Universidad del Valle de Guatemala

Facultad de ingeniería

Inteligencia Artificial

Catedrático: Alberto Suriano

Laboratorio 1

Nelson Eduardo García Bravatti 22434

Ricard Andrés Chuy 221007

Guatemala, enero de 2025

Task 1 - Regresión Lineal

1.1 Considera un modelo de regresión lineal con dos características, X_1 y X_2, y sus pesos correspondientes w_1 y w_2. Si el modelo predice una salida y mediante la ecuación $y = 2w_1X_1 + 3w_2X_2 + 1$, ¿cuál es la interpretación del coeficiente $3w_2$ en el contexto del modelo?

R// En principio el termino w_2 es parte del vector w con la cantidad de features de las muestras que se probarán en el modelo y ayuda a representar la pendiente de la regresión lineal múltiple junto con w_1. El 3 vendría siendo una constante la cual complementa al peso w_2, dándole más cambio a X_2, se puede asumir que X_2 tiene una mayor importancia que X_1 representando una mayor tasa de cambio en la salida del modelo.

1.2 Explica el concepto de **multicolinealidad** en el contexto de la regresión lineal. ¿Cómo afecta la multicolinealidad a la interpretación de los coeficientes de regresión individuales?

R// En general la multicolinealidad ocurre cuando hay una correlación muy alta entre 2 o más variables. Ósea que una variable influye directamente de una manera muy evidente sobre otra. En el contexto de la regresión lineal pasa lo mismo pero la diferencia clave es que la multicolinealidad ocurre cuando 2 variables independientes (predictoras/features) tienen una relación muy alta en un modelo. Que 2 variables estén relacionadas puede tener distintas consecuencias, por ejemplo, los datos del modelo se vuelven muy "sensibles". Específicamente con la interpretación de coeficientes de regresión individuales, estos pueden ser inestables y cambiar significativamente si se agregan, cambian o eliminan variables del modelo. Además, que es más difícil determinar cuál variable está afectando a cuál si la correlación es demasiado alta.

Task 2 - Clasificación de Sitios de Phishing Regresión Logística y KNN

Para esta parte se nos solicitó acerca del balanceo de los datos y revisar si este procedimiento ya había sido realizado. Según la página donde se descarga el data set, este ya se encuentra balanceado con 50% de casos legítimos y 50% de phishing. Es más, para comprobarlo se usó pandas y en efecto habían 5715 de cada uno. Posteriormente se separaron los datos en 80% para training y otro 20% para testing de igual manera con la librería de pandas. Es importante mencionar que también nos aseguramos de que al tomar el 80% de los datos estos queden balanceados y fueran seleccionados de forma aleatoria para evitar sesgo. Dentro del repositorio esto se realizó dentro del archivo data_filtering.py.

Para la parte 2.1 se hizo uso de la métrica de desempeño de accuracy. Nos pareció una buena idea tomando en cuenta que intentamos reducir el sesgo lo mayor posible y que los datos estuvieran balanceados en todo momento. Es importante recordar que el accuracy se hace siempre solo con datos que estén correctamente balanceados.

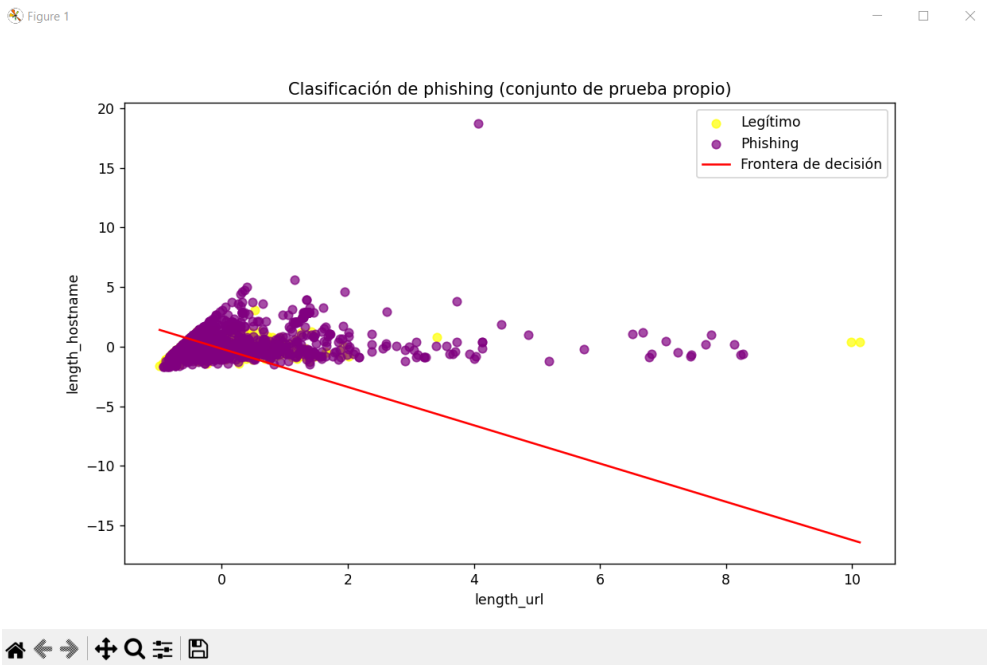
Enlace al repo: https://github.com/chuy-zip/AI_LAB1

2.1 - Regresión Logística

Elegimos las variables del largo del url y el largo del hostname como punto de partida para el entreno del modelo. Según la información de la tabla estas son 2 medidas que tienden a ser

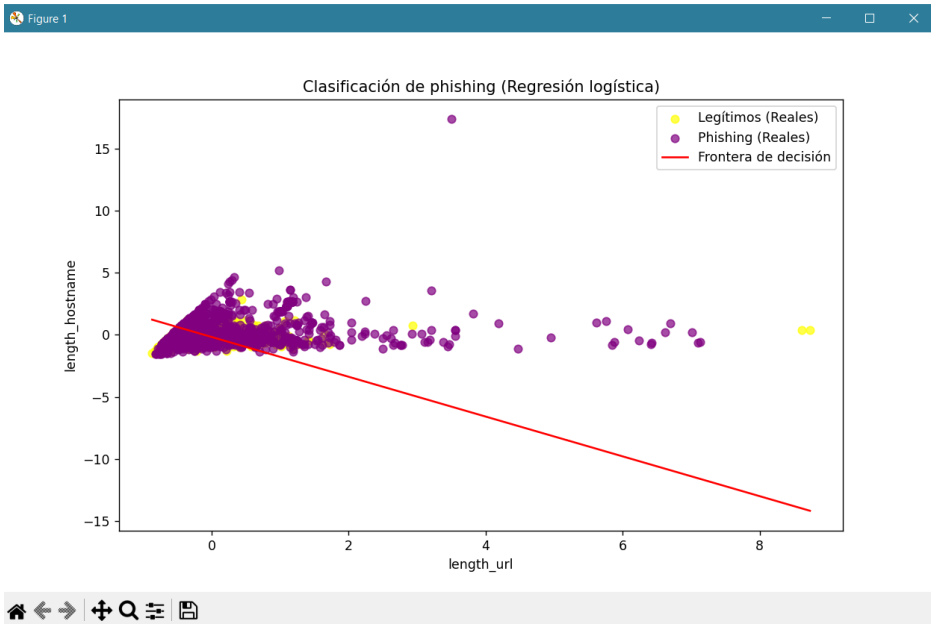
buenos indicadores de phishing. La idea era tratar de mantener consistencia entre los datos teniendo valores de rangos similares para los features. Muchas de las columnas tenían valores binarios por lo que quisimos probar con columnas con distintos valores. Otro ajuste que se realizó a la data set antes de procesarlo, cambiar la columna de estatus a algo binario para poder hacer una clasificación. Colocamos a el status de legitimo como 0 y phishing como 1.

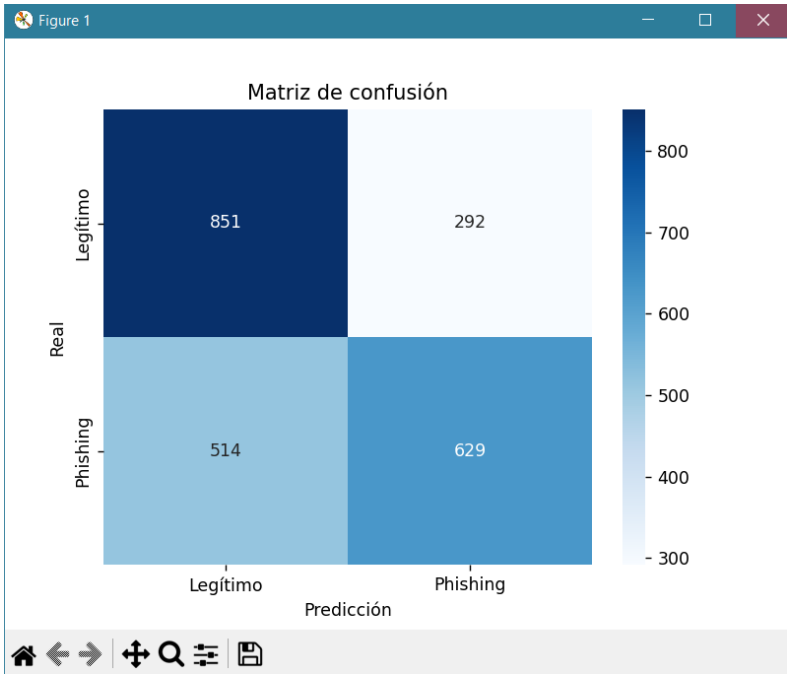
Realizándolo paso a paso




```
Length: 89, dtype: object
Pesos finales: [0.90489477 0.56413875]
Sesgo final: 0.09770203502777959
Precisión del modelo en el conjunto de prueba: 0.65
```

Usando scikit learn






```
PS C:\Users\andre\Desktop\Chuy\CUARTO 1\IA\lab1> & C:/Users/andre/AppData/Local/Programs/Python/Python312/python.exe
Precisión del modelo: 0.65
Matriz de confusión:
 [[851 292]
 [514 629]]
Informe de clasificación:
              precision    recall  f1-score   support

           0       0.62      0.74      0.68      1143
           1       0.68      0.55      0.61      1143

    accuracy                           0.65      2286
   macro avg       0.65      0.65      0.64      2286
weighted avg       0.65      0.65      0.64      2286
```

Reflexión de los resultados:

¿Cuál implementación fue mejor? ¿Por qué?

Realmente los resultados de ambas en términos de acccuracy fueron prácticamente los mismos. Ambos con un 0.65 (65%) de accuracy. Ahora bien, esto no quiere decir que el modelo sea perfecto, seguramente haya una mejor combinación de variables para el entreno del modelo, pero los resultados al final terminaron siendo consistentes. Lo que más se logró diferenciar es el decision boundary (la línea recta del gráfico). Pero sin duda alguna la mejor implementación es la que hace uso de la librería, mayormente por temas de simplicidad y de utilidad. La implementación fue bastante más sencilla y además se tienen varias más opciones para ver los resultados. Como se puede observar hasta colocamos la matriz de confusión para ver el desempeño del entrenamiento y hay más información con respecto a las métricas.

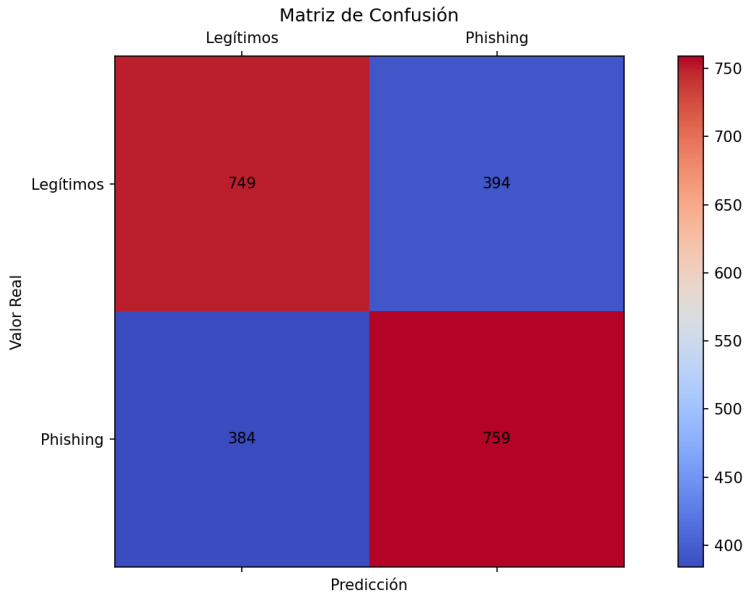
2.2 - K-Nearest Neighbors

Realizándolo paso a paso

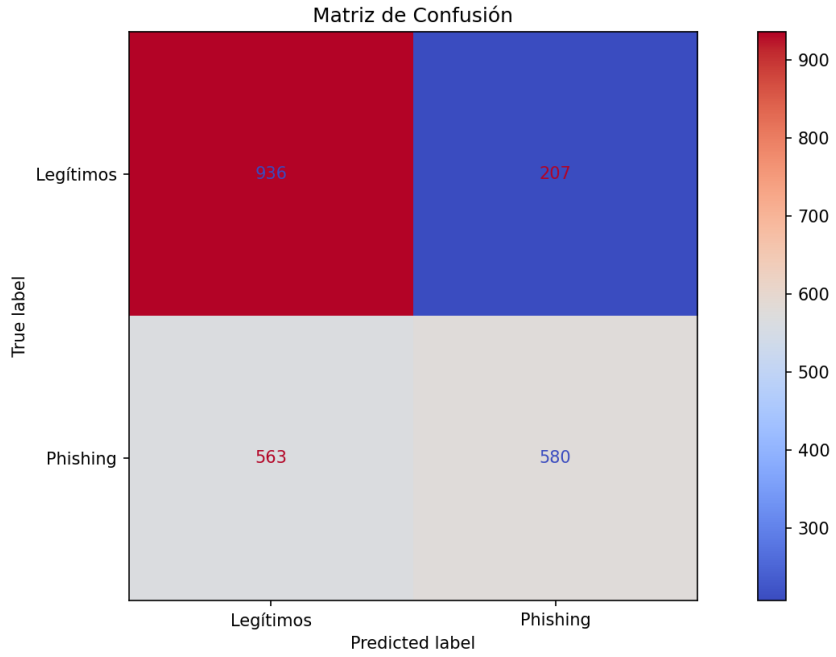
Se utilizó el valor de k=3, ya que es un número impar que minimiza los empates en problemas de clasificación binaria. Este valor puede ajustarse para observar su efecto en el desempeño.

La normalización de las características no fue necesaria, ya que las dos variables (longitud del URL y longitud del hostname) están en escalas comparables.

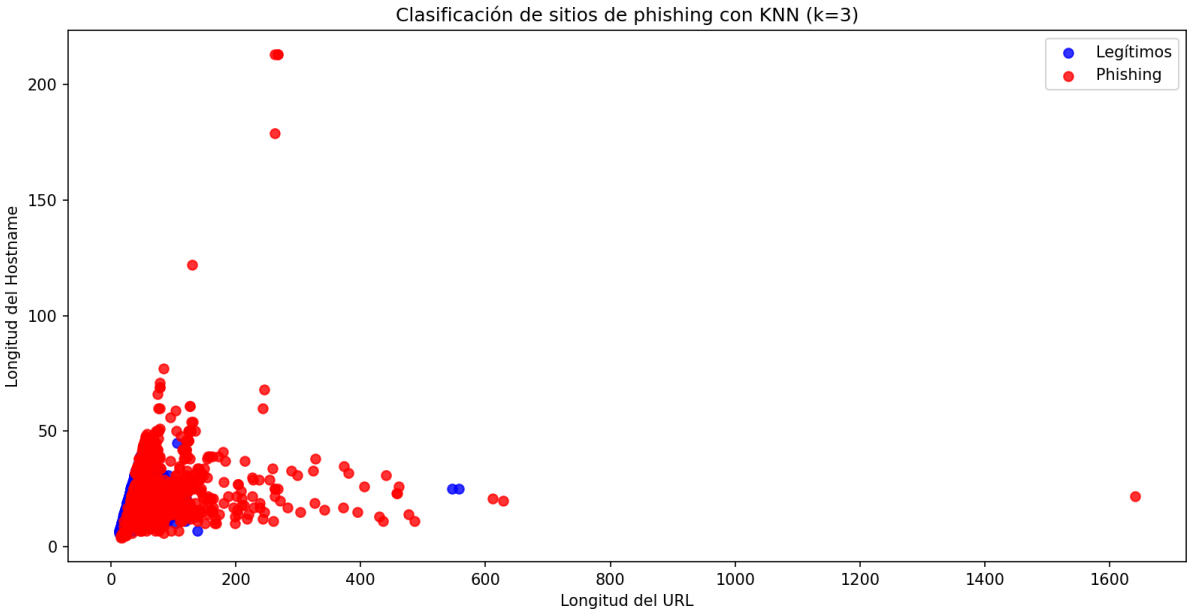
```
Matriz de confusión:
[[749 394]
 [384 759]]
Precisión del modelo: 0.66
```



Usando scikit learn




```
Matriz de confusión:
[[936 207]
 [563 580]]
Precisión del modelo (manual): 0.66
```



Reflexión de los resultados:

¿Cuál implementación fue mejor? ¿Por qué?

Cuál es mejor depende del contexto:

Si se prefiere minimizar falsos positivos (impacto en sitios legítimos): la implementación de scikit-learn es mejor porque clasifica más sitios legítimos correctamente (menos FP).

Si se prefiere minimizar falsos negativos (impacto en seguridad): la implementación desde cero es mejor porque detecta más sitios phishing correctamente (menos FN).

Consideración adicional:

La implementación con scikit-learn es más eficiente y fácil de usar para ajustar hiperparámetros o experimentar con otros valores de k o métricas.

La implementación desde cero muestra un comportamiento diferente, lo cual puede deberse a ligeras diferencias en la lógica (por ejemplo, resolución de empates entre vecinos o cálculos).

En este caso, si el objetivo principal es detectar phishing con mayor precisión (menos FN), la implementación desde cero sería más adecuada. Si deseas priorizar la clasificación correcta de sitios legítimos, entonces la implementación con scikit-learn es la mejor opción.