**Machine Learning Week 4 Assignment Report**

Given Dataset: # id:15--15--15-0

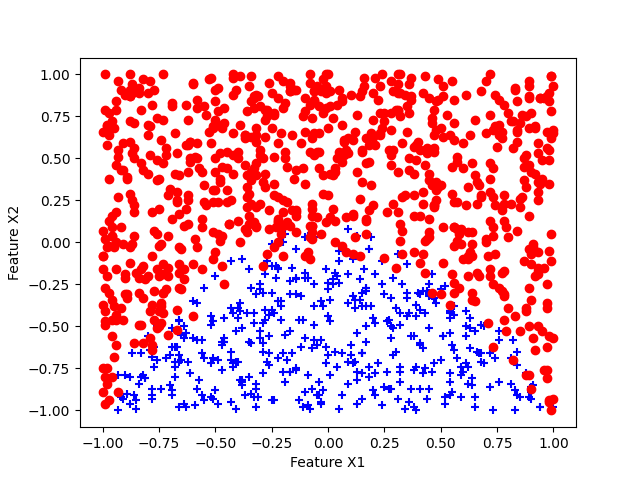
* **Logistic Regression Model**

Logistic Regression is a classification model which classifies an event in one of two outcomes where we take input features. It’s a simple model which is widely used for binary classification. For example, predict if something is true or false, or if a person is diabetic or non-diabetic, or if a email is spam or not.

The given dataset includes 2 input/feature parameters and 1 output or label parameter. Let’s consider, X**1** and **X2** as the input parameters and **Y** as the output parameter. As it is a substantial number of data, we use scatter plot to visualize the data from the dataset to understand the input parameters (**X1**, **X2**) with respect to output parameter **Y**.

* Data-Visualization

To understand the data, we need to visualize it through some graphs or plot. We use “**Matplotlib**” as a library to plot the input and output data. Here, we use scatter plot as a visualization graph. The x-axis represents the x input parameter and y-axis of the plot represents the y input parameter.



Before we train the model, We decide a range of C values and range of polynomial degrees.

Here we consider C Values in range **(0.001, 0.01, 0.1,1, 10, 100)** and degrees in range (2,3,4,5)

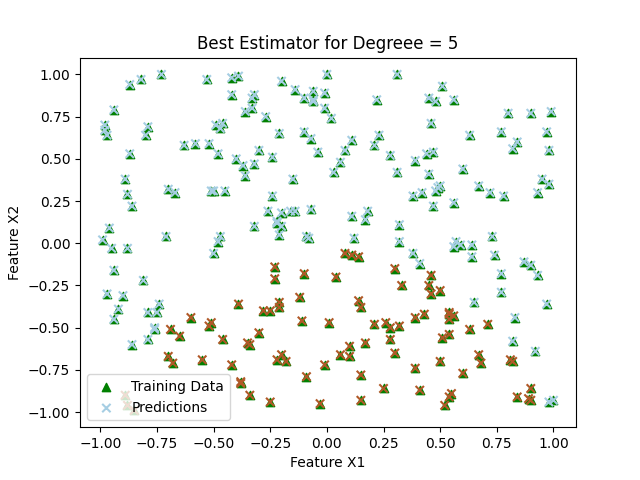
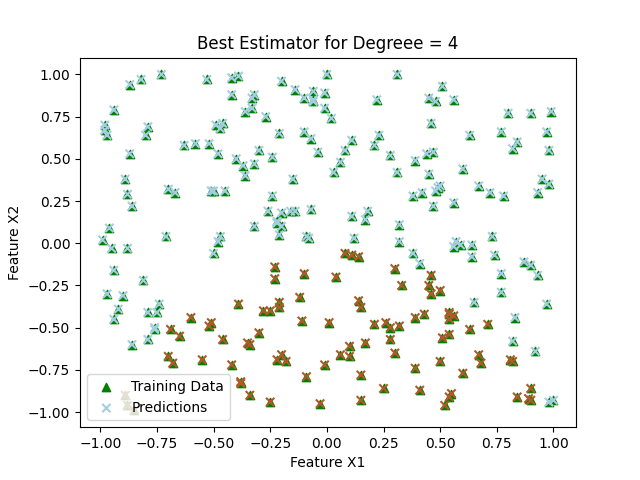
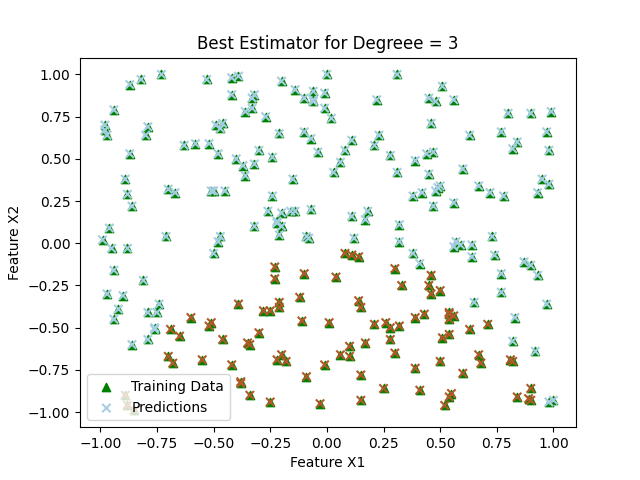
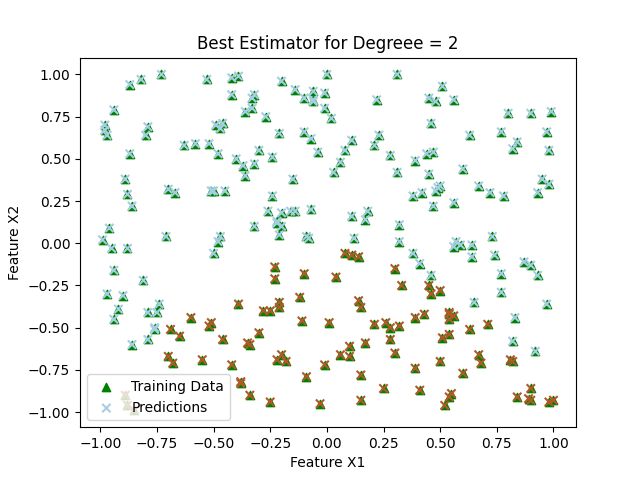
Here, we use cross-validation technique to determine which model, and which subset of data when trained will give us best performing model for a particular dataset.

So we use, K-fold cross-validation techniques for various degrees of polynomials and C values.

**Cross-validation:**

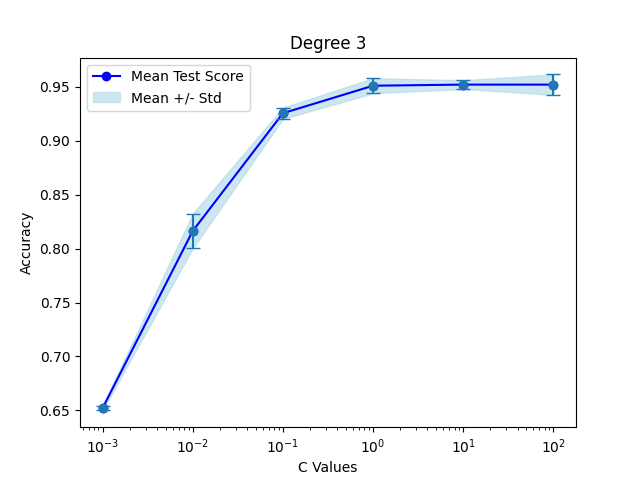
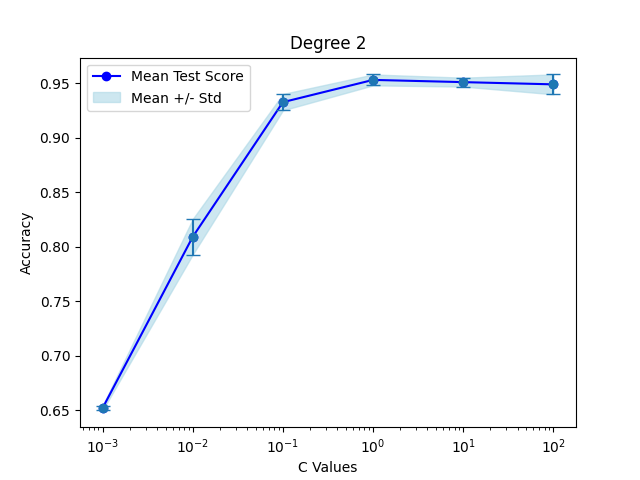
Cross-validation is a technique where we divide the training data into sub-sets and repeatedly train and test the model on different subsets. The most common type of cross validataion is k-fold. In k-fold validations, we divide the dataset into k number of folds equally. The common choices of k being 5 or 10. Which can also depend on the size of the data. And iterate over to subsets.

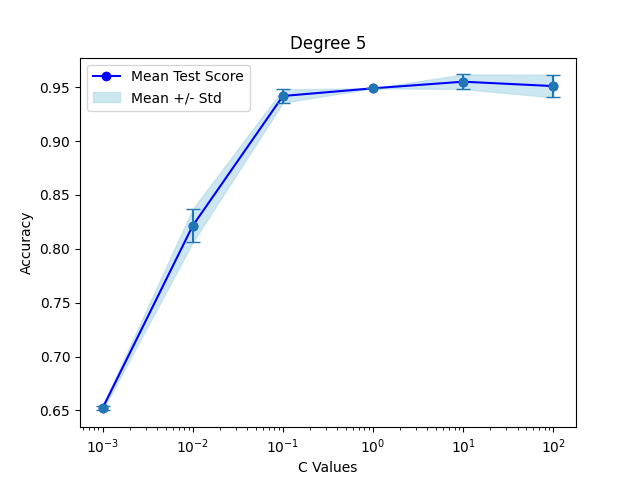
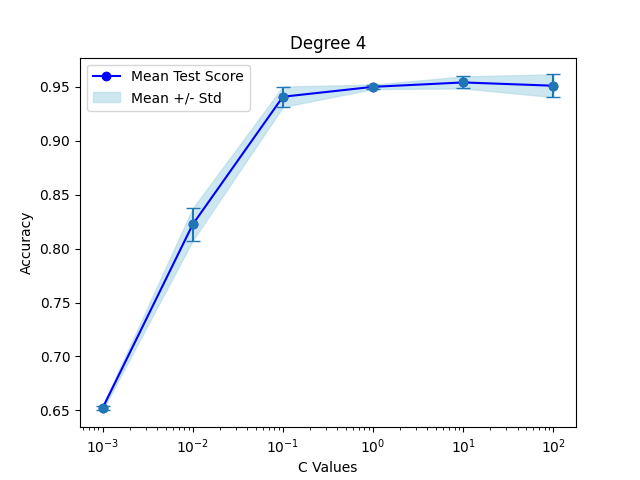
For each of k iteration, one fold is used as a validation set and k-1 folds are used as a training data. After training it on k-1 folds, evaluate the model on validation set. Here we calculate mean performance metrics like accuracy for each such iteration. Calculate the mean of all the performance metric for each iteration. This will give the overall models performance.

The performance values for each iteration can give us the best suited subsets to evaluate the model for each value of C. We can map the mean error and standard deviation between the performance metric for all values of C and find the optimal value of C to be used. Here are the results of logistic regression and its predictions on test data 

**Perform StratifiedKFold Cross-validation:**

1. Choose a Range of C Values: To determine the range of C values to plot, it's essential to consider a broad range that covers potential values for your specific problem. You might start with a wide range, such as 0.001, 0.01, 0.1, 1, 10, 100, 1000, and so on. However, you can adapt this range based on prior knowledge about your data or through experimentation. The goal is to see how the model's performance varies with different levels of regularization.
2. Perform 5-Fold Cross-Validation: Use your dataset to perform 5-fold cross-validation for each value of C. In each fold, you'll train the model on 4/5 of the data and validate it on the remaining 1/5. Repeat this process for all values of C, calculating the prediction error (e.g., mean squared error) for each fold.
3. Calculate Mean and Standard Deviation: After running cross-validation for all values of C, calculate the mean and standard deviation of the prediction error across the five folds for each C value. This will give you an idea of the model's performance variability.
4. Plot the Results: Use the matplotlib errorbar function to create a plot. Plot C values on the x-axis and the mean prediction error on the y-axis. Use the standard deviation as the error bars to show the variability. Your plot will help visualize how prediction error changes with different C values.

We performed cross validation for C values and over a range of polynomial degrees. Here are results of cross-validation plot between C values and the mean accuracy



Justification of choices:

* We chose range of C values **(0.001, 0.01, 0.1,1, 10, 100)** because for the minimum value of C, that is 0.001, the model tends to underfit and reduces the accuracy.
* For upper range, we chose 100 as the highest values because for the values above the 100, the model goes in to overfitting the training dataset. It tries to fit each datapoint and makes classification errors.
* After performing cross-validation for a range of polynomial degrees, (2,3,4,5), we select degree 5 as the upper limit as the model does not maximize the accuracy. The accuracy becomes stagnent.

Observations:

Best Score = 0.9404761904761905

Best C value = 100

Best degree = 5

We use GridSearchCV() funtion with the cross-validator to iterate over the C values and get the best\_score , best\_estimator and best\_parameters of the model. We use this to compare with other models.

**KNN Classifier:**

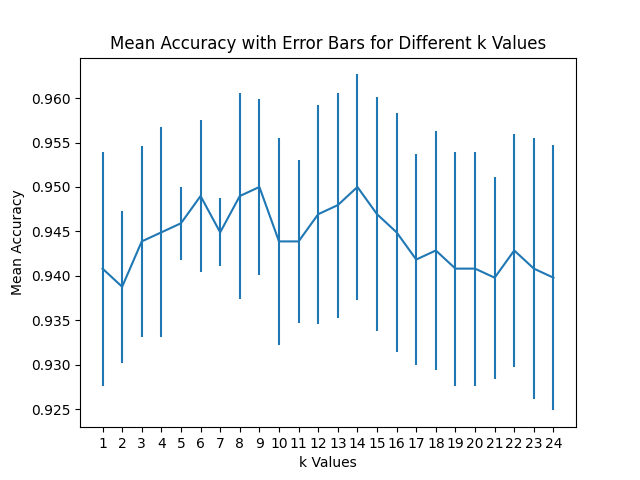
The k-Nearest Neighbors (k-NN) classifier is a simple and intuitive machine learning algorithm used for both classification and regression tasks. It is a non-parametric and instance-based algorithm, meaning it doesn't make any underlying assumptions about the data distribution and makes predictions based on the similarity between data points.

Here, we range over k values of a list (1 to 24) and cross-validate. By using GridSearchCV() we get the best\_estimator and optimal value of k.

We plot the cross-validation plot, to see the how the accuracy varies for different values of k.

So we plot accuracy with k values.

Here we do not use polynomial features as kNN can already capture nonlinear decision boundaries.



Here we can clearly see, for K = 9, the graph is highest accuracy. So, we chose k as 9 and the model as the best model.

Justification of choices:

* Here we choose a wide range of k values from 1 to 24 to better understand how the model behaves and which value of k would be optimal.
* The range chosen here is arbitrary range using trial and error method.
* For this range we perform cross-validation to get the optimal k values for which the performance of model is higher than the other model.

Observation :

* Optimal K value – 9

**Confusion Matrix:**

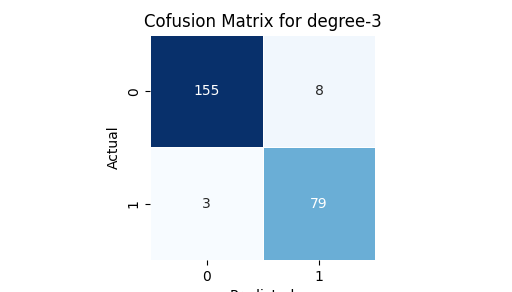
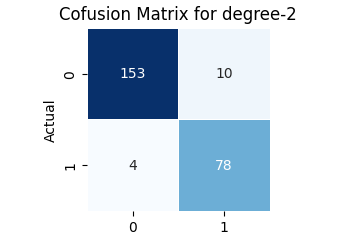
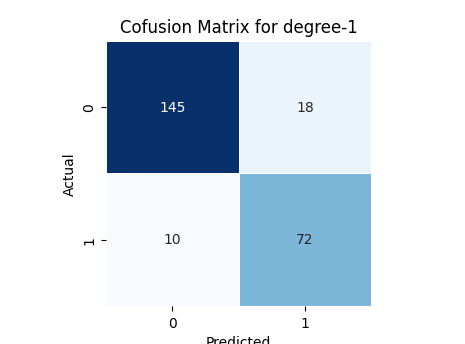
**Logistic Regression:**

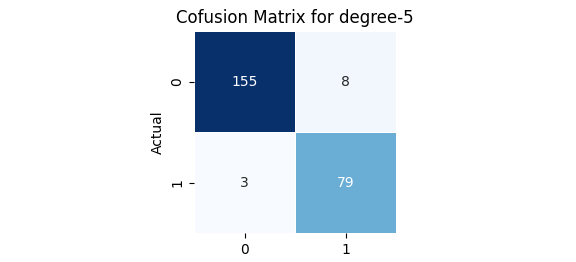
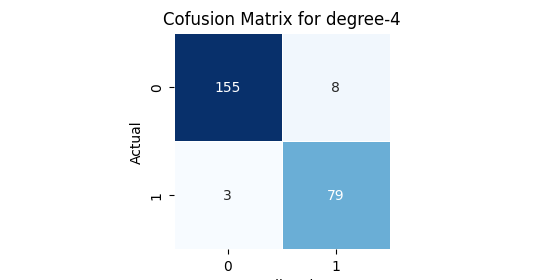
A confusion matrix is a table that helps you understand how well a classification model, like logistic regression, is performing. It provides a way to count the number of correct and incorrect predictions made by the model.

* True Positives (TP): The model correctly predicted that an example belongs to the positive class.
* True Negatives (TN): The model correctly predicted that an example belongs to the negative class
* False Positives (FP): The model incorrectly predicted that an example belongs to the positive class when it actually belongs to the negative class
* False Negatives (FN): The model incorrectly predicted that an example belongs to the negative class when it actually belongs to the positive class

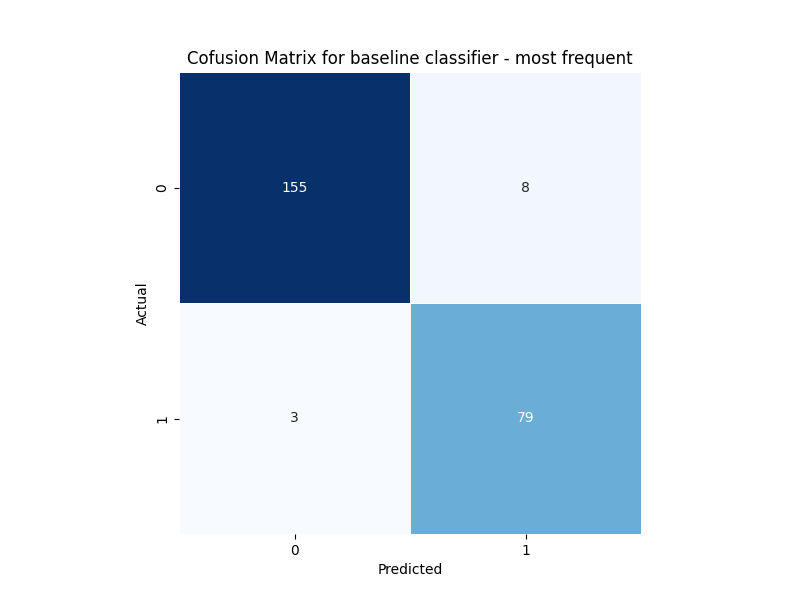
We plot confusion matrix, with the help of seaborn modules SNS package. We use heatmap of the SNS library to map the true positives and negatives also False positives and negatives.

Here are all the plots for each degree of polynomial.





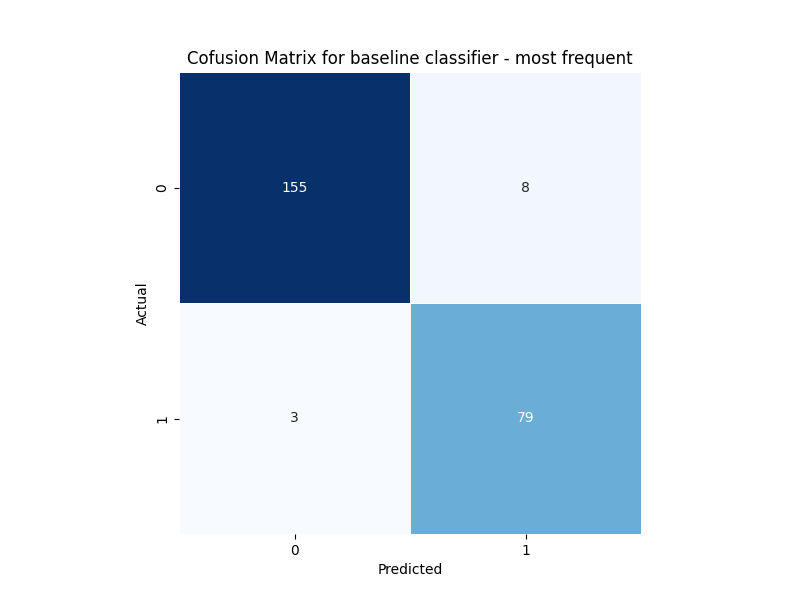
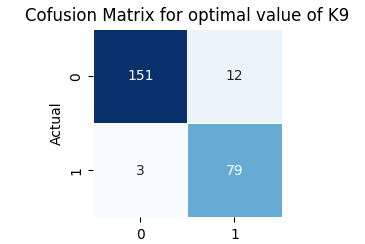
Confusion matrix of all model for degrees 1,2,3,4,5



BaseLine Classifier Confusion Matrix

**KNN:**

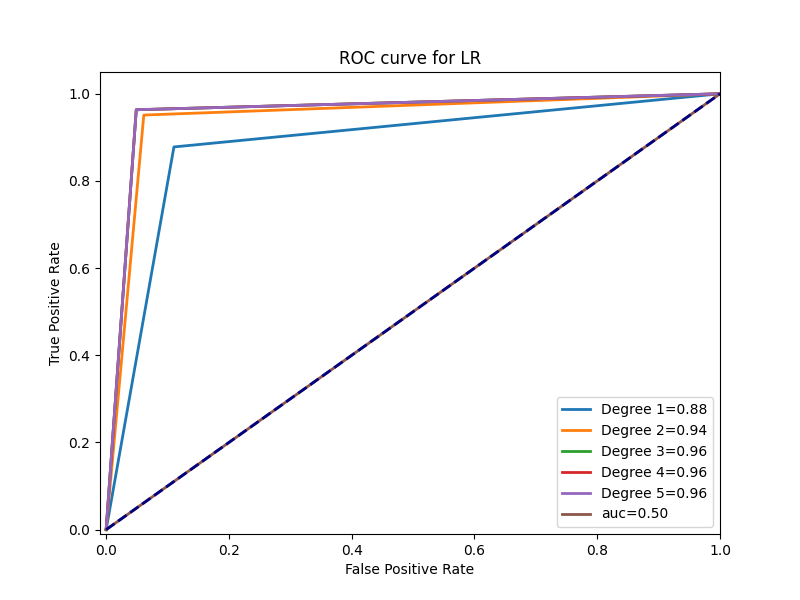
We plot the confusion matrix for the optimal model where k = 9, and compare it with the baseline classifier – most frequent.



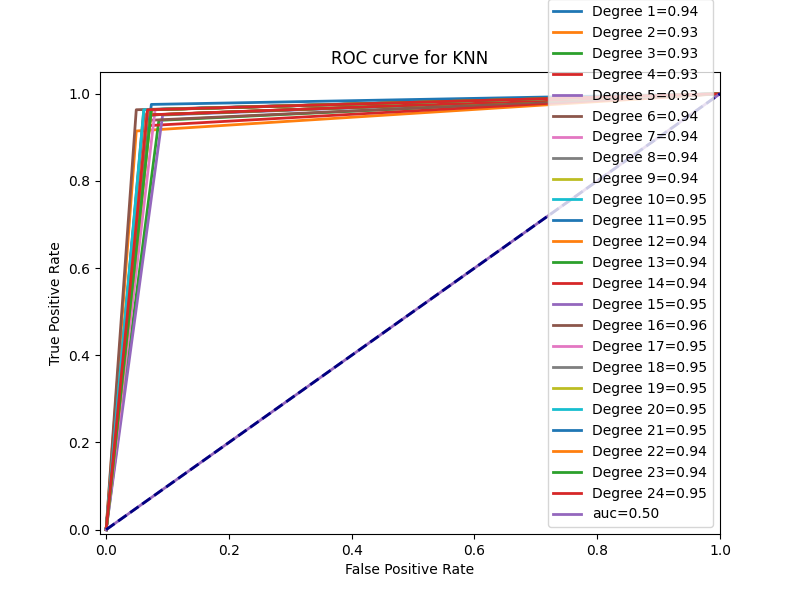
**ROC Curves:**

An ROC curve, which stands for Receiver Operating Characteristic curve, is a graphical representation that illustrates the performance of a binary classification model, such as logistic regression, at various threshold settings. It's a widely used tool for assessing and visualizing the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) of a classification model.

Let us plot ROC curves for both Logistic regression and KNN, of the best model for each degree and different K values.



ROC curve for degrees 2,3,4,5 and baseline Classifier – most\_frequent



ROC curve for KNN for range of 1-24 with baseline classifier

Score Table:

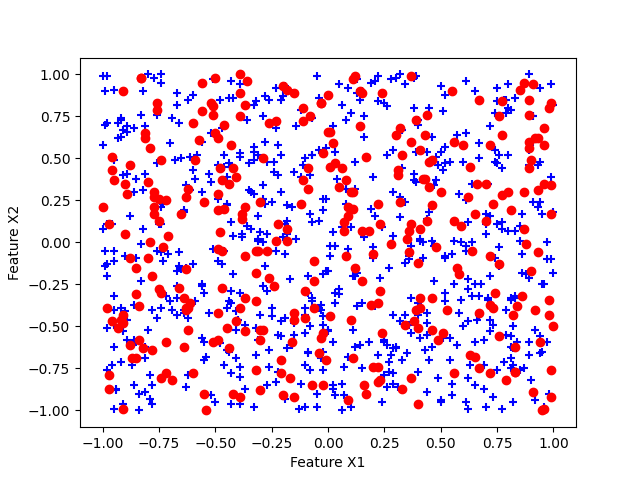
|  |  |  |  |
| --- | --- | --- | --- |
|  | Best Score | Best C value | Best degree or K |
| Logistic Classifier | 0.9404 | 100 | 5 |
| KNN classifier | 0.95 | - | 9 |
| Baseline Classifier | 0.66530 | - | - |

From the score table we can make a conclusion of best performing model for the give dataset.

**Dataset 2:**

Here, we have another dataset, so we use the same model and code to run on this dataset.

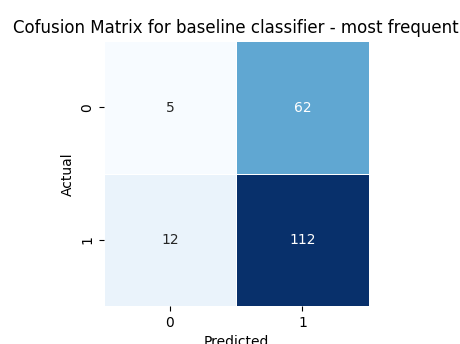
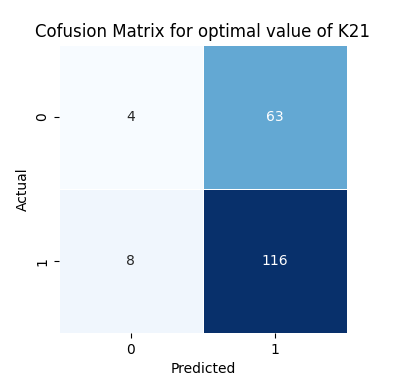
Lets visualize, the data



Here, we can clearly see the data here is very inconsistent and mostly is its noise for our model. Lets train this dataset, and see the result. After doing the cross-validation, we can see the accuracy value has a pattern of spike whenever there is increase in the k value.

**Confusion Matrix:**

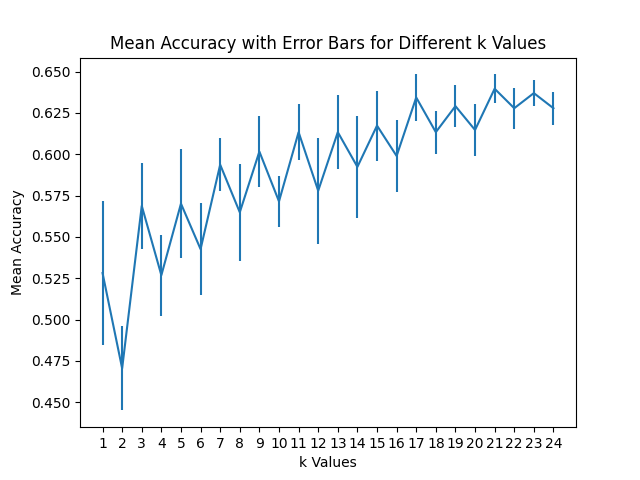
Here, when we plot the confusion matrix, we see that the true positive rate is very low as the model is not performing good for this dataset. The dataset is not good for training a KNN model.



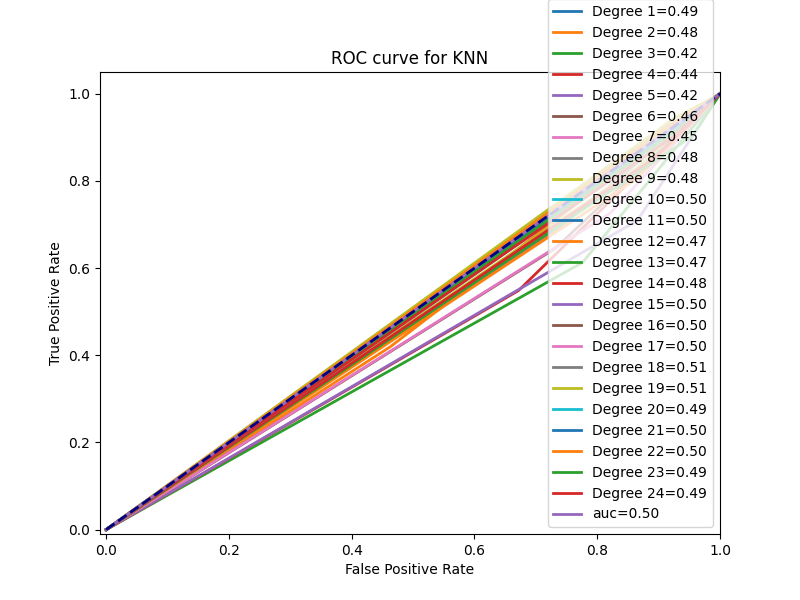
Comparison between KNN and Baseline Classifier

Here, the baseline classifier also performs almost same as the KNN. And for this dataset we get the optimal values of k = 21

ROC Curve :



And after ploting the Roc curve we can understand that the model under-performs with respect to a baseline model. The AUC is below 0.5 which is not good for model.



Dataset 2: Score table

|  |  |  |  |
| --- | --- | --- | --- |
|  | Best Score | Best C value | Best degree or K |
| Logistic Classifier | 0.7873 | 0.001 | 5 |
| KNN classifier | 0.6395 | - | 21 |
| Baseline Classifier | 0.6492 | - | - |

From the given table we can make conclusions on best performing model for the given dataset

Appendix :

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import GridSearchCV, StratifiedKFold

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix

import pandas as pd

from sklearn.dummy import DummyClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn import metrics

# load dataset

col\_names = ['x', 'y','out']

dataset = pd.read\_csv("dataset2.csv", header=None, names=col\_names)

print(dataset.head())

X1=dataset.iloc[:,0]

X2=dataset.iloc[:,1]

X=np.column\_stack((X1,X2))

Y=dataset.iloc[:,2]

#split the data

negative\_mask = dataset['out'] == -1

positive\_mask = dataset['out'] == 1

negative\_data = dataset[negative\_mask]

positive\_data = dataset[positive\_mask]

print(negative\_data.head())

print(positive\_data.head())

negative\_X=negative\_data.iloc[:,0]

negative\_Y=negative\_data.iloc[:,1]

positive\_X=positive\_data.iloc[:,0]

positive\_Y=positive\_data.iloc[:,1]

#plot the data

plt.scatter(positive\_X, positive\_Y, marker='+', label='+1', c='blue') # + marker for target +1

plt.scatter(negative\_X, negative\_Y, marker='o', label='-1', c='red') # o marker for target -1

plt.xlabel('Feature X1')

plt.ylabel('Feature X2')

plt.show()

Model = "KNN" #change here to run different models

#define list to store the best values

best\_parameters = {}

best\_scores = {}

best\_model = {}

degrees = []

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

mean\_score\_dict = []

std\_dev\_dict = []

if(Model == 'LR'):

degrees = [2,3,4,5] #set the range for degrees

C\_val = [0.001,0.01,0.1,1,10,100] #Set C values

#split the dataset into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

#creating polynomial features from input features

for d in degrees:

mean\_scores\_logreg = []

std\_dev\_logreg = []

poly = PolynomialFeatures(degree=d)

X\_poly = poly.fit\_transform(X\_train)

#Create a logistic regression model

log\_Reg = LogisticRegression(penalty='l2', solver='lbfgs')

# Create parameter grid to search

param\_grid = {

'C': C\_val,

'max\_iter': [10000]

}

#Perform cross validataion

cvLR = StratifiedKFold(n\_splits=5)

grid\_search = GridSearchCV(log\_Reg, param\_grid, cv=cvLR, scoring='accuracy')

grid\_search.fit(X\_poly, y\_train)

# Get the results for mean and standard deviation

mean\_test\_scores = grid\_search.cv\_results\_['mean\_test\_score']

std\_test\_scores = grid\_search.cv\_results\_['std\_test\_score']

mean\_score\_dict.append(mean\_test\_scores)

std\_dev\_dict.append(std\_test\_scores)

# Print the mean and standard deviation for each parameter combination

for mean\_score, std\_score, params in zip(mean\_test\_scores, std\_test\_scores, grid\_search.cv\_results\_['params']):

print(f"Mean Test Score: {mean\_score:.4f}, Std Test Score: {std\_score:.4f}, Parameters: {params}")

# Get the best parameters and best cross-validation score

best\_param = grid\_search.best\_params\_

best\_score = grid\_search.best\_score\_

best\_logReg\_model = grid\_search.best\_estimator\_

best\_parameters[d] = best\_param

best\_scores[d] = best\_score

best\_model[d] = best\_logReg\_model

elif(Model == 'KNN'):

#define range for k

degrees = list(range(1, 25))

mean\_scores = []

std\_scores = []

for k in degrees:

#train Model

knn = KNeighborsClassifier(n\_neighbors=k)

# Create parameter grid to search

param\_grid = {

'n\_neighbors': [k],

}

cv = StratifiedKFold(n\_splits=5)

grid\_search = GridSearchCV(knn, param\_grid, cv=cv, scoring='accuracy')

grid\_search.fit(X, Y)

# Get the best parameters and best cross-validation score

best\_param = grid\_search.best\_params\_

best\_score = grid\_search.best\_score\_

best\_knn\_model = grid\_search.best\_estimator\_

best\_parameters[k] = best\_param

best\_scores[k] = best\_score

best\_model[k] = best\_knn\_model

for k in degrees:

knn = KNeighborsClassifier(n\_neighbors=k)

scores = cross\_val\_score(knn, X\_train, y\_train, cv=cv, scoring='accuracy')

mean\_scores.append(np.mean(scores))

std\_scores.append(np.std(scores))

# Plot the mean accuracy with error bars for different k values

plt.errorbar(degrees, mean\_scores, yerr=std\_scores)

plt.xlabel('k Values')

plt.ylabel('Mean Accuracy')

plt.title('Mean Accuracy with Error Bars for Different k Values')

plt.xticks(degrees)

plt.show()

# Find the optimal k value

optimal\_k = degrees[np.argmax(mean\_scores)]

print(f"Optimal k: {optimal\_k}")

else:

print("wrong model selected. Select LG or KNN")

#plot cross-validation plots

if(Model == 'LR'):

# Plot mean test scores as a blue line

for i, d in enumerate(degrees):

plt.plot(C\_val, mean\_score\_dict[i], 'b', marker='o', label='Mean Test Score')

plt.errorbar(C\_val, mean\_score\_dict[i], yerr=std\_dev\_dict[i], fmt='o', capsize=5)

# Calculate the upper and lower bounds for error bars

upper\_bound = [mean + std for mean, std in zip(mean\_score\_dict[i], std\_dev\_dict[i])]

lower\_bound = [mean - std for mean, std in zip(mean\_score\_dict[i], std\_dev\_dict[i])]

# Plot error bars

plt.fill\_between(C\_val, upper\_bound, lower\_bound, color='lightblue', alpha=0.6, label='Mean +/- Std')

# Set plot labels and title

plt.xlabel('C Values')

plt.ylabel('Accuracy')

plt.title(f'Degree {d}')

plt.xscale('log') # Use a logarithmic scale for C values

# Show a legend

plt.legend()

# Show the plot

plt.show()

#initialize list to store scores, tpr , fpr and confusion matrices

f1\_scores = {}

list\_tpr = {}

list\_fpr = {}

confusion\_matrices = {}

roc\_curves = {}

if(Model == 'LR'):

for d in degrees:

#best logistic implement

poly = PolynomialFeatures(degree=d)

X\_poly = poly.fit\_transform(X\_train)

X\_test\_poly = poly.transform(X\_test)

best\_logreg\_model = best\_model[d]

best\_logreg\_model.fit(X\_poly,y\_train)

y\_pred\_logreg = best\_logreg\_model.predict(X\_test\_poly)

falsePositiveRate, truePositiveRate,\_= metrics.roc\_curve(y\_test, y\_pred\_logreg)

#true positive rate and false positive rate

list\_tpr[d] = truePositiveRate

list\_fpr[d] = falsePositiveRate

#confusion matrix

confusion\_matrix\_logreg = confusion\_matrix(y\_test, y\_pred\_logreg)

confusion\_matrices[d] = confusion\_matrix\_logreg

#plot predictions vs training data for best estimators for each degree

plt.scatter(X\_test[:, 0], X\_test[:, 1], c='green', cmap=plt.cm.Paired, marker='^', label='Training Data')

plt.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_pred\_logreg, cmap=plt.cm.Paired, marker='x',label='Predictions')

plt.xlabel('Feature X1')

plt.ylabel('Feature X2')

plt.title(f'Best Estimator for Degreee = {d}')

plt.legend()

plt.show()

elif(Model == 'KNN'):

for d in degrees:

#best KNN implement

best\_KNN\_model = best\_model[d]

best\_KNN\_model.fit(X\_train,y\_train)

y\_pred\_KNN = best\_KNN\_model.predict(X\_test)

#ROC curve

falsePositiveRate, truePositiveRate, \_ = metrics.roc\_curve(y\_test, y\_pred\_KNN)

#true positive rate and false positive rate

list\_tpr[d] = truePositiveRate

list\_fpr[d] = falsePositiveRate

#confusion matrix

confusion\_matrix\_KNN = confusion\_matrix(y\_test, y\_pred\_KNN)

confusion\_matrices[d] = confusion\_matrix\_KNN

#Adding baseline classifier - strategy - most\_frequent

DummyClassifier = DummyClassifier(strategy="most\_frequent")

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

y\_pred\_dummy =DummyClassifier.predict(X\_test)

falsePositiveRate\_dummy, truePositiveRate\_dummy, \_ = metrics.roc\_curve(y\_test, y\_pred\_dummy)

roc\_auc\_dummy = metrics.auc(falsePositiveRate\_dummy,truePositiveRate\_dummy)

#plot ROC

plt.figure(figsize=(8, 6))

for d in degrees:

roc\_auc = metrics.auc(list\_fpr[d], list\_tpr[d])

#create ROC curve

plt.plot(list\_fpr[d],list\_tpr[d],lw=2,label=f"Degree {d}={roc\_auc:.2f}")

plt.plot(falsePositiveRate\_dummy,truePositiveRate\_dummy,lw=2,label=f"auc={roc\_auc\_dummy:.2f}")

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([-0.01, 1.0])

plt.ylim([-0.01, 1.05])

plt.title(f"ROC curve for {Model}")

plt.ylabel('True Positive Rate')

plt.xlabel('False Positive Rate')

plt.legend(loc=4)

plt.show()