Ganeshan M

EN.U4CSE19320

AM.

Phase 2

Problem Definition

Banks play a crucial role in market economies. They decide who can get finance and on what terms and can make or break investment decisions. For markets and society to function, individuals and companies need access to credit.

Credit scoring algorithms, which make a guess at the probability of default, are the method banks use to determine whether or not a loan should be granted. This competition requires participants to improve on the state of the art in credit scoring, by predicting the probability that somebody will experience financial distress in the next two years.
The goal is to build a model that borrowers can use to help make the best financial decisions.

Improve on the state of the art in credit scoring by predicting the probability that somebody will experience financial distress in the next two years.

[GitHub (https://github.com/mganeshan29/Credit-Prediction-model)](https://github.com/mganeshan29/Credit-Prediction-model)

Dataset

Give Me Some Credit

In this Model, credit scoring data sets from Kaggle competition called ‘Give me some credit’ is used to build classifiers

Training dataset [cs-training.csv file (https://www.kaggle.com/brycecf/give-me-some-credit-dataset?select=cs-training.csv)](https://www.kaggle.com/brycecf/give-me-some-credit-dataset?select=cs-training.csv) which will be used for model training and test data [cs-test.csv file (https://www.kaggle.com/brycecf/give-me-some-credit-dataset?select=cs-test.csv)](https://www.kaggle.com/brycecf/give-me-some-credit-dataset?select=cs-test.csv) .

Prepare Data

In this part, the data and variables is done. Target ratio, variables values distributions are also be investigated. Null value analysis is done and null values in variables, filled with statistical approach to prevent them to influence modelling in a bad way.

After the describing inputs, missing values are also investigated, and for 2 inputs (‘MonthlyIncome’ , ‘NumberOfDependents’) some missing values are observed. Handling with missing values is very important to create accurate models.

Data Dictionary :

Variable Name	Description	Type
SeriousDlqin2yrs	Person experienced 90 days past due delinquency or worse	Y/N
RevolvingUtilizationOfUnsecuredLines	Total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits	percentage
age	Age of borrower in years	integer
NumberOfTime30-59DaysPastDueNotWorse	Number of times borrower has been 30-59 days past due but no worse in the last 2 years.	integer
DebtRatio	Monthly debt payments, alimony,living costs divided by monthy gross income	percentage
MonthlyIncome	Monthly income	real
NumberOfOpenCreditLinesAndLoans	Number of Open loans (installment like car loan or mortgage) and Lines of credit (e.g. credit cards)	integer
NumberOfTimes90DaysLate	Number of times borrower has been 90 days or more past due.	integer
NumberRealEstateLoansOrLines	Number of mortgage and real estate loans including home equity lines of credit	integer
NumberOfTime60-89DaysPastDueNotWorse	Number of times borrower has been 60-89 days past due but no worse in the last 2 years.	integer
NumberOfDependents	Number of dependents in family excluding themselves (spouse, children etc.)	integer

Python packages

NumPy

NumPy is a well known general-purpose array-processing package. An extensive collection of high complexity mathematical functions make NumPy powerful to process large multi-dimensional arrays and matrices. NumPy is very useful for handling linear algebra, Fourier transforms, and random numbers.

Scikit-learn

The Python library, Scikit-Learn, is built on top of the matplotlib, NumPy, and SciPy libraries. This Python ML library has several tools for data analysis and data mining tasks.

Pandas

Pandas is a popular Python library for data analysis. It is not directly related to Machine Learning. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation.

Matplotlib

Matplotlib is a very popular Python library for data visualization. Like Pandas, it is not directly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots.

seaborn

Seaborn is a library for making statistical graphs in Python. It is built on top of matplotlib and also integrated with pandas data structures.

Scikit-plot

Scikit-plot is the result of an unartistic data scientist’s dreadful realization that visualization is one of the most crucial components in the data science process, not just a mere afterthought.

Importing Libraries

In [83]:
```
import itertools
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import scikitplot as skplt
from sklearn import metrics
import plotly.graph_objs as go
from plotly.offline import iplot
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import confusion_matrix

from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix, roc_curve, auc, recall_score,roc_auc_sc

from sklearn.metrics import recall_score
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Data analysis

In [2]:
```
train_data = pd.read_csv("cs-training.csv")
test_data = pd.read_csv("cs-test.csv")
```

Review data

In [3]:
```
print(train_data.describe)
print(train_data.head)
```

```
149997                              1                                      0
149998                              0                                      0
149999                              2                                      0

        NumberOfDependents
0                      2.0
1                      1.0
2                      0.0
3                      0.0
4                      0.0
...                    ...
149995                 0.0
149996                 2.0
149997                 0.0
149998                 0.0
149999                 0.0

[150000 rows x 12 columns]>
<bound method NDFrame.head of          Unnamed: 0  SeriousDlqin2yrs  RevolvingUtilizationOfUnsecur
edLines  \
```

In [4]:
```
train_data.shape
```

Out[4]:
```
(150000, 12)
```

In [5]:
```
y = train_data.iloc[:,0].values #Taking first col (credit worthiness value)
X = train_data.iloc[:, 1:11].values #Taking the rest of the cols
```

Print columns headers of the dataset

In [6]:
```
train_data.columns
```

Out[6]:
```
Index(['Unnamed: 0', 'SeriousDlqin2yrs',
       'RevolvingUtilizationOfUnsecuredLines', 'age',
       'NumberOfTime30-59DaysPastDueNotWorse', 'DebtRatio', 'MonthlyIncome',
       'NumberOfOpenCreditLinesAndLoans', 'NumberOfTimes90DaysLate',
       'NumberRealEstateLoansOrLines', 'NumberOfTime60-89DaysPastDueNotWorse',
       'NumberOfDependents'],
      dtype='object')
```

Drop lines with Missing data

In [7]:
```
train_data.dropna(axis=1)
cols_with_missing = [col for col in train_data.columns
                         if train_data[col].isnull().any()]
reduced_train_data = train_data.drop(cols_with_missing, axis=1)
reduced_test_data = test_data.drop(cols_with_missing,axis=1)
```

In [8]:
```
reduced_train_data.columns
```

Out[8]:
```
Index(['Unnamed: 0', 'SeriousDlqin2yrs',
       'RevolvingUtilizationOfUnsecuredLines', 'age',
       'NumberOfTime30-59DaysPastDueNotWorse', 'DebtRatio',
       'NumberOfOpenCreditLinesAndLoans', 'NumberOfTimes90DaysLate',
       'NumberRealEstateLoansOrLines', 'NumberOfTime60-89DaysPastDueNotWorse'],
      dtype='object')
```

Applying Machine Learning Algorithms for Classification Problem

logistic-regression

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

specify the target variable

In [19]:
```
train_y = reduced_train_data.SeriousDlqin2yrs
test_y = reduced_test_data.SeriousDlqin2yrs
```

create list of features

In [20]:
```
feature_names = ['RevolvingUtilizationOfUnsecuredLines', 'age',
       'NumberOfTime30-59DaysPastDueNotWorse', 'DebtRatio',
       'NumberOfOpenCreditLinesAndLoans', 'NumberOfTimes90DaysLate',
       'NumberRealEstateLoansOrLines', 'NumberOfTime60-89DaysPastDueNotWorse']
#create data corresponding to the features
train_X = reduced_train_data[feature_names]
test_X = reduced_test_data[feature_names]
```

Spiliting Dataset into Train and Test set.

In [21]:
```
x_train, x_test, y_train, y_test = train_test_split(train_X, train_y, test_size = 0.3, random_state
```

Logistic regression User Defined

In [22]:

```
class logistic_regression:
    def __init__(self,x,y):
        self.intercept = np.ones((x.shape[0], 1))
        self.x = np.concatenate((self.intercept, x), axis=1)
        self.weight = np.zeros(self.x.shape[1])
        self.y = y

    def sigmoid(self, x, weight):
        z = np.dot(x, weight)
        return 1 / (1 + np.exp(-z))

    def loss(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    def gradient_descent(self, X, h, y):
        return np.dot(X.T, (h - y)) / y.shape[0]

    def fit(self, lr , iterations):
        for i in range(iterations):
            sigma = self.sigmoid(self.x, self.weight)
            loss = self.loss(sigma,self.y)
            dW = self.gradient_descent(self.x , sigma, self.y)
            #Updating the weights
            self.weight -= lr * dW
        return print('Working successfully')
    def predict(self, x_new , treshold):
        x_new = np.concatenate((self.intercept, x_new), axis=1)
        result = self.sigmoid(x_new, self.weight)
        result = result >= treshold
        y_pred = np.zeros(result.shape[0])
        for i in range(len(y_pred)):
            if result[i].any() == True:
                y_pred[i] = 1
            else:
                continue

        return y_pred
```

In [23]:
```
regressor = logistic_regression(X,y)
regressor.fit(0.1 , 5000)
pred_lr = regressor.predict(X,0.5)
```

```
Working successfully
```

In [24]:
```
logmodel = LogisticRegression(random_state=42)
logmodel.fit(x_train,y_train)
y_pred_l = logmodel.predict(x_test)
pred_lr = y_pred_l.astype(int)
```

In [25]:
```
roc=roc_auc_score(y_test, pred_lr)
acc = accuracy_score(y_test, pred_lr)
prec = precision_score(y_test, pred_lr)
results = pd.DataFrame([['Logistic Regression', acc,prec,roc]],
columns = ['Model', 'Accuracy', 'Precision','ROC_AUC'])
results
```

Out[25]:

	Model	Accuracy	Precision	ROC_AUC
0	Logistic Regression	0.932778	0.532468	0.512651

Built in Logistic Regression

In [26]:
```
logreg = LogisticRegression(random_state=1)

#fit the model
logreg.fit(x_train,y_train)
```

Out[26]:
```
LogisticRegression(random_state=1)
```

Make predictions

In [27]:
```
predictions_train = logreg.predict(x_train)
y_pred = logreg.predict(x_test)
t_pred = logreg.predict(x_test)
```

In [28]:
```
# I had to do this because confusion matrix was throwing errors
print(predictions_train)
print(y_pred)
print(y_pred.shape)
print(y_pred.dtype)
print(y_test.shape)
print(y_test.dtype)
```

```
[0 0 0 ... 0 0 0]
[0 0 0 ... 0 0 0]
(45000,)
int64
(45000,)
int64
```

In [29]:
```
y_predi = y_pred.astype(int) #converting float to int
```

create confuson matrics in text view

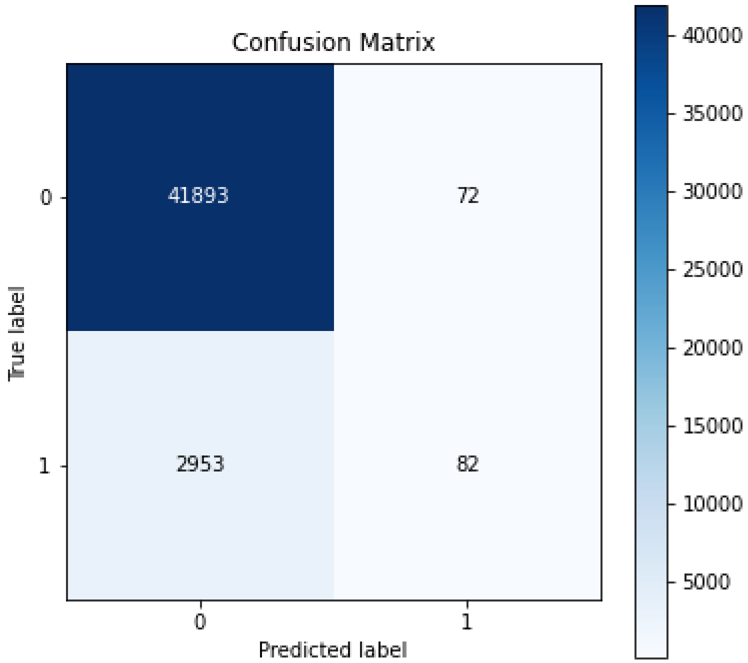
In [30]:
```
tn, fp, fn, tp = confusion_matrix( y_test,y_predi).ravel()
(tn, fp, fn, tp)
```

Out[30]:
```
(41893, 72, 2953, 82)
```

consusion matrix

In [31]:
```
skplt.metrics.plot_confusion_matrix(y_test,y_predi,figsize=(6,6))
```

Out[31]:
```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



calculating ROC

In [32]:
```
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_predi)
print(fpr)
print(fpr.shape)
print(tpr)
print(tpr.shape)
print(thresholds)
```

```
[0.         0.00171572 1.        ]
(3,)
[0.         0.02701812 1.        ]
(3,)
[2 1 0]
```

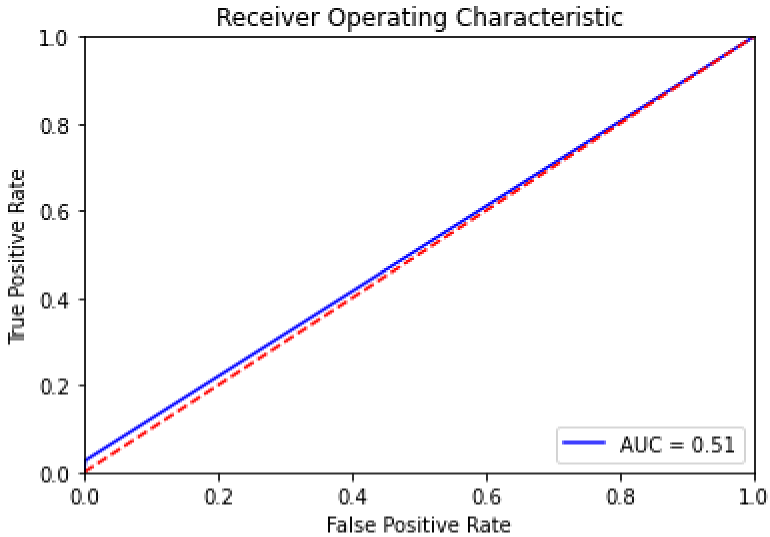
calculating AUC

In [33]:
```
from sklearn import metrics
# AUC
roc_auc = metrics.auc(fpr, tpr)
print(roc_auc)
```

```
0.5126512032169273
```

Plotting ROC

In [34]:
```
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Calculate F1 score

In [35]:
```
#calculate F1 score
from sklearn.metrics import f1_score
f1_score(y_test, y_predi)
```

Out[35]: 0.051426779554719346

Accuracy score

In [36]:
```
accuracy_score(y_test, y_predi)
```

Out[36]: 0.9327777777777778

In [37]:
```
form = []
form.append(['Logistic Regression',accuracy_score(y_test, y_predi)])
```

Precision

In [38]:
```
precision_score(y_test, y_predi)
```

Out[38]: 0.5324675324675324

Recall

In [39]:
```

recall_score(y_test, y_predi)
```

Out[39]: 0.027018121911037891

Cost-sensitive accuracy

In [40]:
```
fp_cost = 1
fn_cost = 0
cost_sensitive_accuracy = (tp + tn) / (tp + tn + fp*fp_cost + fn*fn_cost)
print(cost_sensitive_accuracy)
```

```
0.9982876305087164
```

In [41]:
```
results = pd.DataFrame([['Logistic Regression', accuracy_score(y_test, y_predi) ,precision_score(y_t
columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC_AUC'])
results
```

Out[41]:

	Model	Accuracy	Precision	Recall	F1 Score	ROC_AUC
0	Logistic Regression	0.932778	0.532468	0.027018	0.051427	0.512651

Decision Tree

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.

In [42]:
```
from sklearn.tree import DecisionTreeClassifier

#specify the model, set any numeric valye as parameter to ensure reproducibility
dTree = DecisionTreeClassifier(random_state=1)

#fit the model
dTree.fit(x_train,y_train)
```

Out[42]:
```
DecisionTreeClassifier(random_state=1)
```

Make predictions

In [43]:
```
predictions_train = dTree.predict(x_train)
y_pred = dTree.predict(x_test)
```

In [44]:
```
#this section investigates resulting data
print(predictions_train)
print(y_pred)
print(y_pred.shape)
print(y_pred.dtype)
print(y_test.shape)
print(y_test.dtype)
```

```
[0 0 0 ... 1 0 1]
[0 0 0 ... 0 0 0]
(45000,)
int64
(45000,)
int64
```

In [45]:
```
#conver float to int
y_predi = y_pred.astype(int)
```

Creating confusion matrics in text view

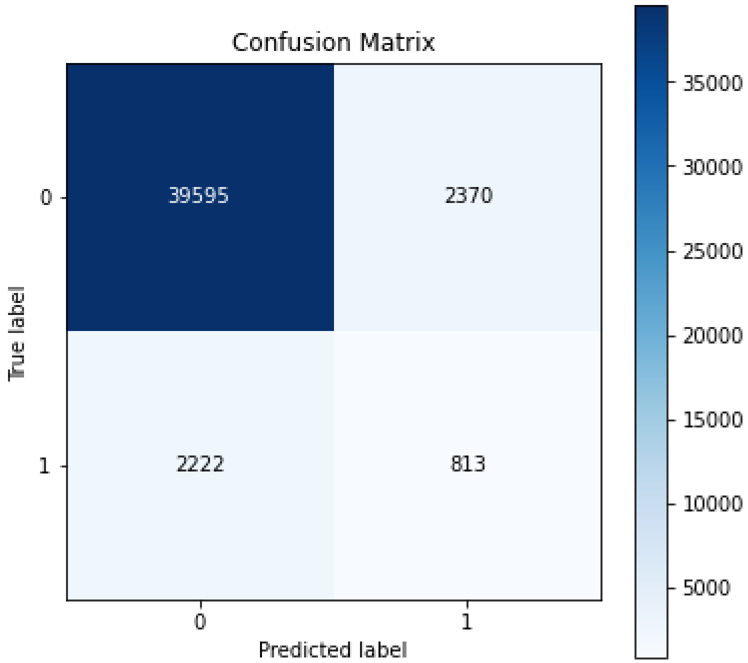
In [46]:
```
tn, fp, fn, tp = confusion_matrix( y_test,y_predi).ravel()
(tn, fp, fn, tp)
```

Out[46]:
```
(39595, 2370, 2222, 813)
```

Plotting Confusion Matrix

In [47]:
```
skplt.metrics.plot_confusion_matrix(y_test,y_predi,figsize=(6,6))
```

Out[47]:
```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



ROC

In [48]:
```
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_predi)
print(fpr)
print(fpr.shape)
print(tpr)
print(tpr.shape)
print(thresholds)
```

```
[0.         0.05647563 1.        ]
(3,)
[0.         0.26787479 1.        ]
(3,)
[2 1 0]
```

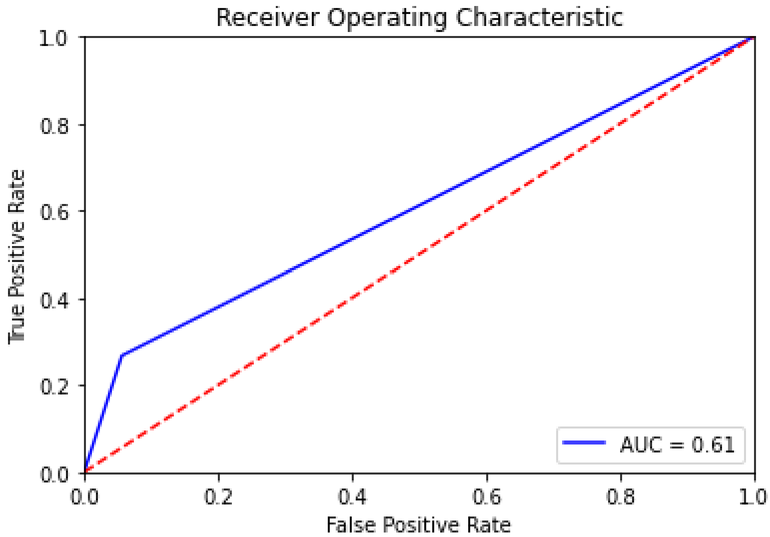
AUC

In [49]:
```
roc_auc = metrics.auc(fpr, tpr)
roc_auc
```

Out[49]:
```
0.6056995798059535
```

Plotting ROC

In [50]:
```
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



F1 score

In [51]:
```
f1_score=f1_score(y_test, y_predi)
f1_score
```
Out[51]: 0.26149887423608875

Accuracy score

In [52]:
```
a_score=accuracy_score(y_test, y_predi)
a_score
```
Out[52]: 0.8979555555555555

Precision

In [53]:
```
p_score=precision_score(y_test, y_predi)
p_score
```
Out[53]: 0.2554194156456173

Recall

In [54]:
```
recall_score=recall_score(y_test, y_predi)
recall_score
```
Out[54]: 0.26787479406919273

cost-sensitive accuracy

In [55]:
```
fp_cost = 1
fn_cost = 0
cost_sensitive_accuracy = (tp + tn) / (tp + tn + fp*fp_cost + fn*fn_cost)
print(cost_sensitive_accuracy)
```

```
0.9445976904016083
```

In [56]:
```
dt = pd.DataFrame([['Decision Tree', a_score ,p_score,recall_score, f1_score ,roc_auc]],
columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC_AUC'])
dt
```

Out[56]:

	Model	Accuracy	Precision	Recall	F1 Score	ROC_AUC
0	Decision Tree	0.897956	0.255419	0.267875	0.261499	0.6057

In [57]:
```
results = results.append(dt, ignore_index = True)
```

In [58]:
```
results
```

Out[58]:

	Model	Accuracy	Precision	Recall	F1 Score	ROC_AUC
0	Logistic Regression	0.932778	0.532468	0.027018	0.051427	0.512651
1	Decision Tree	0.897956	0.255419	0.267875	0.261499	0.605700

K-Nearest Neighbor(KNN) Algorithm

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

In [59]:
```
model_KNN = KNeighborsClassifier()

neighbors = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
model_KNN.fit(x_train, y_train)
knn = KNeighborsClassifier(n_neighbors = 20)
knn.fit(x_train, y_train)
knn_scores_proba = knn.predict_proba(x_train)
knn_scores = knn_scores_proba[:,1]
print("AUC Score :", roc_auc_score(y_train,knn_scores))
```

```
AUC Score : 0.8886195756715168
```

In [60]:
```
scorelist = []
n_neighbors, maxauc = -1, 0
for k in range(100, 1000+1, 100):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)
    y_pred = knn.predict_proba(x_test)[:,1]
    score = roc_auc_score(y_test, y_pred)
    print(k, score)
    if score > maxauc:
        n_neighbors, maxauc = k, score
    scorelist.append(score)
print()
print(n_neighbors, maxauc)
```

```
100 0.7738365716625469
200 0.7751776319444048
300 0.7679608507207013
400 0.7659084657313275
500 0.760544731027327
600 0.7546051182920732
700 0.750963969150569
800 0.7484402963087424
900 0.7465734585834943
1000 0.7445061871006885

200 0.7751776319444048
```

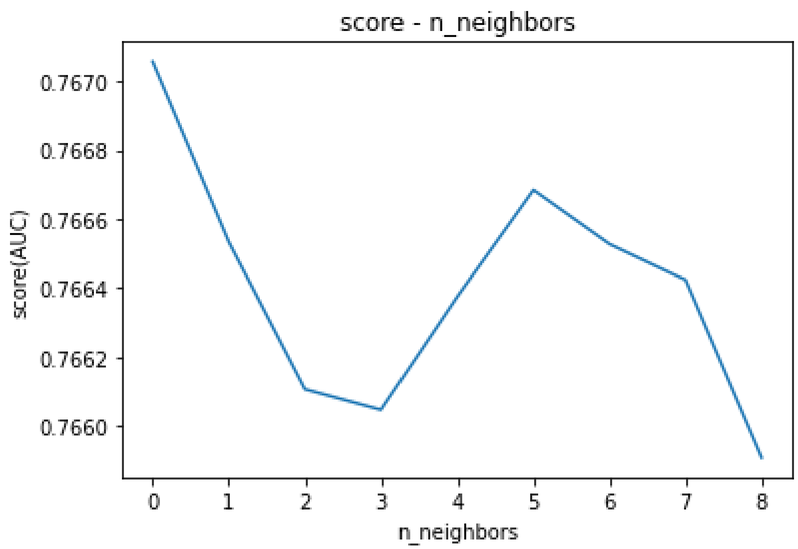
In [61]:
```
scorelist = []
n_neighbors, maxauc = -1, 0
for k in range(320, 400+1, 10):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train,y_train)
    y_pred = knn.predict_proba(x_test)[:,1]
    score = roc_auc_score(y_test, y_pred)
    print(k, score)
    if score > maxauc:
        n_neighbors, maxauc = k, score
    scorelist.append(score)
print()
print(n_neighbors, maxauc)
```

```
320 0.7670568377860976
330 0.7665339104466713
340 0.7661070700833106
350 0.766047414188218
360 0.7663753449518907
370 0.7666842632451809
380 0.7665274996756338
390 0.7664235965053643
400 0.7659084657313275

320 0.7670568377860976
```

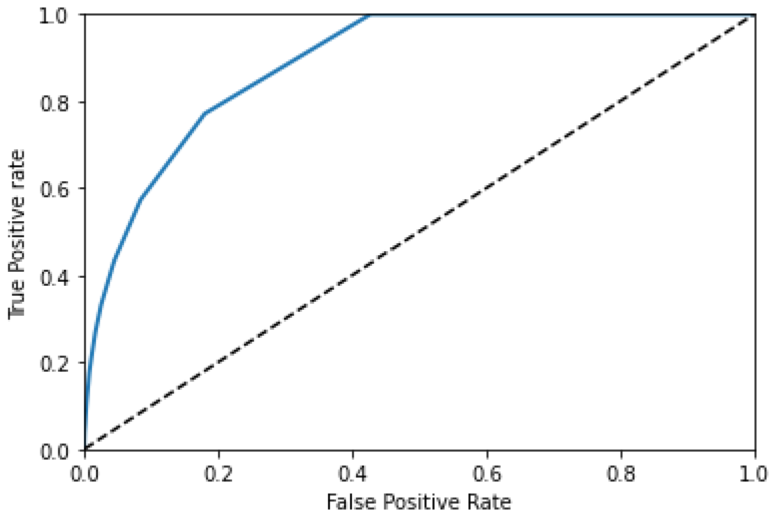
In [62]:
```
plt.plot(scorelist)
plt.title('score - n_neighbors')
plt.xlabel('n_neighbors')
plt.ylabel('score(AUC)')
```

Out[62]: Text(0, 0.5, 'score(AUC)')



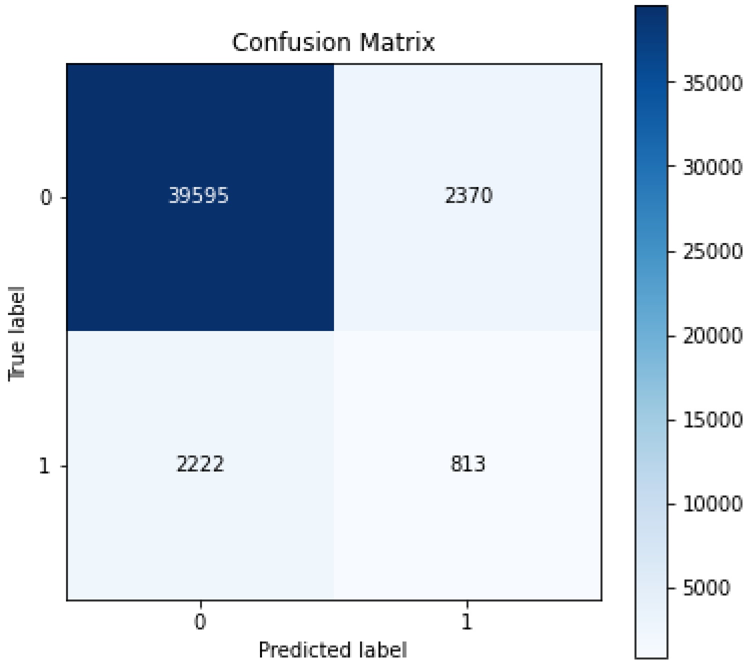
Plotting ROC

In [65]:
```
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0,1],[0,1], "k--")
    plt.axis([0,1,0,1])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive rate")
fpr_gbc, tpr_gbc, thresh_gbc = roc_curve(y_train, knn_scores)
plot_roc_curve(fpr_gbc, tpr_gbc)
```



In [66]:
```
skplt.metrics.plot_confusion_matrix(y_test,y_predi,figsize=(6,6))
```

Out[66]:
```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



ROC

In [67]:
```
roc=roc_auc_score(y_test, y_pred)
roc
```

Out[67]: 0.7659084657313275

AUC

In [68]:
```
auc = metrics.auc(fpr, tpr)
auc
```

Out[68]: 0.6056995798059535

Accuracy

In [69]:
```
acc=accuracy_score(y_test, y_pred.round(), normalize=True)
acc
```

Out[69]: 0.9331333333333334

Precision

In [70]:
```
prec = precision_score(y_test, y_pred.round())
prec
```

Out[70]: 0.6444444444444445

In [71]:
```
k_results = pd.DataFrame([['K-Nearest Neighbour ', acc ,prec ,roc_auc_score(y_train,knn_scores)]],
columns = ['Model', 'Accuracy', 'Precision','ROC_AUC'])
k_results
```

Out[71]:

	Model	Accuracy	Precision	ROC_AUC
0	K-Nearest Neighbour	0.933133	0.644444	0.88862

In [73]:
```
results = results.append(k_results, ignore_index = True)
```

Gaussian Naive Bayes

Gaussian Naive Bayes is a variant of Naive Bayes that follows Gaussian normal distribution and supports continuous data. We have explored the idea behind Gaussian Naive Bayes along with an example.

Before going into it, we shall go through a brief overview of Naive Bayes.
Naive Bayes are a group of supervised machine learning classification algorithms based on the Bayes theorem. It is a simple classification technique, but has high functionality. They find use when the dimensionality of the inputs is high.<\i>

In [74]:
```
from sklearn.naive_bayes import GaussianNB

gaussian = GaussianNB()
gaussian.fit(x_train, y_train)
gaussian_scores_proba = gaussian.predict_proba(x_train)
gaussian_scores = gaussian_scores_proba[:,1]
y_pred = gaussian.predict_proba(x_test)[:,1]
y_predi=gaussian.predict(x_test)

gaussian_scores = roc_auc_score(y_test, y_pred)
print(gaussian_scores)
```

```
0.7097319979719509
```

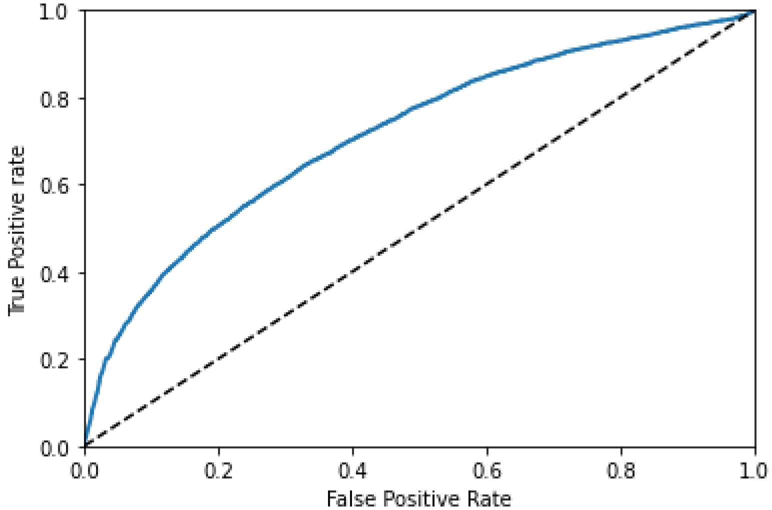
In [75]:
```
print(y_test)
```

```
16269     0
140471    0
78683     0
2605      0
81156     0
         ..
148024    0
59238     0
111773    0
107702    0
89084     0
Name: SeriousDlqin2yrs, Length: 45000, dtype: int64
```

In [76]:
```
gaussian = GaussianNB()
gaussian.fit(x_train, y_train)
gaussian_scores_proba = gaussian.predict_proba(x_train)
gaussian_scores = gaussian_scores_proba[:,1]
roc_auc=roc_auc_score(y_train,gaussian_scores)
roc_auc
```

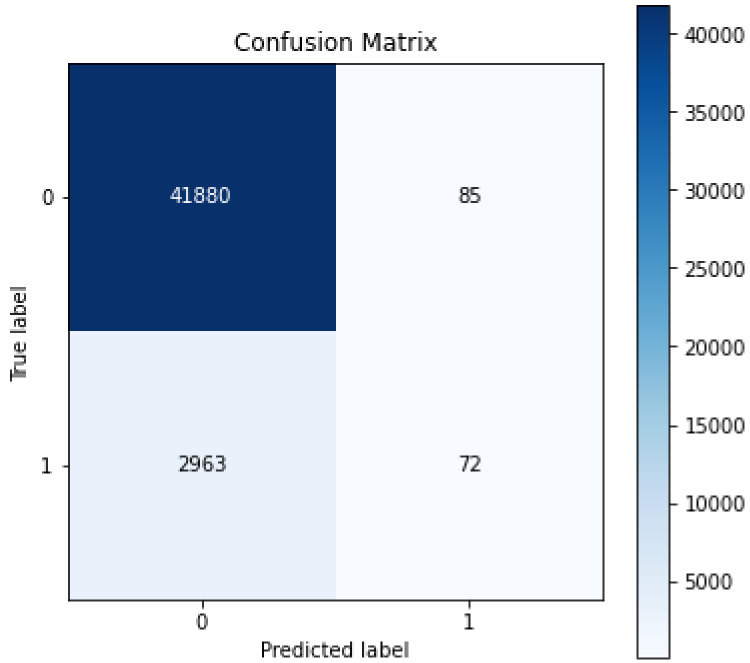
Out[76]:
```
0.714745677265423
```

In [77]:
```
fpr_gbc, tpr_gbc, thresh_gbc = roc_curve(y_train, gaussian_scores)
plot_roc_curve(fpr_gbc, tpr_gbc)
```



In [78]:
```
skplt.metrics.plot_confusion_matrix(y_test,y_predi,figsize=(6,6))
```

Out[78]: <AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>



In [79]:
```
prec = precision_score(y_test, y_predi)
prec
```

Out[79]: 0.4585987261146497

In [80]:
```
acc=accuracy_score(y_test, y_predi.round(), normalize=True)
acc
```

Out[80]: 0.9322666666666667

Model Tuning using KFold Validation

Cross-validation is a statistical method used to estimate the skill of machine learning models.

It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods.

In [86]:
```
from sklearn.model_selection import KFold
X, y = make_classification(n_samples=100, n_features=20, n_informative=15, n_redundant=5, random_sta
# prepare the cross-validation procedure
cv = KFold(n_splits=10, random_state=1, shuffle=True)
kfold = KFold(n_splits=10, random_state=None)
```

K-fold Cross-Validation on Logistic Regression Model

In [87]:
```
form = []
```

In [88]:
```
# evaluate a logistic regression model using k-fold cross-validation
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
# create dataset
X, y = make_classification(n_samples=100, n_features=20, n_informative=15, n_redundant=5, random_sta
# prepare the cross-validation procedure
cv = KFold(n_splits=10, random_state=1, shuffle=True)
# create model
lr_model = LogisticRegression()
# evaluate model
lr_scores = cross_val_score(logreg, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(lr_scores), std(lr_scores)))
LRscore=mean(lr_scores)
```

```
Accuracy: 0.850 (0.128)
```

In [89]:
```
form.append(['logistic regression Tuned'.format(lr_scores),mean(lr_scores)])
```

K-fold Cross-Validation on Decision Tree

In [90]:
```
a, b = make_classification(n_samples=100, n_features=20, n_informative=15, n_redundant=5, random_sta
# prepare the cross-validation procedure
cv = KFold(n_splits=10, random_state=1, shuffle=True)
# create model
Dt_model = DecisionTreeClassifier()
# evaluate model
scores = cross_val_score(dTree, a, b, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
DTCscore=mean(scores)
```

```
Accuracy: 0.650 (0.143)
```

In [91]:
```
form.append(['Decision Tree Tuned'.format(scores),mean(scores)])
```

K-fold Cross-Validation on KNN

In [92]:
```
knn_model = KNeighborsClassifier()
# evaluate model
knn_scores = cross_val_score(model_KNN, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(knn_scores), std(knn_scores)))
KNNscore=mean(knn_scores)
```

```
Accuracy: 0.770 (0.110)
```

In [93]:
```
form.append(['KNN Tuned'.format(knn_scores),mean(knn_scores)])
```

K-fold Cross-Validation Gaussian Naive Bayes

In [94]:
```
gnb_model = GaussianNB()
# evaluate model
gnb_scores = cross_val_score(gaussian, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(gnb_scores), std(gnb_scores)))
form.append(['Gaussian Naive Bayes Tuned'.format(gnb_scores),mean(gnb_scores)])
NBscore=mean(gnb_scores)
```

```
Accuracy: 0.720 (0.117)
```

In [95]:
```
form
```

Out[95]:
```
[['logistic regression Tuned', 0.85],
 ['Decision Tree Tuned', 0.65],
 ['KNN Tuned', 0.7699999999999999],
 ['Gaussian Naive Bayes Tuned', 0.72]]
```

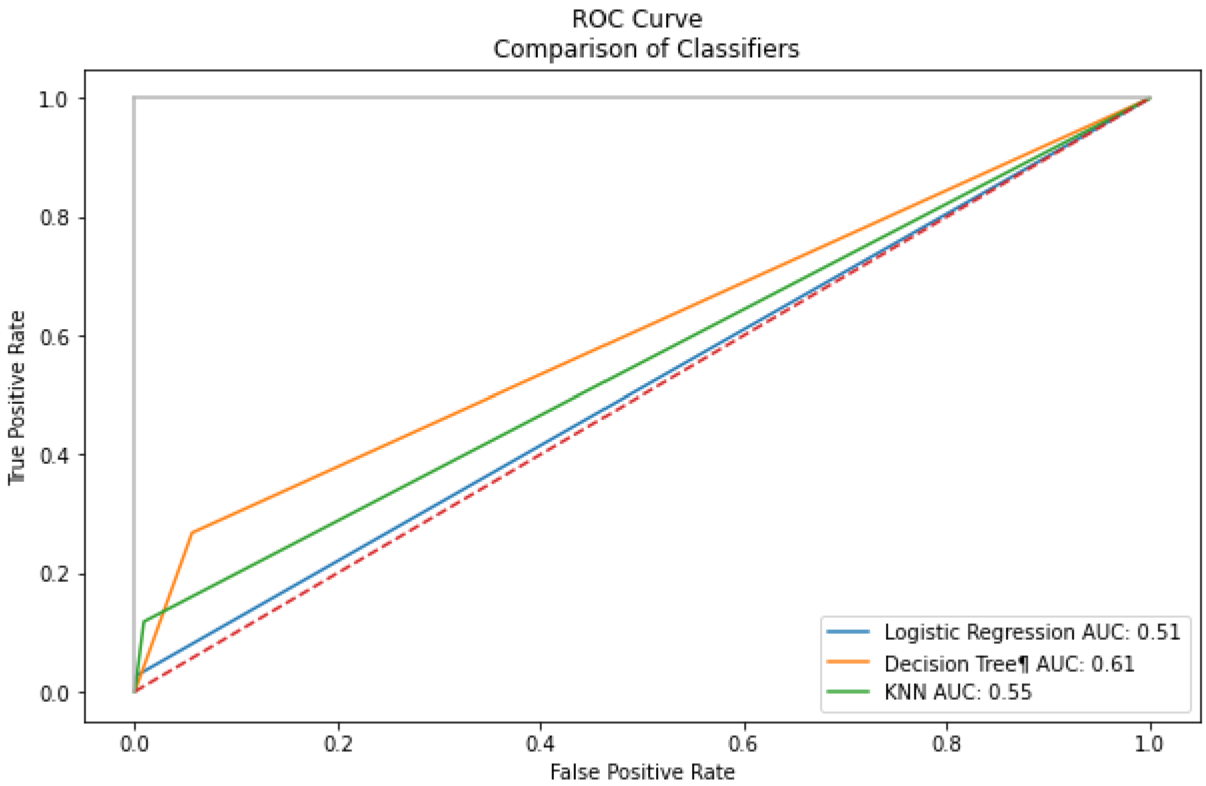
ROC Curve Comparison of Classifiers

In [96]:
```
logreg.fit(x_train, y_train)
dTree.fit(x_train, y_train)
knn_model.fit(x_train, y_train)
log_pred = logreg.predict(x_test)
dt_pred= dTree.predict(x_test)
knn_pred = knn_model.predict(x_test)
```

In [97]:
```
log_fpr, log_tpr, log_threshold   = roc_curve(y_test, log_pred)
dt_fpr, rfc_tpr, rfc_threshold   = roc_curve(y_test, dt_pred)
knn_fpr, knn_tpr, knn_threshold   = roc_curve(y_test, knn_pred)
```

In [98]:
```
fig  = plt.figure(figsize=(10,6))
plt.title('ROC Curve \n Comparison of Classifiers')
plt.plot(log_fpr, log_tpr, label ='Logistic Regression AUC: {:.2f}'.format(roc_auc_score(y_test, log
plt.plot(dt_fpr, rfc_tpr, label ='Decision Tree¶ AUC: {:.2f}'.format(roc_auc_score(y_test, dt_pred))
plt.plot(knn_fpr, knn_tpr, label ='KNN AUC: {:.2f}'.format(roc_auc_score(y_test, knn_pred)))

plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



Scatter Plot For Algorithm Comparison

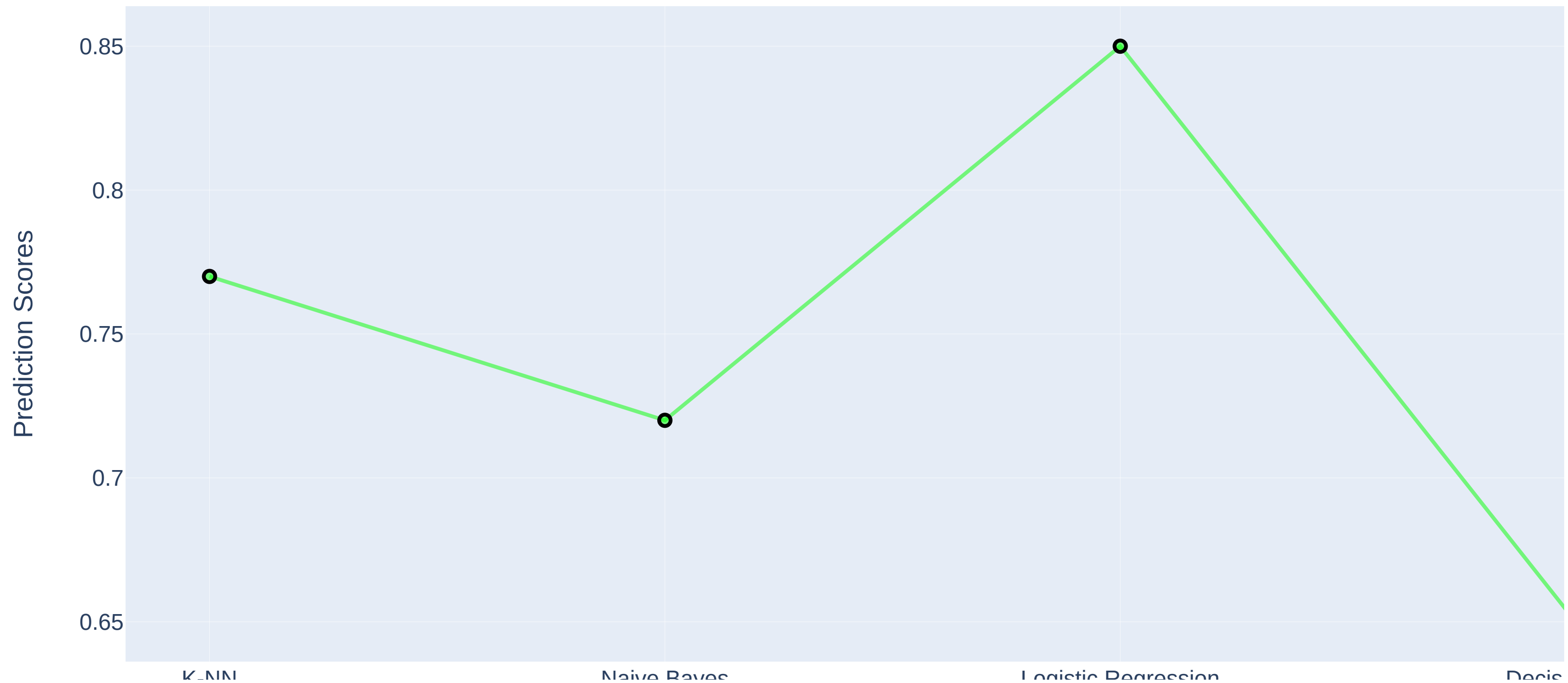
In [100]:
```
scores=[KNNscore,NBscore,LRscore,DTCscore,]
AlgorthmsName=["K-NN","Naive Bayes","Logistic Regression","Decision Tree"]

#create traces

trace1 = go.Scatter(
    x = AlgorthmsName,
    y= scores,
    name='Algortms Name',
    marker =dict(color='rgba(0,255,0,0.5)',
               line =dict(color='rgb(0,0,0)',width=2)),
                text=AlgorthmsName
)
data = [trace1]

layout = go.Layout(barmode = "group",
                  xaxis= dict(title= 'ML Algorithms',ticklen= 5,zeroline= False),
              yaxis= dict(title= 'Prediction Scores',ticklen= 5,zeroline= False))
fig = go.Figure(data = data, layout = layout)
```

In [102]:
```
iplot(fig)
```



Conclusion

*We found best result with **Logistic Regression** .*

***Logistic Regression** has the best performance with **0.85 Accuracy** compared to other three classifiers KNN, Gaussain Naive Bayes, and decision tree .*