Clasificación de textos con aprendizaje máquina (Programa 1)

Harold Eustaquio Amaya

UNAM, FI Procesamiento del Lenguaje Natural 2025-2

8 de septiembre de 2024

Resumen

Este trabajo evalúa diversos modelos de Machine Learning para la clasificación de correos electrónicos en spam y no spam, utilizando diferentes técnicas de vectorización y algoritmos. El objetivo principal fue comparar el rendimiento de estos modelos en términos de precisión, recall, y tiempos de entrenamiento y predicción.

Los modelos evaluados incluyen Árboles de Decisión con criterios de entropía y gini, Regresión Logística con distintos métodos de optimización y SVM con CountVectorizer y kernel lineal. Los Árboles de Decisión con el criterio gini y CountVectorizer alcanzaron una alta precisión de 0.851 y un F1 Score de 0.760, aunque con un tiempo de entrenamiento prolongado. Por otro lado, la Regresión Logística con TfidfVectorizer y newton-cg mostró un buen equilibrio entre precisión (0.718) y recall (0.811), con un tiempo de entrenamiento más corto. En contraste, el SVM con CountVectorizer y kernel lineal presentó un rendimiento deficiente, con una precisión muy baja y un recall elevado.

Los modelos basados en Árboles de Decisión y Regresión Logística demostraron un mejor balance entre precisión y recall, mientras que el SVM resultó menos efectivo, destacando la importancia de seleccionar el modelo adecuado para el problema y los datos disponibles.

1. Introducción

Este problema aborda la clasificación de correos electrónicos fraudulentos utilizando algoritmos de aprendizaje automático como Naive Bayes, Árboles de Decisión, SVM y Regresión Logística, en un contexto donde detectar fraudes de manera eficiente es crucial para la seguridad digital. La relevancia de este proyecto radica en su aplicación directa para identificar correos maliciosos, protegiendo tanto a empresas como a usuarios de posibles ciberataques.

El principal reto fue realizar una adecuada limpieza de datos, seleccionar las características relevantes y decidir qué vectorizador usar, ya que podríamos optar

por CountVectorizer o TF-IDFVectorizer. Además, fue necesario considerar el costo computacional de cada modelo.

En comparación con Naive Bayes y Árboles de Decisión, **Regresión Logística** con **newton-cg** y **class_weight: balanced** resultó ser la opción más eficiente. A pesar de que NB y Árboles ofrecieron tiempos de entrenamiento menores, no alcanzaron los niveles de precisión deseados. Además, el SVM, si no especificamos el máximo de iteraciones va a demorar deamasiado en converger, lo que lo hace inadecuado en términos de gasto computacional. Por ello, Regresión Logística es la más adecuada para cumplir con los objetivos del problema.

2. Metodología

2.1. Importing Libraries

- `pandas`
- `punctuation`
- `CountVectorizer`
- `TfidfVectorizer`
- `time`
- `MultinomialNB`
- `DecisionTreeClassifier`
- `LogisticRegression`
- `SVC`
- `accuracy_score`
- `precision_score`
- `recall_score`
- `f1_score`
- `ConfusionMatrixDisplay`
- `numpy`
- `matplotlib.pyplot`

2.2. Loading and Exploring the Data

Se cargan los archivos CSV `fraud_email_train.csv` y `fraud_email_test.csv`, ubicado en la carpeta `data`, en un DataFrame llamado `train` y `test` respectivamente, utilizando `pandas`. El parámetro `low_memory=False` se usa para evitar advertencias sobre el manejo de tipos de datos.

Finalmente, exploramos los datos para evaluar su calidad y obtener una visión general de su estructura. Esto incluye la revisión de las primeras filas, resúmenes estadísticos, información general del DataFrame y la distribución de las etiquetas en los conjuntos de entrenamiento y prueba. Estos pasos son cruciales para entender la distribución de las clases y la integridad de los datos.

2.3. Data Cleaning and Preparation

2.3.1. Concat and create a new column called id

Para tratar los datos de los conjuntos de entrenamiento y prueba de manera conjunta, se concatenan ambos DataFrames en uno solo. Para evitar la mezcla de datos y mantener la distinción entre los conjuntos, se crea una nueva columna llamada `id` en cada DataFrame, asignando el valor `'train'` al conjunto de entrenamiento y `'test'` al conjunto de prueba. Esto facilita el manejo y análisis de los datos sin perder la identificación de su origen.

2.3.2. Slicing important columns from data

Para enfocar el análisis en las columnas relevantes, se seleccionan y mantienen únicamente las columnas importantes del DataFrame. Esto incluye las columnas `From`, `To`, `Subject`, `Body`, `Bcc`, `Label`, y `id`. Este paso asegura que solo la información necesaria esté disponible para el análisis y procesamiento posterior.

2.3.3. Reformatting From and To

Para estandarizar el formato de las columnas `From` y `To`, se realizan modificaciones en los datos. Se reemplazan los puntos y el símbolo @ con espacios, y se elimina el dominio `com`. Esto se hace para uniformizar la representación de las direcciones de correo electrónico, facilitando su análisis y procesamiento posterior.

2.3.4. Creating new column: text

Se crea una nueva columna llamada `text` que combina los valores de las columnas `Subject`, `From`, `To`, `Bcc` y `Body` en un solo campo. Esto permite consolidar toda la información relevante en una única columna para facilitar el análisis. Posteriormente, se eliminan las columnas originales (`From`, `Body`, `To`, `Bcc` y `Subject`) del DataFrame para simplificar la estructura de los datos y reducir la redundancia.

2.3.5. Convert text to string

Para asegurar la consistencia en el manejo de datos, se convierte el contenido de la columna `text` a tipo cadena en el DataFrame. Esto se hace aplicando una función que convierte cada valor a una cadena si no es un valor nulo; de lo contrario, se mantiene como `NaN`. Este proceso garantiza que todos los datos de texto estén en formato de cadena, facilitando su posterior procesamiento y análisis.

2.3.6. Delete punctuation

Para limpiar el texto en el DataFrame, se eliminan los signos de puntuación. Se convierte todo el texto a minúsculas y se reemplazan los signos de puntuación

por una cadena vacía. Este proceso asegura que el texto sea uniforme y libre de caracteres innecesarios, lo que facilita el análisis posterior.

2.3.7. Split in train and test

Para separar los datos en conjuntos de entrenamiento y prueba, se divide el DataFrame combinado en dos DataFrames distintos basados en la columna `id`. Se filtra el DataFrame para obtener los datos correspondientes a `'train'` y `'test'`, y luego se elimina la columna `id` que se utilizó para la concatenación. Esto asegura que los datos estén correctamente segregados y listos para su análisis o modelado, sin la columna adicional que se usó para identificarlos.

2.3.8. Drop duplicated values

Para asegurar la calidad de los datos y evitar la redundancia, se eliminan los valores duplicados de los conjuntos de entrenamiento y prueba. Esto se realiza utilizando la función `drop_duplicates()` en ambos DataFrames, asegurando que cada conjunto contenga solo entradas únicas y, por lo tanto, mejorar la precisión en el análisis y modelado posterior.

2.3.9. Dealing with missing values

Para manejar los valores faltantes en los datos, se utiliza el `TfidfVectorizer` para transformar el texto en una matriz de características, excluyendo palabras comunes y limitando el número de características a 5000. Primero, se separan los textos de las clases 0 y 1 en los conjuntos de entrenamiento y prueba. Luego, se eliminan los valores nulos en la clase 1 del conjunto de entrenamiento y se ajusta el `TfidfVectorizer` en los datos limpios. A continuación, se crea una cadena de palabras características (features) y se usa para rellenar los valores faltantes en los textos de ambas clases. Esto asegura que los valores faltantes en los datos sean reemplazados de manera consistente sin perder información clave.

2.4. Splitting and Vectorizing Data

2.4.1. CountVectorizer

Para convertir el texto en una representación numérica, se utiliza `CountVectorizer` con la opción `stop_words='english'` para eliminar palabras comunes en inglés que no aportan valor analítico.

2.4.2. TfidfVectorizer

Para convertir el texto en una representación numérica, se utiliza `TfidfVectorizer` con la opción `stop_words='english'` para eliminar palabras comunes en inglés que no aportan valor analítico.

3. Experimentos y resultados

3.1. Naive Bayes

Ninguno de los modelos cumple completamente con el objetivo de tener un máximo de 82 FN. El modelo con TF-IDF tiene menos FP, cumpliendo con el límite de 141, pero aún tiene más FN de los deseado. Si es posible, ajustar los parámetros o técnicas podría ayudar a alcanzar los objetivos de la matriz de confusión.

3.1.1. MultinomialNB with CountVectorizer

Accuracy	0.996166
Precision	0.668192
Recall	0.674365
F1 Score	0.671264
Train Duration	0.397408
Predict Duration	0.085757
Confusion Matrix	$\begin{bmatrix} 74023 & 145 \\ 141 & 292 \end{bmatrix}$

Cuadro 1: NB with CountVectorizer | alpha: 1e-15

3.1.2. MultinomialNB with TfidfVectorizer

Accuracy	0.997198
Precision	0.861290
Recall	0.616628
F1 Score	0.718708
Train Duration	0.244750
Predict Duration	0.071484
Confusion Matrix	$\begin{bmatrix} 74125 & 43 \\ 166 & 267 \end{bmatrix}$

Cuadro 2: NB with TF-IDF Vectorizer | alpha: 1e-15

3.2. Decision Trees

Ninguno de los modelos cumple completamente con el objetivo de un máximo de `82 FN` y `141 FP`.

- El modelo con `gini | CountVectorizer` muestra la mejor precisión y F1 Score, con FP dentro del límite deseado pero FN superior.
- El modelo con `gini-balanced | CountVectorizer` tiene más FP y FN que el límite

- El modelo `entropy | TfidfVectorizer` y `gini | TfidfVectorizer` también tienen FN superiores a 82, aunque el FP está dentro del límite en algunos casos.

3.2.1. entropy | CountVectorizer

Accuracy	0.997158
Precision	0.791557
Recall	0.692841
F1 Score	0.738916
Train Duration	90.058236
Predict Duration	0.086102
Confusion Matrix	$\begin{bmatrix} 74089 & 79 \\ 133 & 300 \end{bmatrix}$

Cuadro 3: Decision Tree with CountVectorizer | entropy

3.2.2. entropy-balanced | CountVectorizer

Accuracy	0.996769
Precision	0.700837
Recall	0.773672
F1 Score	0.735456
Train Duration	95.310890
Predict Duration	0.070017
Confusion Matrix	$\begin{bmatrix} 74025 & 143 \\ 98 & 335 \end{bmatrix}$

Cuadro 4: Decision Tree with CountVectorizer | entropy-balanced

3.2.3. entropy | TfidfVectorizer

Accuracy	0.997172
Precision	0.777500
Recall	0.718245
F1 Score	0.746699
Train Duration	182.346055
Predict Duration	0.077474
Confusion Matrix	$\begin{bmatrix} 74079 & 89 \\ 122 & 311 \end{bmatrix}$

Cuadro 5: Decision Tree with TfidfVectorizer | entropy

3.2.4. entropy-balanced | TfidfVectorizer

Accuracy	0.996756
Precision	0.716553
Recall	0.729792
F1 Score	0.723112
Train Duration	142.370577
Predict Duration	0.075831
Confusion Matrix	$\begin{bmatrix} 74043 & 125 \\ 117 & 316 \end{bmatrix}$

Cuadro 6: Decision Tree with TfidfVectorizer | entropy-balanced

3.2.5. gini | CountVectorizer

Accuracy	0.997480
Precision	0.851003
Recall	0.685912
F1 Score	0.759591
Train Duration	299.344015
Predict Duration	0.091702
Confusion Matrix	$\begin{bmatrix} 74116 & 52 \\ 136 & 297 \end{bmatrix}$

Cuadro 7: Decision Tree with CountVectorizer | gini

3.2.6. gini-balanced | CountVectorizer

.

Accuracy	0.995429
Precision	0.576923
Recall	0.796767
F1 Score	0.669253
Train Duration	148.802921
Predict Duration	0.073667
Confusion Matrix	$\begin{bmatrix} 73915 & 253 \\ 88 & 345 \end{bmatrix}$

Cuadro 8: Decision Tree with CountVectorizer | gini-balanced

3.2.7. gini | TfidfVectorizer

Accuracy	0.997051
Precision	0.773779
Recall	0.695150
F1 Score	0.732360
Train Duration	678.130970
Predict Duration	0.122023
Confusion Matrix	$\begin{bmatrix} 74080 & 88 \\ 132 & 301 \end{bmatrix}$

Cuadro 9: Decision Tree with TfidfVectorizer | gini

3.2.8. gini-balanced | TfidfVectorizer

Accuracy	0.996113
Precision	0.645030
Recall	0.734411
F1 Score	0.686825
Train Duration	213.855259
Predict Duration	0.096307
Confusion Matrix	$\begin{bmatrix} 73993 & 175 \\ 115 & 318 \end{bmatrix}$

Cuadro 10: Decision Tree with TfidfVectorizer | gini-balanced

3.3. Logistic Regression

3.3.1. newton-cg | CountVectorizer

Accuracy	0.997265
Precision	0.811989
Recall	0.688222
F1 Score	0.745000
Train Duration	152.914762
Predict Duration	0.053366
Confusion Matrix	$\begin{bmatrix} 74099 & 69 \\ 135 & 298 \end{bmatrix}$

Cuadro 11: Logistic Regression with CountVectorizer | newton-cg

3.3.2. newton-cg | balanced | CountVectorizer

El modelo `newton-cg | balanced | CountVectorizer` cumple con el objetivo de mantener `Falsos Negativos (FN) = 82` y `Falsos Positivos (FP) = 138`, ajustándose a las condiciones deseadas. Este modelo presenta una alta

precisión (0.7178) y un F1 Score (0.7614) equilibrado, destacándose en la detección de positivos con un buen compromiso entre precisión y recall. Comparado con otros modelos de regresión logística, como newton-cg | CountVectorizer, que tiene un mayor FN = 135, o modelos con TfidfVectorizer, que tienen menor precisión y F1 Score, newton-cg | balanced | CountVectorizer ofrece una combinación más efectiva de precisión y control de FN y FP, a pesar de su mayor tiempo de entrenamiento y predicción.

Accuracy	0.997051
Precision	0.717791
Recall	0.810624
F1 Score	0.761388
Train Duration	152.528993
Predict Duration	0.041005
Confusion Matrix	$\begin{bmatrix} 74030 & 138 \\ 82 & 351 \end{bmatrix}$

Cuadro 12: Logistic Regression with balanced | CountVectorizer | newton-cg

3.3.3. newton-cg | TfidfVectorizer

Accuracy	0.995188
Precision	0.824561
Recall	0.217090
F1 Score	0.343693
Train Duration	10.309901
Predict Duration	0.027204
Confusion Matrix	$\begin{bmatrix} 74148 & 20 \\ 339 & 94 \end{bmatrix}$

Cuadro 13: Logistic Regression with TfidfVectorizer | newton-cg

3.3.4. newton-cg | balanced | Tfidfvectorizer

Accuracy	0.987922
Precision	0.313099
Recall	0.905312
F1 Score	0.465282
Train Duration	9.461586
Predict Duration	0.023322
Confusion Matrix	$\begin{bmatrix} 73308 & 860 \\ 41 & 392 \end{bmatrix}$

Cuadro 14: Logistic Regression with balanced | Tfidfvectorizer | newton-cg

3.4. SVM

El modelo `SVM: CountVectorizer | linear` tiene una precisión muy baja (`0.0061`) y un alto número de `Falsos Positivos (FP) = 69378`, a pesar de tener pocos `Falsos Negativos (FN) = 6`. Esto se debe a que se utilizaron pocas iteraciones para evitar un alto gasto computacional, ya que el `kernel: sigmoid` en SVM demoró 4 horas en converger en pruebas anteriores.

3.4.1. kernel: linear

Accuracy	0.069932
Precision	0.006117
Recall	0.986143
F1 Score	0.012159
Train Duration	30.404905
Predict Duration	7.997924
Confusion Matrix	$\begin{bmatrix} 4790 & 69378 \\ 6 & 427 \end{bmatrix}$

Cuadro 15: SVM with CountVectorizer | linear

4. Conclusiones

El modelo `newton-cg | balanced | CountVectorizer` es el único que cumple con el objetivo de mantener un máximo de `82 Falsos Negativos (FN)` y `141 Falsos Positivos (FP)`, con `FN = 82` y `FP = 138`. Este modelo ofrece un buen equilibrio entre precisión y recall.

En comparación, los modelos de regresión logística con `TfidfVectorizer` y el `SVM: CountVectorizer | linear` muestran un rendimiento inferior debido al desbalance de datos y la complejidad del problema. El modelo `SVM` tiene una precisión extremadamente baja y una alta tasa de `FP`. En general, los modelos de regresión logística ofrecen mejores resultados en términos de precisión y control de `FN` y `FP`, a pesar de las dificultades en el manejo de datos desbalanceados y la complejidad inherente a los modelos.

Referencias

scikit-learn developers, *scikit-learn: Machine Learning in Python*, 2024. `https://scikit-learn.org/stable/`

pandas development team, *pandas: Python Data Analysis Library*, 2024. `https://pandas.pydata.org/`

Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, O'Reilly Media, 2019.