```
# gerardo Herrera... svm (kernel radial "rbf") con 28k instacias de normal y recovering y
```

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

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

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that gets preserved
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#matplotlib inline
```

```
#sensor77 = pd.read_csv('../input/vombas/sensor_procesado.csv')
#sensor77 = pd.read_csv('../input/vombas-28955-balanced-25sensor/sensor2-ordenado_status_s
#sensor77 = pd.read_csv('../input/bombas-sensores-conocidos/sensor2.csv')
#sensor77 = pd.read_csv('../input/10ks25/s25balanced10k.csv')
#sensor77 = pd.read_csv('../input/28k-s24-balan-vombas/sensor2-ordenado_status_sin_broken_
sensor77 = pd.read_csv('/content/drive/My Drive/datasets/sensor2-ordenado_status_sin_broke
```

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

```
sensor77.replace(cleanup_nums, inplace=True)
```

```
print(sensor77.shape)
sensor77.head()
```

```
(28002, 27)
```

	Unnamed: 0	timestamp	sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	sens
0	0	2018-04-01 00:00:00	2.465394	47.09201	53.2118	46.310760	634375.0000	47
1	1	2018-04-01 00:01:00	2.465394	47.09201	53.2118	46.310760	634375.0000	47
2	2	2018-04-01 00:02:00	2.444734	47.35243	53.2118	46.397570	638.8889	48
3	3	2018-04-01 00:03:00	2.460474	47.09201	53.1684	46.397568	628125.0000	48

```
sensor77.isnull()
```

	Unnamed: 0	timestamp	sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	s
0	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	
27997	False	False	False	False	False	False	False	
27998	False	False	False	False	False	False	False	
27999	False	False	False	False	False	False	False	
28000	False	False	False	False	False	False	False	
28001	False	False	False	False	False	False	False	

28002 rows × 27 columns

```
# sensor77.dropna()
# sensor77.fillna(0, inplace=True)
sensor77.fillna(sensor77.mean(), inplace=True)
```

```
print(sensor77.shape)
sensor77.head()
```

```
(28002, 27)
```

	Unnamed: 0	timestamp	sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	sens
0	0	2018-04-01 00:00:00	2.465394	47.09201	53.2118	46.310760	634375.0000	47
1	1	2018-04-01 00:01:00	2.465394	47.09201	53.2118	46.310760	634375.0000	47
2	2	2018-04-01 00:02:00	2.444734	47.35243	53.2118	46.397570	638.8889	48
3	3	2018-04-01 00:03:00	2.460474	47.09201	53.1684	46.397568	628125.0000	48
4	4	2018-04-01 00:04:00	2.445718	47.13541	53.2118	46.397568	636.4583	49

```
is_NaN = sensor77.isnull()
row_has_NaN = is_NaN.any(axis=1)
rows_with_NaN = sensor77[row_has_NaN]

print(rows_with_NaN)
```

```
Empty DataFrame
Columns: [Unnamed: 0, timestamp, sensor_00, sensor_01, sensor_02, sensor_03, sensor_(
Index: []
```

```
#Show the number of missing (NAN, NaN, na) data for each column
sensor77.isnull().sum()
```

```
Unnamed: 0       0
timestamp        0
sensor_00        0
sensor_01        0
sensor_02        0
sensor_03        0
sensor_04        0
sensor 11        0
```

```
#sensor77.drop('sensor_15', axis=1, inplace=True);
#sensor77.drop('sensor_15', axis=1, inplace=True)
```

```
sensor_19        0
```

```
#sensor77.drop('sensor_05', axis=1, inplace=True);sensor77.drop('sensor_06', axis=1, inpla
```

```
sensor_22        0
```

```
#sensor77.drop('sensor_07', axis=1, inplace=True);sensor77.drop('sensor_08', axis=1, inpla
```

```
sensor_26        0
```

```
#sensor77.drop('sensor_09', axis=1, inplace=True);sensor77.drop('sensor_10', axis=1, inpla
```

```
sensor_30        0
```

```
#sensor77.drop('sensor_12', axis=1, inplace=True);sensor77.drop('sensor_13', axis=1, inpla
```

```
sensor_50        0
```

```
#sensor77.drop('timestamp', axis=1, inplace=True)
```

```
dtype: int64
```

```
#sensor77.drop('Unnamed: 0', axis=1, inplace=True)


print(sensor77.shape)
sensor77.head()
```

```
(28002, 27)
```

	Unnamed: 0	timestamp	sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	sens
0	0	2018-04-01 00:00:00	2.465394	47.09201	53.2118	46.310760	634375.0000	47
1	1	2018-04-01 00:01:00	2.465394	47.09201	53.2118	46.310760	634375.0000	47
2	2	2018-04-01 00:02:00	2.444734	47.35243	53.2118	46.397570	638.8889	48
3	3	2018-04-01 00:03:00	2.460474	47.09201	53.1684	46.397568	628125.0000	48
4	4	2018-04-01 00:04:00	2.445718	47.13541	53.2118	46.397568	636.4583	49

```
import time


X=sensor77[['sensor_00', 'sensor_01', 'sensor_02', 'sensor_03','sensor_04', 'sensor_11', '
#y=sensor['target']  # Labels
y=sensor77['machine_status']  # Labels



print(X.shape)
X.head()
```

```
(28002, 24)
```

	sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	sensor_11	sensor_14	
0	2.465394	47.09201	53.2118	46.310760	634375.0000	47.52422	419.5747	
1	2.465394	47.09201	53.2118	46.310760	634375.0000	47.52422	419.5747	
2	2.444734	47.35243	53.2118	46.397570	638.8889	48.17723	420848.0000	
3	2.460474	47.09201	53.1684	46.397568	628125.0000	48.65607	420.7494	4
4	2.445718	47.13541	53.2118	46.397568	636.4583	49.06298	419.8926	

```
# Scale the data to be between -1 and 1
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)




# no coorrer
#import time


#X_train=X
#start = time.time()
#X = sensor77.drop('machine_status', axis=1)
#y = sensor77['machine_status']

#from sklearn import preprocessing
#X_train = preprocessing.scale(X_train)
#X_test = preprocessing.scale(X_test)

#from sklearn.model_selection import train_test_split
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)

#from sklearn.preprocessing import MinMaxScaler
#scaling = MinMaxScaler(feature_range=(-1,1)).fit(X_train)
#X_train = scaling.transform(X_train)
#X_test = scaling.transform(X_test)

#!pip install sklearn.grid search
```

```
#!pip install sklearn.grid_search

#from sklearn.svm import SVR
#from sklearn.grid_search import GridSearchCV
#svclassifier = SVR(kernel='linear')
#svclassifier.fit(X_train, y_train)

#from sklearn.svm import SVC
#svclassifier = SVC(kernel='linear')
#svclassifier.fit(X_train, y_train)

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


start = time.time()
#X = sensor77.drop('machine_status', axis=1)
#y = sensor77['machine_status']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)

from sklearn.svm import SVC
svclassifier = SVC(kernel='rbf', random_state=0, gamma=.01, C=1)
svclassifier.fit(X_train, y_train)

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

```
Training time: 14.533451318740845s
```

```
y_pred = svclassifier.predict(X_test)

#Evaluacion
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[2579  232]
 [ 565 2225]]
              precision    recall  f1-score   support

           0       0.82      0.92      0.87      2811
           1       0.91      0.80      0.85      2790

    accuracy                           0.86      5601
   macro avg       0.86      0.86      0.86      5601
weighted avg       0.86      0.86      0.86      5601
```

```
# validacion cruzada


from sklearn.model_selection import cross_validate


#from sklearn.model_selection import cross_val_score
```

```
#from sklearn.model_selection import cross_val_score
# https://scikit-learn.org/stable/modules/cross_validation.html


#start3 = time.time()
from sklearn import svm
from sklearn.model_selection import cross_val_score
start3 = time.time()
clf = svm.SVC(kernel='rbf', random_state=0, gamma=.01, C=1)
#scores = cross_val_score(clf, X, y, cv=5)
scores = cross_val_score(clf, X, y, cv=10)
scores
print(scores.mean())
stop3 = time.time()
print(f"CV Training time: {stop3 - start3}s")
```

```
0.854941003213138
CV Training time: 175.27739644050598s
```

```
#start3 = time.time()
from sklearn import svm
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
start3 = time.time()
clf = svm.SVC(kernel='rbf', random_state=0, gamma=.1, C=1)
#scores = cross_val_score(clf, X, y, cv=5)
#scores = cross_val_score(clf, X, y, cv=10)
#cv = cross_val_score(clf, X, y, cv=10)
cv = cross_validate(clf, X, y, cv=10)
cv1 = cross_validate(clf, X, y, cv=10,scoring ='accuracy')
#scores
#cv
#print(scores.mean())
#print(cv.mean())
#
f1=cross_validate(clf, X,y, cv=10, scoring ='f1')
recall_score=cross_validate(clf, X,y, cv=10, scoring ='recall')
pre_score=cross_validate(clf, 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"accu:")
print(cv1['test_score'])
print(cv1['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())
```

```
print(f1['test_score'].mean())
#
stop3 = time.time()
print(f"CV Training time: {stop3 - start3}s")
```

```
[[2579  232]
 [ 565 2225]]
precision_macro_score:
[0.88144615 0.84796086 0.89941654 0.91531037 0.9331989  0.94477786
 0.94581513 0.93750558 0.91862923 0.89742729]
0.912148790685948
test_score:
[0.87325955 0.82113531 0.88964286 0.90821429 0.92892857 0.94285714
 0.94428571 0.9375     0.91178571 0.89142857]
0.9049037716121795
accu:
[0.87325955 0.82113531 0.88964286 0.90821429 0.92892857 0.94285714
 0.94428571 0.9375     0.91178571 0.89142857]
0.9049037716121795
recall:
[0.8        0.68236974 0.81142857 0.84285714 0.87928571 0.91
 0.915      0.93928571 0.84785714 0.83      ]
0.8458084021617213
f1score:
[0.86319846 0.79237464 0.88027896 0.90179595 0.92521608 0.94091581
 0.94260486 0.93761141 0.90576116 0.88432268]
0.8974079999777069
CV Training time: 833.3704061508179s
```

```
# print(scores[])
#print(scores.mean())
#stop3 = time.time()
#print(f"CV Training time: {stop3 - start3}s")
```