PYTHON CODES
FOR
CLASSICAL MACHINE LEARNING

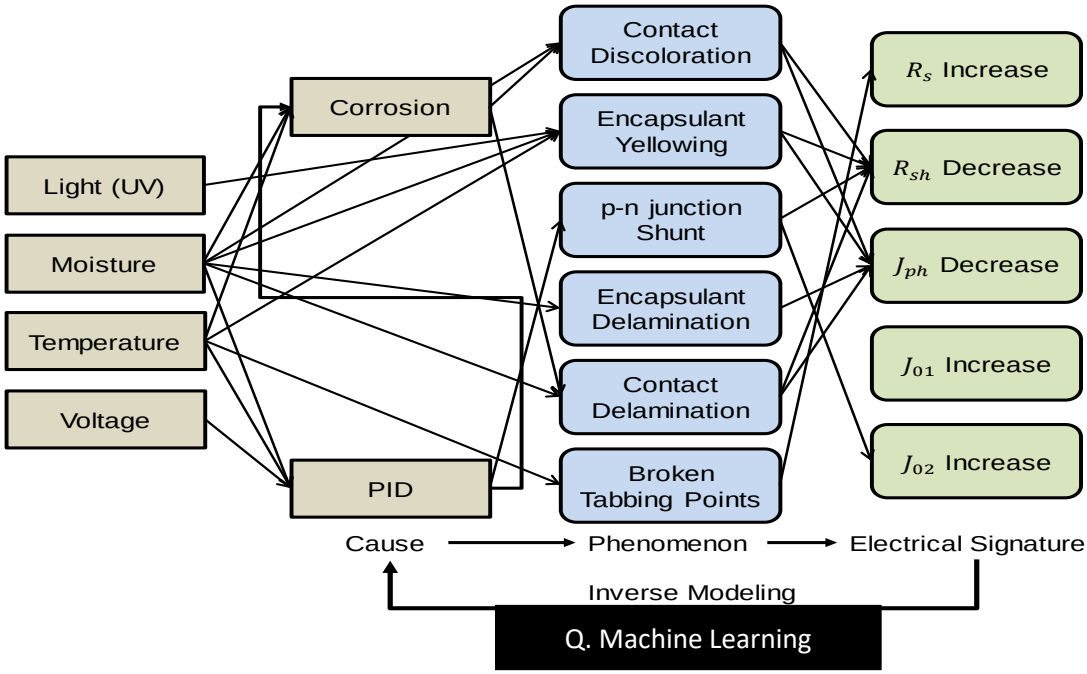
Kumar Ghosh

Classical machine learning for Solar farm project:

In a rapidly-growing solar energy industry, there has been a constant effort to reduce the levelized cost of energy (LCOE) for which the reliability of the PV system has attracted major attentions. The lifetimes and performances of various systems have been measured through field, qualification and accelerated stress tests, yet there is no clear proof of identification and deconvolution of variegated degradation or fault mechanisms until recently. The degradation mechanisms involved are of numerous varieties and high complexity. Moreover, the mechanisms are intertwined, meaning that each mechanism affects more than one feature or parameter of the field data. This poses a major issue in analyzing the faults/degradation mechanisms, deriving analytical forms and physical models and further deconvolve the mechanisms. With the advent of machine learning techniques, we realize that our problem is a classic case of pattern recognition that can be solved using these techniques. Hence, we solve a simplified version of the complex problem of identification and deconvolution of degradation mechanisms in PV systems using machine learning to gauge the viability of our approach. Figure below describes the workflow of our methodology. A real-world problem involves field data and weather data from a wide variety of farms across the globe. This will help make important decisions regarding deployment of solar farms at a particular geographical location based on economic and environmental viability. However, this will require faster machine learning algorithms, since the data is large and there are several physical and environmental parameters affecting various degradation mechanisms simultaneously.

We are collaboating with a solar farm project, where we predict the efficiency degradation of a solar cell depending on few parameters for eg. contact corrosion, potential induced degradation (PID) etc. If the efficiency of a solar cell is less than a threshold (75%), then it is treated as a bad PV cell. This problem can be interprited as a classifier problem where the whole data (fianl output) can be divided into two classes namely: good and bad solar cell.

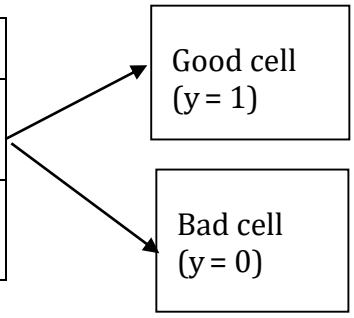
A rough outline of this project is desribed by the following diagram:



In the above diagram we show different causes' path to impact on performance and electrical signature, for e.g. corrosion, Potential Induced Degradation (PID) etc and how it effects the output variables for e.g. current and resistance.

A schematic table of the data structure (from the above diagram) is described in the following, where we have three main independent parameters (Contact Corrosion (x_1), Potential Induced Degradation (PID) (x_2), Encapsulant Yellowing (x_3)) and output dependent variable (y) is efficiency of the solar cell. Although efficiency is a cntinuous variable but we can classify these variables into two classes, namely good class, y= 1 (efficiency is above the threshold), and bad class, y= 0 (efficiency is below the threshold)

R_S Series resistance (x_1)	Contact Corrosion (x_1)
R_{sh} Shunt Resistance (x_2)	Potential Induced Degradation (PID) (x_2)
J_{ph} Photo-current (x_3)	Encapsulant Yellowing (x_3)



We first split our data into training (60%) and test (40%) sets. We use different supervised machine learning algorithm for e.g. Logistic regression, k-nearest neighbours, Random Forest, Support Vector Machine algorithms to classify the data. In the below we summarize the best results got from different machine learning algorithms.

ML Algorithm	Highest testing accuracy achieved
Random Forest	99 %
Support Vector Machine	95 %
k-nearest neighbors	98%
Logistic regression	95 %

Although the overall testing accuracy is comparable for all the algorithms above, but Random forest gives the best result for individual class accuracy.

In the next page I am attaching python codes for data analysis and machine learning for the solar farm data:

Data analysis part

Importing the packages

```
import numpy as np
from sklearn import datasets, svm
from __future__ import division, print_function
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import pandas as pd
import seaborn as sns
from time import time
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
```

```
df = pd.read_excel('test2.xlsx')
df. columns
```

```
Index(['Rs (Corrosion)', 'Jph (Yellowing)', 'Rsh (PID)', 'Eff_loss (25%)',
       'Class', 'Column number'],
      dtype='object')
```

```
df['Class'].value_counts()
```

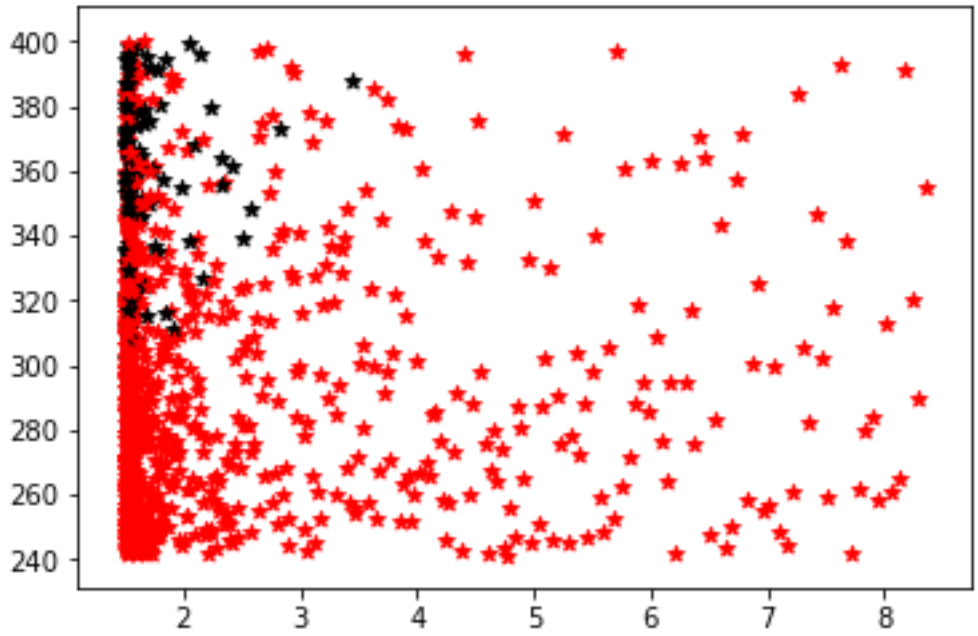
```
0    902
1     98
Name: Class, dtype: int64
```

```
x_data=df[['Rs (Corrosion)', 'Jph (Yellowing)', 'Rsh (PID)']]
y_data=df['Class']
```

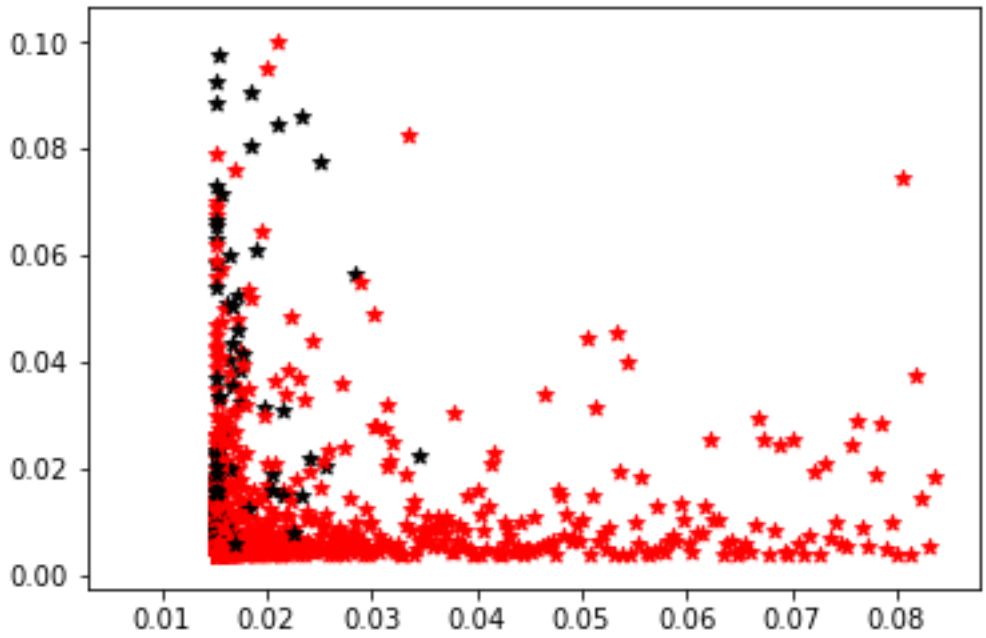
```
yx= x_data.iloc[0,0] #iloc[row,column]# or loc[r,c] also works
```

Plotting the different features of the data

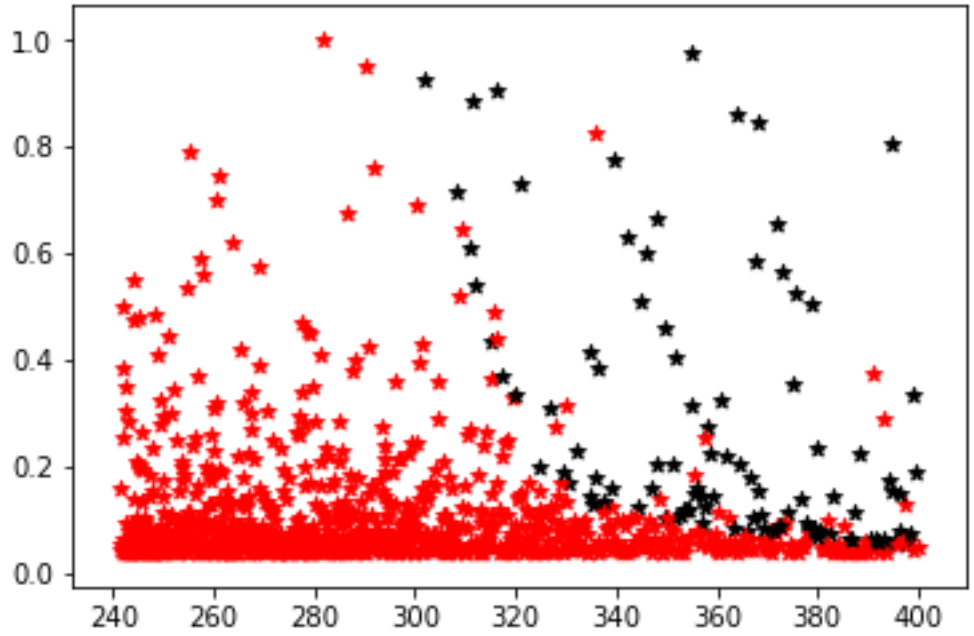
```
for i in range(0, len(y_data)):
    if y[i] ==1:
        #print(x_data.iloc[i,1])
        #print(x[i])
        plt.scatter(x_data.iloc[i,0]*10000, x_data.iloc[i,1], color='k', m
arker='*')
    else:
        plt.scatter(x_data.iloc[i,0]*10000, x_data.iloc[i,1], color='r', m
arker='*')
```




```
for i in range(0, len(y_data)):
    if y[i] ==1:
        #print(x_data.iloc[i,1])
        #print(x[i])
        plt.scatter(x_data.iloc[i,0]*100, x_data.iloc[i,2], color='k', mar
ker='*')
    else:
        plt.scatter(x_data.iloc[i,0]*100, x_data.iloc[i,2], color='r', mar
ker='*')
```



```
for i in range(0, len(y_data)):
    if y[i] ==1:
        #print(x_data.iloc[i,1])
        #print(x[i])
        plt.scatter(x_data.iloc[i,1], x_data.iloc[i,2]*10, color='k', mark
er='*')
    else:
        plt.scatter(x_data.iloc[i,1], x_data.iloc[i,2]*10, color='r', mark
er='*')
```



Machine learnin part

K-Nearest-Neighbors Classifier (with varying parameter k=1 to 20)

Importing the packages

```
import numpy as np
from sklearn import datasets, svm
from __future__ import division, print_function
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import pandas as pd
import seaborn as sns
```

```
from time import time
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report

df = pd.read_excel('test2.xlsx')
df. columns
```

```
Index(['Rs (Corrosion)', 'Jph (Yellowing)', 'Rsh (PID)', 'Eff_loss (20%)',
       'Class', 'Column number'],
      dtype='object')
```

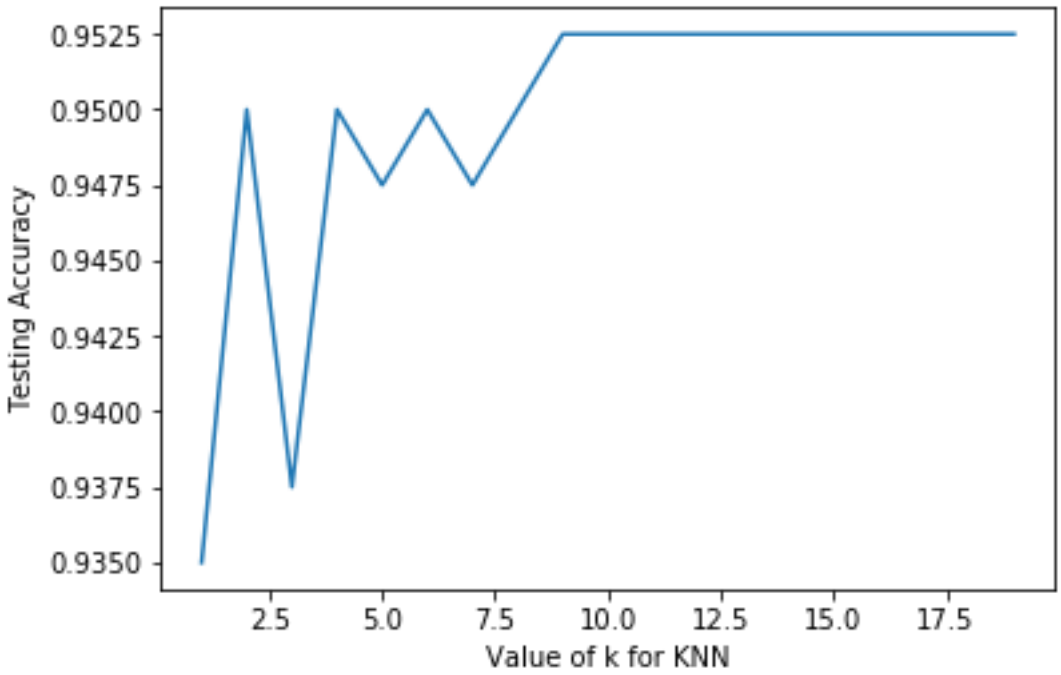
```
x_data=df[['Rs (Corrosion)', 'Jph (Yellowing)', 'Rsh (PID)']]
y_data=df['Class']


# split  x and y into training and testing sets


from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_data,y_data,test_siz
e=0.4, random_state=4)

#KNN with N=3
from  sklearn.neighbors import KNeighborsClassifier
knn= KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train,y_train)

y_pred = knn.predict(x_test)

print(metrics.accuracy_score(y_test,y_pred))
```

```
0.9375
```

```
# KNN with N= 1 to 20

k_range = list(range(1,20))
scores=[]
for i in k_range:
    knn= KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    y_pred = knn.predict(x_test)
    print(metrics.accuracy_score(y_test,y_pred))
    scores.append(metrics.accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred,  digits=4))

print(scores)
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(k_range, scores)
plt.xlabel('Value of k for KNN')
plt.ylabel('Testing Accuracy')
```

```
[0.935, 0.95, 0.9375, 0.95, 0.9475, 0.95, 0.9475, 0.95, 0.9525, 0.9525,
```

0.9525, 0.9525, 0.9525, 0.9525, 0.9525, 0.9525, 0.9525, 0.9525, 0.9525]



Support Vector Machine Classifier (with linear and rbf kernel)

```
# support vector classifier linear kernel


from sklearn.svm import SVC # "Support Vector Classifier"
svmclf = SVC(kernel='linear')

# fitting x samples and y classes
svmclf.fit(x_train,y_train)
#svmclf.predict([[387.09, 0.011]])
#x3_new = [[0.00015, 387.09, 0.011], [0.00015, 312.95, 0.0117]]
svmclf.predict(x_test)
print(metrics.accuracy_score(y_test,y_pred))
```

```
0.9525
```

```
# support vector classifier rbf kernel


from sklearn.svm import SVC # "Support Vector Classifier"
svmclf = SVC(kernel='rbf', gamma=.7)

# fitting x samples and y classes
svmclf.fit(x_train,y_train)
#svmclf.predict([[387.09, 0.011]])
#x3_new = [[0.00015, 387.09, 0.011], [0.00015, 312.95, 0.0117]]
```

```
svmclf.predict(x_test)
print(metrics.accuracy_score(y_test,y_pred))
```

```
0.9525
```

Random Forest Classifier (varying different parameters):

```
df = pd.read_excel('test2.xlsx')
df. columns
```

```
Index(['Rs (Corrosion)', 'Jph (Yellowing)', 'Rsh (PID)', 'Eff_loss (20%)',
       'Class', 'Column number'],
      dtype='object')
```

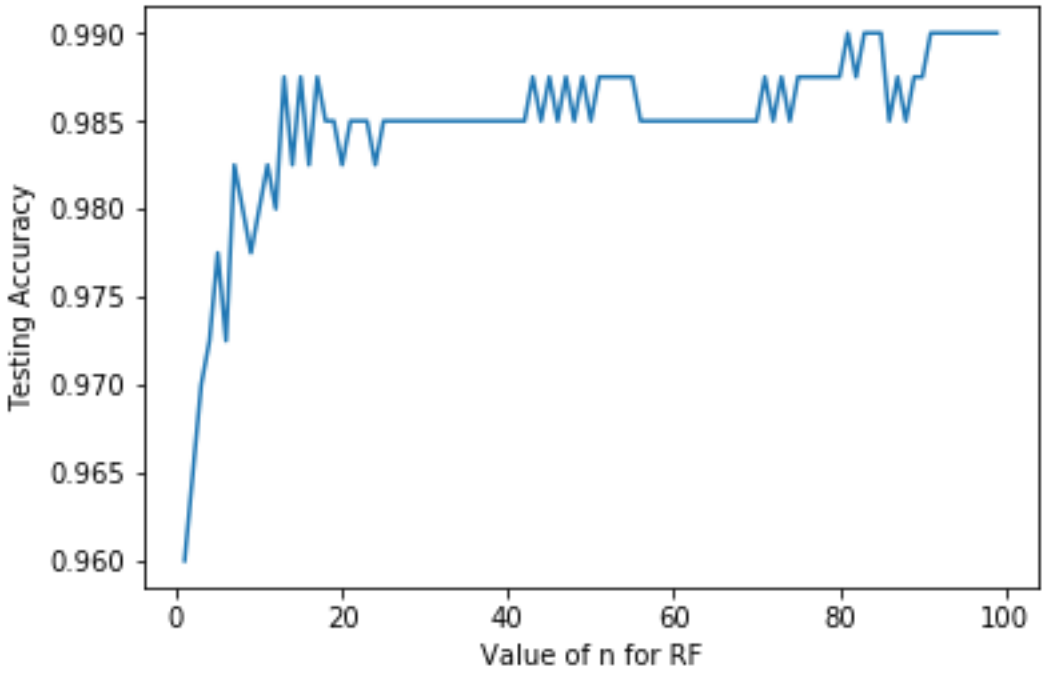
```
#df

x_data=df[['Rs (Corrosion)', 'Jph (Yellowing)', 'Rsh (PID)']]
y_data=df['Class']

from sklearn.ensemble import RandomForestClassifier
import pandas as pd

#creating a random forest classifier
clf = RandomForestClassifier(n_jobs=2, random_state=0)
#training classifier
clf.fit(x_data,y_data)


#step 1: split x and y into training and testing sets


from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_data,y_data,test_siz
e=0.4, random_state=4)

#creating a random forest classifier
clf = RandomForestClassifier(n_jobs=2, random_state=0, n_estimators=10)


#training and testing the classifier


clf.fit(x_train,y_train)

y_pred = clf.predict(x_test)
print(metrics.accuracy_score(y_test,y_pred))
print(classification_report(y_test, y_pred,  digits=4))
```

```
0.98
              precision    recall f1-score   support

           0     0.9895    0.9895    0.9895       381
           1     0.7895    0.7895    0.7895        19

   micro avg     0.9800    0.9800    0.9800       400
```

```
   macro avg     0.8895    0.8895    0.8895       400
weighted avg     0.9800    0.9800    0.9800       400
```

```
n_range = list(range(1,100))
scores=[]
for i in n_range:
    clf = RandomForestClassifier(n_jobs=2, random_state=0, n_estimators= i
)
    clf.fit(x_train,y_train)
    y_pred = clf.predict(x_test)
    #print(metrics.accuracy_score(y_test,y_pred))
    scores.append(metrics.accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred,  digits=4))

print(scores)
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(n_range, scores)
plt.xlabel('Value of n for RF')
plt.ylabel('Testing Accuracy')
```

```
[0.96, 0.965, 0.97, 0.9725, 0.9775, 0.9725, 0.9825, 0.98, 0.9775, 0.98, 0.
9825, 0.98, 0.9875, 0.9825, 0.9875, 0.9825, 0.9875, 0.985, 0.985, 0.9825,
0.985, 0.985, 0.985, 0.9825, 0.985, 0.985, 0.985, 0.985, 0.985, 0.985, 0.9
85, 0.985, 0.985, 0.985, 0.985, 0.985, 0.985, 0.985, 0.985, 0.985, 0.985,
0.985, 0.9875, 0.985, 0.9875, 0.985, 0.9875, 0.985, 0.9875, 0.985, 0.9875,
0.9875, 0.9875, 0.9875, 0.9875, 0.985, 0.985, 0.985, 0.985, 0.985, 0.985,
0.985, 0.985, 0.985, 0.985, 0.985, 0.985, 0.985, 0.985, 0.985, 0.9875, 0.9
85, 0.9875, 0.985, 0.9875, 0.9875, 0.9875, 0.9875, 0.9875, 0.9875, 0.99, 0
.9875, 0.99, 0.99, 0.99, 0.985, 0.9875, 0.985, 0.9875, 0.9875, 0.99, 0.99,
0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99]
```



Logistic Regression Classifier

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression() #using default paameters
logreg.fit(x_train,y_train)
```

```
y_pred = logreg.predict(x_test)
print(metrics.accuracy_score(y_test,y_pred))
```

```
0.955
```

multilayer perceptron (MLP) Classifier

```
from sklearn.linear_model import MLPClassifier
```

```
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(5, 2), random_state=1)
```

```
clf.fit(x_train,y_train)
```

```
y_pred = clf.predict(x_test)
```

```
print(metrics.accuracy_score(y_test,y_pred))
```

```
0.9525
```