebrado en Ohio.

III. Definiciones

1. tf-idf

TF-IDF es un método fiable para estimar la relevancia de un documento para un término. Cuanto mayor es la frecuencia de un término en un texto, se consigue un TF IDF alto, y menor es el número de documentos que mencionan ese término.

Este análisis se compone de dos partes TF (Term Frequency) y el IDF (Inverse Document Frequency)

TF: Sklearn calcula esta parte como el número de veces que una palabra aparece en un documento, es decir, su frecuencia absoluta. Cada documento tiene su propia frecuencia de términos. Se calcula de la siguiente manera:

$$TF_{i,j} = \sum_{k} n_{i,j} \quad (1)$$

IDF: Es una medida de qué tan común o raro es un término a lo largo de todos los documentos. Si la palabra es muy común, el idf (normalizado) se acercará a 0, por el contrario se acercará a 1 si es muy común. La forma en que Sklearn calcula esta métrica es:

$$IDF(t) = log\frac{1+n}{1+df(t)} + 1 \quad (2)$$

donde n es el número total de documentos y df(t) es la frecuencia del término t en el documento.

Y por último, el estadístico TF-IDF se calcula:

$$TFIDF(t,d) = tf(t,d) * idf(t) \quad (3)$$

2. Grid Search

La búsqueda de cuadrícula es esencialmente un algoritmo de optimización que permite seleccionar los mejores parámetros

para un problema de optimización de una lista de opciones de parámetros que se proporcionan, automatizando así el método de 'prueba y error'. Aunque se puede aplicar a muchos problemas de optimización, es más conocido por su uso en el Machine Learning para obtener los parámetros en los que el modelo ofrece la mejor precisión.

3. *Métricas de desempeño*

El desempeño de un modelo para clasificación puede ser medido mediante la matriz de confusión. Esta matriz muestra la proporción de verdaderos positivos (proporción de casos que fueron correctamente clasificados por el modelo en la clase correspondiente TP), falsos positivos (proporción de casos que fueron incorrectamente clasificados por el modelo en la clase correspondiente FP), verdaderos negativos (proporción de casos que fueron clasificados correctamente como no pertenecientes a la clase en cuestión TN) y falsos negativos (proporción de casos que fueron clasificados incorrectamente como no pertenecientes a la clase en cuestión FN).

Tomando en cuenta lo anterior , la matriz de confusión se observa como:

TABLA I
DISEÑO MATRIZ DE CONFUSIÓN

	Predicción	
Real	**Negativo**	**Positivo**
Negativo	TN	FP
Positivo	FN	TP

El modelo óptimo es aquél donde las diagonales de la matriz de confusión (TN y TP) presentan el 100% de los verdaderos valores.

Dos métricas muy valiosas para medir el desempeño final en base a la matriz de confusión son la sensibilidad (recall) y la precisión (precision). La primera es el número de elementos identificados correctamente como positivos del total de positivos verdaderos, mientras que la segunda es el número de elementos identificados correctamente como positivo de un total de elementos identificados como positivos. La forma de calcular ambas métricas es la siguiente:

$$sensibilidad = \frac{TP}{TP+TN} \quad (4)$$

$$precisión = \frac{TN}{TP+FP} \quad (5)$$

Por otra parte, el desempeño en términos generales se mide en la capacidad del modelo de detectar correctamente ambas clases de entre el total de observaciones. La exactitud (accuracy) de un modelo se puede determinar como:

$$exactitud = \frac{TP+TN}{TP+TN+FP+FN} \quad (6)$$

IV. METODOLOGÍA

A continuación, se presenta el planteamiento propuesto para dar solución a la problemática descrita en el apartado anterior.

Es importante destacar que todos los análisis fueron realizados con ayuda del lenguaje de programación Python, en la herramienta de Google Colab.

1. *Separación en set de entrenamiento y set de prueba*

Partiendo del conjunto de datos limpio, se realiza una separación automática con la función de *train_test_split* del módulo sklearn, separando en 80% entrenamiento y 20% prueba.

2. *Vectorización de comentarios*

Con la función de *TfidfVectorizer()*, usada para calcular las métricas de TF-IDF, se realiza una vectorización del texto en el que cada término o palabra se convierte en una característica y es colocado como columna, obteniendo una dimensión de n registros por m términos. En el caso en cuestión, las dimensiones son las siguientes:

Fig. 1. Dimensiones de conjuntos después de Vectorización TF-IDF

```
Set de entrenamiento: (11096, 8622)
-----------------------
Set de prueba: (2775, 8622)
```

Fuente: Elaboración propia

3. *Creación de modelos*

Se elaboró una función que hiciera una búsqueda en rejilla de los modelos probados que fuera iterando cada combinación de parámetros para cada modelo obteniendo la clasificación, predicción y evaluación de resultados en un mismo resultado.

A. *Random Forest*

Un modelo Random Forest está formado por un conjunto de árboles de decisión individuales, cada uno entrenado con una muestra ligeramente distinta de los datos de entrenamiento generada mediante bootstrapping). La predicción de una nueva observación se obtiene agregando las predicciones de todos los árboles individuales que forman el modelo.

El parámetro más importante en este algoritmo es el número de características aleatorias a muestrear en cada punto de división (max_features). Otro parámetro importante para el bosque aleatorio es el número de árboles (n_estimadores). Lo ideal es aumentarlo hasta que no se vea ninguna mejora en el modelo. Un buen valor podría ser una escala logarítmica de 10 a 1.000.
Nota: Por defecto n_estimators=100, max_features='sqrt'.

Los resultados del Random Forest fueron los siguientes:

Fig. 2. Resultados Random Forest

```
======================================================
Model: RandomForestClassifier()
------------------------------------------------------
Best Params:
{'max_features': 'sqrt', 'n_estimators': 200}
------------------------------------------------------
Accuracy Score: 0.8735135135135135
------------------------------------------------------
Classification Report:
              precision    recall  f1-score   support

    Negative       0.93      0.74      0.82       757
     Neutral       0.81      0.96      0.88       893
    Positive       0.90      0.90      0.90      1125

    accuracy                           0.87      2775
   macro avg       0.88      0.86      0.87      2775
weighted avg       0.88      0.87      0.87      2775
```

Fuente: Elaboración propia

Se observa que, con los parámetros seleccionados por la búsqueda en malla, la exactitud general del modelo fue muy buena (87%) y la clase para la que tuvo más problema en identificar fue la Neutral,obteniendo la menor precisión, pero la mayor sensibilidad, mientras que la clase negativa tuvo la menor precisión pero la mayor sensibilidad. Dado que el conjunto de datos es un conjunto relativamente balanceado, no se hace mucho hincapié en el f1-score.

La generalización de la matriz de confusión de este modelo se visualiza como:

TABLA II
MATRIZ DE CONFUSIÓN - RANDOM FOREST

	Predicción			
Real	**Negativo**	**Neutral**	**Positivo**	**Total**
Negativo	559	111	87	**757**
Neutral	15	857	21	**893**
Positivo	30	87	1,008	**1,125**
Total	**604**	**1,055**	**1,116**	**2,775**

B. Ridge Regression

La regresión Ridge es un modelo de regresión lineal penalizado para predecir un valor numérico, sin embargo, puede ser muy eficaz cuando se aplica a la clasificación.
Quizá el parámetro más importante que hay que ajustar es la fuerza de regularización (alfa). Un buen punto de partida podrían ser valores en el rango [0,1 a 1,0].
Nota: Por defecto alpha=1.0.

Los resultados del Ridge Regression para clasificación fueron los siguientes:

Fig. 3. Resultados Ridge Regression

```
======================================================
Model: RidgeClassifier()
------------------------------------------------------
Best Params:
{'alpha': 0.9}
------------------------------------------------------
Accuracy Score: 0.8702702702702703
------------------------------------------------------
Classification Report:
              precision    recall  f1-score   support

    Negative       0.88      0.80      0.84       757
     Neutral       0.85      0.89      0.87       893
    Positive       0.88      0.90      0.89      1125

    accuracy                           0.87      2775
   macro avg       0.87      0.86      0.87      2775
weighted avg       0.87      0.87      0.87      2775
```

Fuente: Elaboración propia

Se observa en este caso el modelo con el mejor parámetro de alfa fue de 0.9 y los resultados fueron más homogéneos entre clases, obteniendo entre 80% y 90% tanto la precisión como la sensibilidad de todas las clases. Aunado a esto, la exactitud general del modelo fue casi tan buena como la del Random Forest.

La matriz de confusión toma los siguientes valores:

TABLA III
MATRIZ DE CONFUSIÓN - RIDGE REGRESSION

	Predicción			
Real	**Negativo**	**Neutral**	**Positivo**	**Total**
Negativo	609	70	78	**757**
Neutral	46	791	56	**893**
Positivo	40	70	1,015	**1,125**
Total	**695**	**931**	**1,149**	**2,775**

C. K Nearest Neighbors

Este algoritmo de aprendizaje automático supervisado simple es uno que se basa en datos de entrada para calcular la "distancia" entre una observación y otra, luego clasifica las distancias y devuelve k vecinos más cercanos como similares.

El hiper parámetro más importante para KNN es el número de vecinos (n_neighbors), sin embargo, para este caso en particular, este parámetro será fijo, pues el número de vecinos siempre serán 3 (las tres categorías).

También puede ser interesante probar diferentes métricas de distancia (metric) para elegir la composición del vecindario, o probar la contribución de los miembros del vecindario mediante diferentes ponderaciones (weights).

Fig. 4. Resultados K Nearest Neighbors

```
======================================================
Model: KNeighborsClassifier(n_neighbors=3)
------------------------------------------------------
Best Params:
{'metric': 'manhattan', 'weights': 'distance'}
------------------------------------------------------
Accuracy Score: 0.5881081081081081
------------------------------------------------------
Classification Report:
              precision    recall  f1-score   support

    Negative       0.96      0.39      0.55       757
     Neutral       0.44      0.99      0.61       893
    Positive       0.98      0.41      0.58      1125

    accuracy                           0.59      2775
   macro avg       0.79      0.59      0.58      2775
weighted avg       0.80      0.59      0.58      2775
```

Fuente: Elaboración propia

Este algoritmo no mostró buenos resultados, obteniendo valores muy bajos en precisión para la clase Neutral y muy bajos en sensibilidad para la clase Negativa. Dados los resultados de este modelo, queda descartado como posible solución al problema en cuestión.

La matriz de confusión toma los siguientes valores:

TABLA IV
MATRIZ DE CONFUSIÓN - K NEAREST NEIGHBORS

	Predicción			
Real	**Negativo**	**Neutral**	**Positivo**	**Total**
Negativo	293	460	4	**757**
Neutral	7	880	6	**893**
Positivo	6	660	459	**1,125**
Total	**306**	**2,000**	**469**	**2,775**

Este algoritmo clasificó más de la mitad de los comentarios como sentimiento neutral, es por eso que pierde mucha precisión en esta clase para el conjunto de prueba, así como tiene muy poca capacidad para detectar los que efectivamente pertenecen a las clases Negativo y Positivo.

D. Support Vector Machines

El objetivo del algoritmo SVM es encontrar un hiperplano que separe de la mejor forma posible dos o más clases diferentes de puntos de datos. “De la mejor forma posible” implica el hiperplano con el margen más amplio entre las clases, representado por los signos más y menos. El margen se define como la anchura máxima de la región paralela al hiperplano que no tiene puntos de datos interiores.

El algoritmo sólo puede encontrar este hiperplano en problemas que permiten separación lineal; en la mayoría de los problemas prácticos, el algoritmo maximiza el margen flexible permitiendo un pequeño número de clasificaciones erróneas.

Este algoritmo es muy popular, muy eficaz y ofrece un gran número de hiper parámetros para ajustar.

Quizás el primer parámetro importante es la elección del núcleo que controlará la forma en que se proyectarán las variables de entrada. Hay muchos entre los que elegir, pero los más comunes son el lineal, el polinómico y el RBF, quizás sólo el lineal y el RBF en la práctica.

Otro parámetro crítico es la penalización (C), que puede tomar un rango de valores y tiene un efecto dramático en la forma de las regiones resultantes para cada clase. Una escala logarítmica podría ser un buen punto de partida.

Nota: Por default los parámetros están establecidos como C=1.0, kernel='rbf'.

Fig. 5. Resultados SVM

```
======================================================
Model: SVC()
------------------------------------------------------
Best Params:
{'kernel': 'linear'}
------------------------------------------------------
Accuracy Score: 0.8803603603603604
------------------------------------------------------
Classification Report:
              precision    recall  f1-score   support

    Negative       0.88      0.80      0.84       757
     Neutral       0.86      0.91      0.88       893
    Positive       0.90      0.91      0.90      1125

    accuracy                           0.88      2775
   macro avg       0.88      0.87      0.88      2775
weighted avg       0.88      0.88      0.88      2775
```

Fuente: Elaboración propia

Con un kernel lineal, se observa que hasta ahora el algoritmo de SVM ha obtenido la mejor exactitud a nivel general (88%), así como la mejor relación y balance de precisión y sensibilidad en todas las clases.

La matriz de confusión del SVM se observa:

TABLA V
MATRIZ DE CONFUSIÓN - SVM

	Predicción			
Real	**Negativo**	**Neutral**	**Positivo**	**Total**
Negativo	609	72	76	**757**
Neutral	44	813	36	**893**
Positivo	43	61	1,021	**1,125**
Total	**696**	**946**	**1,133**	**2,775**

Tal como las métricas anteriores, la diagonal principal de esta matriz de confusión también contiene valores muy elevados y más parecidos a los valores reales.

E. MultiNomial Naive Bayes

Los estimadores Naive Bayes son estimadores probabilísticos basados en el teorema de Bayes con la suposición de que existe una fuerte independencia entre las características. El teorema de Bayes nos ayuda a averiguar la probabilidad de que se produzcan sucesos basándose en un conocimiento previo de las condiciones que pueden estar relacionadas con el suceso. Los clasificadores Bayes ingenuos han funcionado bastante bien para la clasificación de documentos y las aplicaciones de filtrado de spam. Requiere una pequeña cantidad de datos de entrenamiento para establecer las probabilidades del teorema de Bayes y, por lo tanto, funciona con bastante rapidez.

Nota: Por defecto, alpha = 1.

Fig. 6. Resultados MNB

```
======================================================
Model: MultinomialNB()
------------------------------------------------------
Best Params:
{'alpha': 0.1}
------------------------------------------------------
Accuracy Score: 0.7556756756756757
------------------------------------------------------
Classification Report:
              precision    recall  f1-score   support

    Negative       0.77      0.69      0.73       757
     Neutral       0.86      0.65      0.74       893
    Positive       0.70      0.88      0.78      1125

    accuracy                           0.76      2775
   macro avg       0.78      0.74      0.75      2775
weighted avg       0.77      0.76      0.75      2775
```

Fuente: Elaboración propia

Pese a que este algoritmo suele funcionar bien para clasificación de texto, en esta ocasión su desempeño no fue tan bueno,comparándolo con el resto de los modelos. La exactitud apenas alcanza el 75%.

TABLA VI
MATRIZ DE CONFUSIÓN - MNB

	Predicción			
Real	**Negativo**	**Neutral**	**Positivo**	**Total**
Negativo	525	43	189	**757**
Neutral	81	578	234	**893**
Positivo	79	52	994	**1,125**
Total	**685**	**673**	**1,417**	**2,775**

La matriz de confusión también presenta poca variabilidad en los resultados, es decir, las predicciones no están tan desbalanceadas.

V. COMPARATIVA

A continuación se muestran representaciones gráficas de las matrices de confusión, donde es importante destacar que mientras más oscuro sea el color, mayor concentración de observaciones existe en ese cajón. Lo que se espera, como se ha mencionado con anterioridad, es que la concentración se vea en la diagonal, pues esto quiere decir que las predicciones fueron lo más acertadas posibles a los valores reales.

Fig. 7. Visualización Matrices de Confusión

Fuente: Elaboración propia

Gráficamente es mucho más fácil notar que los resultados del Random Forest, Ridge y SVM fueron muy similares y tuvieron los mejores resultados. Por otra parte, el clasificador con el peor desempeño fue el de KNN y es más visible la concentración de las predicciones en la categoría Neutral. Por último, el MNB tuvo un desempeño regular ya que sí se concentran los valores en la diagonal pero también hay más dispersión en el resto.

Posteriormente, para cada modelo se obtuvieron las siguientes métricas:

Fig. 8. Métricas de ejecución

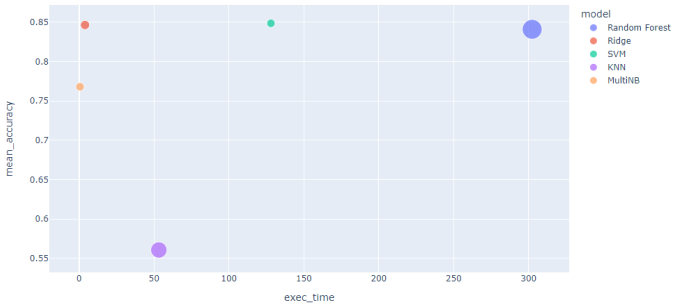
	model	accuracy	mean_accuracy	std_accuracy	exec_time
0	Random Forest	0.873514	0.841025	0.011431	302.392820
1	Ridge	0.870270	0.846522	0.002402	3.995471
2	SVM	0.880360	0.848684	0.002063	128.070101
3	KNN	0.588108	0.560743	0.007686	53.281089
4	MultiNB	0.755676	0.768025	0.002079	0.643825

Fuente: Elaboración propia

Donde accuracy es el desempeño del modelo con los parámetros seleccionados por el algoritmo de búsqueda en rejilla, el *mean_accuracy* es el promedio de la exactitud de todas las combinaciones de parámetros hechas por la búsqueda en rejilla, *std_accuracy* es la desviación estándar de la exactitud obtenida de todas las combinaciones de parámetros; y por último, *exec_time* es el tiempo que tardó la función de búsqueda en rejilla, predicción, clasificación y evaluación para cada modelo.

Con las métricas anteriores, se puede generar el siguiente gráfico:

Fig. 9. Visualización Métricas de ejecución



Fuente: Elaboración propia

Donde el eje X representa el tiempo de ejecución del modelo, el eje Y representa la exactitud promedio en cada iteración y el tamaño de la burbuja representa la desviación estándar de la exactitud por modelo.

Con este gráfico, es fácil notar que el modelo más tardado fue el Random Forest por mucho, tardando aproximadamente 5 minutos en ejecutar todas las combinaciones y además es el que presenta la mayor desviación en sus exactitudes, por lo que los resultados obtenidos en este ejercicio podrían no ser confiables en la realidad. En segundo lugar por tiempo de ejecución, tenemos el SVM, que tardó poco más de 2 minutos en ejecutarse pero es clasificador que presenta menor variabilidad en su exactitud, por lo que sus resultados son mucho más confiables. Después, tenemos el algoritmo de Ridge que tardó apenas 4 segundos en ejecutarse y tiene la tercera menor variabilidad en su exactitud, lo que lo hace tener ventaja en cuanto al tiempo de ejecución. Por último, el estimador de KNN y MNB no se toman en cuenta debido a su mal desempeño.

VI. Conclusiones

Dados los resultados obtenidos por los 5 algoritmos de clasificación, en este ejercicio se considera que los dos mejores modelos son SVM y Ridge.

En caso de que se tenga prioridad sobre el tiempo de ejecución y se tenga la opción de sacrificar un poco la exactitud, el algoritmo Ridge es la mejor opción, pues fue el tercer mejor clasificador en cuanto a métricas de desempeño, las cuales fueron muy cercanas a los dos algoritmos con mejor desempeño y además fue el segundo mejor en cuanto a tiempo de ejecución, con una diferencia considerable del resto.

Por otra parte, si se desea tener mayor control de las métricas de desempeño y se dispone del tiempo suficiente para entrenarlo, entonces se estaría considerando un SVM o Random Forest. Cabe destacar que mientras mayor sea el conjunto de datos, el tiempo de ejecución jugará en contra y los clasificadores se volverán aún más lentos.

Referencias

[1] Github Personal: https://github.com/KinMichelle/FCFM/blob/e9610a6400ea9f9bf30c72821cb902ea383f45aa/Procesa_y_clasif/Tarea_3/Tarea%203%20-%20KMNR.ipynb

[2] Conjunto de datos: First GOP Debate Twitter Sentiment. Analyze tweets on the first 2016 GOP Presidential Debate. https://www.kaggle.com/datasets/crowdflower/first-gop-debate-twitter-sentiment?resource=download

[3] En apoyo de: https://github.com/mayraberrones94/FCFM/blob/master/Semana_3_Prediccion.ipynb

[4] Máquinas de Vector Soporte (Support Vector Machines, SVMs) by Joaquín Amat Rodrigo, available under a Attribution 4.0 International (CC BY 4.0) at https://www.cienciadedatos.net/documentos/34_maquinas_de_vector_soporte_support_vector_machines

[5] Random Forest con Python by Joaquín Amat Rodrigo, available under a Attribution 4.0 International (CC BY 4.0) at https://www.cienciadedatos.net/documentos/py08_random_forest_python.html