Representación plot de datasets, selección de características y entrenamiento de modelos

Enlace a github: github
Enlace al recurso utilizado: Kaggle

El objetivo de esta actividad es poner en práctica los conocimientos adquiridos para el preprocesamiento de datos, selección de características y entrenamiento de modelos.

Para ello es necesario seleccionar un Dataset que consideres oportuno, de clasificación o regresión, y distinto a los utilizados en clase.

Los puntos a desarrollar son los siguientes:

Sumario

1. (5%) Describir el origen y breve explicación del Dataset, así como de cada una de las características.

He podido experimentar con dos datasets:
- Uno grande y desbalanceado
- Uno pequeño y balanceado
El que mejores resultados me ha dado ha sido el primero, pero he puesto los dos en el github.
Ambos datasets provienen del resurso, cuyo link está al principio, junto al link de mi github.

Estos datasets ya fueron probados, limpiados y configurados específicamente para la tarea de detectar páginas de phishing, de modo que apenas fue necesario realizar una limpieza de datos.

Dataset Grande (dataset_full.csv):
- Número total de instancias: 88,647
- Número de páginas web legítimas (Etiquetado como 0 en la característica phishing): 58,000
- Número de páginas web phishing (Etiquetado como 1 en la característica phishing): 30,647
- Número de características: 112 (Incluyendo la característica phishing)

Dataset Pequeño (dataset_small.csv):
- Número total de instancias: 58,645
- Número de páginas web legítimas (Etiquetado como 0 en la característica phishing): 27,998
- Número de páginas web phishing (Etiquetado como 1 en la característica phishing): 30,647
- Número de características: 112 (Incluyendo la característica phishing)

Aquí especifico todas sus características:

Característica	Descripción
qty_dot_url	count (.) en la URL
qty_hyphen_url	count (-) en la URL
qty_underline_url	count (_) en URL
qty_slash_url	count (/) en la URL
qty_questionmark_url	count (?) en la URL
qty_equal_url	count (=) en URL
qty_at_url	recuento (@) en URL
qty_and_url	recuento (&) en URL
qty_exclamation_url	count (!) en la URL
qty_space_url	count ( ) en la URL
qty_tilde_url	count (~) en la URL
qty_comma_url	count (,) en la URL
qty_plus_url	recuento (+) en URL
qty_asterisk_url	count (*) en la URL
qty_hashtag_url	count (#) en la URL
qty_dollar_url	count ($) en la URL
qty_percent_url	recuento (%) en URL
qty_tld_url	longitud del dominio de nivel superior
length_url	Longitud de la URL
qty_dot_domain	count (.) en el dominio
qty_hyphen_domain	count (-) en el dominio
qty_underline_domain	count (_) en el dominio
qty_slash_domain	count (/) en dominio
qty_questionmark_domain	count (?) en el dominio
qty_equal_domain	count (=) en el dominio
qty_at_domain	Recuento (@) en dominio
qty_and_domain	Recuento (&) en dominio
qty_exclamation_domain	count (!) en el dominio
qty_space_domain	count ( ) en el dominio
qty_tilde_domain	count (~) en el dominio

Característica	Descripción
qty_comma_domain	count (,) en el dominio
qty_plus_domain	Recuento (+) en dominio
qty_asterisk_domain	count (*) en el dominio
qty_hashtag_domain	count (#) en el dominio
qty_dollar_domain	count ($) en el dominio
qty_percent_domain	Recuento (%) en dominio
qty_vowels_domain	Contar vocales en dominio
domain_length	Longitud del dominio
domain_in_ip	Dominio URL en formato de dirección IP
server_client_domain	domain contiene las palabras clave "servidor" o "cliente"
qty_dot_directory	count (.) en el directorio
qty_hyphen_directory	count (-) en el directorio
qty_underline_directory	count (_) en el directorio
qty_slash_directory	count (/) en el directorio
qty_questionmark_directory	count (?) en el directorio
qty_equal_directory	count (=) en el directorio
qty_at_directory	count (@) en el directorio
qty_and_directory	count (&) en el directorio
qty_exclamation_directory	count (!) en el directorio
qty_space_directory	count ( ) en el directorio
qty_tilde_directory	count (~) en el directorio
qty_comma_directory	count (,) en el directorio
qty_plus_directory	count (+) en el directorio
qty_asterisk_directory	count (*) en el directorio
qty_hashtag_directory	count (#) en el directorio
qty_dollar_directory	count ($) en el directorio
qty_percent_directory	recuento (%) en el directorio
directory_length	Longitud del directorio
qty_dot_file	count (.) en el archivo
qty_hyphen_file	count (-) en el archivo

Característica	Descripción
qty_underline_file	count (_) en el archivo
qty_slash_file	count (/) en el archivo
qty_questionmark_file	count (?) en el archivo
qty_equal_file	count (=) en el archivo
qty_at_file	count (@) en el archivo
qty_and_file	Recuento (&) en el archivo
qty_exclamation_file	count (!) en el archivo
qty_space_file	recuento ( ) en el archivo
qty_tilde_file	count (~) en el archivo
qty_comma_file	count (,) en el archivo
qty_plus_file	Recuento (+) en el archivo
qty_asterisk_file	recuento (*) en el archivo
qty_hashtag_file	count (#) en el archivo
qty_dollar_file	count ($) en el archivo
qty_percent_file	Recuento (%) en el archivo
file_length	Longitud del archivo
qty_dot_params	count (.) en parámetros
qty_hyphen_params	count (-) en parámetros
qty_underline_params	count (_) en parámetros
qty_slash_params	count (/) en parámetros
qty_questionmark_params	count (?) en parámetros
qty_equal_params	count (=) en parámetros
qty_at_params	count (@) en parámetros
qty_and_params	count (&) en parámetros
qty_exclamation_params	count (!) en parámetros
qty_space_params	count ( ) en parámetros
qty_tilde_params	count (~) en parámetros
qty_comma_params	count (,) en parámetros
qty_plus_params	Recuento (+) en parámetros
qty_asterisk_params	count (*) en parámetros

Característica	Descripción
qty_hashtag_params	count (#) en parámetros
qty_dollar_params	count ($) en parámetros
qty_percent_params	Recuento (%) en parámetros
params_length	Parámetros Longitud
tld_present_params	Presencia de TLD en los argumentos
qty_params	Número de parámetros
email_in_url	correo electrónico presente en la URL
time_response	Tiempo de búsqueda (respuesta) Dominio (búsqueda)
domain_spf	el dominio tiene SPF
asn_ip	Número de AS (o ASN)
time_domain_activation	Tiempo (en días) de la activación del dominio
time_domain_expiration	Tiempo (en días) de vencimiento del dominio
qty_ip_resolved	número de direcciones IP resueltas
qty_nameservers	número de servidores de nombres resueltos (NameServers - NS)
qty_mx_servers	número de servidores MX
ttl_hostname	Valor de tiempo de vida (TTL) asociado con el nombre de host
tls_ssl_certificate	Certificado TLS / SSL válido
qty_redirects	Número de redireccionamientos
url_google_index	comprobar si la URL está indexada en Google
domain_google_index	comprobar si el dominio está indexado en Google
url_shortened	comprobar si la URL está acortada
phishing	es un sitio web de phishing

Todas las características aportan información relevante. La mayoría de características simplemente es el número de veces que podemos encontrar un carácter en particular dentro de la url (esos caracteres se utilizan como operadores, así que aportan mucha información). Además, entre algunas de estas características, podemos ver que hay otras especialmente interesantes, como:

- url_google_index (Comprueba si google indexa la página)
- qty_redirects (Comprueba si redirecciona a otra página, y de hacerlo cuántas veces lo hace)
- tls_ssl_certificate (Comprueba si tiene el certificado tls ssl)

2. (5%) Procesamiento de datos en el dataset: ajustes en características con datos no informados, conversión de variables categóricas, etc.

Como se mencionó anteriormente, no fue necesario realizar ningún cambio dentro del dataset, ya que ya venía preparado para utilizarse en este tipo de tareas. Sin embargo, si hubiera tenido que realizar algún cambio de balance, lo habría hecho con esta herramienta: Phishing Datasets Web App, que me permitía customizar el dataset que obtengo, por si lo hubiera necesitado para otra cosa.

3. Utilizar las siguientes herramientas explicadas en clase para la selección de características:

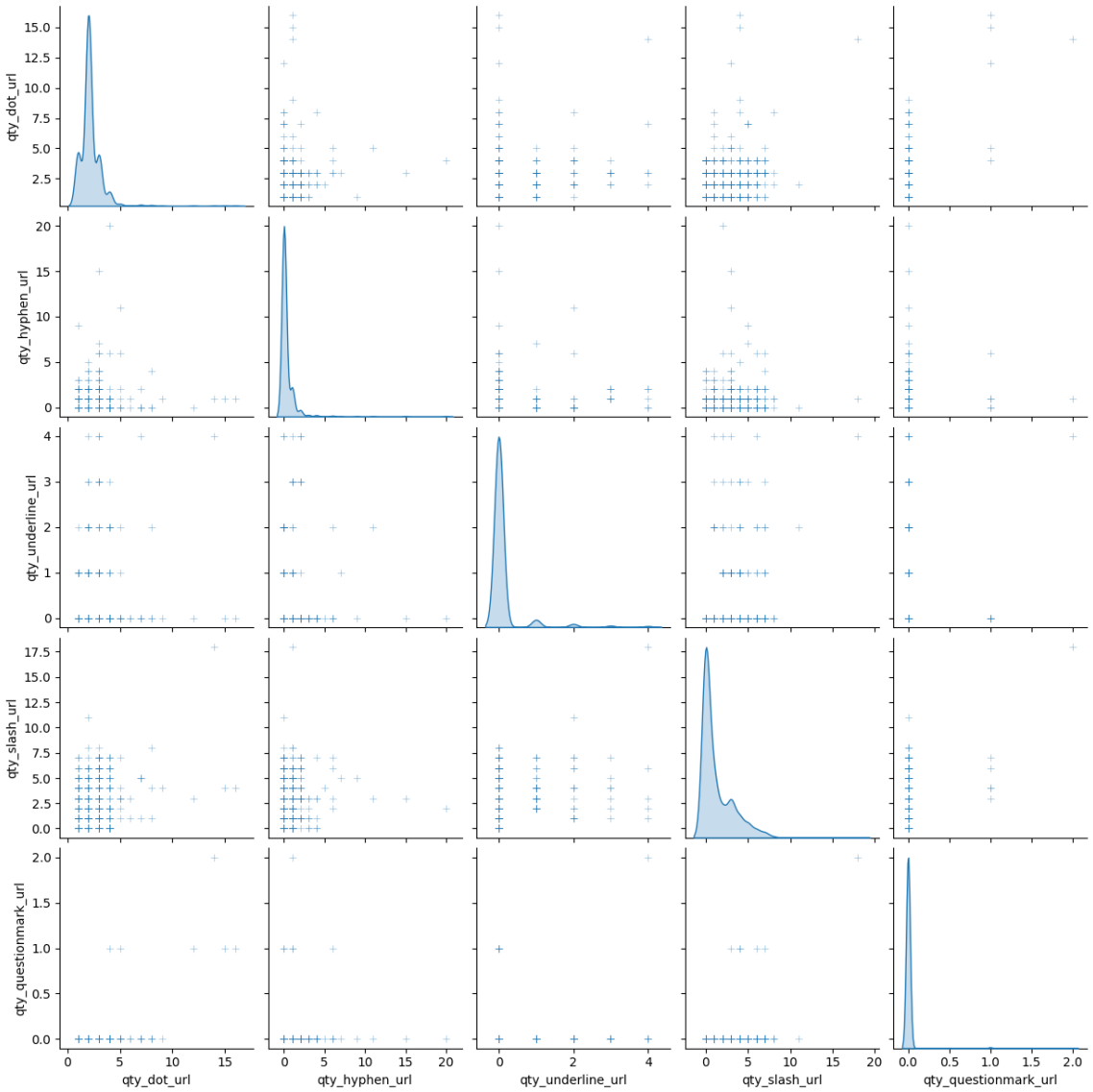
3.1. (10%) Matriz de gráficos de correlación.

Para mostrar la matriz de gráficos de correlación, usé el siguiente código:

```
import seaborn as sns
import matplotlib.pyplot as plt
def showCorrelationMatrix(dataFrame):
  # Tomar una muestra del dataset (por ejemplo, 1000 filas)
  df_sample = dataFrame.sample(n=1000, random_state=1)
  # Selecciono 5 variables (ya que si las selecciono todas, tardará demasiado)
  selected_columns = df_sample.columns[:5]
  df_sample = df_sample[selected_columns]
  # Crear la matriz de gráficos de correlación
  sns.pairplot(df_sample, diag_kind='kde', markers='+', plot_kws={'alpha': 0.5})
  # Mostrar la gráfica
  plt.show()
correlation = showCorrelationMatrix(df)
correlation
```

Aquí los resultados:

3.2.

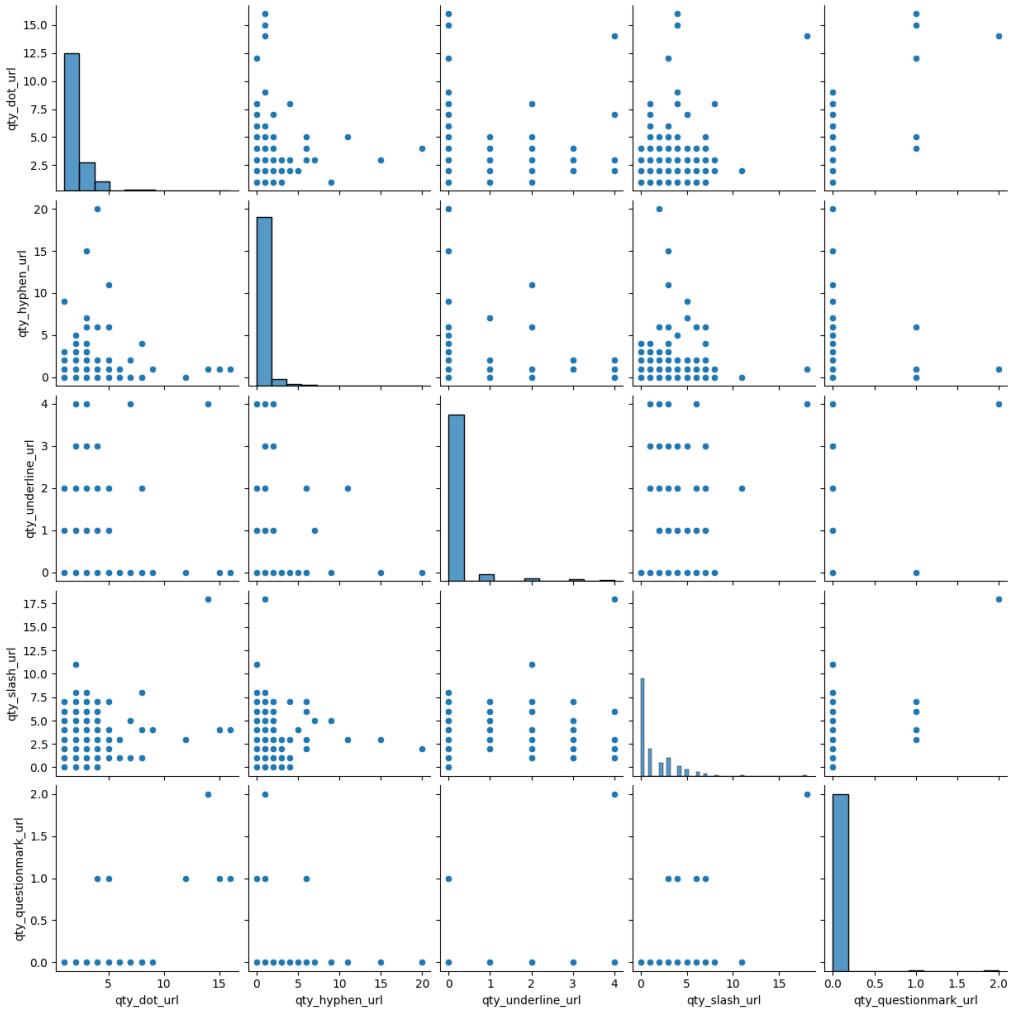


(10%) Matriz de gráficos de dispersión.

Para mostrar la matriz de gráficos de dispersión, utilicé este código:

```
def showDispersion(dataFrame):
  # Seleccionar una muestra del conjunto de datos
  df_sample = dataFrame.sample(n=1000, random_state=1)

  # Selecciono 5 variables (ya que si las selecciono todas, tardará demasiado)
  selected_columns = df_sample.columns[:5]
  df_sample = df_sample[selected_columns]

  # Crear la matriz de gráficos de dispersión
  sns.pairplot(df_sample)

  # Mostrar la matriz
  plt.show()

dispersion = showDispersion(df)
dispersion
```

Aquí los resultados:



3.3. (10%) SelectKBest.

SelectKBest nos permite seleccionar las mejores características del dataset para no tener que gastar demasiado tiempo en el entrenamiento (ya que tendremos que tener menos características en cuenta).
Este es el código utilizado:

```
from sklearn.feature_selection import SelectKBest, f_classif
selector = SelectKBest(score_func=f_classif, k=50)  # Selecciona las 50 mejores características
Xtrain_new = selector.fit_transform(Xtrain, ytrain)
Xtest_new = selector.transform(Xtest)

model = GaussianNB()
model.fit(Xtrain_new, ytrain)
y_model = model.predict(Xtest_new)

# Evaluar el modelo
accuracy = accuracy_score(ytest, y_model)
print("Accuracy:", accuracy)
```

Si entrenamos al modelo con las 50 características que ha seleccionado, obtenemos una precisión del 87%:

```
from sklearn.feature_selection import SelectKBest, f_classif
selector = SelectKBest(score_func=f_classif, k=50)  # Selecci
Xtrain_new = selector.fit_transform(Xtrain, ytrain)
Xtest_new = selector.transform(Xtest)

model = GaussianNB()
model.fit(Xtrain_new, ytrain)
y_model = model.predict(Xtest_new)

# Evaluar el modelo
accuracy = accuracy_score(ytest, y_model)
print("Accuracy:", accuracy)
```

✓ 0.2s

```
Accuracy: 0.8681598785183579
```

Sin embargo, al no utilizar SelectKBest, obtenemos una precisión del 91%:

```
# 5 evaluación
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(ytest, y_model) # Precisión del modelo
print(accuracy)
```

[22] ✓ 0.0s

```
0.9105157546504691
```

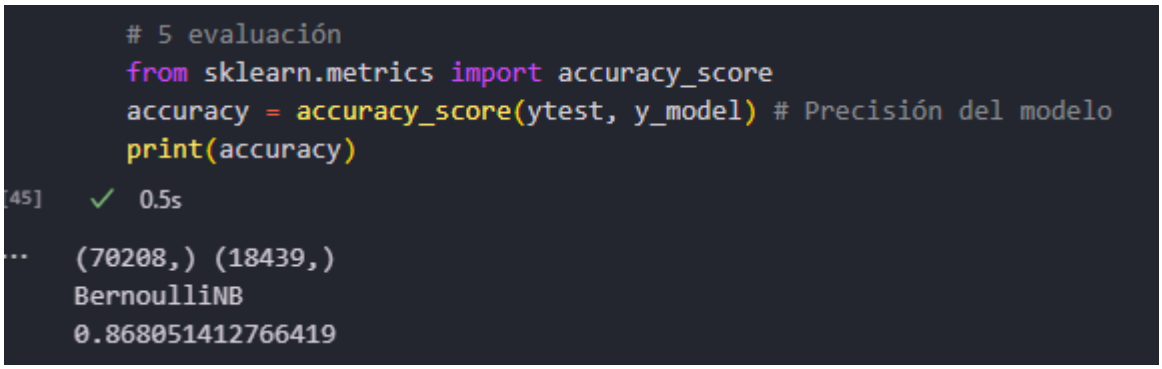
4. (5%) Una pequeña reflexión sobre la elección de las características elegidas.

Desconozco por qué entrenar el modelo con las características recomendadas baja tanto la precisión. Me parece raro, ya que también he probado a usar todas las características junto con SelectKBest, pero sigo sin obtener la misma precisión.

5. Con las librerías para NaiveBayes vistas en clase, entrenar el modelo que consideres más adecuado.

Entre los modelos de Naive Bayes, encontramos el BernoulliNB, que se especializa en clasificaciones binarias.

Parece ideal para clasificar URLs en phishing/no phishing, así que lo pruebo:



Como se puede ver, obtengo una precisión decente, pero aún inferior a la obtenida con el NB Gaussiano (91%). De modo que el más adecuado para seguir, a juzgar por los resultados, es el NB Gaussiano. Y es por eso que será el modelo que utilizaré.

5.1. (10%) Sin utilizar Cross Validation.

Para mostrar los resultados de la comparación de usar y no usar cross-validation, he usado el siguiente código:

```
from sklearn.model_selection import cross_val_score
# 1. Elegimos clasificador
from sklearn.naive_bayes import GaussianNB
# 2. Instanciamos el modelo
model = GaussianNB()
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
print(model.__class__.__name__)
for column in Xtrain.columns:
    Xtrain[column] = label_encoder.fit_transform(Xtrain[column])
    # Apply label encoding to each column in Xtrain
for column in Xtest.columns:
    Xtest[column] = label_encoder.fit_transform(Xtest[column])
    # Now you can fit your model
model.fit(Xtrain, ytrain)
# Evaluar el modelo sin usar cross-validation
y_model = model.predict(Xtest)
```

```
accuracy = accuracy_score(ytest, y_model)

print("Accuracy sin cross-validation:", accuracy)

# Aplicar validación cruzada

scores = cross_val_score(model, Xtrain, ytrain, cv=5)  # cv=5 para 5-fold cross-validation

print("Cross-validation scores:", scores)

print("Mean cross-validation score:", scores.mean())
```

- Aquí temenos el resultado sin cross-validation:

```
Accuracy sin cross-validation: 0.9105157546504691
```

5.2. (15%) Utilizando Cross Validation.

- Aquí tenemos el resultado con cross-validation:

```
Cross-validation scores: [0.91026919 0.90998433 0.91119499 0.91161598 0.90947938]
Mean cross-validation score: 0.9105087750372253
```

6. (5%) Obtener una conclusión sobre los resultados obtenidos en la predicción y evaluación al utilizar o no Cross Validation.

- Sin Cross-Validation: La precisión obtenida puede ser específica de la partición de datos utilizada y no necesariamente representativa del rendimiento general del modelo.

- Con Cross-Validation: Los puntajes de validación cruzada y su promedio proporcionan una evaluación más robusta y confiable del rendimiento del modelo, ya que consideran múltiples particiones de los datos.