```
# gerardo Herrera... random forest (100 arboles) con 28k instacias de normal y recovering
```

```
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_sc
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
```

```
# 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(as
#feature_imp = pd.Series(clf.feature_importances_,index=sensor.columns[19:27]).sort_values
#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()



X=sensor[['sensor_00', 'sensor_01', 'sensor_02', 'sensor_03','sensor_04', 'sensor_11', 'se
#y=sensor['target']  # Labels
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 a

# 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 an


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")
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarnir
  import pandas.util.testing as tm
Requirement already satisfied: lightgbm in /usr/local/lib/python3.6/dist-packages (2
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages
Collecting catboost
  Downloading https://files.pythonhosted.org/packages/90/86/c3dcb600b4f9e7584ed90ea9c
    |████████████████████████████████| 66.1MB 60kB/s
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from ca
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: graphviz in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (fron
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.6/dis
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages
```

```
# 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_ba
sensor = pd.read_csv('/content/drive/My Drive/datasets/sensor2-ordenado_status_sin_broken_
#sensor.drop(['Unnamed: 0'], axis=1, inplace=True)
```

```
sensor.head()
```

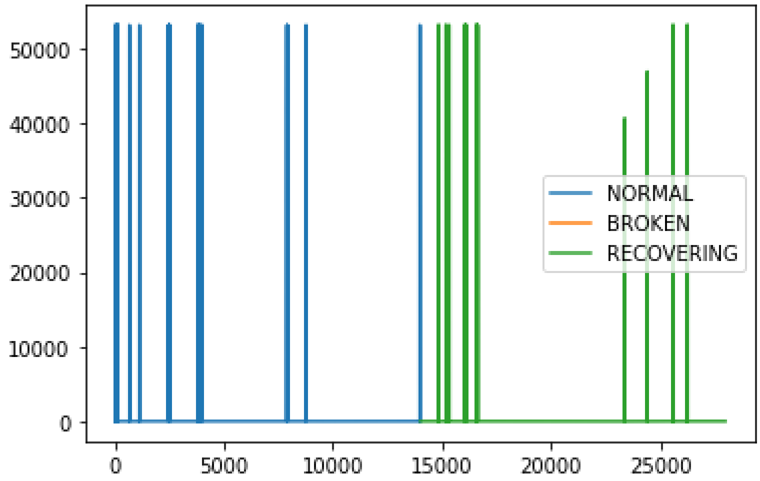
	Unnamed: 0	timestamp	sensor_00	sensor_01	sensor_02	s
0	0	2018-04-01 00:00:00	2.465394	47.092010000000002	53.211799999999997	46.3107600
1	1	2018-04-01 00:01:00	2.465394	47.092010000000002	53.211799999999997	46.3107600
2	2	2018-04-01 00:02:00	2.444734	47.352429999999998	53.211799999999997	46.3975700
3	3	2018-04-01 00:03:00	2.460474	47.092010000000002	53.168399999999998	46.3975677
4	4	2018-04-01 00:04:00	2.445718	47.135410000000000	53.211799999999997	46.3975677

```
#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='RECOVER
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fd19c7e2390>
```




```
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_0
0	0	2.465394	47.092010000000002	53.211799999999997	46.3107600000000
1	1	2.465394	47.092010000000002	53.211799999999997	46.3107600000000
2	2	2.444734	47.352429999999998	53.211799999999997	46.3975700000000
3	3	2.460474	47.092010000000002	53.168399999999998	46.3975677490234
4	4	2.445718	47.135410000000000	53.211799999999997	46.3975677490234
5	5	2.453588	47.092010000000002	53.168399999999998	46.3975677490234
6	6	2.455556	47.048609999999996	53.168399810790994	46.3975677490234
7	7	2.449653	47.135410000000000	53.168399810790994	46.3975677490234
8	8	2.463426	47.092010000000002	53.168399810790994	46.3975677490234
9	9	2.445718	47.178820000000002	53.168399999999998	46.3975677490234
10	10	2.464410	47.482640000000004	53125.000000000000000	46.3975677490234
11	11	2.444734	47.916660000000000	53.168399999999998	46.3975677490234
12	12	2.460474	48.263890000000004	53125.000000000000000	46.3975677490234
13	13	2.448669	48.437500000000000	53.168399999999998	46.3975677490234

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

```
# bosque aleatorio
```

17	17	2.463426	48.480899999999998	53.689240000000012	46.31076049804689

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

```
sensor.head()
```

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

28	28	2.464410	48.350690000000000	53.168399999999998	46.2673599999999

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

```
(28002, 26)
```

```
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,
```

```
Training time: 2.8685848712921143s
Accuracy: 0.9996429209069809
array([1])
```

```
#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]])
```

```
array([1])
```

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

```
array([1])
```

```
# 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','sens
#                 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")
```

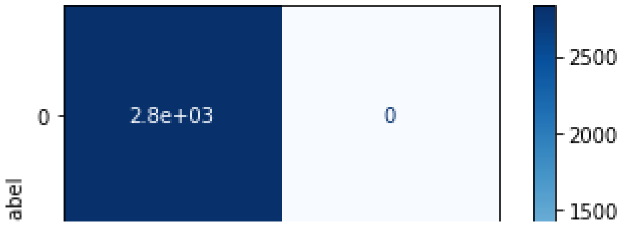
```
[0.99464477 0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.85642857]
0.9850716325802009
Training time: 31.289194583892822s
```

```
#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=100)

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, cmap=plt.cm.Blues)

plt.show()
```

```
[[2828    0]
 [   2 2771]]
[0.99678686 0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.86214286]
0.9858572703626255
Training time: 31.3827486038208s
```

```
# 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=100)

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(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()
```

```
[[2828    0]
 [   2 2771]]
precision_macro_score:
[0.99470339 0.99964311 1.         1.         1.         1.
 1.         1.         1.         0.86600578]
0.9860352279204246
test_score:
[0.99714388 0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.93678571]
0.9933572576120773
recall:
[0.98642857 0.99928622 1.         1.         1.         1.
 1.         1.         1.         0.99857143]
0.9984286224125626
f1score:
[0.99749373 0.99964298 1.         1.         1.         1.
 1.         1.         1.         0.8680534 ]
0.9865190118549533
Training time: 125.59279036521912s
[[2828    0]
 [   2 2771]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2828
           1       1.00      1.00      1.00      2773

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




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

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

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

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


#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]])
```

```
array([1])
```

```
# 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]])
```

```
array([1])
```

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

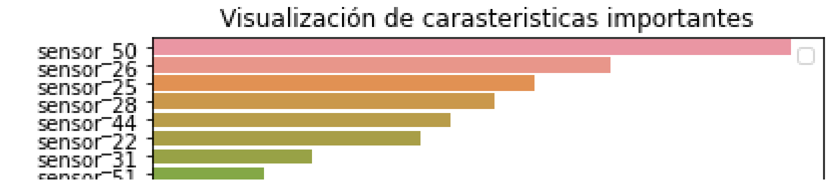
```
array([1])
```

```
# 2 es broken
loaded_rf.predict([[2.258796,47.26563,52.73437,43.4461784362793,200.11573791503898,43.6232
```

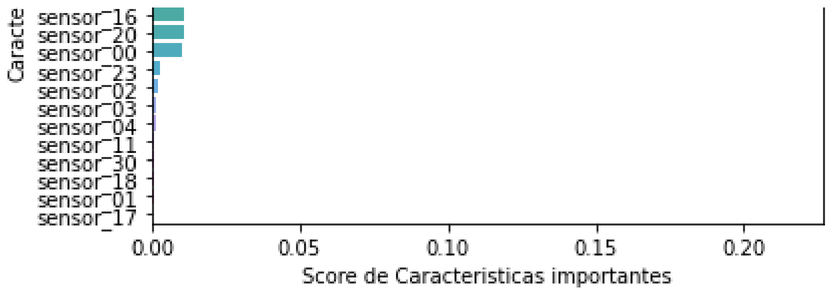
```
array([1])
```

```
import pandas as pd
#feature_imp = pd.Series(clf.feature_importances_,index=iris.feature_names).sort_values(as
#feature_imp = pd.Series(clf.feature_importances_,index=X.columns[1:8]).sort_values(ascend
feature_imp = pd.Series(clf.feature_importances_,index=X.columns[0:24]).sort_values(ascend
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.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.
sensor_50    0.215916918172932
sensor_26    0.154926089421661
sensor_25    0.128967351923029
sensor_28    0.115787939226255
sensor_44    0.100605509159674
sensor_22    0.090685535738714
sensor_31    0.054359178840330
sensor_51    0.037829247277325
sensor_27    0.021963297775157
sensor_19    0.013749260653028
sensor_21    0.013089560310996
sensor_14    0.011999772525839
sensor_16    0.010866154105111
sensor_20    0.010529401001523
sensor_00    0.009865498082195
sensor_23    0.002712213500527
sensor_02    0.001784758702430
sensor_03    0.001327415576626
sensor_04    0.001236767824419
sensor_11    0.000532141878278
sensor_30    0.000411923212256
sensor_18    0.000401726198045
sensor_01    0.000379114263316
sensor_17    0.000073224630334
dtype: float64
```




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




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