**Bank Note Authentication**

**Pre-processing:**

1. data information:

RangeIndex: 1372 entries, 0 to 1371

Data columns (total 5 columns):

Variance 1372 non-null float64

Skewness 1372 non-null float64

Cutosis 1372 non-null float64

Entropy 1372 non-null float64

Class 1372 non-null int64

dtypes: float64(4), int64(1)

memory usage: 53.7 KB

2.Number of features: 5

3.The features are: 'Variance', 'Skewness', 'Cutosis', 'Entropy', 'Class'

4. labels (classes) of the dataset:

Variance:[ 3.6216 4.5459 3.866 ... -3.7503 -3.5637 -2.5419]

Skewness: [ 8.6661 8.1674 -2.6383 ... -13.4586 -8.3827 -0.65804]

Cutosis: [-2.8073 -2.4586 1.9242 ... 17.5932 12.393 2.6842]

Entropy: [-0.44699 -1.4621 0.10645 ... -1.2953 -0.55949 -2.7771 ]

Class: [0 1]

Standardize the dataset along x axis. Center to the mean and component wise scale to unit variance[0,1]

**Training the model:**

Case-1: using 20% (0.2) of data for testing

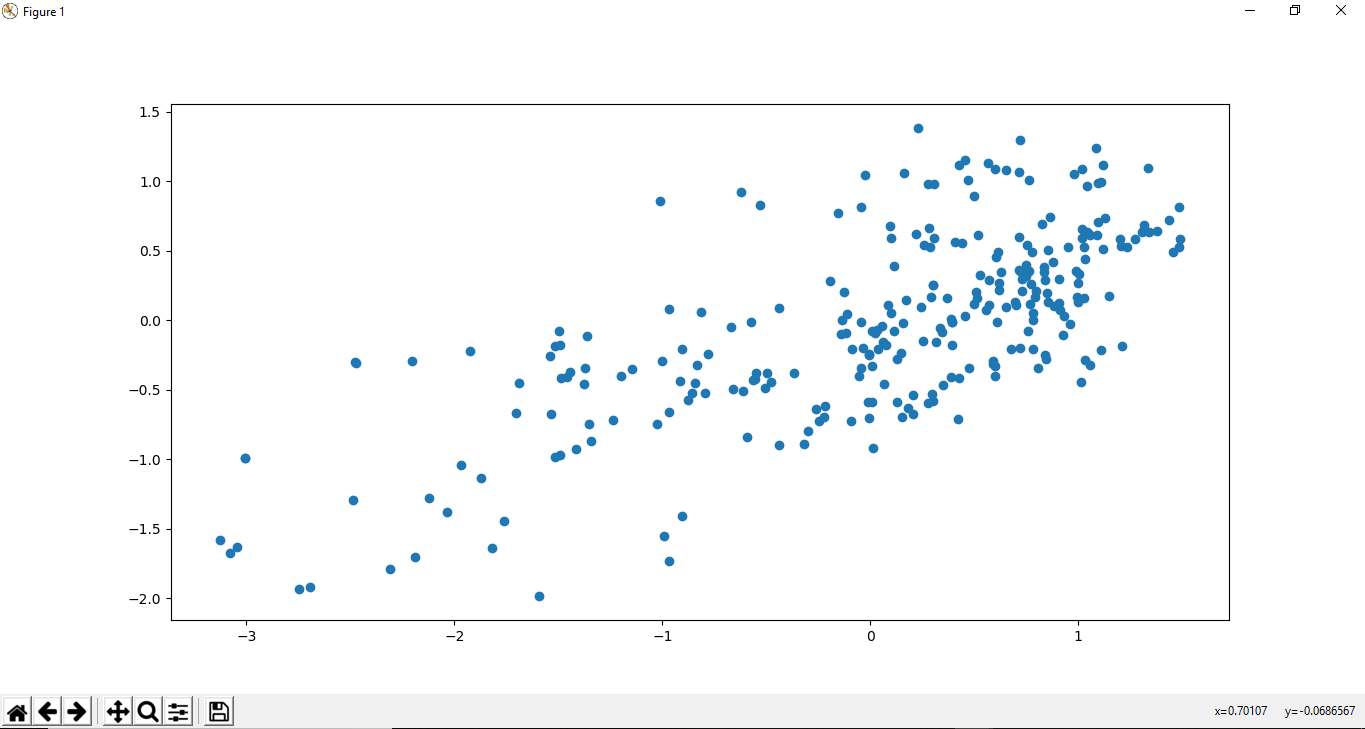


Figure01: Training classifier

Case-2: using 35% (0.35) of data for testing

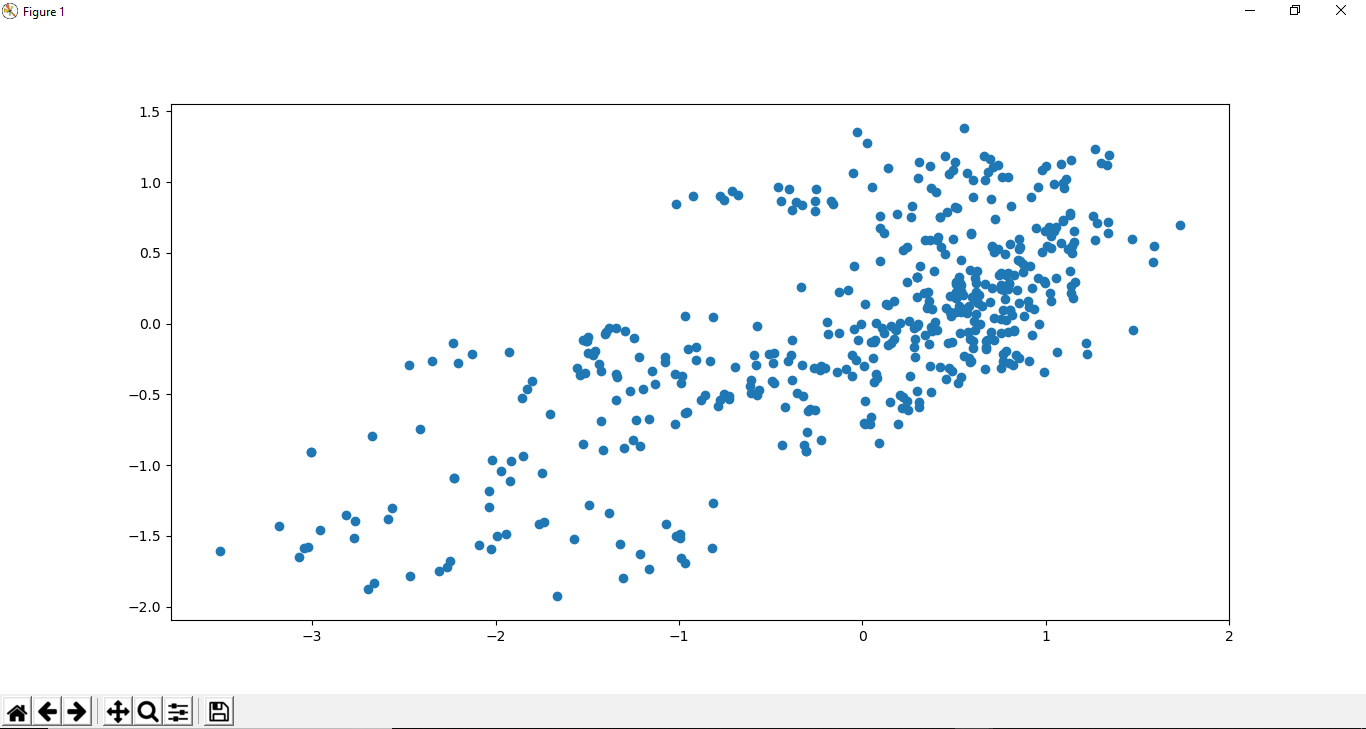
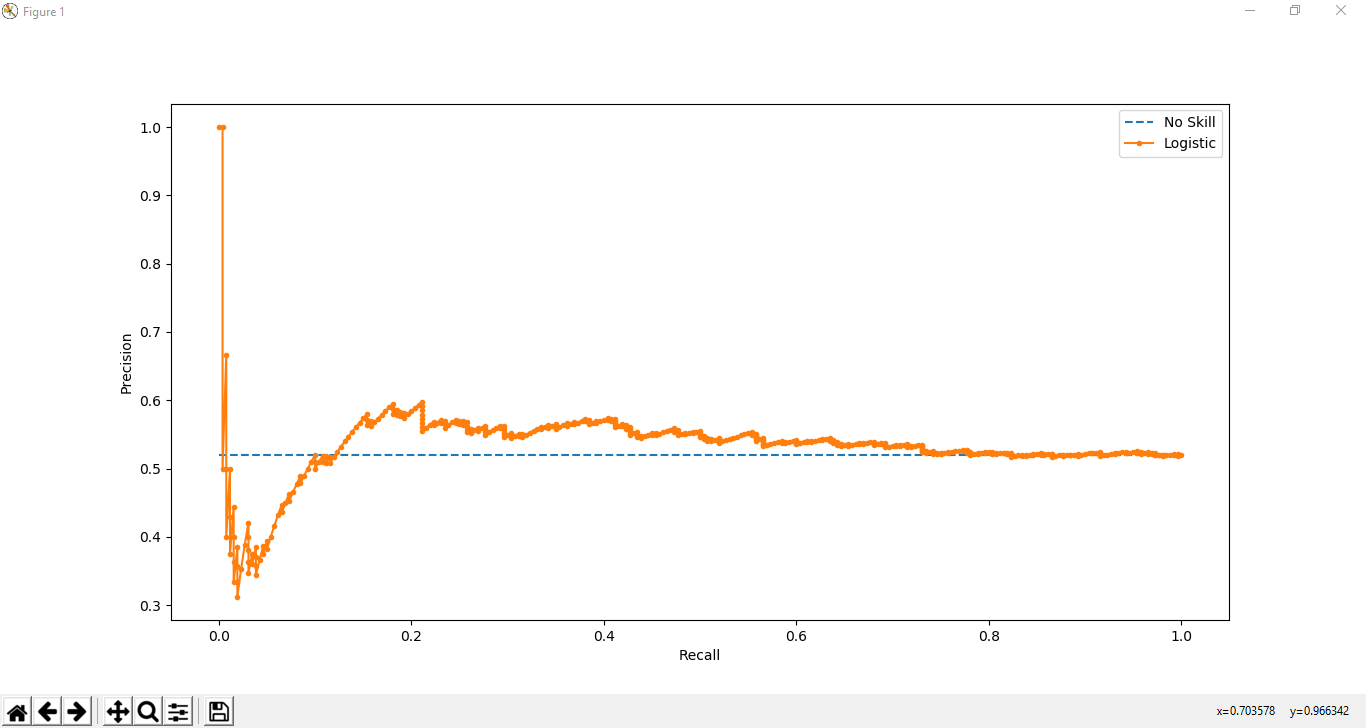


Figure02: Training classifier

**Test accuracy for Sklearn algorithm(Using Precision and Recalls)**



In case of KNN, model works well when the input dimensions are small, as the input dimension increases the performance of KNN decreases, because increase in dimension weakens the most important assumption on which KNN is built, which is that closer points belongs to same class.

1. **Discussion:**

In this project we use the data set named as: data\_banknote\_authentication.csv. We load the data using pandas library. Additionally, you will require to use the following packages.

• numpy

• matplotlib

• knn\_numpy and so on.

We found the dataset in bellowed link:

https://archive.ics.uci.edu/ml/datasets/banknote+authentication

<https://drive.google.com/file/d/18m_lBOVp37BOH6osFmux6VPS1S_WfXoX/view?usp=sharing>

First we get the features and labels (classes) of the dataset. Then Using train\_test\_split method from sklearn we split data into 2 parts. One used for training a classifier. Another to test how generalized is our classifier.Case-1: use 20% (0.2) of data for testing and Case-2: use 35% (0.35) of data for testing. We scaled data between 0 and 1), for better performance/prediction

We predict data using linear Regression and also Predict using Sklearn KNN. Using two more params called Precision and Recalls we test accuracy for Sklearn algorithm. Precision and Recalls. Finally we generate necessary graphs for visualizations using seaborn and matplotlib packages and add them in this report.

1. **Appendix**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

from sklearn.preprocessing import scale

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import precision\_recall\_curve

from matplotlib import pyplot

from sklearn.linear\_model import LogisticRegression

#import seaborn as sns

df = pd.read\_csv("E:/Python/data\_banknote\_authentication.csv")

print ("Number of features : {}".format(len(df.columns.values)))

print(df.columns.values)

print(df.info())

print('Variance\n',df['Variance'].unique())#To see all labels write: print('Variance\n',set(df.Variance))

print('Skewness\n',df['Skewness'].unique())#To see all labels write: print('Skewness\n',set(df.Skewness))

print('Cutosis\n',df['Cutosis'].unique())#To see all labels write: print('Cutosis\n',set(df.Cutosis))

print('Entropy\n',df['Entropy'].unique())#To see all labels write: print('Entropy\n',set(df.Entropy))

print('Class\n',df['Class'].unique())#To see all labels write: print('Class\n',set(df.Class))

df['Class'] = df['Class'].astype(float)

X = df.drop('Entropy',axis=1)

y = df.Entropy

X=scale(X, axis=0, with\_mean=True, with\_std=True, copy=True)

y=scale(y, axis=0, with\_mean=True, with\_std=True, copy=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

lm = LinearRegression()

lm.fit(X\_train,y\_train)

predictions = lm.predict(X\_test)

plt.scatter(y\_test,predictions)

plt.show()

# fit a model

model = LogisticRegression(solver='lbfgs')

model.fit(X\_train,y\_train)

# predict probabilities

lr\_probs = model.predict\_proba(X\_train)

# keep probabilities for the positive outcome only

lr\_probs = lr\_probs[:, 1]

# predict class values

yhat = model.predict(X\_test)

lr\_precision, lr\_recall, \_ = precision\_recall\_curve(y\_test, lr\_probs)

# plot the precision-recall curves

no\_skill = len(y\_test[y\_test==1]) / len(y\_test)

pyplot.plot([0, 1], [no\_skill, no\_skill], linestyle='--', label='No Skill')

pyplot.plot(lr\_recall, lr\_precision, marker='.', label='Logistic')

# axis labels

pyplot.xlabel('Recall')

pyplot.ylabel('Precision')

# show the legend

pyplot.legend()

# show the plot

pyplot.show()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.35)

lm = LinearRegression()

lm.fit(X\_train,y\_train)

predictions = lm.predict(X\_test)

plt.scatter(y\_test,predictions)

plt.show()

# fit a model

model = LogisticRegression(solver='lbfgs')

model.fit(X\_train,y\_train)

# predict probabilities

lr\_probs = model.predict\_proba(X\_train)

# keep probabilities for the positive outcome only

lr\_probs = lr\_probs[:, 1]

# predict class values

yhat = model.predict(X\_test)

lr\_precision, lr\_recall, \_ = precision\_recall\_curve(y\_test, lr\_probs)

# plot the precision-recall curves

no\_skill = len(y\_test[y\_test==1]) / len(y\_test)

pyplot.plot([0, 1], [no\_skill, no\_skill], linestyle='--', label='No Skill')

pyplot.plot(lr\_recall, lr\_precision, marker='.', label='Logistic')

# axis labels

pyplot.xlabel('Recall')

pyplot.ylabel('Precision')

# show the legend

pyplot.legend()

# show the plot

pyplot.show()