Clasificacion de fallas electricas

El objetivo de este trabajo es poder predecir si hubo una falla electrica a partir de los valores de corriente y voltage registrados en cada fase, se analizaran los siguientes metodos de clasificacion para poder ver cual es la mas adecuada para este problema:

- Regresion logistica
- KNN (K Vecinos Mas Cercanos por sus siglas en ingles)
- SVM (Maquina de Soporte Vectorial por sus siglas en ingles)
- Naive Bayes
- Arbol de decision
- Random Forest

Este trabajo se baso en el paper Deteccion y clasificacion de fallas (electricas) utilizando Machine Learning [1] realizado por Suresh Kumar, Abhinav Varma, Devika Rani, Nishanth del Chaitanya Bharathi Institute of Technology.

Los investigadores recrearon un sistema electrico utilizando MATLAB para poder registrar los datos, realizar estas simulaciones es muy bueno para poder aislar mejor el sistema a analizar, sin que este sea afectados por variables externas.

A diferencia del paper original donde se intento predecir individualmente cada tipo de falla, este trabajo solo se limitara a predecir si hubo o no una falla electrica.

Los datos utilizados pueden encontrarse en kaggle - electrical fault detection and classification [2].

Dependencias

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from matplotlib.colors import ListedColormap
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import (
    confusion_matrix, precision_score, accuracy_score,
    recall_score, f1_score, roc_auc_score, classification_report,
    )
import random # para tener colores random
import warnings
warnings.filterwarnings("ignore")
```

Revisando el dataset

El dataset contiene las siguientes columnas:

- `G` : puede ser 0 o 1, donde 1 indica que hay una falla en la conexion a tierra.
- `C` : puede ser 0 o 1, donde 1 indica que hay una falla en la fase C.
- `B` : puede ser 0 o 1, donde 1 indica que hay una falla en la fase B.
- `A` : puede ser 0 o 1, donde 1 indica que hay una falla en la fase A.
- `Ia` : corriente en la fase A.
- `Ib` : corriente en la fase B.
- `Ic` : corriente en la fase C.
- `Va` : voltaje en la fase A.
- `Vb` : voltaje en la fase B.
- `Vc` : voltaje en la fase C.

In [ ]:
```
dataset = pd.read_csv('./content/fallas-electricas.csv')
dataset.head()
```

Out[ ]:

	G	C	B	A	Ia	Ib	Ic	Va	Vb	Vc
0	1	0	0	1	-151.291812	-9.677452	85.800162	0.400750	-0.132935	-0.267815
1	1	0	0	1	-336.186183	-76.283262	18.328897	0.312732	-0.123633	-0.189099
2	1	0	0	1	-502.891583	-174.648023	-80.924663	0.265728	-0.114301	-0.151428
3	1	0	0	1	-593.941905	-217.703359	-124.891924	0.235511	-0.104940	-0.130570
4	1	0	0	1	-643.663617	-224.159427	-132.282815	0.209537	-0.095554	-0.113983

In [ ]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7861 entries, 0 to 7860
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   G       7861 non-null   int64  
 1   C       7861 non-null   int64  
 2   B       7861 non-null   int64  
 3   A       7861 non-null   int64  
 4   Ia      7861 non-null   float64
 5   Ib      7861 non-null   float64
 6   Ic      7861 non-null   float64
 7   Va      7861 non-null   float64
 8   Vb      7861 non-null   float64
 9   Vc      7861 non-null   float64
dtypes: float64(6), int64(4)
memory usage: 614.3 KB
```

Primer procesado y Analisis exploratorio

Es indiferente si el fallo se produjo entre las fases AB o si ocurrio en la fases BC, por lo cual se agregara una nueva columna con las siguientes categorias para clasificar la falla

segun la cantidad de fases y si fallo a tierra.

- `sin falla` : no hubo ninguna falla en el sistema.
- `falla LG` : falla entre una fase y tierra.
- `falla LL` : falla entre 2 fases.
- `falla LLG` : falla entre 2 fases y tierra
- `falla LLL` : falla entre 3 fases.
- `falla LLLG` : falla entre 3 fases y tierra.

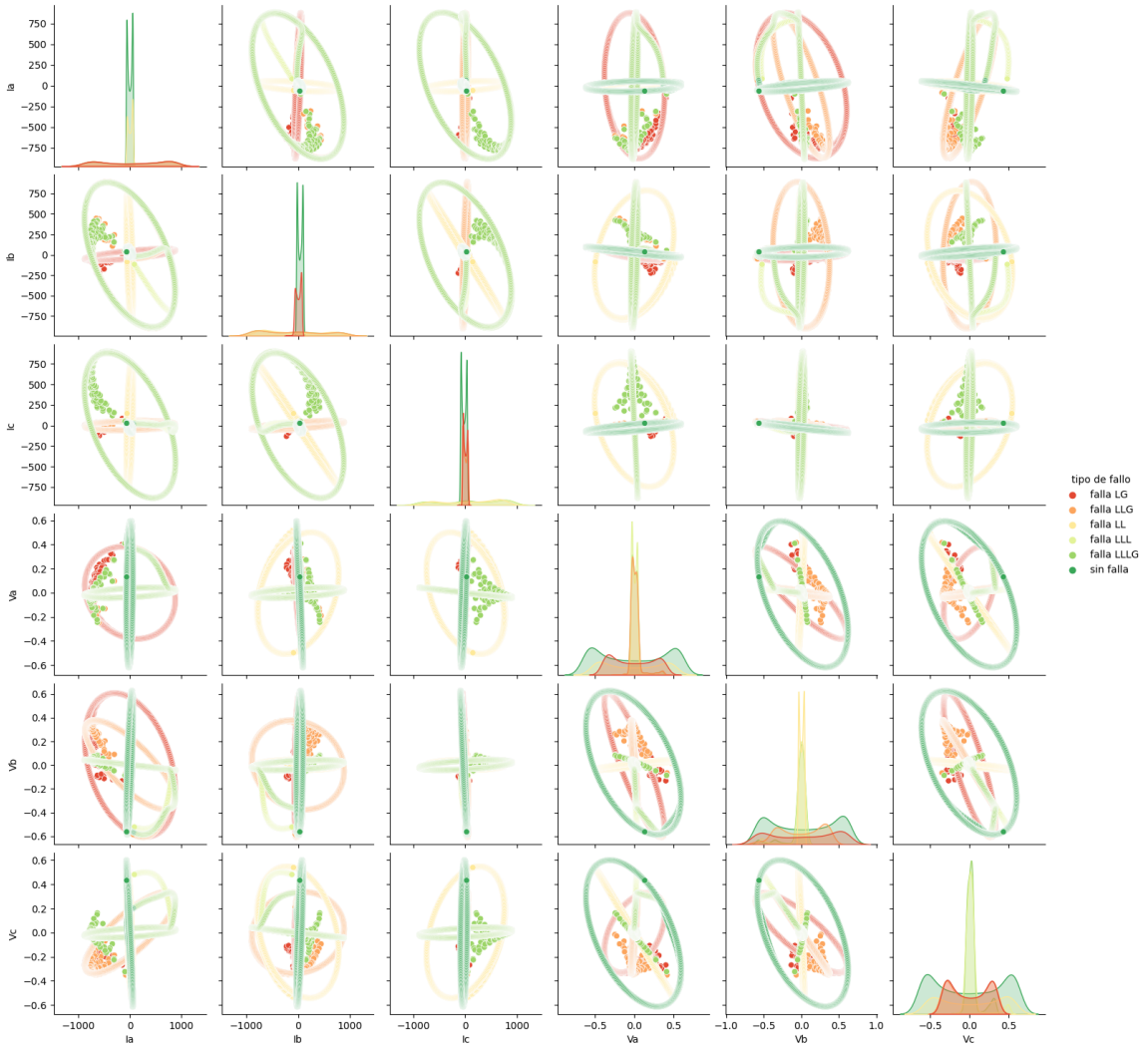
In [ ]:
```
def clasificar_falla(x):
    resultado = ""
    for col in ['C', 'B', 'A']:
        if x[col]:
            resultado += 'L'
    if x['G']:
        resultado += 'G'
    return 'falla ' + resultado if resultado else 'sin falla'
#
dataset['tipo de fallo'] = dataset.apply(clasificar_falla, axis=1)
dataset.sample(5)
```

Out[ ]:

	G	C	B	A	Ia	Ib	Ic	Va	Vb	Vc	tipo de fal
7442	0	0	0	0	-70.418950	53.194970	14.238906	-0.038754	-0.492640	0.531394	s fal
3917	0	1	1	1	865.304206	-273.947673	-589.298030	0.026948	-0.041739	0.014791	fal LL
7707	0	0	0	0	15.675897	69.912520	-88.498430	-0.514771	0.562541	-0.047769	s fal
6493	0	0	0	0	27.017330	61.892588	-92.180038	-0.430328	0.609660	-0.179332	s fal
4203	0	1	1	1	-10.055085	-759.261996	771.319777	-0.037776	0.002285	0.035491	fal LL

Procedamos a realizar un grafico de correlaciones para ver que relacion hay entre las variables de corriente y voltage para cada fase

In [ ]:
```
# Mas estilos se EDA se pueden encontrar en
# https://nlpfy.com/2018/07/21/exploratory-data-analysis-and-knn-classifi
sns.pairplot(dataset, hue='tipo de fallo', vars=['Ia', 'Ib', 'Ic', 'Va',
plt.show()
```



Podemos ver que cada `tipo de fallo` y `sin fallo` siguen un patron especifico, cercano a circulos o eclipses, esto nos indica dos cosas:

1. Podemos realizar una clasificacion ya que se observa correlacion entre las variables.
2. No podemos separar linealmente los datos por lo tanto el metodo de Regresion Logistica no sera el mas adecuado para este problema, aun asi vamos a utilizarlo para corroborarlo.

Tambien procederemos a crear una nueva columna `fallo` con los valores 0 para los `tipo de fallo` que sean `sin falla` y 1 para el resto. Esta columna es la principal que queremos predecir en este trabajo.

In [ ]:
```
dataset['fallo'] = dataset['tipo de fallo'].apply(lambda x: 0 if x == 'si
```

Procedemos a observar cuantos registros tenemos para cada `tipo de fallo` y si es necesario balancear el dataset.

In [ ]:
```
dataset['tipo de fallo'].value_counts()
```

Out[ ]:
```
sin falla     2365
falla LLG     1134
falla LLLG    1133
falla LG      1129
falla LLL     1096
falla LL      1004
Name: tipo de fallo, dtype: int64
```

Se puede observar que hay proximadamente unos 5400 registros para los fallos y solo 2365 para sin fallo, por lo cual procederemos a balancear el dataset seleccionando aleatoriamente registros de fallos hasta tener una cantidad equivalente a los registros de sin fallo.

La idea es tener una misma cantidad de registros para los tipos `sin fallo` y el resto, pero manteniendo registros de todos los tipos de fallos y asi evitar un sesgo en los modelos, por ejemplo si en el conjunto de testeo tenemos 900 fallos y 100 sin fallo, nos podria ocurrir de que el modelo nos clasifique a todos los datos como fallo y aun asi tener un 90% de precision, lo cual esta muy mal.

In [ ]:
```
cantidad_fallos = len(dataset[dataset["fallo"] == 1])
cantidad_sin_fallos = len(dataset[dataset["fallo"] == 0])
cantidad_a_borrar = abs(cantidad_sin_fallos - cantidad_fallos)
dataset = dataset.drop(dataset[dataset['fallo'] == 1].sample(cantidad_a_b
```

Veamos el resultado del balanceado:

In [ ]:
```
dataset['tipo de fallo'].value_counts()
```

Out[ ]:
```
sin falla     2365
falla LLLG     514
falla LLG      499
falla LG       461
falla LLL      460
falla LL       431
Name: tipo de fallo, dtype: int64
```

Ahora si tenemos una cantidad equivalente de `sin fallo` y de fallos, sin excluir ningun tipo de fallo.

Definiendo las variables dependientes e independientes

Las variables independientes seran los voltages y corrientes en cada fase, y como variable independiente tendremos la columna `fallo`.

In [ ]:
```
X = dataset.iloc[:, 4: 10].values
y = dataset.iloc[:, 11].values
X, y
```

Out[ ]:
```
(array([[-6.43663617e+02, -2.24159427e+02, -1.32282815e+02,
          2.09536880e-01, -9.55537510e-02, -1.13983129e-01],
        [-5.57391809e+02, -1.19468643e+02, -2.95294498e+01,
          2.10003736e-01, -7.67124530e-02, -1.33291283e-01],
        [-3.85668729e+02, -9.79898386e+01, -1.00768239e+01,
          3.34648677e-01, -5.77954490e-02, -2.76853228e-01],
        ...,
        [-6.54466976e+01,  3.64720551e+01,  2.61065537e+01,
          1.13106580e-01, -5.58210927e-01,  4.45104348e-01],
        [-6.50296327e+01,  3.54770884e+01,  2.66847311e+01,
          1.22404174e-01, -5.61094199e-01,  4.38690025e-01],
        [-6.45984013e+01,  3.44807988e+01,  2.72500649e+01,
          1.31668579e-01, -5.63834635e-01,  4.32166056e-01]]),
 array([1, 1, 1, ..., 0, 0, 0]))
```

Separando conjunto de entrenamiento y de testeo

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2
```

Estandarizado de los datos

In [ ]:
```
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Clasificacion con Regresion Logistica

Este metodo de clasificacion basicamente intenta predecir con reglas de la probabilidad que tan probable es que haya un fallo o no, utilizando una funcion sigmoide.

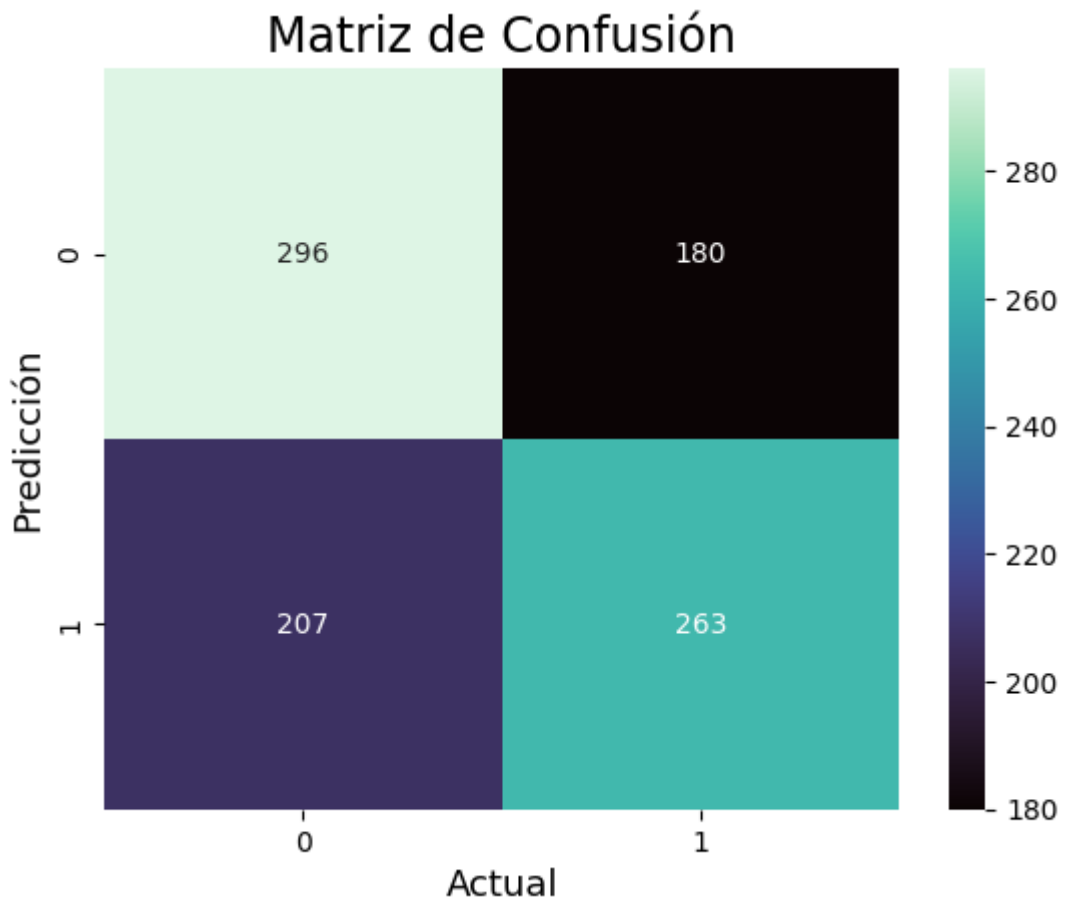
In [ ]:
```
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Prediccion': y_pred}).sample(10)
```

Out[ ]:

	Actual	Prediccion
175	0	0
834	1	1
648	0	1
5	0	1
349	1	1
697	0	0
942	1	0
539	1	1
244	1	0
624	1	0

Se observa en el muestreo que algunos valores se predijeron bien y que le erro en otros, veamos la matriz de confusion para ver cuantos aciertos y errores tuvo el modelo.

In [ ]:
```
def plot_confusion_matrix(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    palletes = [
        'Blues', 'Reds', 'mako', 'crest'
        ]
    sns.heatmap(
        cm,
        annot=True,
        fmt='g',
        cmap=random.choice(palletes),
        )
    plt.ylabel('Predicción', fontsize=13)
    plt.xlabel('Actual', fontsize=13)
    plt.title('Matriz de Confusión', fontsize=17)
    plt.show()
#
plot_confusion_matrix(y_test, y_pred)
```



Podemos observar que obtuvimos 277 aciertos para clasificar el tipo `sin fallos` y 254 aciertos para los fallos, lo cual comparado con los falsos valores (de 199 y 216 respectivamente), es una prediccion media mala.

Veamos exactamente las metricas que hemos obtenido con el modelo

In [ ]:
```
def plot_metricas(y_test, y_pred):
    print(f'Precisión del modelo: {precision_score(y_test, y_pred):.2f}')
```

```
    print(f'Exactitud del modelo: {accuracy_score(y_test, y_pred):.2f}')
    print(f'Sensibilidad del modelo: {recall_score(y_test, y_pred):.2f}')
    print(f'Puntaje F1 del modelo: {f1_score(y_test, y_pred):.2f}')
    print(f'Curva ROC - AUC del modelo: {roc_auc_score(y_test, y_pred):.2
#
plot_metricas(y_test, y_pred)
```

```
Precisión del modelo: 0.59
Exactitud del modelo: 0.59
Sensibilidad del modelo: 0.56
Puntaje F1 del modelo: 0.58
Curva ROC - AUC del modelo: 0.59
```

vemos una precision del 0.57, lo cual es medio malo comparandolo con el 0.50 que se obtendria si se predijera al azar.

Clasificacion con KNN (K Vecinos Mas Cercanos)

Basicamente este metodo de clasificacion intenta predecir si fallo o no, basandose en que tan cercanos son los datos en un espacio multidimensional.

In [ ]:
```
classifier = KNeighborsClassifier(n_neighbors = 5, metric = "minkowski",
classifier.fit(X_train, y_train)
```

Out[ ]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Prodecemos a realizar la prediccion

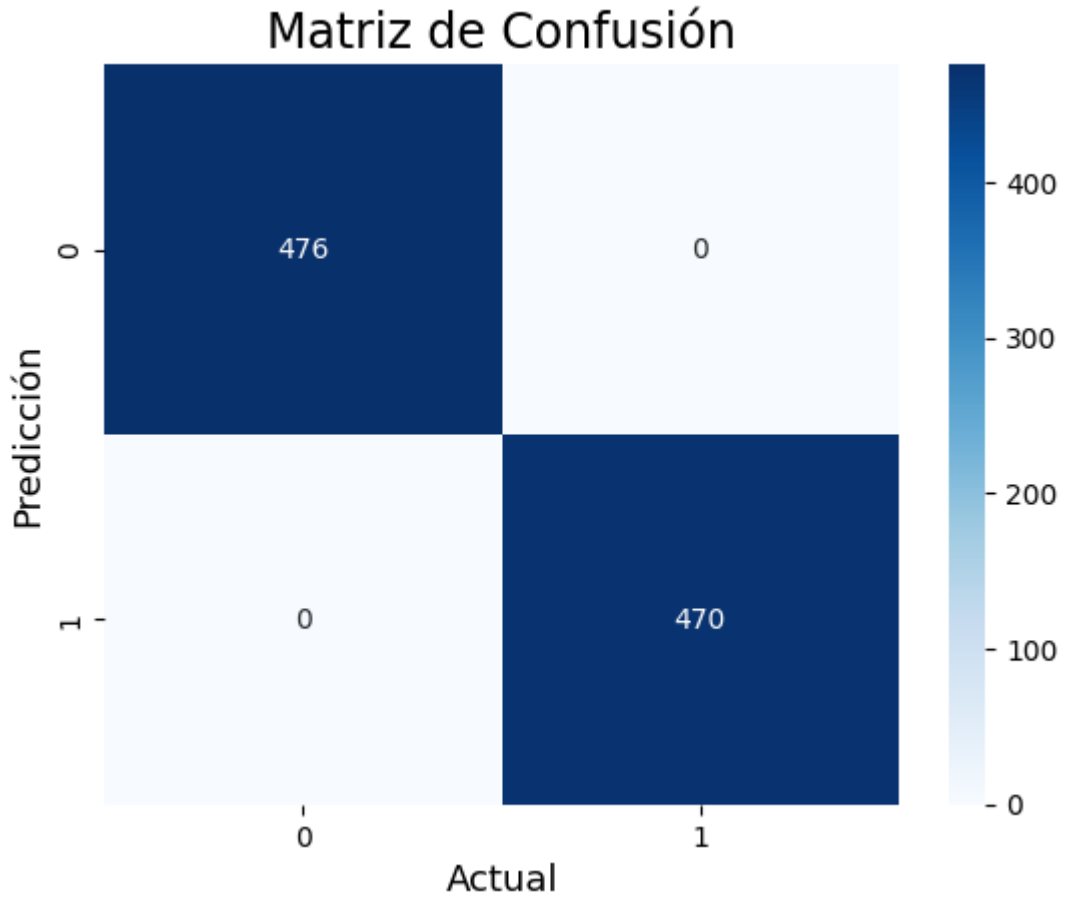
In [ ]:
```
y_pred = classifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Predicción': y_pred}).sample(10)
```

Out[ ]:

	Actual	Predicción
727	0	0
840	1	1
133	1	1
532	1	1
464	1	1
100	0	0
701	0	0
45	1	1
662	0	0
392	1	1

Vemos en el muestro que el modelo predijo bien algunos valores, mucho mejor que en la clasificacion anterior con Regresion Lineal, veamos la matriz de confusion para ver el total de aciertos y errores.

In [ ]:
```
plot_confusion_matrix(y_test, y_pred)
```



Podemos ver en la matriz de confusion que se obtuvieron 476 aciertos para predecir `sin fallo`, 470 aciertos para predecir `falla LLG` y 1 falsos positivos o negativos, lo cual hace que tenga una muy buena precision.

Veamos las metricas obtenidas con KNN:

In [ ]:
```
plot_metricas(y_test, y_pred)
```

```
Precisión del modelo: 1.00
Exactitud del modelo: 1.00
Sensibilidad del modelo: 1.00
Puntaje F1 del modelo: 1.00
Curva ROC - AUC del modelo: 1.00
```

Vemos que se obtuvo un 100% de precision, lo cual puede decirnos dos cosas, que el modelo se adapta muy bien para clasificar los errores

Clasificacion con Maquina de Soporte Vectorial

Basicamente este metodo de clasificacion intenta predecir una categoria, separando los datos con hiperplanos, el negativo y positivo que son los mas cercanos a los datos y el hiperplano optimo que es el intermedio entre ambos.

Utilizaremos el Kernel RBF ya que debemos clasificar utilizando 6 dimensiones (una dimension para cada voltage y corriente en cada fase) y puede que los conjuntos no sea

linealmente separable.

In [ ]:
```
classifier = SVC(kernel = "rbf", random_state = 0)
classifier.fit(X_train, y_train)
```

Out[ ]:
```
SVC(random_state=0)
```

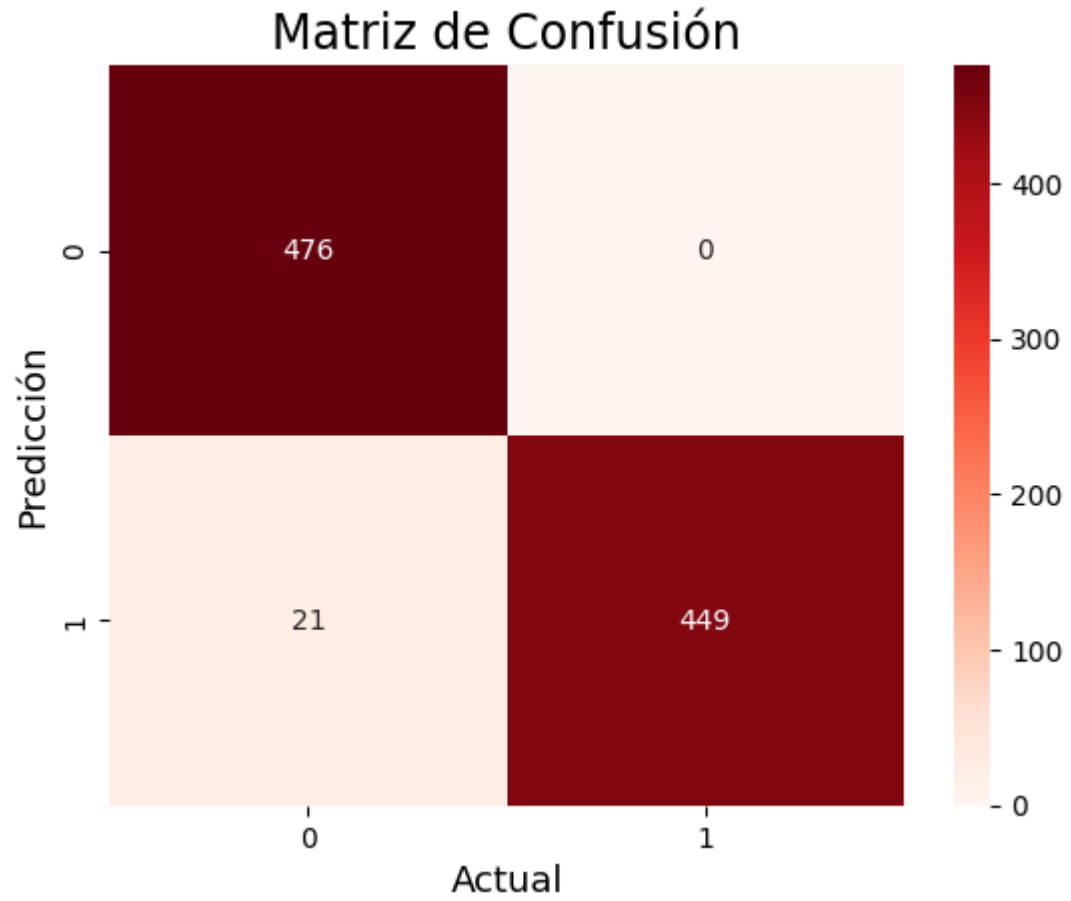
In [ ]:
```
y_pred = classifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Predicción': y_pred}).sample(10)
```

Out[ ]:

	Actual	Predicción
851	0	0
835	1	1
325	0	0
306	1	1
345	1	1
0	0	0
644	0	0
596	1	1
641	0	0
322	0	0

Nuevamente podemos ver una muy buena similitud entre los valores Actuales y los de Prediccion, procedamos a realizar una matriz de confusion para ver cuantos aciertos obtuvimos

In [ ]:
```
plot_confusion_matrix(y_test, y_pred)
```



Podemos observar que obtuvimos una muy buena cantidad de aciertos, comparando los verdaderos positivos y negativos con los falsos positivos y negativos, veamos las metricas que el modelo obtuvo:

In [ ]:
```
plot_metricas(y_test, y_pred)
```

```
Precisión del modelo: 1.00
Exactitud del modelo: 0.98
Sensibilidad del modelo: 0.96
Puntaje F1 del modelo: 0.98
Curva ROC - AUC del modelo: 0.98
```

Nuevamente una precision del 100% como en KNN

Clasificacion con Naive Beyes

Basicamente este metodo de clasificacion utiliza el Teorema de Bayes, el cual intenta predecir cual es la probabilidad de que un evento ocurra comparando las caracteristicas de este evento vs la probabilidad de que ocurra un evento similar.

In [ ]:
```
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Out[ ]:
```
▾ GaussianNB
GaussianNB()
```

Muestreo de algunos valores de prediccion vs actuales

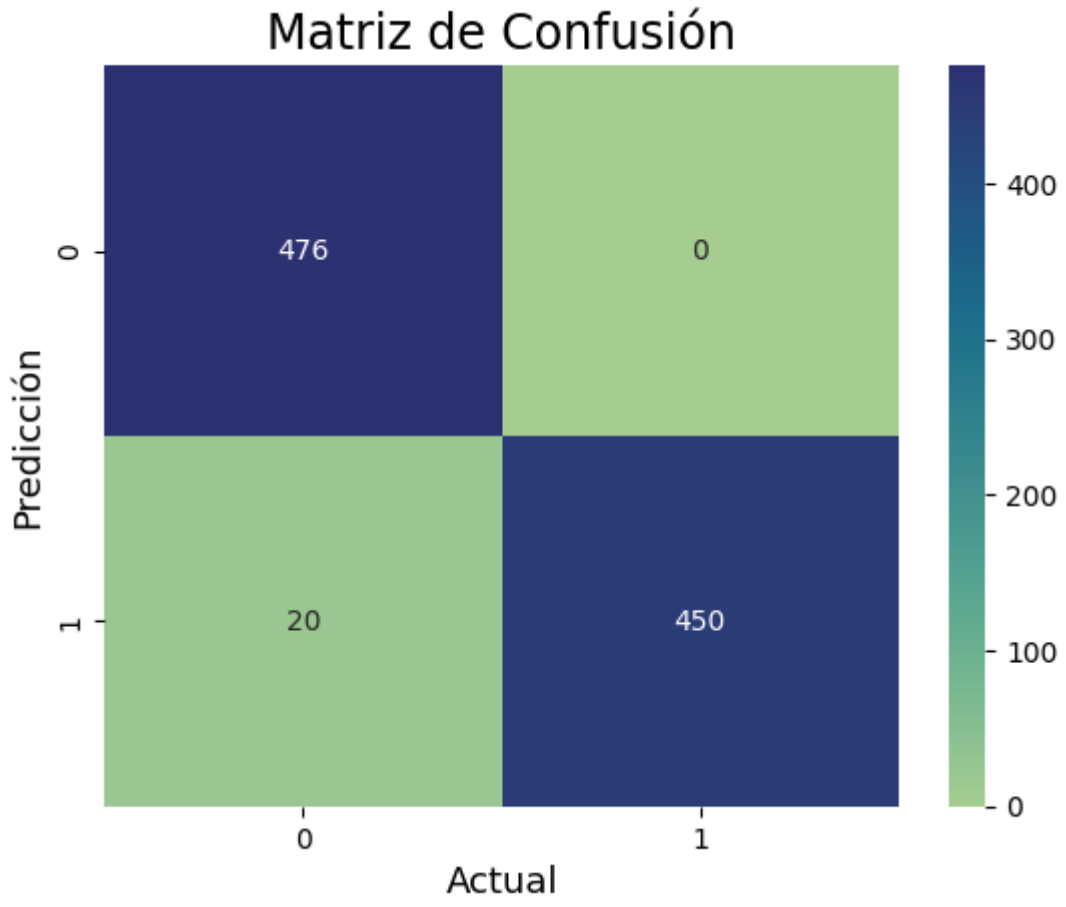
In [ ]:
```
y_pred = classifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Predicción': y_pred}).sample(10)
```

Out[ ]:

	Actual	Predicción
687	0	0
593	1	1
882	0	0
420	0	0
706	0	0
722	1	1
500	0	0
642	1	1
362	0	0
8	1	1

Podemos observar que hay una alta similitud entre algunos valores actuales y los predichos, veamos la matriz de confusion para tener un panorama mas claro

In [ ]:
```
plot_confusion_matrix(y_test, y_pred)
```



Podemos ver que en una primera impresion, que el modelo predijo muy bien los valores, veamos las metricas que obtuvo

In [ ]:
```
plot_metricas(y_test, y_pred)
```

```
Precisión del modelo: 1.00
Exactitud del modelo: 0.98
Sensibilidad del modelo: 0.96
Puntaje F1 del modelo: 0.98
Curva ROC - AUC del modelo: 0.98
```

Nuevamente obtuvimos una precision del 100%

Clasificacion con Arboles de Decision

Este metodo de clasificacion funciona subdividiendo los datos en ramas, siguiendo como criterio el indice de Gini para realizar los cortes en las variables independientes.

In [ ]:
```
classifier = DecisionTreeClassifier(criterion = "entropy", random_state =
classifier.fit(X_train, y_train)
```

Out[ ]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

Veamos algunos valores de prediccion vs actuales

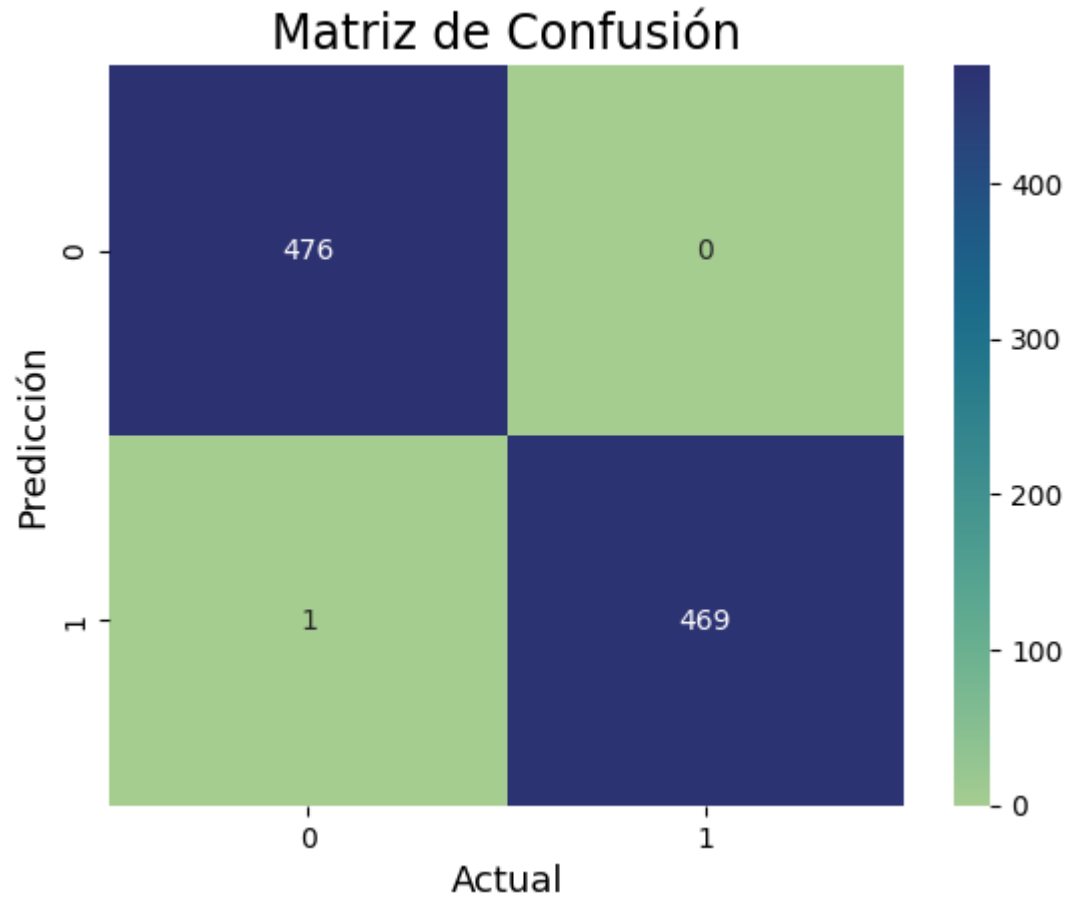
In [ ]:
```
y_pred = classifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Predicción': y_pred}).sample(10)
```

Out[ ]:

	Actual	Predicción
78	0	0
185	0	0
394	1	1
426	0	0
532	1	1
131	1	1
397	0	0
372	1	1
881	0	0
353	1	1

Podemos ver una alta similitud entre algunos valores actuales y predichos, realizemos una matriz de confusion para ver cuantos aciertos tuvimos en total

In [ ]:
```
plot_confusion_matrix(y_test, y_pred)
```



Podemos ver que obtuvimos muchos aciertos en la prediccion aunque algunos falsos negativos, veamos las metricas que obtuvimos

In [ ]:
```
plot_metricas(y_test, y_pred)
```

```
Precisión del modelo: 1.00
Exactitud del modelo: 1.00
Sensibilidad del modelo: 1.00
Puntaje F1 del modelo: 1.00
Curva ROC - AUC del modelo: 1.00
```

Podemos ver que practicamente tenemos una precision del 100% nuevamente

Clasificacion con Random Forest

Es la version mejorada de los Arboles de Decision ya que combina multiples de estos en un mismo resultado, basicamente se seleccionan aleatoriamente los datos y se agrupan en diferentes subconjuntos, con cada uno de estos subconjuntos se construye un arbol de decision y para realizar la prediccion, se agrupan los resultados de estos arboles y se selecciona la clasificacion mas concurrida.

In [ ]:
```
classifier = RandomForestClassifier(n_estimators = 10, criterion = "entro
classifier.fit(X_train, y_train)
```

Out[ ]:

```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

veamos algunos valores de prediccion vs actuales

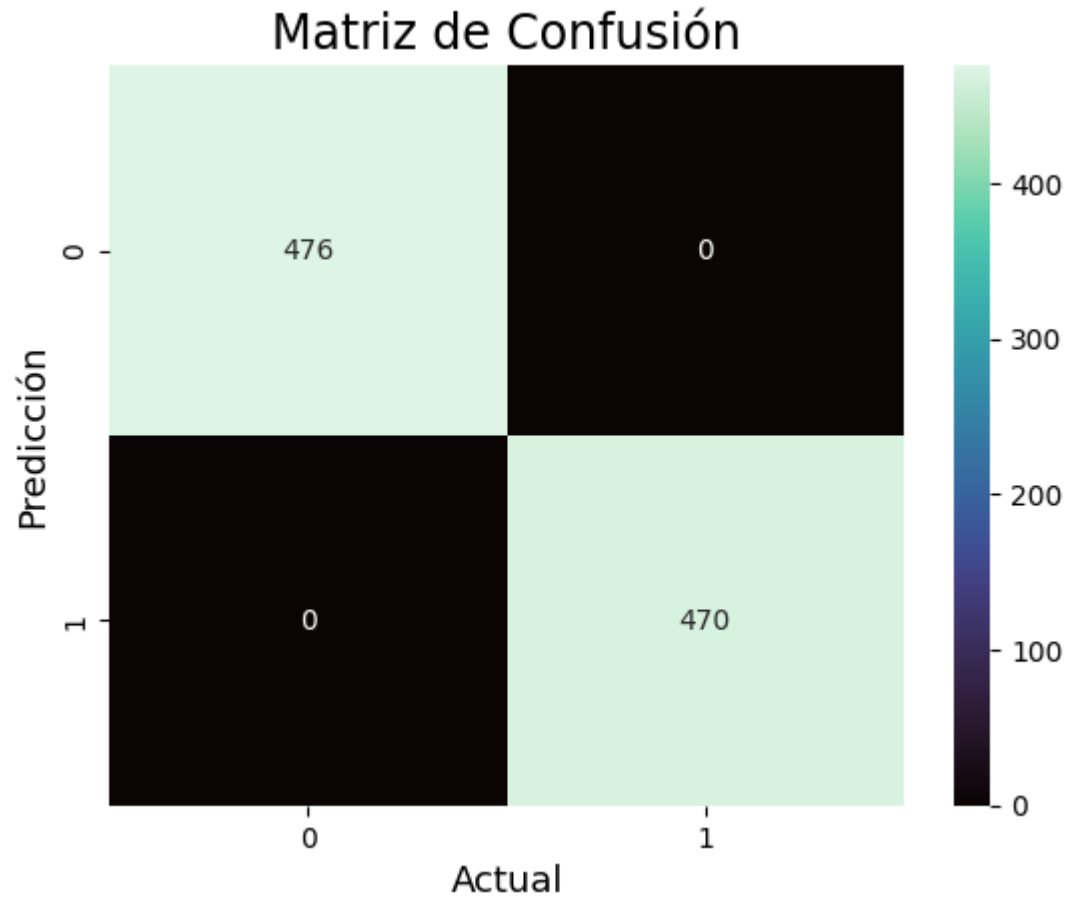
In [ ]:
```
y_pred = classifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Predicción': y_pred}).sample(10)
```

Out[ ]:

	Actual	Predicción
401	1	1
20	1	1
682	1	1
903	0	0
937	1	1
249	1	1
255	1	1
695	1	1
549	0	0
168	0	0

A simple vista se puede observar que son muy similares los valores actuales y los predichos, al igual que con las clasificaciones anteriores, procederemos a realizar una matriz de confusion para ver cuantos aciertos tuvimos

In [ ]:
```
plot_confusion_matrix(y_test, y_pred)
```



Nuevamente, se obtuvo muy pocos falsos negativos y muchos aciertos, veamos las metricas que obtuvimos

In [ ]:
```
plot_metricas(y_test, y_pred)
```

```
Precisión del modelo: 1.00
Exactitud del modelo: 1.00
Sensibilidad del modelo: 1.00
Puntaje F1 del modelo: 1.00
Curva ROC - AUC del modelo: 1.00
```

Podemos observar que la precision fue del 100% nuevamente

Conclusiones

Hemos logrado el objetivo de este trabajo, de poder predecir si hubo o no una falla electrica con una muy buena precision utilizando los metodos de clasificacion: KNN, SVM, Naive Bayes, Arboles de Decision y Random Forest.

Tambien hemos podido predecir y observar que con el metodo de Regresion Logistica tendriamos una precision muy pobre, esto se debe a que las categorias de fallas electricas no eran linealmente separables ya que estos datos se distribuian en forma de circulos o eclipses.

Referencias

[1] Paper Deteccion y clasificacion de fallas utilizando Machine Learning: https://www.ijraset.com/research-paper/fault-detection-and-classification-using-ml

[2] Dataset utilizado: kaggle - electrical fault detection and classification https://www.kaggle.com/datasets/esathyaprakash/electrical-fault-detection-and-classification