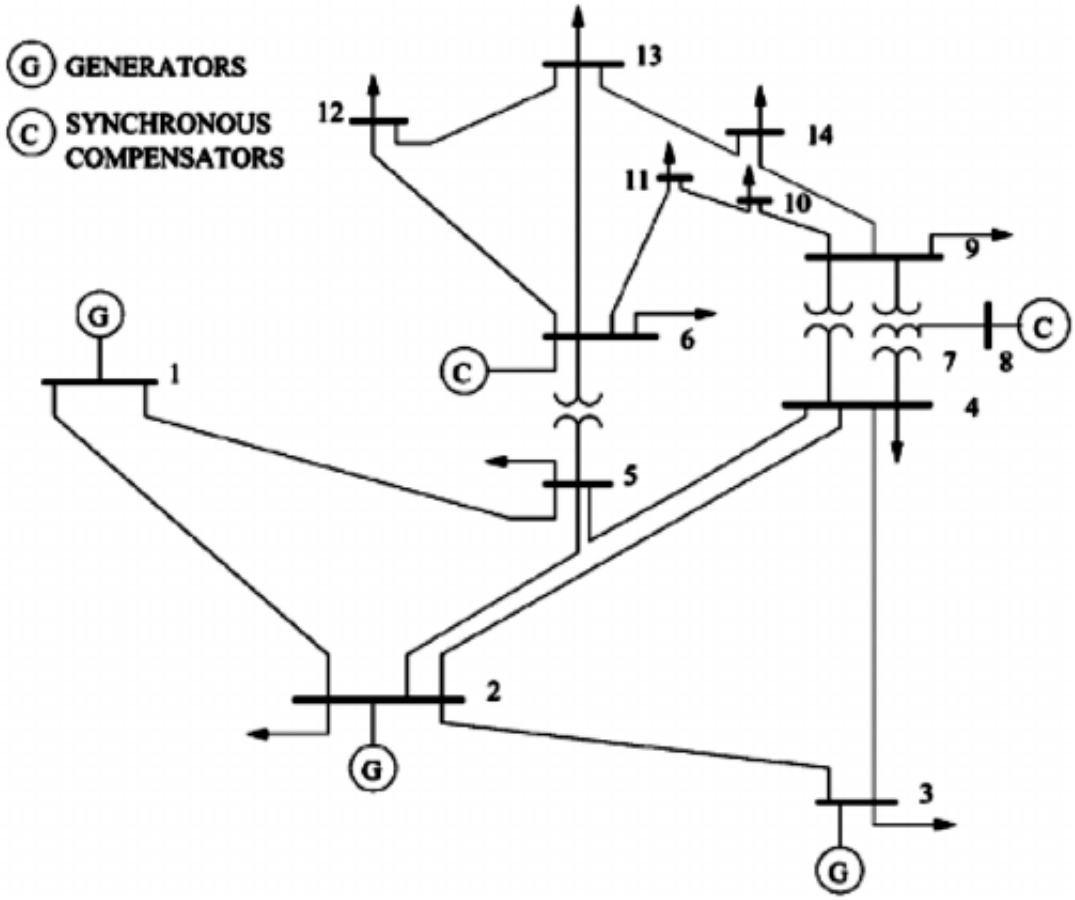
Lab 2: Detecting Bad Sensors in Power System Monitoring

Ke Xu 3190110360

In this lab, our goal is to detect bad sensor data measured on the IEEE 14 bus test system shown below. The power flow equations that couple the voltages and power flows are nonlinear in nature, as discussed in class. We will load the sensor data from the file 'sensorData14Bus.csv', and utilize SVM to perform the bad data detection. We aim to understand how various parameters such as the nature of the corrupt data, the number of corrupt data, etc., affect our abilities to classify the data.



First, we need to call the needed libraries

In [1]:
```
import numpy as np
import os
from sklearn import preprocessing, svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from IPython.display import Image
import pandas as pd
```

Loading the data

Load the sensor data from the IEEE 14 bus test system, that has 14 buses and 20 branches. The data has been generated by adding a small noise to feasible voltages and power flows.

```
    Columns 1-14 contain bus voltage magnitudes.

    Columns 15-28 contain bus voltage phase angles.

    Columns 29-48 contain real power flow on all branches.

    Columns 49-68 contain reactive power flow on all
   branches.
```

In [2]:
```
nBuses = 14
nBranches = 20

# Select the bus numbers you monitor. For convenience, we have selected i
# The '-1' makes them columns as per Python's convention of starting to n
# from 0.
busesToSample = np.array([1, 2, 5, 10, 13]) - 1
columnsForBuses = np.concatenate((busesToSample, busesToSample + 14))

# Select the branches that you monitor.
branchesToSample = np.array([1, 3, 5, 10, 11, 15, 17, 20]) - 1
columnsForBranches = np.concatenate((branchesToSample + 28,
                                     branchesToSample + 48))

# Load the sensor data from the file 'sensorData14Bus.csv' in 'X' from th
# specified in 'columnsForBuses' and 'columnsForBranches'. The csv file i
# separated. Read a maximum of 5000 lines. Make sure your data is a numpy
# with each column typecast as 'np.float32'.
X = np.genfromtxt('sensorData14Bus.csv', dtype=np.float32, delimiter=',',
                  usecols=np.concatenate((columnsForBuses, columnsForBran
                  max_rows=5000)

nDataPoints = np.shape(X)[0]
nFeatures = np.shape(X)[1]

print("Loaded sensor data on IEEE 14 bus system.")
print("Number of data points = %d, number of features = %d"
      % (nDataPoints, nFeatures))
```

```
Loaded sensor data on IEEE 14 bus system.
Number of data points = 5000, number of features = 26
```

Curroption Models

Intentionally corrupt the first 'nCorrupt' rows of the data by adding a quantity to one or two sensor measurements that is not representative of our error model. We aim to study what nature of corruption is easier or difficult to detect. Specifically, we shall study 3 different models:

1. 'corruptionModel' = 1 : Add a random number with a bias to one of the measurements.

2. 'corruptionModel' = 2 : Add a random number without bias to one of the measurements.

3. 'corruptionModel' = 3 : Add a random number with a bias to both the measurements.

In all these cases, we will multiply the sensor data by either a uniform or a normal random number multiplied by 'multiplicationFactor'.

In [3]:
```
# Choose a corruption model.
nCorrupt = int(nDataPoints/3)
corruptionModel = 1
multiplicationFactor = 0.5

# Choose which data to tamper with, that can be a voltage magnitude,
# voltage phase angle, real power flow on a branch, reactive power flow
# on a branch. We create functions to extract the relevant column to
# corrupt the corresponding data in the 'ii'-th bus or branch.
voltageMagnitudeColumn = lambda ii: ii

voltageAngleColumn = lambda ii: ii + np.shape(busesToSample)[0]

realPowerColumn = lambda ii: ii + 2*np.shape(busesToSample)[0]
reactivePowerColumn = lambda ii: ii + 2*np.shape(busesToSample)[0] + np.s

# Encode two different kinds of columns to corrupt.
# Option 1: Corrupt real power columns only.
# Option 2: Corrupt real power and voltage magnitude.
columnsToCorruptOption = 2

if columnsToCorruptOption == 1:
    columnsToCorrupt = [realPowerColumn(1),
                        realPowerColumn(2)]
else:
    columnsToCorrupt = [voltageMagnitudeColumn(0),
                        realPowerColumn(1)]

# Corrupt the data appropriately, given the options.
for index in range(nCorrupt):

    if corruptionModel == 1:
        X[index, columnsToCorrupt[0]] \
            *= (1 + multiplicationFactor * np.random.rand())
    elif corruptionModel == 2:
        X[index, columnsToCorrupt[0]] \
            *= (1 + multiplicationFactor * np.random.randn())
    else:
        X[index, columnsToCorrupt[0]] \
            *= (1 + multiplicationFactor * np.random.rand())
        X[index, columnsToCorrupt[1]] \
            *= (1 + multiplicationFactor * np.random.rand())
```

It is always a good practice to scale your data to run SVM. Notice that we are cheating a little when we scale the entire data set 'X', because our training and test sets are derived from 'X'. Ideally, one would have to scale the training and test sets separately. Create the appropriate labels and shuffle the lists 'X' and 'Y' together.

In [4]:
```
X = preprocessing.StandardScaler().fit_transform(X)

# Create the labels as a column of 1's for the first 'nCorrupt' rows, and
# 0's for the rest.
Y = np.concatenate((np.ones(nCorrupt), np.zeros(nDataPoints-nCorrupt)))


# Shuffle the features and the labels together.
XY = list(zip(X, Y))
np.random.shuffle(XY)
X, Y = zip(*XY)
```

Recall from the first lab that 'test_size' determines what fraction of the data becomes your test set.

Task 1 (10 points)

Split the dataset into two parts: training and testing. Store the training set in the variables 'trainX' and 'trainY'. Store the testing set in the variables 'testX' and 'testY'. Reserve 20% of the data for testing. The function 'train_test_split' may prove useful.

In [5]:
```
trainX, testX, trainY, testY = train_test_split(X, Y, test_size=0.2)

print("Scaled and split the data into two parts!")
```

```
Scaled and split the data into two parts!
```

Task 2 (10 points)

Define the support vector machine classifier and train on the variables 'trainX' and 'trainY'. Use the SVC library from sklearn.svm. Only specify three hyper-parameters: 'kernel', 'degree', and 'max_iter'. Limit the maximum number of iterations to 100000 at the most. Set the kernel to be a linear classifier first. You may have to change it to report the results with other kernels. The parameter 'degree' specifies the degree for polynomial kernels. This parameter is not used for other kernels. The functions 'svm.SVC' and 'fit' will prove useful.

In [6]:
```
classifier = svm.SVC(kernel='linear', degree=1, max_iter=100000)
classifier.fit(trainX, trainY)
```

Out[6]:
```
SVC(degree=1, kernel='linear', max_iter=100000)
```

Task 3 (10 points)

Predict the labels on the 'testX' dataset and store them in 'predictY'.

In [7]:
```
predictY = classifier.predict(testX)
```

Task 4 (10 points)

Print the 'classification_report' to see how well 'predictY' matches with 'testY'.

In [8]:
```
print(classification_report(testY, predictY))
```

```
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.97       665
         1.0       1.00      0.89      0.94       335

    accuracy                           0.96      1000
   macro avg       0.97      0.95      0.96      1000
weighted avg       0.97      0.96      0.96      1000
```

Print svm's internal accuracy score as a percentage.

In [9]:
```
print('classifier.score:',classifier.score(testX, testY))
```

```
classifier.score: 0.964
```

Task 5

We would like to compare 'classification_report' with this score for various runs. Let us consider the following cases:

In [10]:
```
def load_data(nBuses=14, nBranches=20, branches=[1, 3, 5, 10, 11, 15, 17,

    busesToSample = np.array([1, 2, 5, 10, 13]) - 1
    columnsForBuses = np.concatenate((busesToSample, busesToSample + 14))
    branchesToSample = np.array(branches) - 1
    columnsForBranches = np.concatenate((branchesToSample + 28,
                                         branchesToSample + 48))

    X = np.genfromtxt('sensorData14Bus.csv', dtype=np.float32, delimiter=
                      usecols=np.concatenate((columnsForBuses, columnsFor
                      max_rows=5000)

    return X


def add_corruption(X, corruptionModel=1, columnsToCorruptOption=2, multip

    nDataPoints = np.shape(X)[0]
    nFeatures = np.shape(X)[1]
    nCorrupt = int(nDataPoints/3)

    if columnsToCorruptOption == 1:
```

```
        columnsToCorrupt = [realPowerColumn(1),
                            realPowerColumn(2)]
    else:
        columnsToCorrupt = [voltageMagnitudeColumn(0),
                            realPowerColumn(1)]

    for index in range(nCorrupt):

        if corruptionModel == 1:
            X[index, columnsToCorrupt[0]] \
                *= (1 + multiplicationFactor * np.random.rand())
        elif corruptionModel == 2:
            X[index, columnsToCorrupt[0]] \
                *= (1 + multiplicationFactor * np.random.randn())
        else:
            X[index, columnsToCorrupt[0]] \
                *= (1 + multiplicationFactor * np.random.rand())
            X[index, columnsToCorrupt[1]] \
                *= (1 + multiplicationFactor * np.random.rand())
    return X


def preprocess_data(X):

    nDataPoints = np.shape(X)[0]
    nFeatures = np.shape(X)[1]
    nCorrupt = int(nDataPoints/3)

    X = preprocessing.StandardScaler().fit_transform(X)
    Y = np.concatenate((np.ones(nCorrupt), np.zeros(nDataPoints-nCorrupt)
    XY = list(zip(X, Y))
    np.random.shuffle(XY)
    X, Y = zip(*XY)

    return X, Y


def train_and_test(X, Y, kernel='linear', degree=1):

    trainX, testX, trainY, testY = train_test_split(X, Y, test_size=0.2)

    classifier = svm.SVC(kernel=kernel, degree=degree, max_iter=100000)
    classifier.fit(trainX, trainY)
    predictY = classifier.predict(testX)

    report = classification_report(testY, predictY)
    score = classifier.score(testX, testY)

    return report, score
```

Case 1:

Only have sensor measurements from the first 5 branches. Choose option 1 in the 'columnsToCorruptOption'. Examine how well linear kernels perform when 'corruptionModel' = 1, 'corruptionModel' = 2, and 'corruptionModel'= 3. In case linear kernels do not perform well, you may try 'rbf' or polynomial kernels with degree 2.

In [11]:
```
dataFrame = pd.DataFrame(
    {
        "kernel": [],
        "corruptionModel": [],
        "accuracy": [],
        "report": [],
    }
)


corruptionModels = [1, 2, 3]
kernels = [('linear', 0), ('rbf', 0), ('poly', 2)]


for kernel in kernels:
    for corruptionModel in corruptionModels:

        X = load_data(nBranches=5, branches=[1, 2, 3, 4, 5])
        X = add_corruption(X, corruptionModel=corruptionModel)
        X, Y = preprocess_data(X)
        report, score = train_and_test(X, Y, kernel=kernel[0], degree=ker

        print(f"------------kernel={kernel[0]},corruptionModel={corruptio
        print(report)
        print('classifier.score: ', score)

        dataFrame = dataFrame.append(
            {
                "kernel": kernel[0],
                "corruptionModel": corruptionModel,
                "accuracy": score,
                "report": report
            }, ignore_index=True
        )

    print("\n\n\n")

display(dataFrame)
```

```
------------kernel=linear,corruptionModel=1------------
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98       659
         1.0       1.00      0.91      0.95       341

    accuracy                           0.97      1000
   macro avg       0.98      0.95      0.96      1000
weighted avg       0.97      0.97      0.97      1000

classifier.score:  0.969
/var/folders/vh/vy1h_x1105q5pl_wdj84g_140000gn/T/ipykernel_65097/2689376845.py:27: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  dataFrame = dataFrame.append(
/Users/xuke/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:284: ConvergenceWarning: Solver terminated early (max_iter=100000).  Consider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
------------kernel=linear,corruptionModel=2------------
              precision    recall  f1-score   support

         0.0       0.68      1.00      0.81       655
         1.0       1.00      0.09      0.16       345

    accuracy                           0.69      1000
   macro avg       0.84      0.54      0.48      1000
weighted avg       0.79      0.69      0.58      1000

classifier.score:  0.685
------------kernel=linear,corruptionModel=3------------
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00       666
         1.0       1.00      0.98      0.99       334

    accuracy                           0.99      1000
   macro avg       1.00      0.99      0.99      1000
weighted avg       0.99      0.99      0.99      1000

classifier.score:  0.994
/var/folders/vh/vy1h_x1105q5pl_wdj84g_140000gn/T/ipykernel_65097/2689376845.py:27: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  dataFrame = dataFrame.append(
/var/folders/vh/vy1h_x1105q5pl_wdj84g_140000gn/T/ipykernel_65097/2689376845.py:27: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  dataFrame = dataFrame.append(
```

```
------------kernel=rbf,corruptionModel=1------------
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.97       694
         1.0       1.00      0.87      0.93       306

    accuracy                           0.96      1000
   macro avg       0.97      0.93      0.95      1000
weighted avg       0.96      0.96      0.96      1000

classifier.score:  0.96
/var/folders/vh/vy1h_x1105q5pl_wdj84g_140000gn/T/ipykernel_65097/2689376845.py:27: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  dataFrame = dataFrame.append(
------------kernel=rbf,corruptionModel=2------------
              precision    recall  f1-score   support

         0.0       0.81      1.00      0.89       683
         1.0       0.99      0.49      0.66       317

    accuracy                           0.84      1000
   macro avg       0.90      0.74      0.77      1000
weighted avg       0.87      0.84      0.82      1000

classifier.score:  0.837
/var/folders/vh/vy1h_x1105q5pl_wdj84g_140000gn/T/ipykernel_65097/2689376845.py:27: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  dataFrame = dataFrame.append(
------------kernel=rbf,corruptionModel=3------------
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99       690
         1.0       1.00      0.95      0.98       310

    accuracy                           0.99      1000
   macro avg       0.99      0.98      0.98      1000
weighted avg       0.99      0.99      0.99      1000

classifier.score:  0.986
```

```
/var/folders/vh/vy1h_x1105q5pl_wdj84g_140000gn/T/ipykernel_65097/2689376845.py:27: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  dataFrame = dataFrame.append(
```

```
------------kernel=poly,corruptionModel=1------------
              precision    recall  f1-score   support

         0.0       0.79      1.00      0.88       672
         1.0       1.00      0.46      0.63       328

    accuracy                           0.82      1000
   macro avg       0.90      0.73      0.76      1000
weighted avg       0.86      0.82      0.80      1000

classifier.score:  0.822
```

```
/var/folders/vh/vy1h_x1105q5pl_wdj84g_140000gn/T/ipykernel_65097/2689376845.py:27: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  dataFrame = dataFrame.append(
```

```
------------kernel=poly,corruptionModel=2------------
              precision    recall  f1-score   support

         0.0       0.82      1.00      0.90       681
         1.0       1.00      0.53      0.69       319

    accuracy                           0.85      1000
   macro avg       0.91      0.76      0.80      1000
weighted avg       0.88      0.85      0.83      1000

classifier.score:  0.85
```

```
/var/folders/vh/vy1h_x1105q5pl_wdj84g_140000gn/T/ipykernel_65097/2689376845.py:27: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  dataFrame = dataFrame.append(
```

```
------------kernel=poly,corruptionModel=3------------
              precision    recall  f1-score   support

         0.0       0.85      1.00      0.91       653
         1.0       0.99      0.66      0.79       347

    accuracy                           0.88      1000
   macro avg       0.92      0.83      0.85      1000
weighted avg       0.90      0.88      0.87      1000

classifier.score:  0.879
```

```
/var/folders/vh/vy1h_x1105q5pl_wdj84g_140000gn/T/ipykernel_65097/2689376845.py:27: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  dataFrame = dataFrame.append(
```

	kernel	corruptionModel	accuracy	report
0	linear	1.0	0.969	precision recall f1-score ...
1	linear	2.0	0.685	precision recall f1-score ...
2	linear	3.0	0.994	precision recall f1-score ...
3	rbf	1.0	0.960	precision recall f1-score ...
4	rbf	2.0	0.837	precision recall f1-score ...
5	rbf	3.0	0.986	precision recall f1-score ...
6	poly	1.0	0.822	precision recall f1-score ...
7	poly	2.0	0.850	precision recall f1-score ...
8	poly	3.0	0.879	precision recall f1-score ...

Your Answer: My experiment result is shown as above. It is worth noticing that in cases corruptionModel = 1 and 3, 'linear' kernel will achieve the best accracy, while in case corruptionModel = 2, 'poly' kernel achieves the best accracy. Though 'Rbf' kernel never performs best, it has the best average accuracy for the overall 3 corruptionModel.

Case 2:

Choose 'corruptionModel = 1' with 'linear' kernel. Does it pay to monitor voltage magnitudes than power flows? In other words, do you consistently get better results when you choose 'columnsToCorruptOption' as 2? Make these judgements using the average score of at least 5 runs.

Your task is to investigate the above two cases. You may add a few 'Markdown' and 'Code' cells below with your comments, code, and results. You can also report your results as a pandas DataFrame. You are free to report your results in your own way.

In [12]:

```
columnsToCorruptOptions = [1, 2]
experimentNum = 10


dataFrame = pd.DataFrame(
    {
        "kernel": [],
        "corruptionModel": [],
        "columnsToCorruptOption": [],
        "average accuracy": [],
        "report": [],
    }
)

for columnsToCorruptOption in columnsToCorruptOptions:

    scoreSum = 0
    X = load_data()
    X = add_corruption(X, corruptionModel=1, columnsToCorruptOption=colum
    X, Y = preprocess_data(X)

    for i in range(experimentNum):
        report, score = train_and_test(X, Y)
        scoreSum += score
    score = scoreSum / experimentNum

    print(f"------------columnsToCorruptOption={columnsToCorruptOption}--
    print(report)
    print('average accuracy', score)

    dataFrame = dataFrame.append(
        {
            "kernel": 'linear',
            "corruptionModel": 1,
            "columnsToCorruptOption": columnsToCorruptOption,
            "average accuracy": score,
            "report": report
        }, ignore_index=True
    )

display(dataFrame)
```

```
/Users/xuke/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.p
y:284: ConvergenceWarning: Solver terminated early (max_iter=100000).  Co
nsider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
/Users/xuke/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.p
y:284: ConvergenceWarning: Solver terminated early (max_iter=100000).  Co
nsider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
/Users/xuke/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.p
y:284: ConvergenceWarning: Solver terminated early (max_iter=100000).  Co
nsider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
/Users/xuke/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.p
y:284: ConvergenceWarning: Solver terminated early (max_iter=100000).  Co
nsider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
/Users/xuke/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.p
y:284: ConvergenceWarning: Solver terminated early (max_iter=100000).  Co
nsider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
/Users/xuke/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.p
y:284: ConvergenceWarning: Solver terminated early (max_iter=100000).  Co
nsider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
/Users/xuke/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.p
y:284: ConvergenceWarning: Solver terminated early (max_iter=100000).  Co
nsider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
/Users/xuke/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.p
y:284: ConvergenceWarning: Solver terminated early (max_iter=100000).  Co
nsider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
```

```
------------columnsToCorruptOption=1------------
              precision    recall  f1-score   support

         0.0       0.74      0.95      0.83       668
         1.0       0.77      0.32      0.45       332

    accuracy                           0.74      1000
   macro avg       0.75      0.64      0.64      1000
weighted avg       0.75      0.74      0.71      1000

average accuracy 0.7457
```

```
/var/folders/vh/vy1h_x1105q5pl_wdj84g_140000gn/T/ipykernel_65097/37465422
61.py:31: FutureWarning: The frame.append method is deprecated and will b
e removed from pandas in a future version. Use pandas.concat instead.
  dataFrame = dataFrame.append(
```

```
------------columnsToCorruptOption=2------------
              precision    recall  f1-score   support

         0.0       0.97      1.00      0.98       669
         1.0       1.00      0.94      0.97       331

    accuracy                           0.98      1000
   macro avg       0.98      0.97      0.97      1000
weighted avg       0.98      0.98      0.98      1000

average accuracy 0.9773999999999999
/var/folders/vh/vy1h_x1105q5pl_wdj84g_140000gn/T/ipykernel_65097/3746542261.py:31: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  dataFrame = dataFrame.append(
```

	kernel	corruptionModel	columnsToCorruptOption	average accuracy	report
0	linear	1.0	1.0	0.7457	precision recall f1-score ...
1	linear	1.0	2.0	0.9774	precision recall f1-score ...

Your Answer: My experiment result is shown as above. It pays to monitor voltage magnitudes than power flows. The setting of "columnsToCorruptOption = 2" has better average accuracy.