```
# gerardo Herrera... random forest (25 arboles) con 28k instacias de normal y recovering y 24
# comparacion de los diferentes por diferentes covariancia
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import numpy as np
import pandas as pd
import os

import matplotlib.pyplot as plt
%matplotlib inline
from tqdm import tqdm_notebook
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error
pd.options.display.precision = 15

import time
# Libraries
import numpy as np
import pandas as pd
pd.set_option('max_columns', None)
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

!pip install lightgbm
!pip install catboost

import datetime
import lightgbm as lgb
from scipy import stats
from sklearn.model_selection import train_test_split, StratifiedKFold, KFold, cross_val_score
from sklearn.preprocessing import StandardScaler
import os
import lightgbm as lgb
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn import metrics
from sklearn import linear_model
from tqdm import tqdm_notebook
from catboost import CatBoostClassifier
```

```
Requirement already satisfied: lightgbm in /usr/local/lib/python3.6/dist-packages (2.2.3
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from lig
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from lig
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (f
Collecting catboost
  Downloading https://files.pythonhosted.org/packages/20/37/bc4e0ddc30c07a96482abf1de7ec
    |████████████████████████████████| 65.8MB 58kB/s
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from cat
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from catbo
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from ca
Requirement already satisfied: graphviz in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/li
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (f
Installing collected packages: catboost
Successfully installed catboost-0.24.4
```

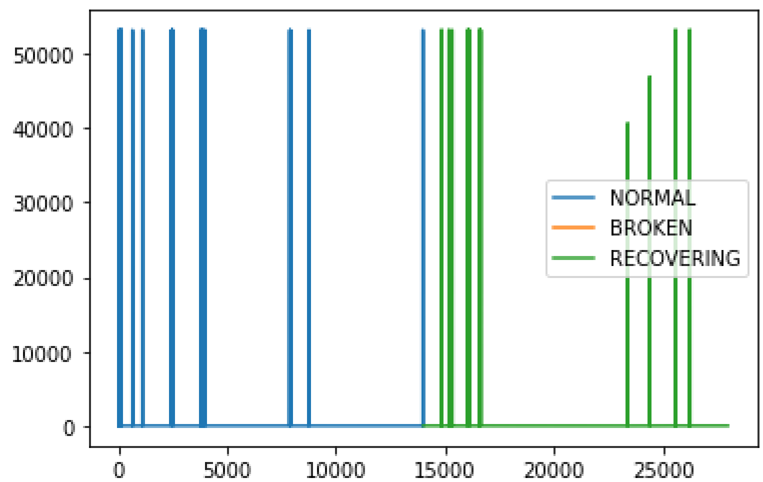
```python
# sensor = pd.read_csv('../input/sensor.csv')
# sensor = pd.read_csv('../input/vombas/sensor_procesado.csv')
#sensor = pd.read_csv('dataset_sensor_procesado.csv')
#sensor = pd.read_csv('../input/bombas-sensores-conocidos/sensor2.csv')
#sensor = pd.read_csv('../input/28k-s24-balan-vombas/sensor2-ordenado_status_sin_broken_balan
#sensor.drop(['Unnamed: 0'], axis=1, inplace=True)

sensor = pd.read_csv('/content/drive/My Drive/datasets/sensor2-ordenado_status_sin_broken_bal


sensor.head()
```

	Unnamed: 0	timestamp	sensor_00	sensor_01	sensor_02	sens
0	0	2018-04-01	2.465394	47.092010000000002	53.211799999999997	46.3107600000
		2018-04-				
2	2	01	2.444734	47.35242999999998	53.211799999999997	46.3975700000

```
#sensor.drop(['sensor_15'], axis=1, inplace=True)
sensor.drop(['timestamp'], axis=1, inplace=True)
```

```
# lineA DE LOS 22K INSTANCIAS
plt.plot(sensor.loc[sensor['machine_status'] == 'NORMAL', 'sensor_02'], label='NORMAL')
plt.plot(sensor.loc[sensor['machine_status'] == 'BROKEN', 'sensor_02'], label='BROKEN')
plt.plot(sensor.loc[sensor['machine_status'] == 'RECOVERING', 'sensor_02'], label='RECOVERING
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa3e6a072b0>
```




```
cleanup_nums = {"machine_status":     {"NORMAL": 0, "RECOVERING": 1,"BROKEN": 2}}


sensor.replace(cleanup_nums, inplace=True)
sensor.head(30)
```

	Unnamed: 0	sensor_00	sensor_01	sensor_02	sensor_03
0	0	2.465394	47.092010000000002	53.211799999999997	46.310760000000002
1	1	2.465394	47.092010000000002	53.211799999999997	46.310760000000002
2	2	2.444734	47.352429999999998	53.211799999999997	46.397570000000002
3	3	2.460474	47.092010000000002	53.168399999999998	46.397567749023402
4	4	2.445718	47.135410000000000	53.211799999999997	46.397567749023402
5	5	2.453588	47.092010000000002	53.168399999999998	46.397567749023402
6	6	2.455556	47.048609999999996	53.168399810790994	46.397567749023402
7	7	2.449653	47.135410000000000	53.168399810790994	46.397567749023402
8	8	2.463426	47.092010000000002	53.168399810790994	46.397567749023402
9	9	2.445718	47.178820000000002	53.168399999999998	46.397567749023402
10	10	2.464410	47.482640000000004	53125.000000000000000	46.397567749023402
11	11	2.444734	47.916660000000000	53.168399999999998	46.397567749023402
12	12	2.460474	48.263890000000004	53125.000000000000000	46.397567749023402
13	13	2.448669	48.437500000000000	53.168399999999998	46.397567749023402
14	14	2.453588	48.567709999999998	53.168399999999998	46.397567749023402
15	15	2.455556	48.394100000000002	53125.000000000000000	46.397570000000002
16	16	2.449653	48.394100000000002	53.168399999999998	46.310760000000002
17	17	2.463426	48.480899999999998	53.689240000000012	46.310760498046896
18	18	2.445718	48.611109999999996	53125.000000000000000	46.310760498046896
19	19	2.464410	48.611109999999996	53.168399999999998	46.310760498046896
20	20	2.445718	49.088540000000002	53.038190000000000	46.310760498046896
21	21	2.460474	49.218750000000000	53125.000000000000000	46.310760000000002
22	22	2.448669	48.784720000000000	53125.000000000000000	46.267359999999996
23	23	2.453588	49.088540000000002	53.168399999999998	46.267360687255895

```
for col in sensor.columns[1:-1]:
    sensor[col] = sensor[col].fillna(sensor[col].mean())
```

25	25	2.449653	49.305549999999997	53.168399999999998	46.267360687255895

```
# bosque aleatorio
```

27	27	2.448669	48.784720000000000	53125.000000000000000	46.267360687255895

```
sensor.fillna(sensor.mean(), inplace=True)
```

```
sensor.head()
```

```
sensor.head()
```

	Unnamed: 0	sensor_00	sensor_01	sensor_02	sensor_03	
0	0	2.465394	47.092010000000002	53.211799999999997	46.310760000000002	6343
1	1	2.465394	47.092010000000002	53.211799999999997	46.310760000000002	6343
2	2	2.444734	47.352429999999998	53.211799999999997	46.397570000000002	6
3	3	2.460474	47.092010000000002	53.168399999999998	46.397567749023402	6281
4	4	2.445718	47.135410000000000	53.211799999999997	46.397567749023402	6

```
print(sensor.shape)
```

```
(28002, 26)
```

```
# Encontrar características importantes en Scikit-learn

# from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
#clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
#clf.fit(X_train,y_train)


# no correr
#import pandas as pd
#feature_imp = pd.Series(clf.feature_importances_,index=iris.feature_names).sort_values(ascen
#feature_imp = pd.Series(clf.feature_importances_,index=sensor.columns[19:27]).sort_values(as
#print(feature_imp)

#Visualización
#import matplotlib.pyplot as plt
#import seaborn as sns
#%matplotlib inline
# Creating a bar plot
#sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
#plt.xlabel('Feature Importance Score')
#plt.ylabel('Features')
#plt.title("Visualizing Important Features")
#plt.legend()
#plt.show()
```

```
# n_estimators=25
```

```
#X=sensor[['sensor_00', 'sensor_01', 'sensor_02', 'sensor_03','sensor_04', 'sensor_11', 'sens
X=sensor[['sensor_50', 'sensor_44', 'sensor_28', 'sensor_31','sensor_26', 'sensor_25', 'senso
# https://www.datacamp.com/community/tutorials/python-rename-column?utm_source=adwords_ppc&ut
#y=sensor['target']  # Labels
X= X.rename(columns = {'sensor_50': 's50', 'sensor_44': 's44', 'sensor_28': 's28', 'sensor_31
y=sensor['machine_status']  # Labels

# Split dataset into training set and test set
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) # 80% training and 2


from sklearn.ensemble import RandomForestClassifier

#Create a Random Forest Classifier
clf=RandomForestClassifier(n_estimators=25)

start = time.time()

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

stop = time.time()
print(f"Training time: {stop - start}s")

y_pred=clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

#predicciones del item 17156 q es 1
#clf.predict([[0.0,53.55902,52.77777,43.402774810790994,204.72509765625,3.7302410000000004,40
```

```
Training time: 0.47502899169921875s
Accuracy: 1.0
```

```
#predicciones
#clf.predict([[0.0,53.55902,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]])



#predicciones
#clf.predict([[0.0,53.55902,52.77777,43.402774810790994,204.72509765625,3.7302410000000004,40


# Extract single tree
estimator = clf.estimators_[5]
```

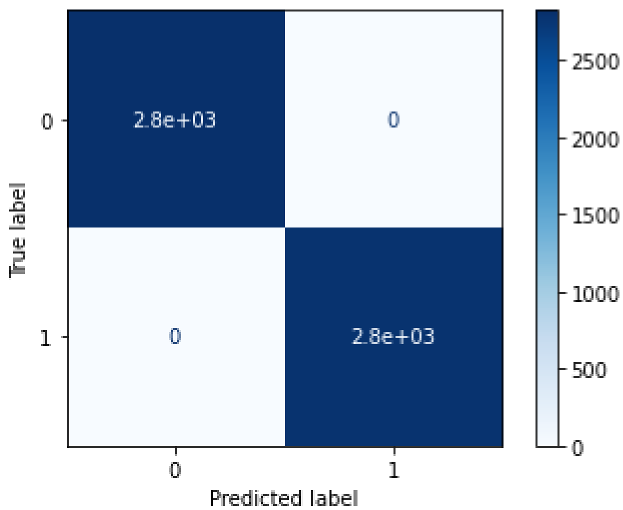
```
#from sklearn.tree import export_graphviz
# Export as dot file
#export_graphviz(estimator, out_file='tree.dot',
#                feature_names = ['sensor_00', 'sensor_01', 'sensor_02', 'sensor_03','sensor_
#                class_names = [ 'machine_status'],
 #               rounded = True, proportion = False,
 #               precision = 2, filled = True)


# validacion cruzada
# https://jamesrledoux.com/code/k_fold_cross_validation


from sklearn.model_selection import cross_validate


start1 = time.time()
model = RandomForestClassifier(random_state=1)
cv = cross_validate(model, X, y, cv=10)
print(cv['test_score'])
print(cv['test_score'].mean())
stop1 = time.time()
print(f"Training time: {stop1 - start1}s")
```

```
[0.99892895 0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.93107143]
0.9929643367164788
Training time: 22.235894918441772s
```

```
#https://stackoverflow.com/questions/20662023/save-python-random-forest-model-to-file


from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

start1 = time.time()
#model = RandomForestClassifier(random_state=1)
model = RandomForestClassifier(n_estimators=25)

cv = cross_validate(model, X, y, cv=10)
print(confusion_matrix(y_test,y_pred))
print(cv['test_score'])
print(cv['test_score'].mean())
stop1 = time.time()
print(f"Training time: {stop1 - start1}s")

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

#plot_confusion_matrix(clf, X_test, y_test)
# plot confusion matrix(clf, X test, y test)
```

```
# plot_confusion_matrix(clf, X_test, y_test)
plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues)

plt.show()
```

```
[[2822    0]
 [   0 2779]]
[1.         0.99964298 1.         1.         0.99857143 1.
 1.         1.         1.         0.99928571]
0.9997500127505482
Training time: 5.762657165527344s
[[2822    0]
 [   0 2779]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2822
           1       1.00      1.00      1.00      2779

    accuracy                           1.00      5601
   macro avg       1.00      1.00      1.00      5601
weighted avg       1.00      1.00      1.00      5601
```




```
# version with multi scroring
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

start1 = time.time()
#model = RandomForestClassifier(random_state=1)
model = RandomForestClassifier(n_estimators=25)

cv = cross_validate(model, X, y, cv=10)
#recall_score=cross_validation.cross_val_score(clf, X,y, cv=10, scoring ='recall')
#recall_score=cross_val_score(model, X,y, cv=10, scoring ='recall')
f1=cross_validate(model, X,y, cv=10, scoring ='f1')
recall_score=cross_validate(model, X,y, cv=10, scoring ='recall')
pre_score=cross_validate(model, X,y, cv=10, scoring ='precision_macro')
print(confusion_matrix(y_test,y_pred))
print(f"precision_macro_score:")
```

```
print(f"precision_macro_score:")
print(pre_score['test_score'])
print(pre_score['test_score'].mean())
print(f"test_score:")
print(cv['test_score'])
print(cv['test_score'].mean())
print(f"recall:")
print(recall_score['test_score'])
print(recall_score['test_score'].mean())
print(f"f1score:")
print(f1['test_score'])
print(f1['test_score'].mean())
stop1 = time.time()
print(f"Training time: {stop1 - start1}s")

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

#plot_confusion_matrix(clf, X_test, y_test)
# plot_confusion_matrix(clf, X_test, y_test)
plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues)

plt.show()
```

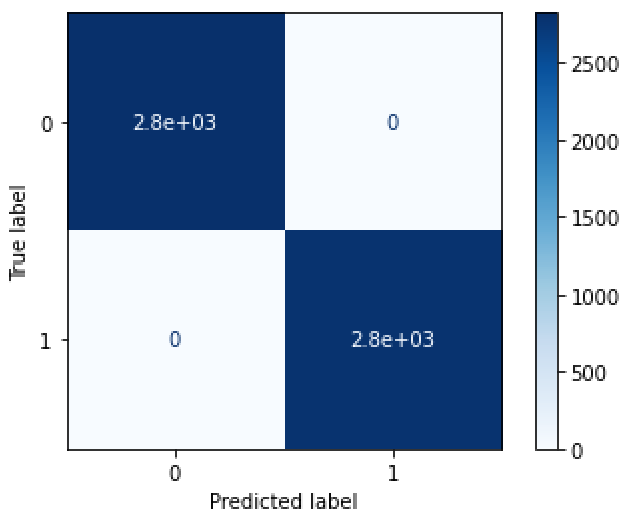
```
[[2822    0]
 [   0 2779]]
precision_macro_score:
[1.         0.99964311 1.         1.         1.         1.
 1.         1.         1.         0.85361752]
0.9853260627811086
test_score:
[1.         0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.93428571]
0.9933928698934054
recall:
[0.99571429 0.99928622 1.         1.         1.         1.
 1.         1.         1.         0.99857143]
0.9993571938411339
f1score:
```

```
# version with multi scroring mejorada
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

start1 = time.time()
#model = RandomForestClassifier(random_state=1)
model = RandomForestClassifier(n_estimators=25)

#GH
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
#GH

cv = cross_validate(model, X, y, cv=10)
#recall_score=cross_validation.cross_val_score(clf, X,y, cv=10, scoring ='recall')
#recall_score=cross_val_score(model, X,y, cv=10, scoring ='recall')

#scoring = ['neg_mean_absolute_error','r2']

scores=cross_validate(model, X,y, cv=10, scoring = ['accuracy','f1','recall','precision'],ret
#recall_score=cross_validate(model, X,y, cv=10, scoring ='recall')
#pre_score=cross_validate(model, X,y, cv=10, scoring ='precision_macro')
print(confusion_matrix(y_test,y_pred))

print(f"multi_metric_scores:")
#print(scores['test_score'])
print(scores)
#print(scores['test_score'].mean())

#print(scores.mean())

#print(f"precision_macro_score:")
#print(pre_score['test_score'])
#print(pre_score['test_score'].mean())

#print(f"test_score:")
#print(cv['test_score'])
```

```
#print(cv['test_score'])
#print(cv['test_score'].mean())

#print(f"recall:")
#print(recall_score['test_score'])
#print(recall_score['test_score'].mean())

#print(f"f1score:")
#print(f1['test_score'])
#print(f1['test_score'].mean())

stop1 = time.time()
print(f"Training time: {stop1 - start1}s")

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

#plot_confusion_matrix(clf, X_test, y_test)
# plot_confusion_matrix(clf, X_test, y_test)
plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues)

plt.show()

gh4 = scores.get("test_accuracy")

print(f"accuracy:")
print(gh4)
print(gh4.mean())

gh3 = scores.get("test_precision")

print(f"precision:")
print(gh3)
print(gh3.mean())

gh = scores.get("test_recall")

print(f"recall:")
print(gh)
print(gh.mean())

gh2 = scores.get("test_f1")

print(f"f1:")
print(gh2)
print(gh2.mean())

CM = confusion_matrix(y_test, y_pred)
print(f"--------")
print(f"matriz de confusion:")
TN = CM[0][0]
FN = CM[1][0]
```

```
TP = CM[1][1]
FP = CM[0][1]
print(f"TN={TN}, FP={FP} ")
print(f"FN={FN}, TP={TP} ")

print(f"--------")
print(f"matriz de confusion %:")
total1=(TN+TP+FN+FP)

print(f"TN={100*TN/total1}, FP={100*FP/total1} ")
print(f"FN={100*FN/total1}, TP={100*TP/total1} ")

print(f"--------")
acc1=(TN+TP)/(TN+TP+FN+FP)
print(f"accuracy1={acc1}")

print(f"--------")
re1=(TP)/(TP+FN)
print(f"reca1={re1}")

print(f"--------")
pre1=(TP)/(TP+FP)
print(f"pre1={pre1}")

print(f"--------")
f1s1=(2*pre1*re1)/(pre1+re1)
print(f"f1score={f1s1}")
```

```
[[2822    0]
 [   0 2779]]
multi_metric_scores:
{'fit_time': array([0.56579471, 0.54468942, 0.55483556, 0.56191278, 0.57669592,
       0.53374624, 0.58592129, 0.55244994, 0.55253959, 0.50533867]), 'score_time': array
       0.01252532, 0.01103711, 0.01109099, 0.01137662, 0.01111388]), 'test_accuracy': ar
       1.        , 1.        , 1.        , 1.        , 0.92285714]), 'test_f1': array([0
       1.        , 1.        , 1.        , 1.        , 0.92828685]), 'test_recall': arra
       1.        , 1.        , 1.        , 1.        , 0.99857143]), 'test_precision': a
       1.        , 1.        , 1.        , 1.        , 0.86724566])}
Training time: 11.806705236434937s
[[2822    0]
 [   0 2779]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2822
           1       1.00      1.00      1.00      2779

    accuracy                           1.00      5601
   macro avg       1.00      1.00      1.00      5601
weighted avg       1.00      1.00      1.00      5601
```




```
accuracy:
[0.99892895 0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.92285714]
0.9921429081450504
precision:
[1.         1.         1.         1.         1.         1.
```

```
import joblib
from sklearn.ensemble import RandomForestClassifier
# create RF
```

```
0.9995714795554196
```

```
# save
joblib.dump(clf, "my_random_forest.joblib")
```

```
['my_random_forest.joblib']
```

```
matriz de confusion:
```

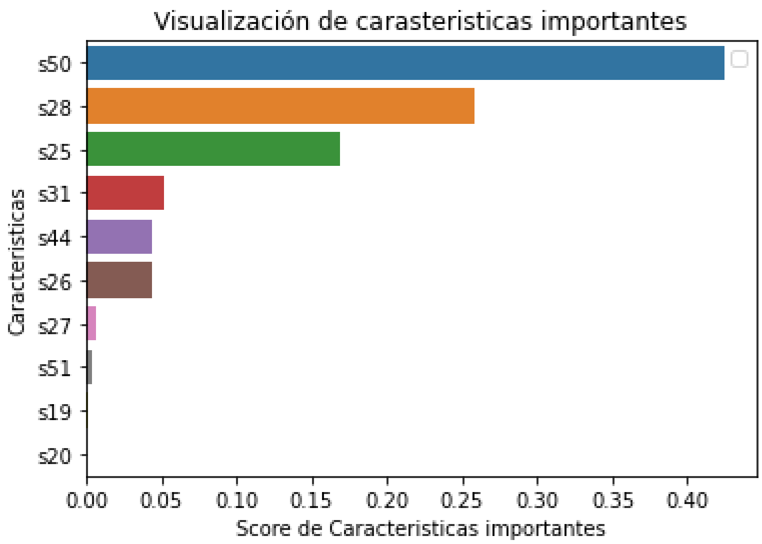
```
# load
```

```
loaded_rf = joblib.load("my_random_forest.joblib")
```

```
     matriz de confusion %:
```

```
#predicciones
#clf.predict([[0.0,53.55902,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]])
#predicciones
#loaded_rf.predict([[0.0,53.55902,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]])

# 1 es recovering
#loaded_rf.predict([[0.0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]])

# 0 es recovering
#loaded_rf.predict([[2.465394,47.092009999999995,53.2118,46.310759999999995,634375,47.52422,4


# 2 es broken
#loaded_rf.predict([[2.258796,47.26563,52.73437,43.4461784362793,200.11573791503898,43.62322,



import pandas as pd
#feature_imp = pd.Series(clf.feature_importances_,index=iris.feature_names).sort_values(ascen
#feature_imp = pd.Series(clf.feature_importances_,index=X.columns[1:8]).sort_values(ascending
#feature_imp = pd.Series(clf.feature_importances_,index=X.columns[0:24]).sort_values(ascendin
feature_imp = pd.Series(clf.feature_importances_,index=X.columns[0:24]).sort_values(ascending
print(feature_imp)

x1=feature_imp
#y1=feature_imp.X.columns[0:24]).sort_values(ascending=False)
y1=feature_imp.index

#Visualización
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
#plt.xlabel('Feature Importance Score')
plt.xlabel('Score de Caracteristicas importantes')
#plt.ylabel('Features')
plt.ylabel('Caracteristicas')
#plt.title("Visualizing Important Features")
plt.title("Visualización de carasteristicas importantes")
plt.legend()
plt.show()

#plt.savefig('destination_path.eps', format='eps' , dpi=1000)

plt.savefig('myimage.svg', format='svg', dpi=1200)
```

```
No handles with labels found to put in legend.
s50    0.424751004788311
s28    0.257588094073515
s25    0.168535262545393
s31    0.051044133401583
s44    0.044079456942063
s26    0.043627367930263
s27    0.006666899019197
s51    0.003074328721534
s19    0.000524926605115
s20    0.000108525973026
dtype: float64
```




```
<Figure size 432x288 with 0 Axes>
```

```
# https://towardsdatascience.com/how-to-visualize-a-decision-tree-from-a-random-forest-in-pyt


# otra rf


# n_estimators=100
#X2=sensor[['sensor_00', 'sensor_01', 'sensor_02', 'sensor_03','sensor_04', 'sensor_11', 'sen
X2=sensor[['sensor_28','sensor_50', 'sensor_25', 'sensor_26', 'sensor_44', 'sensor_31','senso
#y=sensor['target']  # Labels
X2= X2.rename(columns = {'sensor_50': 's50', 'sensor_44': 's44', 'sensor_28': 's28', 'sensor_
X2= X2.rename(columns = {'sensor_22': 's22', 'sensor_14': 's14'}, inplace = False)

y=sensor['machine_status']  # Labels

# Split dataset into training set and test set
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X2, y, test_size=0.2) # 80% training and


from sklearn.ensemble import RandomForestClassifier

#Create a Random Forest Classifier
```

```
clf=RandomForestClassifier(n_estimators=100)

start = time.time()

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

stop = time.time()
print(f"Training time: {stop - start}s")

y_pred=clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

#predicciones del item 17156 q es 1
#clf.predict([[0.0,53.55902,52.77777,43.402774810790994,204.72509765625,3.7302410000000004,40
```

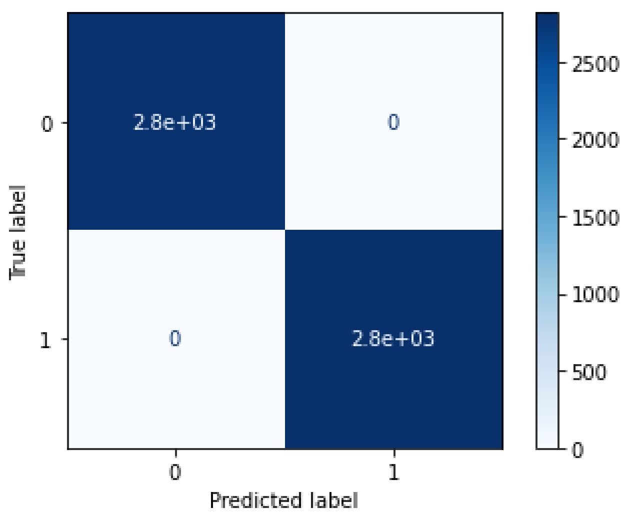
```
Training time: 2.1514952182769775s
Accuracy: 1.0
```

```
# version with multi scroring mejorada
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

start1 = time.time()
#model = RandomForestClassifier(random_state=1)
model = RandomForestClassifier(n_estimators=100)

#GH
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
#GH

cv = cross_validate(model, X2, y, cv=10)
#recall_score=cross_validation.cross_val_score(clf, X,y, cv=10, scoring ='recall')
#recall_score=cross_val_score(model, X,y, cv=10, scoring ='recall')

#scoring = ['neg_mean_absolute_error','r2']

scores=cross_validate(model, X2,y, cv=10, scoring = ['accuracy','f1','recall','precision'],re
#recall_score=cross_validate(model, X,y, cv=10, scoring ='recall')
#pre_score=cross_validate(model, X,y, cv=10, scoring ='precision_macro')
print(confusion_matrix(y_test,y_pred))

print(f"multi_metric_scores:")
#print(scores['test_score'])
print(scores)
```

```
    #print(scores['test_score'].mean())

    #print(scores.mean())

    #print(f"precision_macro_score:")
    #print(pre_score['test_score'])
    #print(pre_score['test_score'].mean())

    #print(f"test_score:")
    #print(cv['test_score'])
    #print(cv['test_score'].mean())

    #print(f"recall:")
    #print(recall_score['test_score'])
    #print(recall_score['test_score'].mean())

    #print(f"f1score:")
    #print(f1['test_score'])
    #print(f1['test_score'].mean())

    stop1 = time.time()
    print(f"Training time: {stop1 - start1}s")

    print(confusion_matrix(y_test,y_pred))
    print(classification_report(y_test,y_pred))

    #plot_confusion_matrix(clf, X_test, y_test)
    # plot_confusion_matrix(clf, X_test, y_test)
    plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues)

    plt.show()

    gh4 = scores.get("test_accuracy")

    print(f"accuracy:")
    print(gh4)
    print(gh4.mean())

    gh3 = scores.get("test_precision")

    print(f"precision:")
    print(gh3)
    print(gh3.mean())

    gh = scores.get("test_recall")

    print(f"recall:")
    print(gh)
    print(gh.mean())

    gh2 = scores.get("test_f1")
```

```
print(f"f1:")
print(gh2)
print(gh2.mean())

CM = confusion_matrix(y_test, y_pred)
print(f"--------")
print(f"matriz de confusion:")
TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
print(f"TN={TN}, FP={FP} ")
print(f"FN={FN}, TP={TP} ")

print(f"--------")
print(f"matriz de confusion %:")
total1=(TN+TP+FN+FP)

print(f"TN={100*TN/total1}, FP={100*FP/total1} ")
print(f"FN={100*FN/total1}, TP={100*TP/total1} ")

print(f"--------")
acc1=(TN+TP)/(TN+TP+FN+FP)
print(f"accuracy1={acc1}")

print(f"--------")
re1=(TP)/(TP+FN)
print(f"reca1={re1}")

print(f"--------")
pre1=(TP)/(TP+FP)
print(f"pre1={pre1}")

print(f"--------")
f1s1=(2*pre1*re1)/(pre1+re1)
print(f"f1score={f1s1}")
```

```
[[2785    0]
 [   0 2816]]
multi_metric_scores:
{'fit_time': array([2.13406587, 2.11660218, 2.21964049, 2.04910231, 2.177001  ,
       2.11884999, 2.16349006, 2.12058425, 2.14469528, 2.07227969]), 'score_time': array
       0.02583504, 0.02701926, 0.02577019, 0.02714133, 0.0261147 ]), 'test_accuracy': ar
       1.        , 1.        , 1.        , 1.        , 0.99928571]), 'test_f1': array([1
       1.        , 1.        , 1.        , 1.        , 0.9992852 ]), 'test_recall': arra
       1.        , 1.        , 1.        , 1.        , 0.99857143]), 'test_precision': a
Training time: 45.115500688552856s
[[2785    0]
 [   0 2816]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2785
           1       1.00      1.00      1.00      2816

    accuracy                           1.00      5601
   macro avg       1.00      1.00      1.00      5601
weighted avg       1.00      1.00      1.00      5601
```




```
accuracy:
[1.         0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.99928571]
0.9998928698934053
precision:
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
1.0
recall:
[1.         0.99928622 1.         1.         1.         1.
 1.         1.         1.         0.99857143]
0.9997857652697053
f1:
[1.         0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.9992852 ]
0.999892818836528
--------
matriz de confusion:
TN=2785, FP=0
```

```
from scipy import stats
```

```
#stats.ttest_rel(df['bp_before'], df['bp_after'])
stats.ttest_rel(gh4, gh3)
#stats.ttest_rel(gh2, gh)
```

```
Ttest_relResult(statistic=-1.4054822401941298, pvalue=0.1934455436174361)
--------
```

```
import pandas as pd
#feature_imp = pd.Series(clf.feature_importances_,index=iris.feature_names).sort_values(ascen
#feature_imp = pd.Series(clf.feature_importances_,index=X.columns[1:8]).sort_values(ascending
feature_imp2 = pd.Series(clf.feature_importances_,index=X2.columns[0:24]).sort_values(ascendi

#feature_imp2 = pd.Series(clf.feature_importances_,index=X2.columns).sort_values(ascending=Fa
print(feature_imp2)
#print(feature_imp2)

#Visualización
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Creating a bar plot
sns.barplot(x=feature_imp2, y=feature_imp2.index)
# Add labels to your graph
#plt.xlabel('Feature Importance Score')
plt.xlabel('Score de Caracteristicas importantes')
#plt.ylabel('Features')
plt.ylabel('Caracteristicas')
#plt.title("Visualizing Important Features")
plt.title("Visualización de carasteristicas importantes")
plt.legend()
plt.show()

#plt.savefig('destination_path.eps', format='eps' , dpi=1000)

plt.savefig('myimage.svg', format='svg', dpi=1200)
```

```
No handles with labels found to put in legend.
s50    0.331783576319425
s28    0.164916906695706
s26    0.159915598917073
s25    0.137043558569975
s44    0.092036586537369
s22    0.078737795254665
s31    0.018374022001212
s51    0.010338762695317
s27    0.005402614533553
s14    0.001450578475704
dtype: float64
```


```
# doble plot
# https://matplotlib.org/gallery/subplots_axes_and_figures/subplots_demo.html
```

```
# n_estimators=500)
#X3=sensor[['sensor_00', 'sensor_01', 'sensor_02', 'sensor_03','sensor_04', 'sensor_11', 'sen
X3=sensor[['sensor_50','sensor_28', 'sensor_26', 'sensor_25', 'sensor_44', 'sensor_31','senso
#y=sensor['target']  # Labels
X3= X3.rename(columns = {'sensor_50': 's50', 'sensor_44': 's44', 'sensor_28': 's28', 'sensor_
X3= X3.rename(columns = {'sensor_22': 's22', 'sensor_14': 's14'}, inplace = False)
y=sensor['machine_status']  # Labels

# Split dataset into training set and test set
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X3, y, test_size=0.2) # 80% training and


from sklearn.ensemble import RandomForestClassifier

#Create a Random Forest Classifier
clf=RandomForestClassifier(n_estimators=500)

start = time.time()

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

stop = time.time()
print(f"Training time: {stop - start}s")

y_pred=clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
#predicciones del item 17156 q es 1
#clf.predict([[0.0,53.55902,52.77777,43.402774810790994,204.72509765625,3.7302410000000004,40
```

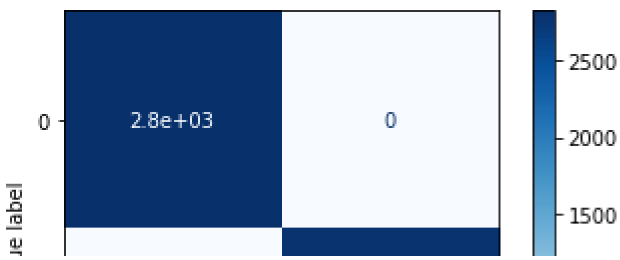
```
Training time: 10.027456045150757s
Accuracy: 1.0
```

```
start1 = time.time()
model = RandomForestClassifier(random_state=1)
cv = cross_validate(model, X3, y, cv=10)
print(cv['test_score'])
print(cv['test_score'].mean())
stop1 = time.time()
print(f"Training time: {stop1 - start1}s")
```

```
[1.         0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.99214286]
0.9991785841791196
Training time: 21.957491397857666s
```

```
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

start1 = time.time()
#model = RandomForestClassifier(random_state=1)
model = RandomForestClassifier(n_estimators=500)

cv = cross_validate(model, X3, y, cv=10)
print(confusion_matrix(y_test,y_pred))
print(cv['test_score'])
print(cv['test_score'].mean())
stop1 = time.time()
print(f"Training time: {stop1 - start1}s")

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

#plot_confusion_matrix(clf, X_test, y_test)
# plot_confusion_matrix(clf, X_test, y_test)
plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues)

plt.show()
```

```
[[2820    0]
 [   0 2781]]
[1.         0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.99892857]
0.9998571556076911
Training time: 110.38733720779419s
[[2820    0]
 [   0 2781]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2820
           1       1.00      1.00      1.00      2781

    accuracy                           1.00      5601
   macro avg       1.00      1.00      1.00      5601
weighted avg       1.00      1.00      1.00      5601
```

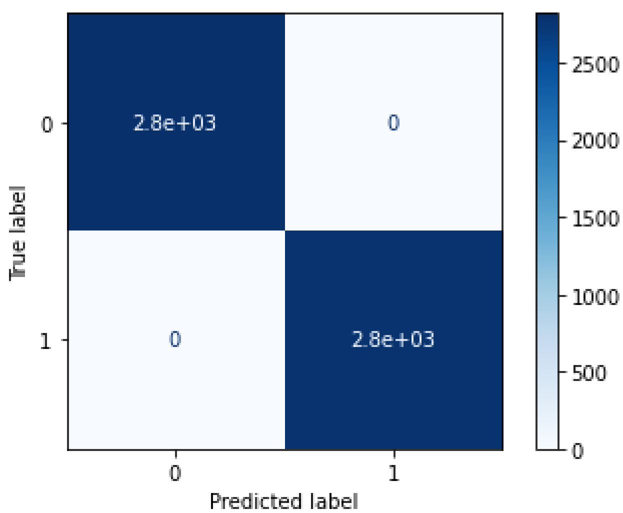



```
# version with multi scroring mejorada
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

start1 = time.time()
#model = RandomForestClassifier(random_state=1)
model = RandomForestClassifier(n_estimators=500)

#GH
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
#GH

cv = cross_validate(model, X3, y, cv=10)
#recall_score=cross_validation.cross_val_score(clf, X,y, cv=10, scoring ='recall')
#recall_score=cross_val_score(model, X,y, cv=10, scoring ='recall')

#scoring = ['neg_mean_absolute_error','r2']

scores=cross_validate(model, X3,y, cv=10, scoring = ['accuracy','f1','recall','precision'],re
#recall_score=cross_validate(model, X,y, cv=10, scoring ='recall')
#pre_score=cross_validate(model, X,y, cv=10, scoring ='precision_macro')
print(confusion_matrix(y_test,y_pred))

print(f"multi_metric_scores:")
#print(scores['test_score'])
print(scores)
#print(scores['test_score'].mean())
```

```
#print(scores['test_score'].mean())

#print(scores.mean())

#print(f"precision_macro_score:")
#print(pre_score['test_score'])
#print(pre_score['test_score'].mean())

#print(f"test_score:")
#print(cv['test_score'])
#print(cv['test_score'].mean())

#print(f"recall:")
#print(recall_score['test_score'])
#print(recall_score['test_score'].mean())

#print(f"f1score:")
#print(f1['test_score'])
#print(f1['test_score'].mean())

stop1 = time.time()
print(f"Training time: {stop1 - start1}s")

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

#plot_confusion_matrix(clf, X_test, y_test)
# plot_confusion_matrix(clf, X_test, y_test)
plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues)

plt.show()

gh4 = scores.get("test_accuracy")

print(f"accuracy:")
print(gh4)
print(gh4.mean())

gh3 = scores.get("test_precision")

print(f"precision:")
print(gh3)
print(gh3.mean())

gh = scores.get("test_recall")

print(f"recall:")
print(gh)
print(gh.mean())

gh2 = scores.get("test_f1")
```

```
    print(f"f1:")
    print(gh2)
    print(gh2.mean())

    CM = confusion_matrix(y_test, y_pred)
    print(f"--------")
    print(f"matriz de confusion:")
    TN = CM[0][0]
    FN = CM[1][0]
    TP = CM[1][1]
    FP = CM[0][1]
    print(f"TN={TN}, FP={FP} ")
    print(f"FN={FN}, TP={TP} ")

    print(f"--------")
    print(f"matriz de confusion %:")
    total1=(TN+TP+FN+FP)

    print(f"TN={100*TN/total1}, FP={100*FP/total1} ")
    print(f"FN={100*FN/total1}, TP={100*TP/total1} ")

    print(f"--------")
    acc1=(TN+TP)/(TN+TP+FN+FP)
    print(f"accuracy1={acc1}")

    print(f"--------")
    re1=(TP)/(TP+FN)
    print(f"reca1={re1}")

    print(f"--------")
    pre1=(TP)/(TP+FP)
    print(f"pre1={pre1}")

    print(f"--------")
    f1s1=(2*pre1*re1)/(pre1+re1)
    print(f"f1score={f1s1}")
```

```
[[2820    0]
 [   0 2781]]
multi_metric_scores:
{'fit_time': array([10.17055106, 10.27183318, 10.8692565 , 10.50330186, 10.26805496,
       10.74668527, 10.89880562, 10.81330132, 10.87387753, 10.35521698]), 'score_time':
       0.10432959, 0.10426211, 0.10720825, 0.10558581, 0.10331106]), 'test_accuracy': ar
       1.        , 1.        , 1.        , 1.        , 0.99892857]), 'test_f1': array([1
       1.        , 1.        , 1.        , 1.        , 0.99892819]), 'test_recall': arra
       1.        , 1.        , 1.        , 1.        , 0.99857143]), 'test_precision': a
       1.       , 1.       , 1.       , 0.9992852])}
Training time: 224.7085771560669s
[[2820    0]
 [   0 2781]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2820
           1       1.00      1.00      1.00      2781

    accuracy                           1.00      5601
   macro avg       1.00      1.00      1.00      5601
weighted avg       1.00      1.00      1.00      5601
```

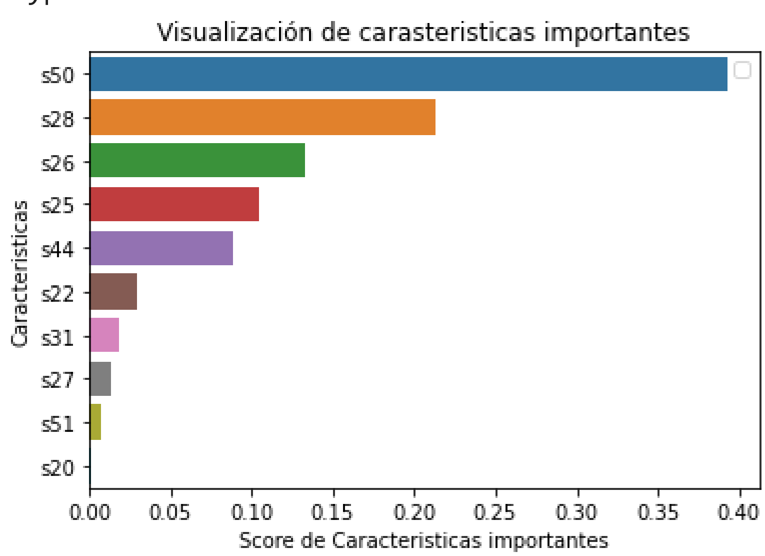



```
accuracy:
[1.         0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.99892857]
0.9998571556076911
precision:
[1.        1.        1.        1.        1.        1.        1.
 1.        1.        0.9992852]
0.9999285203716941
recall:
[1.         0.99928622 1.         1.         1.         1.
 1.         1.         1.         0.99857143]
0.9997857652697053
f1:
[1.         0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.99892819]
0.999857117328714
```

```
import pandas as pd
#feature_imp = pd.Series(clf.feature_importances_,index=iris.feature_names).sort_values(ascen
#feature_imp = pd.Series(clf.feature_importances_,index=X.columns[1:8]).sort_values(ascending
```

```
feature_imp3 = pd.Series(clf.feature_importances_,index=X3.columns).sort_values(ascending=Fal
print(feature_imp3)

#Visualización
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Creating a bar plot
sns.barplot(x=feature_imp3, y=feature_imp3.index)
# Add labels to your graph
#plt.xlabel('Feature Importance Score')
plt.xlabel('Score de Caracteristicas importantes')
#plt.ylabel('Features')
plt.ylabel('Caracteristicas')
#plt.title("Visualizing Important Features")
plt.title("Visualización de carasteristicas importantes")
plt.legend()
plt.show()

#plt.savefig('destination_path.eps', format='eps' , dpi=1000)

plt.savefig('myimage.svg', format='svg', dpi=1200)
```

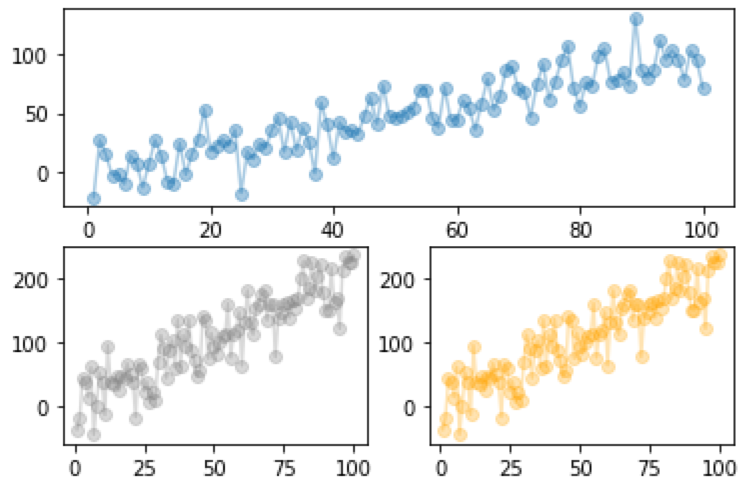
```
No handles with labels found to put in legend.
s50    0.393070120036386
s28    0.212628691091245
s26    0.132563063278011
s25    0.104254266104868
s44    0.088812765842395
s22    0.029298725806180
s31    0.017683349388846
s27    0.013719661096821
s51    0.007182280001457
s20    0.000787077353790
dtype: float64
```




```
<Figure size 432x288 with 0 Axes>
```

```
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
df=pd.DataFrame({'x': range(1,101), 'y': np.random.randn(100)*15+range(1,101), 'z': (np.rando
ax1 = plt.subplot2grid((2, 2), (0, 0), colspan=2)
#ax1.plot( 'x', 'y', marker='o', alpha=0.4)
ax1.plot( 'x', 'y', data=df, marker='o', alpha=0.4)
ax2 = plt.subplot2grid((2, 2), (1, 0), colspan=1)
ax2.plot( 'x','z', data=df, marker='o', color="grey", alpha=0.3)
ax3 = plt.subplot2grid((2, 2), (1, 1), colspan=1)
ax3.plot( 'x','z', data=df, marker='o', color="orange", alpha=0.3)
```

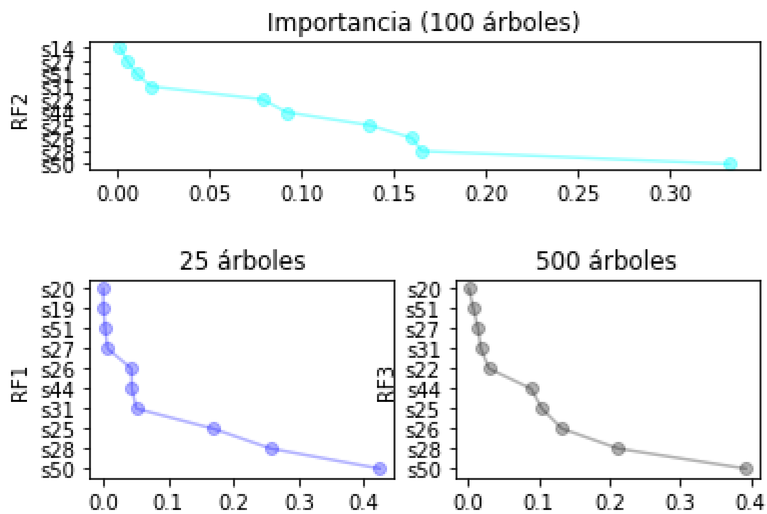
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: Second a
  import sys
[<matplotlib.lines.Line2D at 0x7fa3e4610d30>]
```




```
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
df=pd.DataFrame({'x': range(1,101), 'y': np.random.randn(100)*15+range(1,101), 'z': (np.rando
# n 25
df1=pd.DataFrame({'x': feature_imp, 'y': feature_imp.index})
# n 100
df2=pd.DataFrame({'x': feature_imp2, 'y': feature_imp2.index})
# n 500
df3=pd.DataFrame({'x': feature_imp3, 'y': feature_imp3.index})
plt.ylabel('ylabel', fontsize=6)
#ax1.ylabel('ylabel', fontsize=6)
#plt.yticks(fontsize=6)
plt.yticks(fontsize=6)
#ax1.yticks=[1, 2, 3]
#ax1 = plt.subplot2grid((2, 2), (0, 0), colspan=2)
ax1 = plt.subplot2grid((3, 2), (0, 0), colspan=2)
#ax1.plot( x1, y1, data=df, marker='o', alpha=0.4)
ax1.set_title('Importancia (100 árboles)')
#ax1.set(xlabel='x-label', ylabel='RF1')
ax1.set( ylabel='RF2')
```

```
ax1.set( ylabel='RF2')
#ax1.plot( 'x','y',data=df2, marker='o', alpha=0.4)
ax1.plot( 'x','y',data=df2, marker='o' , color="cyan" , alpha=0.4)
ax2 = plt.subplot2grid((2, 2), (1, 0), colspan=1)
#ax2.plot( 'x','z', data=df, marker='o', color="grey", alpha=0.3)
ax2.set_title('25 árboles')
ax2.set( ylabel='RF1')
#ax2.plot( 'x','y', data=df1, marker='o', color="grey", alpha=0.3)
ax2.plot( 'x','y', data=df1, marker='o', color="blue", alpha=0.3)
ax3 = plt.subplot2grid((2, 2), (1, 1), colspan=1)
#ax3.plot( 'x','z', data=df, marker='o', color="orange", alpha=0.3)
#ax3.plot( 'x','z', data=df3, marker='o', color="orange", alpha=0.3)
ax3.set_title('500 árboles')
ax3.set( ylabel='RF3')
#ax3.label_outer()
ax3.plot( 'x','y', data=df3, marker='o', color="black", alpha=0.3)
#ax3.plot( 'x','y', data=df3, marker='o', color="navy", alpha=0.3)
plt.savefig('filename.png', dpi=1200)
```

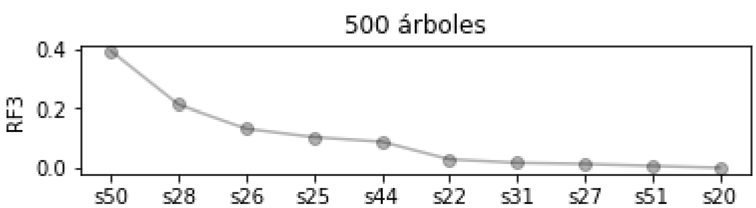
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:23: RuntimeWarning: Second
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:29: RuntimeWarning: Second
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:36: RuntimeWarning: Second
```




```
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
df=pd.DataFrame({'x': range(1,101), 'y': np.random.randn(100)*15+range(1,101), 'z': (np.rando
# n 25
df1=pd.DataFrame({'y': feature_imp, 'x': feature_imp.index})
# n 100
df2=pd.DataFrame({'y': feature_imp2, 'x': feature_imp2.index})
# n 500
df3=pd.DataFrame({'y': feature_imp3, 'x': feature_imp3.index})
plt.ylabel('ylabel', fontsize=6)
#ax1.ylabel('ylabel', fontsize=6)
#plt.yticks(fontsize=6)
```

```
plt.yticks(fontsize=6)
#ax1.yticks=[1, 2, 3]
#ax1 = plt.subplot2grid((2, 2), (0, 0), colspan=2)
ax1 = plt.subplot2grid((1, 1), (0, 0), rowspan=1)
#ax1.plot( x1, y1, data=df, marker='o', alpha=0.4)
ax1.set_title('sensores ó caracteristicas')
#ax1.set(xlabel='x-label', ylabel='RF1')
ax1.set( ylabel='RF2(importancia)')
#ax1.plot( 'x','y',data=df2, marker='o', alpha=0.4)
ax1.plot( 'x','y',data=df2, marker='o' , color="cyan" , alpha=0.4)
ax2 = plt.subplot2grid((2, 1), (0, 0), rowspan=1)
#ax2.plot( 'x','z', data=df, marker='o', color="grey", alpha=0.3)
ax2.set_title('25 árboles')
ax2.set( ylabel='RF1(importancia)')
#ax2.plot( 'x','y', data=df1, marker='o', color="grey", alpha=0.3)
ax2.plot( 'x','y', data=df1, marker='o', color="blue", alpha=0.3)
ax3 = plt.subplot2grid((3, 1), (0, 0), rowspan=1)
#ax3.plot( 'x','z', data=df, marker='o', color="orange", alpha=0.3)
#ax3.plot( 'x','z', data=df3, marker='o', color="orange", alpha=0.3)
ax3.set_title('500 árboles')
ax3.set( ylabel='RF3')
#ax3.label_outer()
ax3.plot( 'x','y', data=df3, marker='o', color="black", alpha=0.3)
#ax3.plot( 'x','y', data=df3, marker='o', color="navy", alpha=0.3)
plt.savefig('filename.png', dpi=1200)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:23: RuntimeWarning: Second
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:29: RuntimeWarning: Second
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:36: RuntimeWarning: Second
```

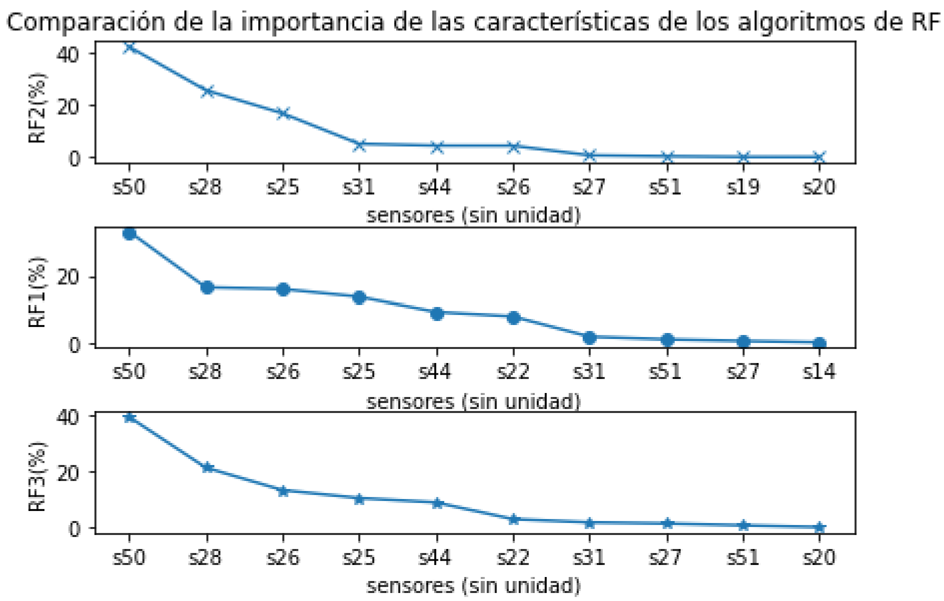



```
x=feature_imp.index
y=feature_imp
x2=feature_imp2.index
y2=feature_imp2
x3=feature_imp3.index
y3=feature_imp3
```

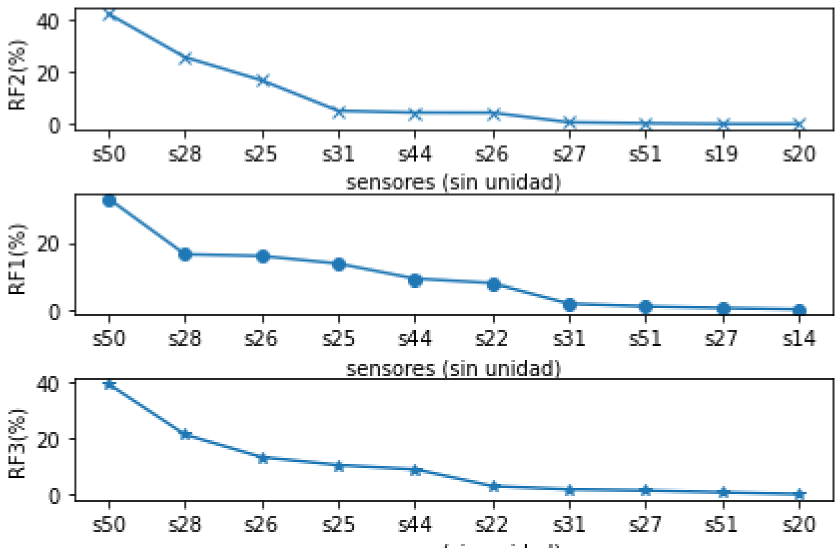
```
#https://matplotlib.org/3.1.0/gallery/subplots_axes_and_figures/subplots_demo.html
fig, (ax1, ax2, ax3) = plt.subplots(3)
plt.tight_layout()
#fig.title('Importancia()')
#fig.suptitle('Importancia()')
ax1.title.set_text('Comparación de la importancia de las características de los algoritmos de
```

```
ax1.title.set_text('Comparación de la importancia de las características de los algoritmos de
ax1.set( ylabel='RF2(%)', xlabel='sensores (sin unidad)')
#ax1.plot(x, y*100)
ax1.plot(x, y*100, marker='x')
# ax2.set_title('Importancia()')
ax2.set( ylabel='RF1(%)', xlabel='sensores (sin unidad)')
ax2.plot(x2, y2*100, marker='o')
# ax3.set_title('Importancia()')
ax3.set( ylabel='RF3(%)', xlabel='sensores (sin unidad)')
#ax3.plot(x3, y3*100)
ax3.plot(x3, y3*100, marker='*')
```

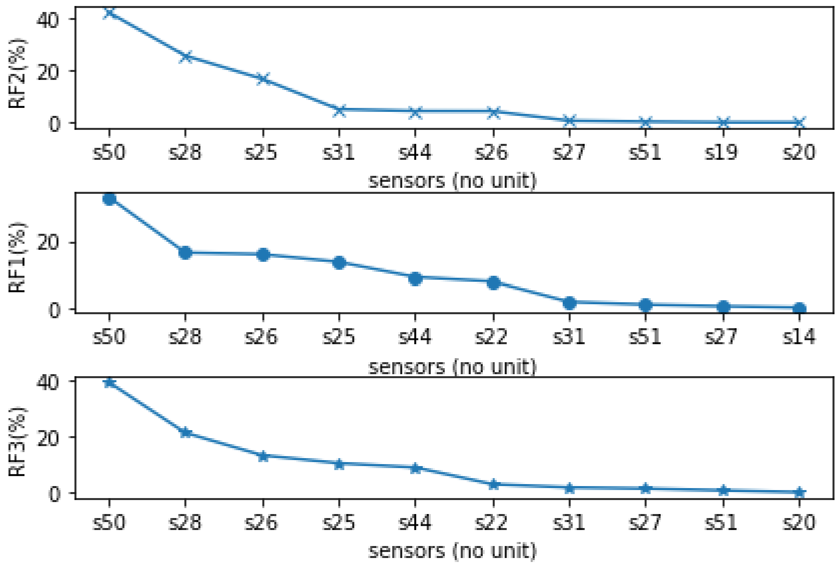
```
[<matplotlib.lines.Line2D at 0x7fa3e43a5da0>]
```



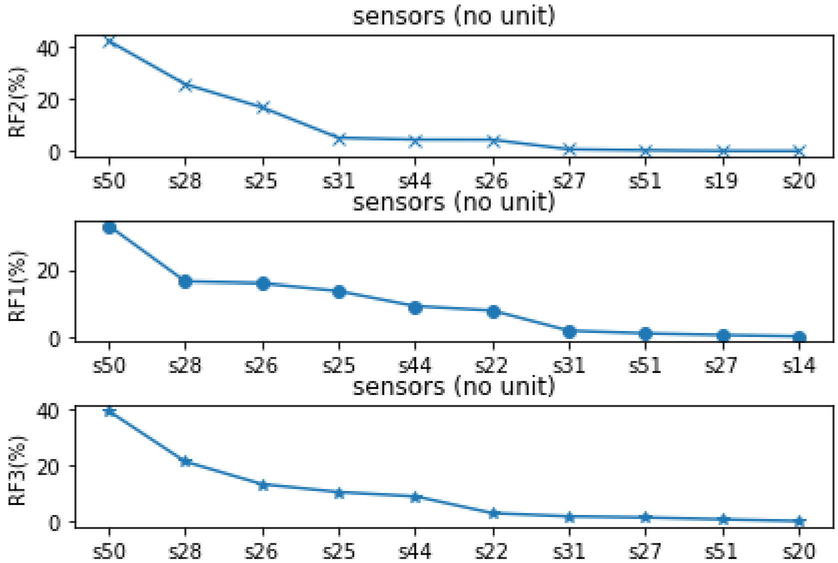

```
#https://matplotlib.org/3.1.0/gallery/subplots_axes_and_figures/subplots_demo.html
fig, (ax1, ax2, ax3) = plt.subplots(3)
plt.tight_layout()
#fig.title('Importancia()')
#fig.suptitle('Importancia()')
#ax1.title.set_text('Comparación de la importancia de las características de los algoritmos d
ax1.set( ylabel='RF2(%)', xlabel='sensores (sin unidad)')
#ax1.plot(x, y*100)
ax1.plot(x, y*100, marker='x')
# ax2.set_title('Importancia()')
ax2.set( ylabel='RF1(%)', xlabel='sensores (sin unidad)')
ax2.plot(x2, y2*100, marker='o')
# ax3.set_title('Importancia()')
ax3.set( ylabel='RF3(%)', xlabel='sensores (sin unidad)')
#ax3.plot(x3, y3*100)
ax3.plot(x3, y3*100, marker='*')
plt.savefig('juntasMejordescripcion.png', dpi=1200)
```




```
#fig, axs = plt.subplots(1, 3)
#axs[0, 0].plot(x, y)
#axs[0, 0].plot(feature_imp2, feature_imp2.index)
#axs[0, 0].set_title("main")
#axs[1, 0].plot(x, y**2)
#axs[1, 0].plot(feature_imp, feature_imp.index)
#axs[1, 0].set_title("shares x with main")
#axs[1, 0].sharex(axs[0, 0])
#axs[0, 1].plot(x + 1, y + 1)
#axs[0, 1].plot(feature_imp, feature_imp.index)
#axs[1, 0].plot(feature_imp, feature_imp)
#axs[0, 1].set_title("unrelated")
#axs[1, 1].plot(x + 2, y + 2)
#axs[1, 3].plot(x + 2, y + 2)
#axs[1, 1].set_title("also unrelated")
#axs[1, 3].set_title("also unrelated")
#fig.tight_layout()


#https://matplotlib.org/3.1.0/gallery/subplots_axes_and_figures/subplots_demo.html
fig, (ax1, ax2, ax3) = plt.subplots(3)
plt.tight_layout()
#fig.title('Importancia()')
#fig.suptitle('Importancia()')
#ax1.title.set_text('Comparación de la importancia de las características de los algoritmos d
ax1.set( ylabel='RF2(%)', xlabel='sensors (no unit)')
#ax1.plot(x, y*100)
ax1.plot(x, y*100, marker='x')
# ax2.set_title('Importancia()')
ax2.set( ylabel='RF1(%)', xlabel='sensors (no unit)')
ax2.plot(x2, y2*100, marker='o')
# ax3.set_title('Importancia()')
ax3.set( ylabel='RF3(%)', xlabel='sensors (no unit)')
#ax3.plot(x3, y3*100)
ax3.plot(x3, y3*100, marker='*')
plt.savefig('juntasMejordescripcion.png', dpi=1200)
```




```
#https://matplotlib.org/3.1.0/gallery/subplots_axes_and_figures/subplots_demo.html
fig, (ax1, ax2, ax3) = plt.subplots(3)
plt.tight_layout()
#fig.title('Importancia()')
#fig.suptitle('Importancia()')
#ax1.title.set_text('Comparación de la importancia de las características de los algoritmos d
#ax1.set( ylabel='RF2(%)', xlabel='sensors (no unit)')
ax1.set( ylabel='RF2(%)')
ax1.set_title('sensors (no unit)')
#ax1.plot(x, y*100)
ax1.plot(x, y*100, marker='x')
# ax2.set_title('Importancia()')
#ax2.set( ylabel='RF1(%)', xlabel='sensors (no unit)')
ax2.set( ylabel='RF1(%)')
ax2.set_title('sensors (no unit)')
ax2.plot(x2, y2*100, marker='o')
# ax3.set_title('Importancia()')
#ax3.set( ylabel='RF3(%)', xlabel='sensors (no unit)')
ax3.set( ylabel='RF3(%)')
ax3.set_title('sensors (no unit)')
#ax3.plot(x3, y3*100)
ax3.plot(x3, y3*100, marker='*')
plt.savefig('juntasMejordescripcion77.png', dpi=1200)

#for ax in fig.get_axes():
#    ax.label_outer()
```


sensors (no unit)
RF2(%)
s50 s28 s25 s31 s44 s26 s27 s51 s19 s20
sensors (no unit)
RF1(%)
s50 s28 s26 s25 s44 s22 s31 s51 s27 s14
sensors (no unit)
RF3(%)
s50 s28 s26 s25 s44 s22 s31 s27 s51 s20