Importing all libraries

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
from sklearn.model_selection import cross_validate,
train_test_split,GridSearchCV,KFold,validation_curve,StratifiedKFold,StratifiedShuffleSplit,RandomizedSearchC
V
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.metrics import make_scorer, accuracy_score,mean_squared_error
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression,LogisticRegression
from sklearn.neighbors import KNeighborsRegressor,KNeighborsClassifier
from sklearn.svm import SVR, SVC
from sklearn.tree import DecisionTreeRegressor,DecisionTreeClassifier
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score,
f1_score,confusion_matrix,classification_report,roc_curve, auc
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, RandomForestRegressor,
AdaBoostRegressor
import scikitplot
from sklearn.naive_bayes import GaussianNB
from mlxtend.classifier import StackingClassifier
import xgboost as xgb
```

Quation 1. (50 points) Use numeric prediction techniques to build a predictive model for the HW3.xlsx dataset. This dataset is provided on the course website and contains data about whether or not different consumers made a purchase in response to a test mailing of a certain catalog and, in case of a purchase, how much money each consumer spent. The data file has a brief description of all the attributes in a separate worksheet. Note that this dataset has two possible outcome variables: Purchase (0/1 value: whether or not the purchase was made) and Spending (numeric value: amount spent).

Importing Data

```
data = pd.read_excel('HW3.xlsx')
data.head()
```

	sequence_number	US	source_a	source_c	source_b	source_d	source_e	source_m	source_o	source_h	...
0	1	1	0	0	1	0	0	0	0	0	...
1	2	1	0	0	0	0	1	0	0	0	...
2	3	1	0	0	0	0	0	0	0	0	...
3	4	1	0	1	0	0	0	0	0	0	...
4	5	1	0	1	0	0	0	0	0	0	...

5 rows × 25 columns

```
xVariables = data[["US","source_a","source_c","source_b","source_d",\
                   "source_e","source_m","source_o","source_h",\
                   "source_r","source_s","source_t","source_u",\
                   "source_p","source_x","source_w","Freq",\
                   "last_update_days_ago","1st_update_days_ago",\
                   "Web order","Gender=male","Address_is_res"]]
yVariable = data[["Spending"]]
xVariablesStandard = xVariables
yVariableStandarad = yVariable
xVariables.head()
```

	US	source_a	source_c	source_b	source_d	source_e	source_m	source_o	source_h	source_r	...	source_u	source_p
0	1	0	0	1	0	0	0	0	0	0	...	0	0
1	1	0	0	0	0	1	0	0	0	0	...	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0
3	1	0	1	0	0	0	0	0	0	0	...	0	0
4	1	0	1	0	0	0	0	0	0	0	...	0	0

5 rows × 22 columns

MinMax Scaling

Because most of the variables are sparse , I will only scale only those features that are not sparse

```
scalerFreq = MinMaxScaler(feature_range=(0,1))
scalerLastUpdateDaysAgo = MinMaxScaler(feature_range=(0,1))
scalerFirstUpdateDaysAgo = MinMaxScaler(feature_range=(0,1))
#Yscaler = MinMaxScaler(feature_range=(0,1))
xVariables["Freq"] = scalerFreq.fit_transform(xVariables[["Freq"]])
xVariables["last_update_days_ago"] = \
    scalerLastUpdateDaysAgo.fit_transform(xVariables[["last_update_days_ago"]])
xVariables["1st_update_days_ago"] = \
    scalerFirstUpdateDaysAgo.fit_transform(xVariables[["1st_update_days_ago"]])
```

Normalizing Columns

```
xVariablesStandard["Freq"] = StandardScaler().fit_transform(xVariablesStandard[["Freq"]])
xVariablesStandard["last_update_days_ago"] = \
    StandardScaler().fit_transform(xVariablesStandard[["last_update_days_ago"]])
xVariablesStandard["1st_update_days_ago"] = \
    StandardScaler().fit_transform(xVariablesStandard[["1st_update_days_ago"]])
```

Test Train Split For Data

```
X_train, X_test, y_train, y_test = train_test_split(xVariables, yVariable\
                        , test_size=0.33, random_state=42)

X_train_stan, X_test_stan, y_train_stan, y_test_stan = train_test_split(xVariablesStandard, yVariableStandarad\
                        ,test_size=0.33, random_state=42)
```

(b) (20 points) As a variation on this exercise, create a separate "restricted" dataset (i.e., a subset of the original dataset), which includes only purchase records (i.e., where Purchase = 1). Build numeric prediction models to predict Spending for this restricted dataset. All the same requirements as for task (a) apply.

Filtering dataset to contain data where purchase = 1

```
dataFiltered = data[data['Purchase'] == 1]
xVariablesFiltered = dataFiltered[["US","source_a", "source_c", "source_b", "source_d", \
                   "source_e",  "source_m", "source_o", "source_h",\
                   "source_r","source_s",   "source_t", "source_u", \
                   "source_p",  "source_x", "source_w", "Freq", \
                   "last_update_days_ago",  "1st_update_days_ago",  \
                   "Web order", "Gender=male",  "Address_is_res"]]
yVariableFiltered = dataFiltered[["Spending"]]

xVariablesFilteredStan = xVariablesFiltered
yVariableFilteredStan = yVariableFiltered
```

Min Max Scaling for filtered Dataset

```
scalerFreq = MinMaxScaler(feature_range=(0,1))
scalerLastUpdateDaysAgo = MinMaxScaler(feature_range=(0,1))
scalerFirstUpdateDaysAgo = MinMaxScaler(feature_range=(0,1))
#Yscaler = MinMaxScaler(feature_range=(0,1))
xVariablesFiltered["Freq"] = scalerFreq.fit_transform(xVariablesFiltered[["Freq"]])
xVariablesFiltered["last_update_days_ago"] = \
    scalerLastUpdateDaysAgo.fit_transform(xVariablesFiltered[["last_update_days_ago"]])
xVariablesFiltered["1st_update_days_ago"] = \
    scalerFirstUpdateDaysAgo.fit_transform(xVariablesFiltered[["1st_update_days_ago"]])
```

Normalizing Columns

```
xVariablesFilteredStan["Freq"] = StandardScaler().fit_transform(xVariablesFilteredStan[["Freq"]])
xVariablesFilteredStan["last_update_days_ago"] = \
    StandardScaler().fit_transform(xVariablesFilteredStan[["last_update_days_ago"]])
xVariablesFilteredStan["1st_update_days_ago"] = \
    StandardScaler().fit_transform(xVariablesFilteredStan[["1st_update_days_ago"]])
```

Linear Regression

```
reg = LinearRegression()
nested_score = cross_validate(reg, X=xVariables, y=yVariable, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
reg = LinearRegression()
nested_score = cross_validate(reg, X=xVariablesStandard, y=yVariableStandarad, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -16491.633380178715 Standard Deviation is :  5341.441753470301
Average Mean Squared Error Across All Splits For Normalized Data:  -16491.633380178715 Standard Deviation is :  5341.441753470301
```

Cross Validation on Restricted Dataset

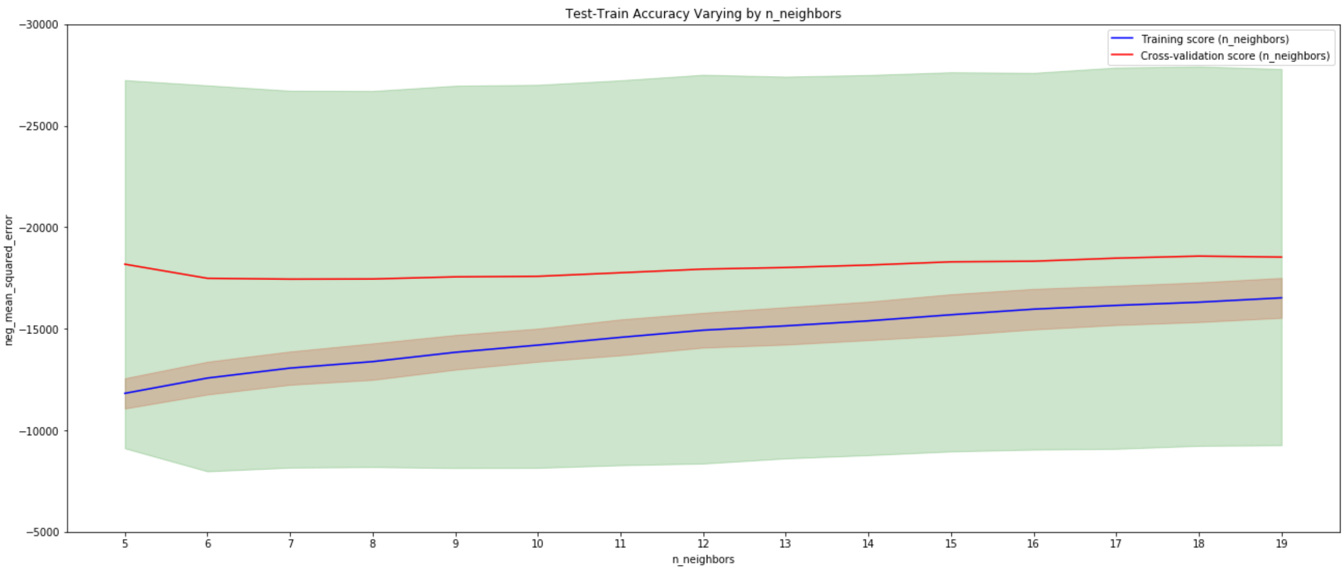
```
reg = LinearRegression()
nested_score = cross_validate(reg, X=xVariablesFiltered, y=yVariableFiltered, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
reg = LinearRegression()
nested_score = cross_validate(reg, X=xVariablesFilteredStan, y=yVariableFilteredStan, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -27493.07288227803 Standard Deviation is :  8119.277352782083
Average Mean Squared Error Across All Splits For Normalized Data:  -27493.07288227803 Standard Deviation is :  8119.277352782083
```

KNN

Hyperparameter Visualizations

```
n = np.arange(5,20)
metric = ['euclidean','manhattan','chebyshev','minkowski']
weights = ['uniform','distance']
parameters_dict = dict(n_neighbors=n, weights=weights, metric=metric)

train_scores, test_scores = validation_curve(
    KNeighborsRegressor(), X_train, y_train, param_name="n_neighbors", cv=10,
    param_range=parameters_dict['n_neighbors'],scoring="neg_mean_squared_error")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))

plt.title("Test-Train Accuracy Varying by n_neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("neg_mean_squared_error")
plt.ylim(-5000,-30000)
plt.fill_between(parameters_dict['n_neighbors'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['n_neighbors'], meanTrainScore, label="Training score (n_neighbors)",
             color="b")
plt.fill_between(parameters_dict['n_neighbors'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['n_neighbors'], meanTestScore, label="Cross-validation score (n_neighbors)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['n_neighbors'])
plt.show()
```




```
n = np.arange(5,20)
```

```
metric = ['euclidean','manhattan','chebyshev','minkowski']
weights = ['uniform','distance']
parameters_dict = dict(n_neighbors=n, weights=weights, metric=metric)
neigh = KNeighborsRegressor()
KNN = GridSearchCV(estimator=neigh, param_grid=parameters_dict, cv=5)
nested_score_KNN = cross_validate(KNN, X=xVariables, y=yVariable, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score_KNN['test_score'].mean(),
     "Standard Deviation is : ",nested_score_KNN['test_score'].std())

nested_score_KNN = cross_validate(KNN, X=xVariablesStandard, y=yVariableStandarad, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score_KNN['test_score'].mean(),
     "Standard Deviation is : ",nested_score_KNN['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -17780.169305920914 Standard Deviation is :  5679.450003112616

Average Mean Squared Error Across All Splits For Normalized Data:  -17780.169305920914 Standard Deviation is :  5679.450003112616
```

Cross Validation on Restricted Dataset

```
n = np.arange(5,20)
metric = ['euclidean','manhattan','chebyshev','minkowski']
weights = ['uniform','distance']
parameters_dict = dict(n_neighbors=n, weights=weights, metric=metric)
neigh = KNeighborsRegressor()
KNN = GridSearchCV(estimator=neigh, param_grid=parameters_dict, cv=5)
nested_score_KNN = cross_validate(KNN, X=xVariablesFiltered, y=yVariableFiltered, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score_KNN['test_score'].mean(),
     "Standard Deviation is : ",nested_score_KNN['test_score'].std())

nested_score_KNN = cross_validate(KNN, X=xVariablesFilteredStan, y=yVariableFilteredStan, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score_KNN['test_score'].mean(),
     "Standard Deviation is : ",nested_score_KNN['test_score'].std())
```

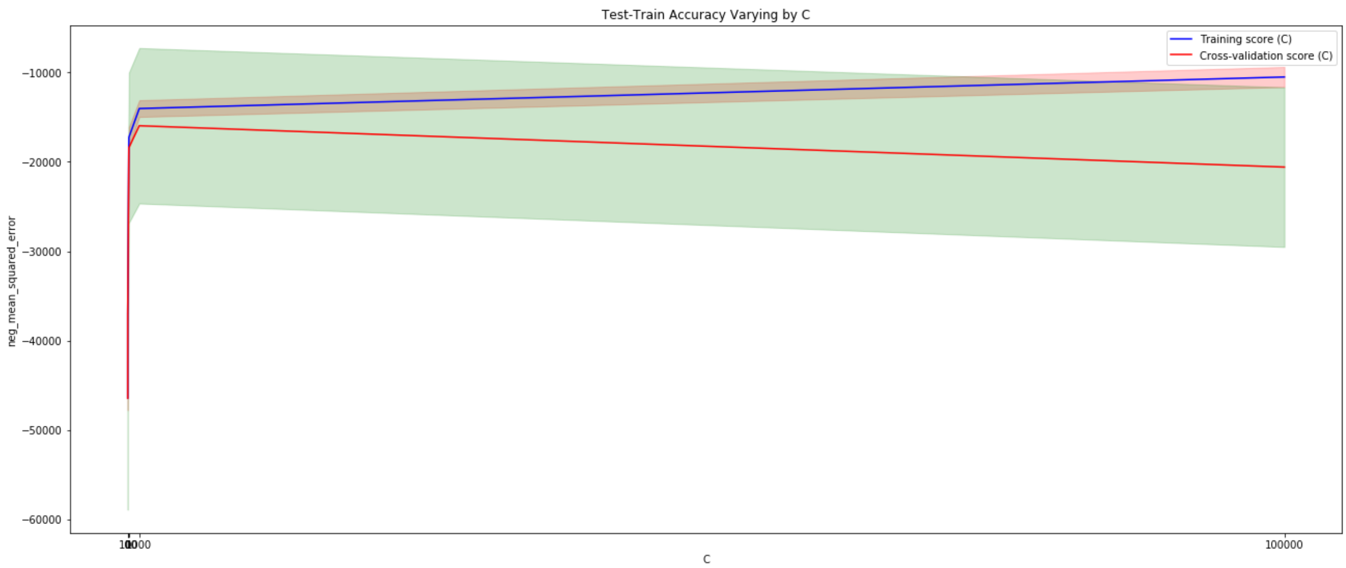
```
Average Mean Squared Error Across All Splits For Min Max Data:  -27938.161086667627 Standard Deviation is :  7727.001156489883

Average Mean Squared Error Across All Splits For Normalized Data:  -27938.161086667627 Standard Deviation is :  7727.001156489883
```

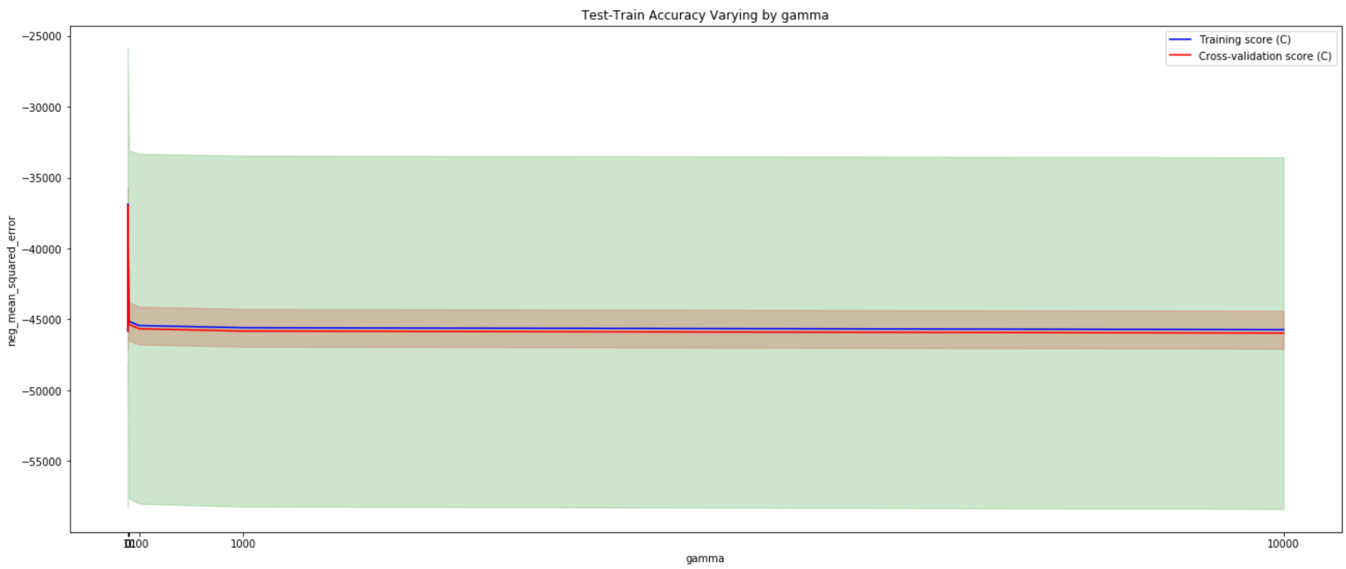
SVR

Hyperparameter Visualizations

```
C = [0.001,0.1,1,100,1000,1e5]
kernel = ['rbf','linear']
gamma = [0.001,0.01,0.1,10,100,1000,10000]
parameters_dict = dict(C=C, kernel = kernel, gamma = gamma)

train_scores, test_scores = validation_curve(
    SVR(max_iter = 15000), X_train, y_train, param_name="C", cv=10,
    param_range=parameters_dict['C'],scoring="neg_mean_squared_error")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))

plt.title("Test-Train Accuracy Varying by C")
plt.xlabel("C")
plt.ylabel("neg_mean_squared_error")
plt.fill_between(parameters_dict['C'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['C'], meanTrainScore, label="Training score (C)",
             color="b")
plt.fill_between(parameters_dict['C'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['C'], meanTestScore, label="Cross-validation score (C)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['C'])
plt.show()

###################################################################################################

train_scores, test_scores = validation_curve(
    SVR(max_iter = 15000), X_train, y_train, param_name="gamma", cv=10,
    param_range=parameters_dict['gamma'],scoring="neg_mean_squared_error")
```

```
#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))

plt.title("Test-Train Accuracy Varying by gamma")
plt.xlabel("gamma")
plt.ylabel("neg_mean_squared_error")
plt.fill_between(parameters_dict['gamma'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['gamma'], meanTrainScore, label="Training score (C)",
             color="b")
plt.fill_between(parameters_dict['gamma'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['gamma'], meanTestScore, label="Cross-validation score (C)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['gamma'])
plt.show()
```





Nested Cross Validation Performance

```
C = [1,100,1000,1e5]
kernel = ['rbf']
gamma = [0.1,10,100,1000]
parameters_dict = dict(C=C, kernel = kernel, gamma = gamma)
svr = SVR(max_iter = 1000)
svrgrid = GridSearchCV(estimator=svr, param_grid=parameters_dict, cv=5)
nested_score_svr_filtered = cross_validate(svrgrid, X=xVariables, \
                                       y=np.array(yVariable).ravel(), cv=5,
                                   scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score_svr_filtered['test_score'].mean(),
```

```
    "Standard Deviation is : ",nested_score_svr_filtered['test_score'].std())

nested_score_svr_filtered = cross_validate(svrgrid, X=xVariablesStandard, y=yVariableStandarad, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score_svr_filtered['test_score'].mean(),
     "Standard Deviation is : ",nested_score_svr_filtered['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -16585.58628143141 Standard Deviation is :  6541.599501200509

Average Mean Squared Error Across All Splits For Normalized Data:  -16585.58628143141 Standard Deviation is :  6541.599501200509
```

Nested Cross Validation Performance on Restricted Dataset

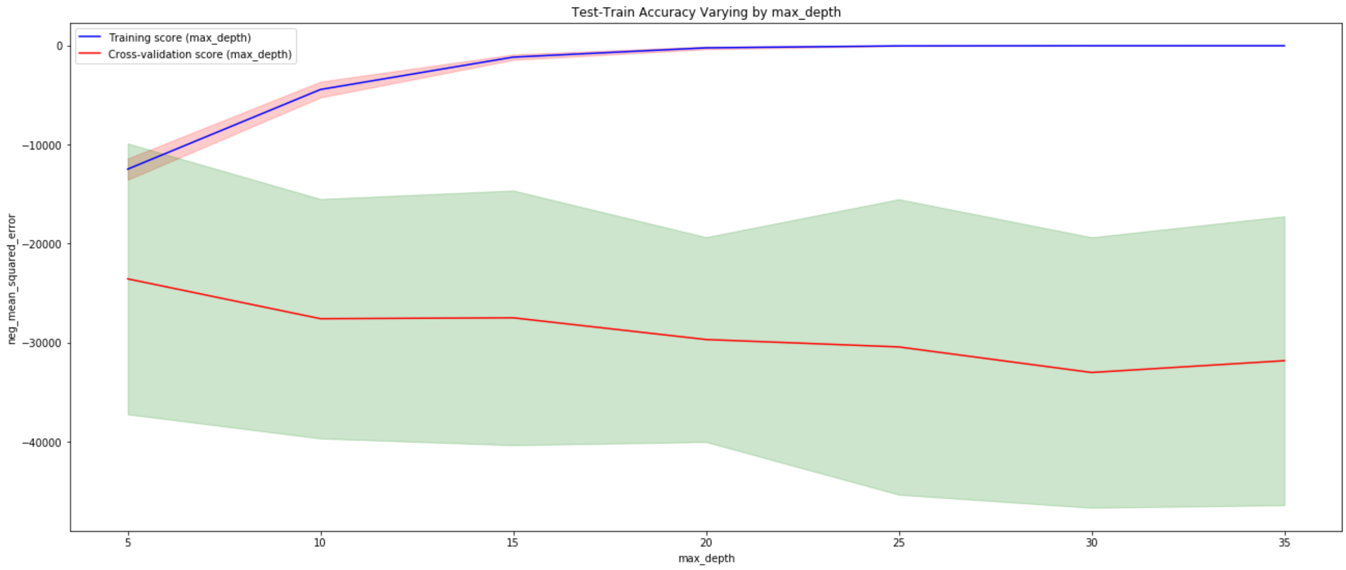
```
C = [1,100,1000,1e5]
kernel = ['rbf']
gamma = [0.1,10,100,1000]
parameters_dict = dict(C=C, kernel = kernel, gamma = gamma)
svr = SVR(max_iter = 1000)
svrgrid = GridSearchCV(estimator=svr, param_grid=parameters_dict, cv=3)
nested_score_svr_filtered = cross_validate(svrgrid, X=xVariablesFiltered, \
                                  y=np.array(yVariableFiltered).ravel(), cv=3,
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score_svr_filtered['test_score'].mean(),
     "Standard Deviation is : ",nested_score_svr_filtered['test_score'].std())

nested_score_svr_filtered = cross_validate(svrgrid, X=xVariablesFilteredStan, y=yVariableFilteredStan, cv=3, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score_svr_filtered['test_score'].mean(),
     "Standard Deviation is : ",nested_score_svr_filtered['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -25732.088161164254 Standard Deviation is :  8276.59815425903

Average Mean Squared Error Across All Splits For Normalized Data:  -25732.088161164254 Standard Deviation is :  8276.59815425903
```

Decision Tree Regressor

Hyperparameter Visualizations

```
criterion = ['mse', 'friedman_mse','mae']
splitter = ['best','random']
max_depth = np.arange(5,40,5)
parameters_dict = dict(criterion=criterion,splitter=splitter,max_depth=max_depth)
train_scores, test_scores = validation_curve(
    DecisionTreeRegressor(), X_train, y_train, param_name="max_depth", cv=10,
    param_range=parameters_dict['max_depth'],scoring="neg_mean_squared_error")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))

plt.title("Test-Train Accuracy Varying by max_depth")
plt.xlabel("max_depth")
plt.ylabel("neg_mean_squared_error")
plt.fill_between(parameters_dict['max_depth'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['max_depth'], meanTrainScore, label="Training score (max_depth)",
             color="b")
plt.fill_between(parameters_dict['max_depth'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['max_depth'], meanTestScore, label="Cross-validation score (max_depth)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['max_depth'])
plt.show()
```



Nested Cross Validation Performance

```
treegrid = GridSearchCV(estimator=regressor, param_grid=parameters_dict, cv=5)
nested_score_tree = cross_validate(treegrid, X=xVariables, \
                                   y=np.array(yVariable).ravel(), cv=5,
                               scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score_tree['test_score'].mean(),
     "Standard Deviation is : ",nested_score_tree['test_score'].std())
nested_score = cross_validate(treegrid, X=xVariablesStandard, y=yVariableStandarad, cv=5, \
                               scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score_tree['test_score'].mean(),
     "Standard Deviation is : ",nested_score_tree['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -18905.56412516842 Standard Deviation is :  5434.059112623316

Average Mean Squared Error Across All Splits For Normalized Data:  -18905.56412516842 Standard Deviation is :  5434.059112623316
```

Nested Cross Validation Performance on Restricted Dataset

```
criterion = ['mse', 'friedman_mse','mae']
splitter = ['best','random']
max_depth = np.arange(5,40,5)
parameters_dict = dict(criterion=criterion,splitter=splitter,max_depth=max_depth)
regressor = DecisionTreeRegressor()
treegrid = GridSearchCV(estimator=regressor, param_grid=parameters_dict, cv=5)
nested_score_tree = cross_validate(treegrid, X=xVariablesFiltered, \
                                   y=np.array(yVariableFiltered).ravel(), cv=5,
                               scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score_tree['test_score'].mean(),
     "Standard Deviation is : ",nested_score_tree['test_score'].std())
nested_score_tree = cross_validate(treegrid, X=xVariablesFilteredStan, y=yVariableFilteredStan, cv=5, \
                               scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score_tree['test_score'].mean(),
     "Standard Deviation is : ",nested_score_tree['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -31293.579675649642 Standard Deviation is :  10264.04926041689

Average Mean Squared Error Across All Splits For Normalized Data:  -33432.88976761054 Standard Deviation is :  12123.252492521222
```

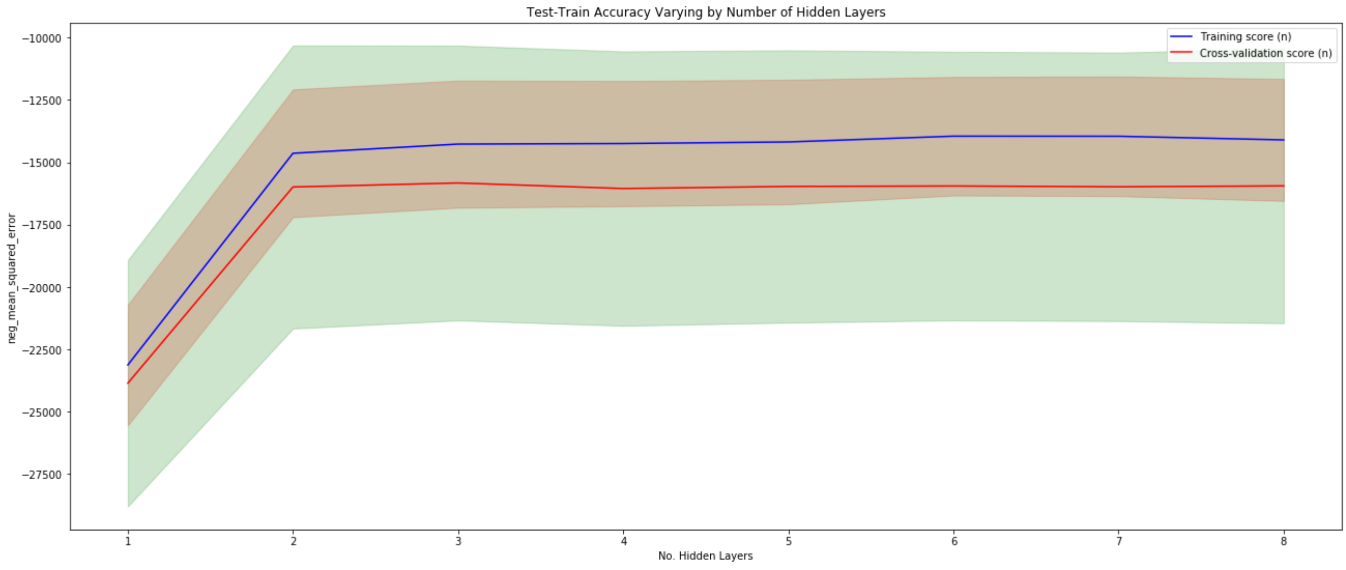
Neural Network

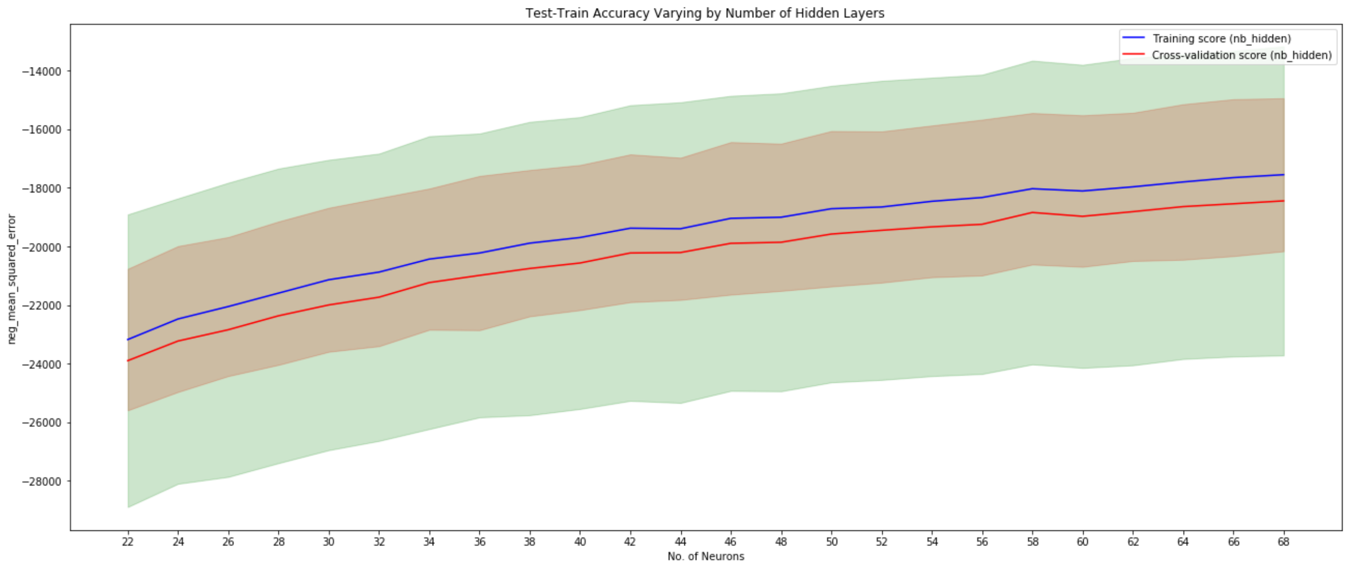
```
def create_model(n = 1,activation='relu',nb_hidden=22,init_mode='uniform',optimizer='adam'):
    #print(n,activation,nb_hidden)
    model = keras.Sequential()
    model.add(layers.Dense(nb_hidden,
                    input_dim=22, kernel_initializer=init_mode,
                    activation=activation))
    if n > 1:
        for i in range(n):
            model.add(layers.Dense(nb_hidden,
                        kernel_initializer=init_mode,
                        activation=activation))

    model.add(layers.Dense(1, kernel_initializer=init_mode, activation='linear'))
    # Compile model
    model.compile(loss='mse',
                  optimizer=optimizer,
                  metrics=['mean_squared_error'])
```

```
    #print(model.summary())
    return model

activations = ['relu']
nb_hiddens = np.arange(22, 70, 2)
ns = np.array([1,2,3,4,5,6,7,8])
from keras.wrappers.scikit_learn import KerasRegressor
parameters_dict = dict(activation=activations, nb_hidden=nb_hiddens, n = ns)
model = KerasRegressor(build_fn=create_model, epochs=30, verbose=0)
```

Hyperparameter Visualizations

```
## Dictionary that holds different parameters and their values.
train_scores, test_scores = validation_curve(
    model, StandardScaler().fit_transform(xVariables.values), yVariable.values, param_name="n", cv=3,
    param_range=parameters_dict['n'],scoring="neg_mean_squared_error")

meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))

plt.title("Test-Train Accuracy Varying by Number of Hidden Layers")
plt.xlabel("No. Hidden Layers")
plt.ylabel("neg_mean_squared_error")
plt.fill_between(parameters_dict['n'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['n'], meanTrainScore, label="Training score (n)",
             color="b")
plt.fill_between(parameters_dict['n'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['n'], meanTestScore, label="Cross-validation score (n)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['n'])
plt.show()

####################################################################################################

train_scores, test_scores = validation_curve(
    model, StandardScaler().fit_transform(xVariables.values), yVariable.values, param_name="nb_hidden", cv=3,
    param_range=parameters_dict['nb_hidden'],scoring="neg_mean_squared_error")

meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))

plt.title("Test-Train Accuracy Varying by Number of Hidden Layers")
plt.xlabel("No. of Neurons")
plt.ylabel("neg_mean_squared_error")
plt.fill_between(parameters_dict['nb_hidden'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['nb_hidden'], meanTrainScore, label="Training score (nb_hidden)",
             color="b")
plt.fill_between(parameters_dict['nb_hidden'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['nb_hidden'], meanTestScore, label="Cross-validation score (nb_hidden)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['nb_hidden'])
plt.show()
```





Nested Cross Validation Performance

```
activations = ['relu']
nb_hiddens = np.arange(70,80, 5)
ns = np.array([2,3,4])
parameters_dict = dict(activation=activations, nb_hidden=nb_hiddens, n = ns)
NN = GridSearchCV(model, param_grid = parameters_dict, cv= 5)
nested_score_NN = cross_validate(NN, X=xVariables.values, y=yVariable.values, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score_NN['test_score'].mean(),
     "Standard Deviation is : ",nested_score_NN['test_score'].std())
nested_score_NN = cross_validate(NN, X=xVariablesStandard.values, y=yVariableStandarad.values, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score_NN['test_score'].mean(),
     "Standard Deviation is : ",nested_score_NN['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -15729.540172068044 Standard Deviation is :  5213.087157284444

Average Mean Squared Error Across All Splits For Normalized Data:  -15467.049262421113 Standard Deviation is :  5187.185422457909
```

Nested Cross Validation Performance on Restricted Dataset

```
activations = ['relu']
nb_hiddens = np.arange(70,80, 5)
ns = np.array([2,3,4])
parameters_dict = dict(activation=activations, nb_hidden=nb_hiddens, n = ns)
NN = GridSearchCV(model, param_grid = parameters_dict, cv= 3)
nested_score_NN = cross_validate(NN, X=xVariablesFiltered.values, y=yVariableFiltered.values, cv=3, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score_NN['test_score'].mean(),
     "Standard Deviation is : ",nested_score_NN['test_score'].std())
nested_score_NN = cross_validate(NN, X=xVariablesFilteredStan.values, y=yVariableFilteredStan.values, cv=3, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score_NN['test_score'].mean(),
     "Standard Deviation is : ",nested_score_NN['test_score'].std())
```

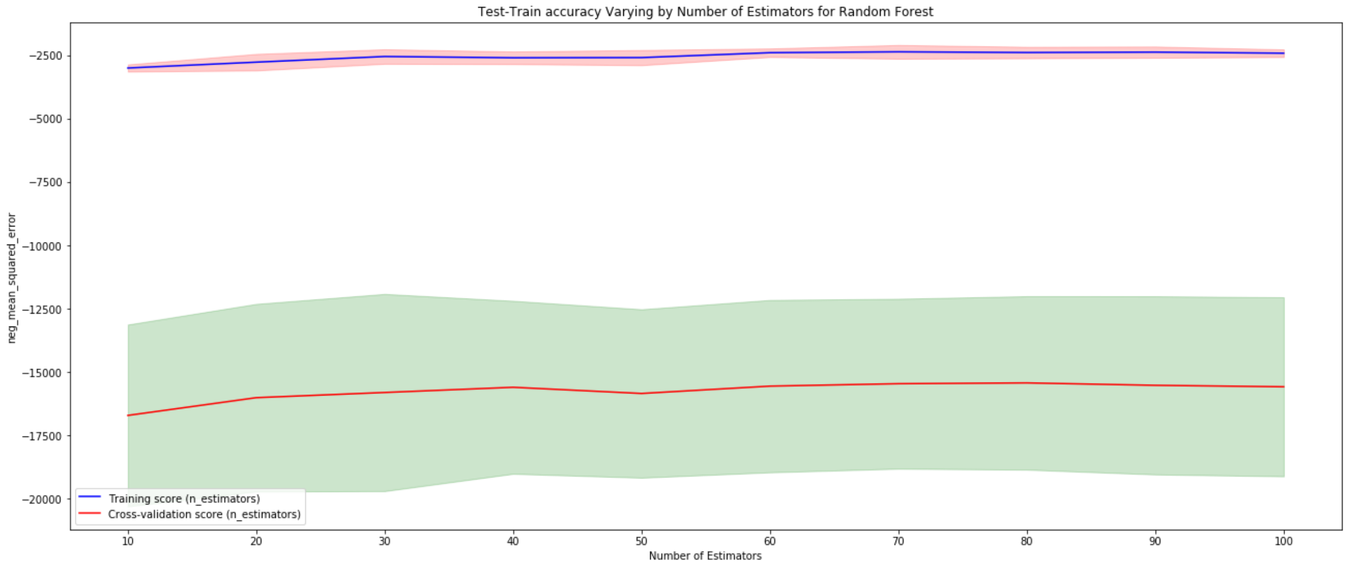
```
Average Mean Squared Error Across All Splits For Min Max Data:  -25820.687119948492 Standard Deviation is :  7462.782362610176

Average Mean Squared Error Across All Splits For Normalized Data:  -26561.243049642446 Standard Deviation is :  5904.586246211025
```

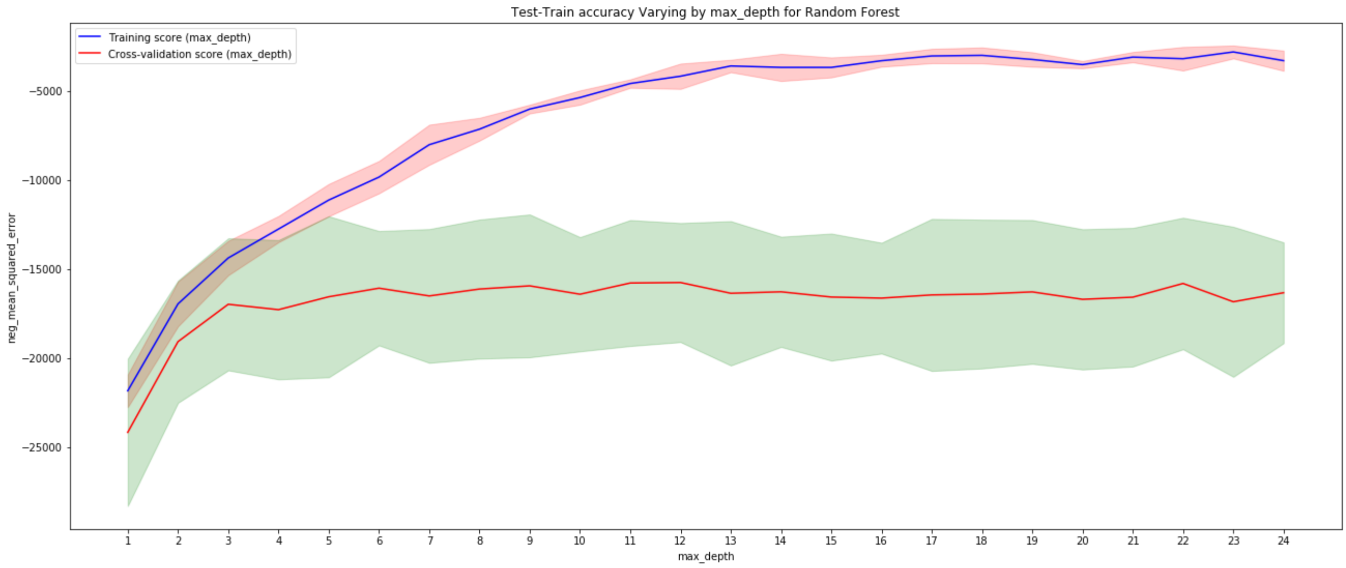
Random Forest Regressor

Hyperparameter Visualization

```
parameters_dict = {
    'bootstrap': [True],
    'max_depth': [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [10,20,30,40,50,60,70,80,90,100]

}
train_scores, test_scores = validation_curve(
    RandomForestRegressor(),X_train, y_train, param_name="n_estimators", cv=5,
    param_range=parameters_dict['n_estimators'],scoring="neg_mean_squared_error")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train accuracy Varying by Number of Estimators for Random Forest")
plt.xlabel("Number of Estimators")
plt.ylabel("neg_mean_squared_error")

plt.fill_between(parameters_dict['n_estimators'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['n_estimators'], meanTrainScore, label="Training score (n_estimators)",
             color="b")
plt.fill_between(parameters_dict['n_estimators'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['n_estimators'], meanTestScore, label="Cross-validation score (n_estimators)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['n_estimators'])
plt.show()

#############################################################################################################

train_scores, test_scores = validation_curve(
    RandomForestRegressor(),X_train, y_train, param_name="max_depth", cv=5,
    param_range=parameters_dict['max_depth'],scoring="neg_mean_squared_error")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train accuracy Varying by max_depth for Random Forest")
plt.xlabel("max_depth")
plt.ylabel("neg_mean_squared_error")
plt.fill_between(parameters_dict['max_depth'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['max_depth'], meanTrainScore, label="Training score (max_depth)",
             color="b")
plt.fill_between(parameters_dict['max_depth'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['max_depth'], meanTestScore, label="Cross-validation score (max_depth)",
             color="r")

plt.legend(loc="best")
```

```
plt.xticks(parameters_dict['max_depth'])
plt.show()
```





Nested Cross Validation Performance

```
parameters_dict = {
    'bootstrap': [True],
    'max_depth': [23,24],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [10,20]}
RandomJhaads = GridSearchCV(RandomForestRegressor(), param_grid = parameters_dict, cv = 3)
nested_score = cross_validate(RandomJhaads, X=xVariables, y=yVariable, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
nested_score = cross_validate(RandomJhaads, X=xVariablesStandard, y=yVariableStandarad, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -16940.626993013644 Standard Deviation is :  6356.129100684072

Average Mean Squared Error Across All Splits For Normalized Data:  -17097.59674021251 Standard Deviation is :  6713.257478105308
```

Nested Cross Validation on Restricted Dataset

```
parameters_dict = {
    'bootstrap': [True],
    'max_depth': [23,24],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [10,20]}
```

```
RandomJhaads = GridSearchCV(RandomForestRegressor(), param_grid = parameters_dict, cv = 3)
nested_score = cross_validate(RandomJhaads, X=xVariablesFiltered, y=yVariableFiltered, cv=3, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
nested_score = cross_validate(RandomJhaads, X=xVariablesFilteredStan, y=yVariableFilteredStan, cv=3, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
```

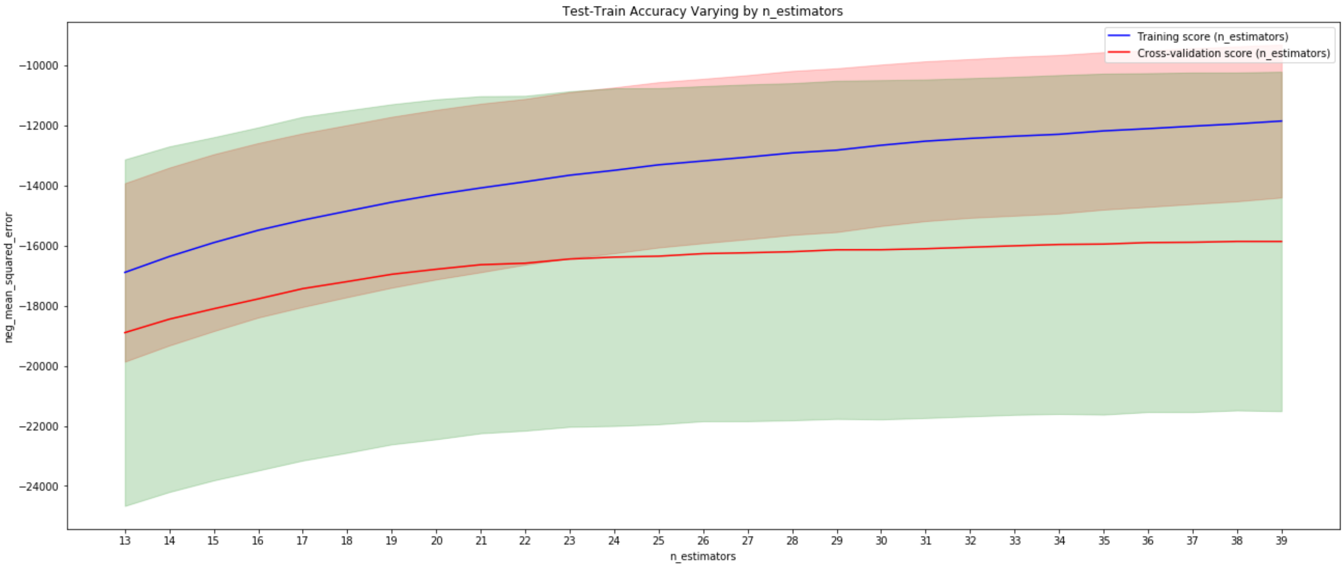
```
Average Mean Squared Error Across All Splits For Min Max Data:  -27794.429399569704 Standard Deviation is :  6992.425551148038

Average Mean Squared Error Across All Splits For Normalized Data:  -27090.920682010754 Standard Deviation is :  7170.501487279928
```

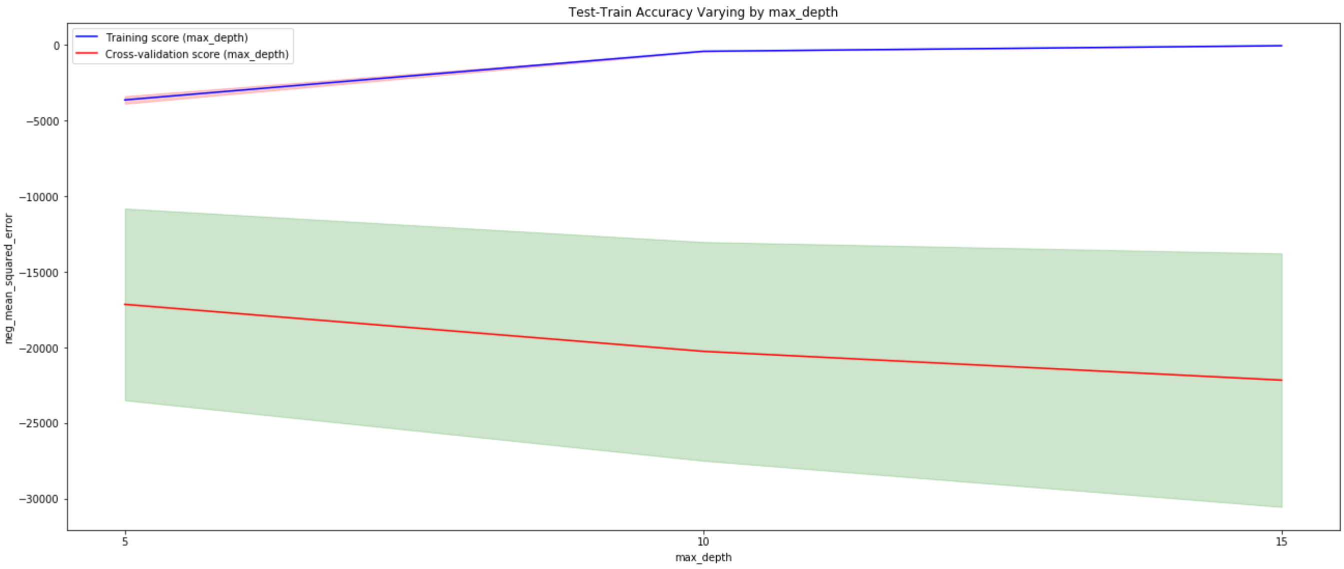
XGBOOST

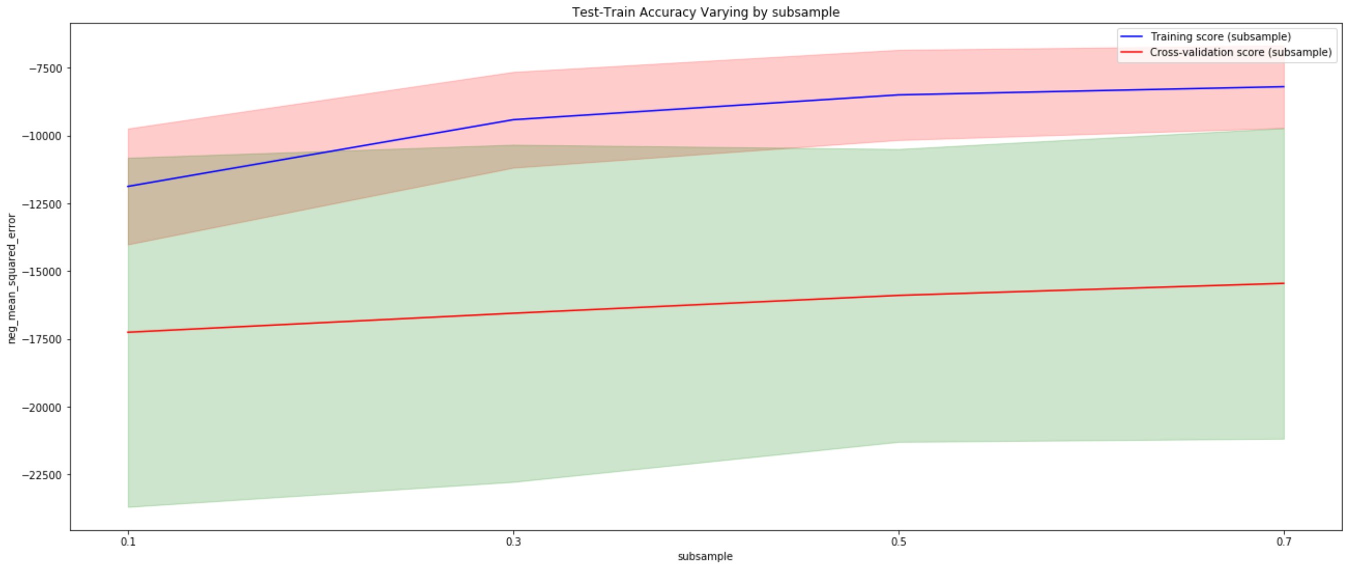
Hyperparameter Visualizations

```
import xgboost as xgb
parameters_dict = {
    'learning_rate': [0.001,0.01,0.1,1],
    'n_estimators': list(range(13,40)),
    'max_depth' : np.arange(5,20,5),
    'subsample':np.arange(0.1,0.8,0.2),
    'gamma' :[0,1,5]
    }
train_scores, test_scores = validation_curve(
    xgb.XGBRegressor(objective = 'reg:squarederror'),xVariables, yVariable, param_name="n_estimators", cv=3,
    param_range=parameters_dict['n_estimators'],scoring="neg_mean_squared_error")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train Accuracy Varying by n_estimators")
plt.xlabel("n_estimators")
plt.ylabel("neg_mean_squared_error")
plt.fill_between(parameters_dict['n_estimators'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['n_estimators'], meanTrainScore, label="Training score (n_estimators)",
             color="b")
plt.fill_between(parameters_dict['n_estimators'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['n_estimators'], meanTestScore, label="Cross-validation score (n_estimators)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['n_estimators'])
plt.show()
#######################################################################
train_scores, test_scores = validation_curve(
    xgb.XGBRegressor(objective = 'reg:squarederror'),xVariables, yVariable, param_name="max_depth", cv=3,
    param_range=parameters_dict['max_depth'],scoring="neg_mean_squared_error")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train Accuracy Varying by max_depth")
plt.xlabel("max_depth")
plt.ylabel("neg_mean_squared_error")
plt.fill_between(parameters_dict['max_depth'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['max_depth'], meanTrainScore, label="Training score (max_depth)",
             color="b")
plt.fill_between(parameters_dict['max_depth'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['max_depth'], meanTestScore, label="Cross-validation score (max_depth)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['max_depth'])
plt.show()

#######################################################################
train_scores, test_scores = validation_curve(
    xgb.XGBRegressor(objective = 'reg:squarederror'),xVariables, yVariable, param_name="subsample", cv=3,
    param_range=parameters_dict['subsample'],scoring="neg_mean_squared_error")
```

```
#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train Accuracy Varying by subsample")
plt.xlabel("subsample")
plt.ylabel("neg_mean_squared_error")
plt.fill_between(parameters_dict['subsample'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['subsample'], meanTrainScore, label="Training score (subsample)",
             color="b")
plt.fill_between(parameters_dict['subsample'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['subsample'], meanTestScore, label="Cross-validation score (subsample)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['subsample'])
plt.show()
```







Nested Cross Validation performance

```
parameters_dict = {
    'learning_rate': [0.001,0.01,0.1,1],
    'n_estimators': [25,25,38],
    'max_depth' : [5,7,8],
    'subsample':[0.1,0.7]
    }
Boost = GridSearchCV(xgb.XGBRegressor(objective = 'reg:squarederror'), param_grid = parameters_dict, cv = 3)
nested_score = cross_validate(Boost, X=xVariables, y=yVariable, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
nested_score = cross_validate(Boost, X=xVariablesStandard, y=yVariableStandarad, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -16245.282637729648 Standard Deviation is :  6197.988698727639
```

```
Average Mean Squared Error Across All Splits For Normalized Data:  -16245.282637729648 Standard Deviation is :  6197.988698727639
```

Nested Cross Validation on Restricted Dataset

```
parameters_dict = {
    'learning_rate': [0.001,0.01,0.1,1],
    'n_estimators': [25,25,38],
    'max_depth' : [5,7,8],
    'subsample':[0.1,0.7]
    }
Boost = GridSearchCV(xgb.XGBRegressor(objective = 'reg:squarederror'), param_grid = parameters_dict, cv = 3)
nested_score = cross_validate(Boost, X=xVariablesFiltered, y=yVariableFiltered, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
nested_score = cross_validate(Boost, X=xVariablesFilteredStan, y=yVariableFilteredStan, cv=5, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
```

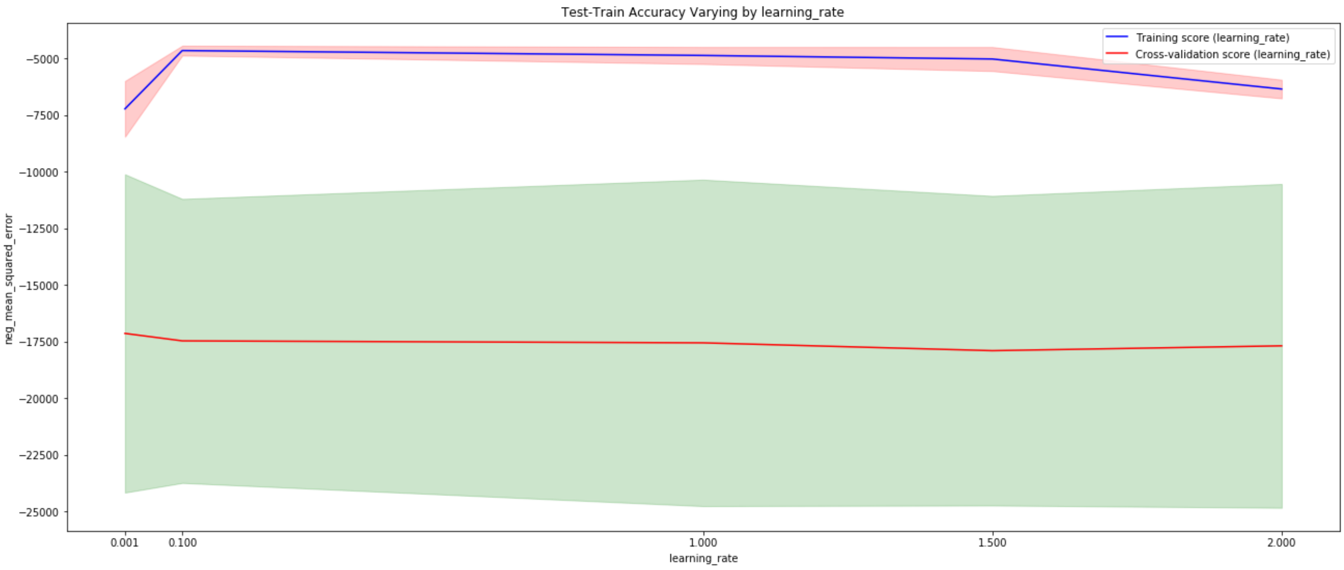
```
Average Mean Squared Error Across All Splits For Min Max Data:  -27138.54933220382 Standard Deviation is :  7083.9872654153205

Average Mean Squared Error Across All Splits For Normalized Data:  -27138.54933220382 Standard Deviation is :  7083.9872654153205
```

AdaBoost

Hyperparameter Visualizations

```
parameters_dict = {
    'n_estimators': list(range(13,16)),
    'learning_rate' : [0.001,0.1,1,1.5,2]
    }
train_scores, test_scores = validation_curve(
    AdaBoostRegressor(DecisionTreeRegressor(max_depth=8)),xVariables, yVariable, param_name="learning_rate", cv=5,
    param_range=parameters_dict['learning_rate'],scoring="neg_mean_squared_error")

#Calculating mean and standard deviations of the scores
```

```
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train Accuracy Varying by learning_rate")
plt.xlabel("learning_rate")
plt.ylabel("neg_mean_squared_error")
plt.fill_between(parameters_dict['learning_rate'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['learning_rate'], meanTrainScore, label="Training score (learning_rate)",
             color="b")
plt.fill_between(parameters_dict['learning_rate'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['learning_rate'], meanTestScore, label="Cross-validation score (learning_rate)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['learning_rate'])
plt.show()
```



Nested Cross Validation Performance

```
parameters_dict = {
    'n_estimators': list(range(13,16)),
    'learning_rate' : [0.001,0.1,1,1.5,2]
    }
AdaB = GridSearchCV(AdaBoostRegressor(DecisionTreeRegressor(max_depth=8)),cv = 3, param_grid = parameters_dict)
nested_score = cross_validate(AdaB, X=xVariables, y=yVariable, cv=3, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
nested_score = cross_validate(AdaB, X=xVariablesStandard, y=yVariableStandarad, cv=3, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -17842.87604067896 Standard Deviation is :  6491.305980248762

Average Mean Squared Error Across All Splits For Normalized Data:  -18145.88141291543 Standard Deviation is :  7761.746399859438
```

Nested Cross Validation Peformance on Restricted Dataset

```
parameters_dict = {
    'n_estimators': list(range(13,16)),
    'learning_rate' : [0.001,0.1,1,1.5,2]
    }
AdaB = GridSearchCV(AdaBoostRegressor(DecisionTreeRegressor(max_depth=8)),cv = 3, param_grid = parameters_dict)
nested_score = cross_validate(AdaB, X=xVariablesFiltered, y=yVariableFiltered, cv=3, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
nested_score = cross_validate(AdaB, X=xVariablesFilteredStan, y=yVariableFilteredStan, cv=3, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -26707.829729356337 Standard Deviation is :  8784.450688694713

Average Mean Squared Error Across All Splits For Normalized Data:  -27783.791481726366 Standard Deviation is :  9888.572742791785
```

Stacking

```
from mlxtend.regressor import StackingRegressor
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
lr = LinearRegression()
svr_lin = SVR(kernel='linear')
ridge = Ridge(random_state=1)
lasso = Lasso(random_state=1)
svr_rbf = SVR(kernel='rbf')
regressors = [svr_lin, lr, ridge, lasso]
stregr = StackingRegressor(regressors=regressors,
                           meta_regressor=svr_rbf)

params = {'lasso__alpha': [0.1,1.0, 10.0],
          'ridge__alpha': [0.1,1.0, 10.0],
          'svr__C': [0.1,1.0, 10.0]}

grid = GridSearchCV(estimator=stregr,
                    param_grid=params,
                    cv=3,
                    refit=True)
nested_score = cross_validate(grid, X=xVariables, y=yVariable, cv=3, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
nested_score = cross_validate(grid, X=xVariablesStandard, y=yVariableStandarad, cv=3, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -41691.94798190915 Standard Deviation is :  6964.111641012676

Average Mean Squared Error Across All Splits For Normalized Data:  -41691.94798190915 Standard Deviation is :  6964.111641012676
```

Nested Cross Validation Restricted Dataset

```
nested_score = cross_validate(grid, X=xVariablesFiltered, y=yVariableFiltered, cv=3, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Min Max Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
nested_score = cross_validate(grid, X=xVariablesFilteredStan, y=yVariableFilteredStan, cv=3, \
                              scoring = 'neg_mean_squared_error')
print("Average Mean Squared Error Across All Splits For Normalized Data: ", nested_score['test_score'].mean(),
     "Standard Deviation is : ",nested_score['test_score'].std())
```

```
Average Mean Squared Error Across All Splits For Min Max Data:  -51281.08574201525 Standard Deviation is :  6222.337995060803

Average Mean Squared Error Across All Splits For Normalized Data:  -51281.08574201525 Standard Deviation is :  6222.337995060803
```

Performance Evaluation of All Models

```
performanceData = {"Min Max Mean":[-16491.633,-17780.169,-16585.586,-18905.564,-15729.54,-16940.62,-16245.282,-17842.87,-41691.94]
,"Min Max Std Dev":[5341.441,5679.45,6541.5995,5434.05,5213.08,6356.129,6197.98,6491.305,6964.1]
,"Normalized Dataset Mean":[-16491.633,-17780.169,-16585.586,-18905.564,-15467.049,-17097.59,-16245.28,-18145.881,-41691.94]
,"Normalized Dataset Std Dev":[5341.441,5679.45,6541.5995,5434.05,5187.18,6713.257,6197.98,7761.7463,6964.1]
,"Min Max Mean (Restricted Dataset)":
[-27493.072882278,-27938.161,-25732.088,-31293.579,-25820.687,-27794.42,-27138.549,-26707.8297,-51281.08574]
,"Min Max Std Dev (Restricted Dataset)":[8119.2773,7727.001,8276.598,10264.04,7462.78236,6992.42,7083.98,8784.45,6222.3]
,"Normalized Dataset Mean (Restricted Dataset)":
[-27493.072882278,-27938.161,-25732.088,-33432.88,-26561.243,-27090.92,-27138.549,-27783.7914,-51281.08574]
,"Normalized Dataset Std Dev (Restricted Dataset)":[8119.2773,7727,8276.598,12123.25,5904.58,7170.5,7083.98,9888.572,6222.3]
}
perDf = pd.DataFrame(performanceData)
perDf.index = ['Linear Regression','KNN','SVR','Decision Tree', 'Neural Networks','Random Forest','Xgboost','AdaBoost','Stacking']
perDf
```

	Min Max Mean	Min Max Std Dev	Normalized Dataset Mean	Normalized Dataset Std Dev	Min Max Mean (Restricted Dataset)	Min Max Std Dev (Restricted Dataset)	Normalized Dataset Mean (Restricted Dataset)	Normalized Dataset Std Dev (Restricted Dataset)
Linear Regression	-16491.633	5341.4410	-16491.633	5341.4410	-27493.072882	8119.27730	-27493.072882	8119.2773
KNN	-17780.169	5679.4500	-17780.169	5679.4500	-27938.161000	7727.00100	-27938.161000	7727.0000
SVR	-16585.586	6541.5995	-16585.586	6541.5995	-25732.088000	8276.59800	-25732.088000	8276.5980
Decision Tree	-18905.564	5434.0500	-18905.564	5434.0500	-31293.579000	10264.04000	-33432.880000	12123.2500
Neural Networks	-15729.540	5213.0800	-15467.049	5187.1800	-25820.687000	7462.78236	-26561.243000	5904.5800
Random Forest	-16940.620	6356.1290	-17097.590	6713.2570	-27794.420000	6992.42000	-27090.920000	7170.5000
Xgboost	-16245.282	6197.9800	-16245.280	6197.9800	-27138.549000	7083.98000	-27138.549000	7083.9800
AdaBoost	-17842.870	6491.3050	-18145.881	7761.7463	-26707.829700	8784.45000	-27783.791400	9888.5720
Stacking	-41691.940	6964.1000	-41691.940	6964.1000	-51281.085740	6222.30000	-51281.085740	6222.3000

I have used neg_mean_squared_error as the error term.

All scorer objects follow the convention that higher return values are better than lower return values. Thus metrics which measure the distance between the model and the data, like metrics.mean_squared_error, are available as neg_mean_squared_error which return the negated value of the metric.

Neural Networks have performed the best with the lowest Mean Squared Error value of 15729.54 on the MinMax Dataset and 15467.048 on the Normalized Dataset. Worst performing model is Stacking. Neural networks work better because they can learn any arbitratry curve and fit the data very well.

There is hadrly any difference in performance between the Min Max Scaled and Normalized Data set for all models.

In general all models performed worst on the restricted data set, perhaps maybe the dataset was restricted so there werent as many instances as the original dataset.

Quation 2. (50 points) Download the dataset on spam vs. non-spam emails from the following URL: http://archive.ics.uci.edu/ml/datasets/Spambase. Specifically, (i) file “spambase.data” contains the actual data, and (ii) files “spambase.names” and “spambase.DOCUMENTATION” contain the description of the data. This dataset has 4601 records, each record representing a different email message. Each record is described with 58 attributes (indicated in the aforementioned .names file): attributes 1-57 represent various content-based characteristics already extracted from each email message (related to the frequency of certain words or certain punctuation symbols in a message as well as to the usage of capital letters in a message), and the last attribute represents the class label for each message (spam or non-spam).

Task: The general task for this assignment is to build two different models for detecting spam messages (based on the email characteristics that are given): (i) the best possible model that you can build in terms of the overall predictive accuracy (i.e., not taking any cost information into account), and (ii) the best cost-sensitive classification model that you can build in terms of the average misclassification cost.

Importing data

```
x = pd.read_csv('spambase.data', header = None, usecols = np.arange(0,56,1))
target = pd.read_csv('spambase.data', header = None, usecols = [57], squeeze = True)
x.head()
```

	0	1	2	3	4	5	6	7	8	9	...	46	47	48	49	50	51	52	53
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.0	0.0	0.00	0.000	0.0	0.778	0.000	0.000
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.0	0.0	0.00	0.132	0.0	0.372	0.180	0.048
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.0	0.0	0.01	0.143	0.0	0.276	0.184	0.010
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.0	0.0	0.00	0.137	0.0	0.137	0.000	0.000
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.0	0.0	0.00	0.135	0.0	0.135	0.000	0.000

5 rows × 56 columns

Normalizing the features and also creating a separate minMax scaled dataset

```
standardScaler = StandardScaler()
transformedData = standardScaler.fit_transform(x)
transformedData[1]
minScaler = MinMaxScaler(feature_range=(0,1))
minData = minScaler.fit_transform(x)
```

```
transformedData[1]
```

```
array([ 0.3453594 ,  0.05190919,  0.43512954, -0.04689958, -0.25611729,
        0.67239929,  0.24474325, -0.0880101 , -0.32330236,  1.0867113 ,
        0.74520639,  0.28818107,  1.84739108,  0.4516628 ,  0.35081184,
       -0.1318249 , -0.16348027,  0.17936681,  1.0183704 , -0.16789311,
        0.64983137, -0.11817151,  0.93749108,  0.75856508, -0.32881467,
       -0.29923993, -0.22789481, -0.23183016, -0.16673145, -0.22523952,
       -0.16053931, -0.14321202, -0.17492026, -0.14521515, -0.19806739,
       -0.24213022, -0.15812941, -0.05983624, -0.18091134, -0.18530385,
       -0.12090468, -0.17259996, -0.20599311, -0.12734332, -0.29777621,
       -0.19738748, -0.0713879 , -0.11154623, -0.15845336, -0.02600722,
       -0.15519768,  0.12620315,  0.42378306,  0.00876271, -0.00244327,
        0.25056283])
```

```
minData[1]
```

```
array([0.04625551, 0.01960784, 0.09803922, 0.        , 0.014     ,
       0.04761905, 0.02888583, 0.00630063, 0.        , 0.05170517,
       0.08045977, 0.08169597, 0.11711712, 0.021     , 0.03174603,
       0.007     , 0.00980392, 0.03080308, 0.18506667, 0.        ,
       0.14311431, 0.        , 0.07889908, 0.0344    , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.01015965, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.01353568,
       0.        , 0.01145391, 0.02998501, 0.0024207 , 0.00373491,
       0.01001201])
```

Creating arrays to hold scoring information per class

```
precisionArraySpam = []
precisionArrayNSpam = []
recallArraySpam = []
recallArrayNSpam = []
f1ArraySpam = []
f1ArrayNSpam = []

def emptyArrays():
    precisionArraySpam = []
    precisionArrayNSpam = []
    recallArraySpam = []
    recallArrayNSpam = []
    f1ArraySpam = []
    f1ArrayNSpam = []
    return precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam
```

```
def scoringFunction(y_pred,y_true):
#    print(precision_score(y_pred,y_true, average = None))
    pScores = precision_score(y_pred,y_true, average = None)
    rScores = recall_score(y_pred,y_true, average = None)
    fScores = f1_score(y_pred,y_true, average = None)
    precisionArraySpam.append(pScores[0])
    precisionArrayNSpam.append(pScores[1])
    recallArraySpam.append(rScores[0])
    recallArrayNSpam.append(rScores[1])
    f1ArraySpam.append(fScores[0])
    f1ArrayNSpam.append(fScores[1])

    return accuracy_score(y_pred,y_true)
```

Creating a function that will hold mean and standard deviation of scores for nested cross validation.

```
data = {}
def createScoreDataFrame(precisionArraySpam,recallArraySpam, \
                         f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam):
    data = {"Mean Precision":[np.array(precisionArraySpam).mean(),np.array(precisionArrayNSpam).mean()],\
        "Standard Deviation Precision":[np.array(recallArraySpam).std(),np.array(precisionArrayNSpam).std()],\
        "Mean Recall":[np.array(recallArraySpam).mean(),np.array(recallArrayNSpam).mean()],\
        "Standard Deviation Recall":[np.array(recallArraySpam).std(),np.array(recallArrayNSpam).std()],\
        "Mean F1":[np.array(f1ArraySpam).mean(),np.array(f1ArrayNSpam).mean()],\
        "Standard Deviation F1":[np.array(f1ArraySpam).std(),np.array(f1ArrayNSpam).std()]}
    return pd.DataFrame(data, index=["Spam","Not Spam"])
```

Creating a dictionary that will hold cost matrix values:

```
costMatrix = {'TP':0, 'FP':10, 'TN':0, 'FN':1}
```

Creating a dictionary that will hold the cost and threshold value

```
costDict = {}
costArray = []
```

Creating Function that will calculate cost

```
def calculateCost(y_true, y_pred):
    cost = 0
    confusionMatrix = confusion_matrix(y_true, y_pred)
    tn,fp,fn,tp = confusionMatrix.ravel()
    cost = tn*costMatrix['TN'] + fp*costMatrix['FP']/y_true.size + fn*costMatrix['FN']/y_true.size + tp*costMatrix['TP']
    print("The cost of the model is:", cost)

def cost_score(y_true, y_pred, labels=None, pos_label=1, sample_weight=None):
    cost = 0
    cost += sum(y_pred > y_true)*-1
    cost += sum(y_pred < y_true)*-10

    return cost

cost_scorer = make_scorer(cost_score,greater_is_better=True)
```

Creating a stratified Sample and Nested Cross Validation Folds and Stratified Test-Train Splits

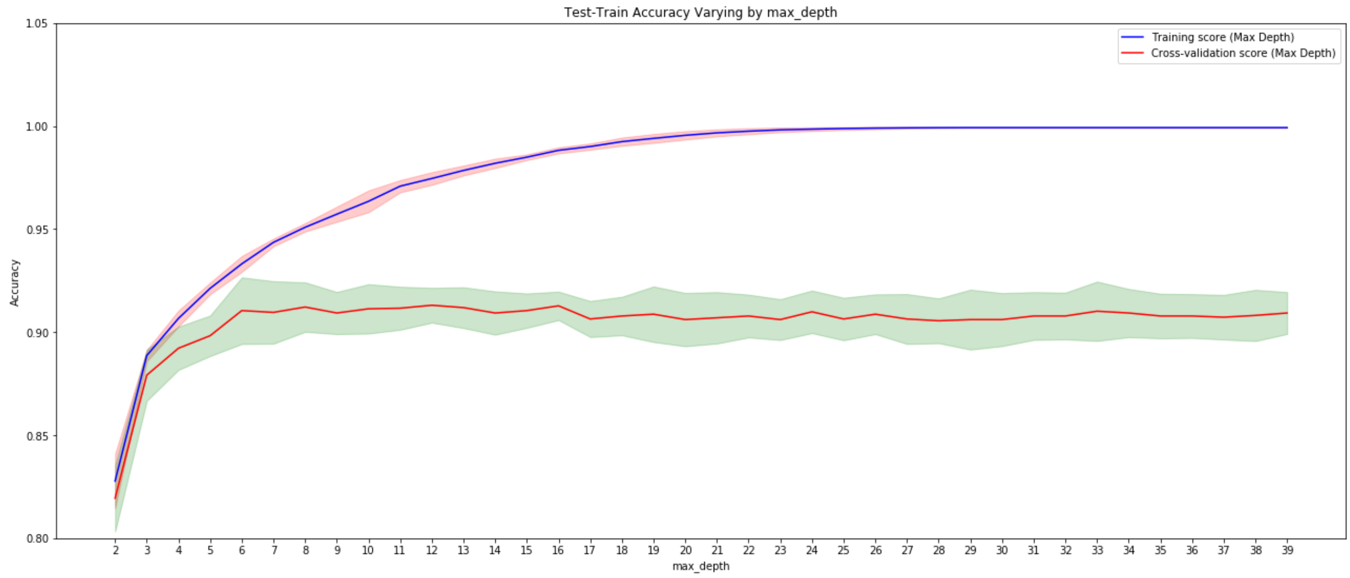
```
inner_cv = StratifiedKFold(n_splits=4, shuffle=True, random_state = 40)
outer_cv = StratifiedKFold(n_splits=4, shuffle=True, random_state = 40)
X_train, X_test, y_train, y_test = train_test_split(transformedData, target,stratify=target, train_size=0.75)
Xmin_train, Xmin_test, ymin_train, ymin_test = train_test_split(minData, target,stratify=target, train_size=0.75)
```

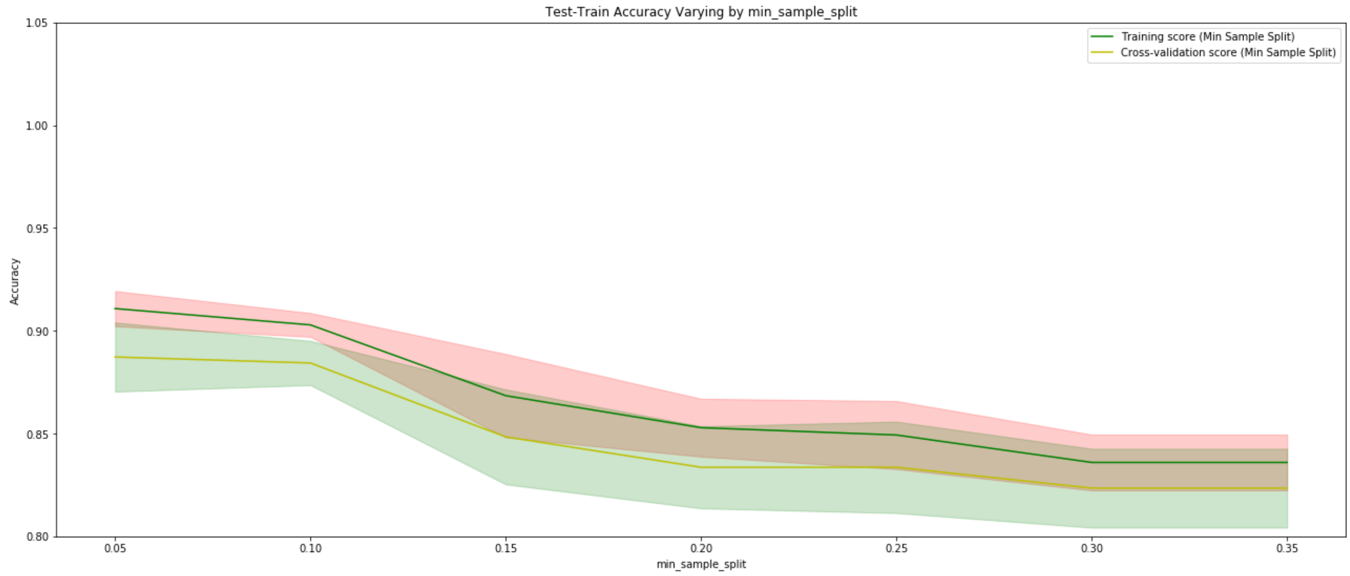
Decision Tree

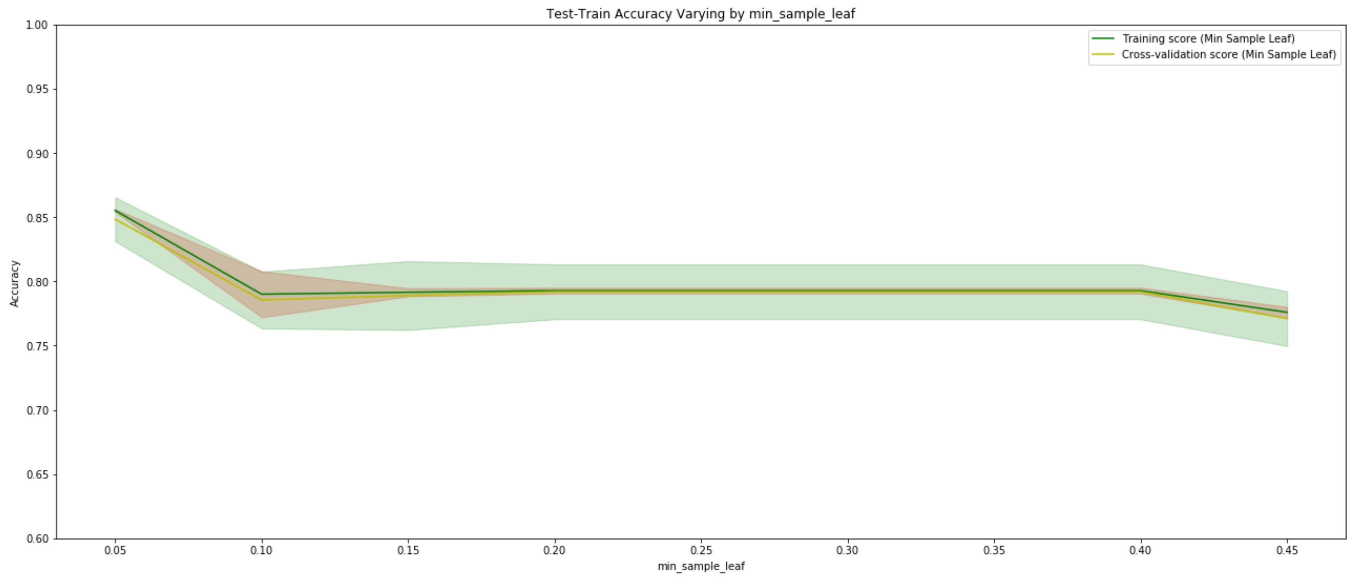
Visualizing Effect of Different Parameters of Decision Tree

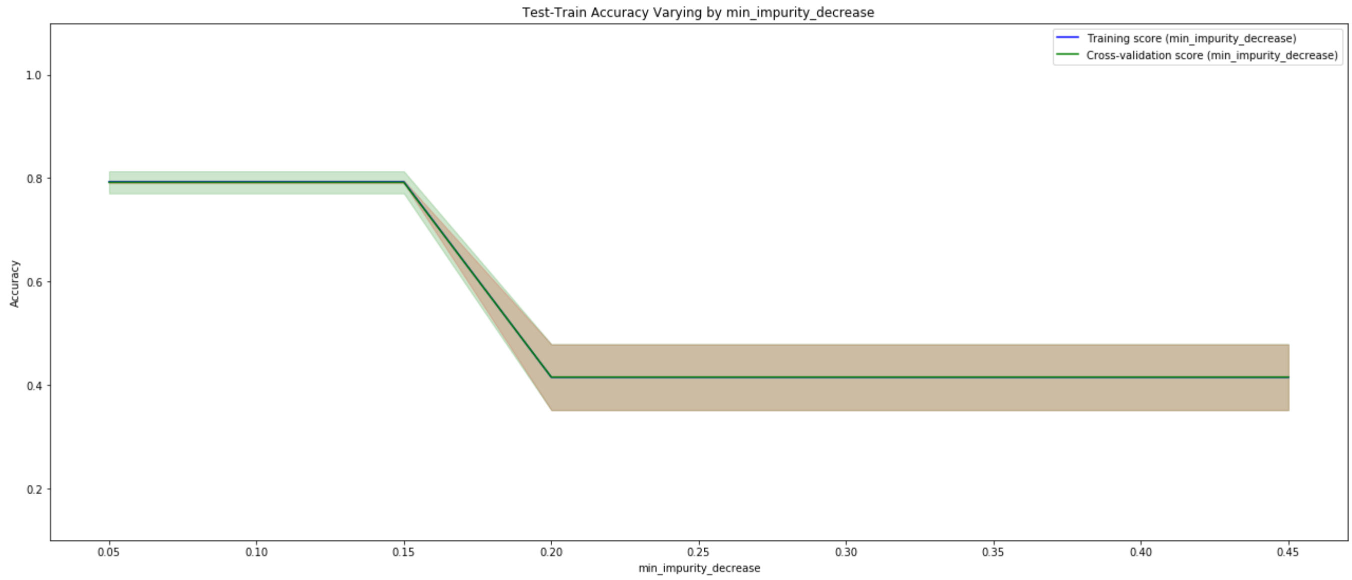
```
## Dictionary that holds different parameters and their values.
parameters_dict = {"max_depth": range(2,40), "min_samples_split" : np.arange(0.05,0.4,0.05), \
                   "min_samples_leaf" : np.arange(0.05,0.5,0.05), "criterion": ["gini","entropy"],
"min_impurity_decrease":np.arange(0.05,0.5,.05)}
train_scores, test_scores = validation_curve(
    DecisionTreeClassifier(class_weight='balanced'), X_train, y_train, param_name="max_depth", cv=10,
    param_range=parameters_dict['max_depth'],scoring="accuracy")

train_scoresRecall, test_scoresRecall = validation_curve(
    DecisionTreeClassifier(class_weight='balanced'), X_train, y_train, param_name="min_samples_split", cv=10,
    param_range=parameters_dict['min_samples_split'],scoring="accuracy")

train_scoresLeaf, test_scoresLeaf = validation_curve(
    DecisionTreeClassifier(class_weight='balanced'), X_train, y_train, param_name="min_samples_leaf", cv=10,
    param_range=parameters_dict['min_samples_leaf'],scoring="accuracy")

train_scoresreduc, test_scoresreduc = validation_curve(
    DecisionTreeClassifier(class_weight='balanced'), X_train, y_train, param_name="min_impurity_decrease", cv=10,
    param_range=parameters_dict['min_impurity_decrease'],scoring="accuracy")


#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

meanTrainScoreRC = np.mean(train_scoresRecall, axis =1)
stdDevTrainRC = np.std(train_scoresRecall, axis=1)
meanTestScoreRC = np.mean(test_scoresRecall, axis=1)
stdTestScoreRC = np.std(test_scoresRecall, axis=1)

meanTrainScoreLeaf = np.mean(train_scoresLeaf, axis =1)
```

```
stdDevTrainLeaf = np.std(train_scoresLeaf, axis=1)
meanTestScoreLeaf = np.mean(test_scoresLeaf, axis=1)
stdTestScoreLeaf = np.std(test_scoresLeaf, axis=1)

meanTrainScorereduc = np.mean(train_scoresreduc, axis =1)
stdDevTrainreduc = np.std(train_scoresreduc, axis=1)
meanTestScorereduc = np.mean(test_scoresreduc, axis=1)
stdTestScorereduc = np.std(test_scoresreduc, axis=1)

#Plotting
plt.figure(figsize=(22,9))

plt.title("Test-Train Accuracy Varying by max_depth")
plt.xlabel("max_depth")
plt.ylabel("Accuracy")
plt.ylim(0.8, 1.05)
plt.fill_between(parameters_dict['max_depth'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['max_depth'], meanTrainScore, label="Training score (Max Depth)",
             color="b")
plt.fill_between(parameters_dict['max_depth'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['max_depth'], meanTestScore, label="Cross-validation score (Max Depth)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['max_depth'])
plt.show()

plt.figure(figsize=(22,9))
plt.title("Test-Train Accuracy Varying by min_sample_split")
plt.xlabel("min_sample_split")
plt.ylabel("Accuracy")
plt.ylim(0.8, 1.05)
plt.fill_between(parameters_dict['min_samples_split'], meanTrainScoreRC - stdDevTrainRC, meanTrainScoreRC + stdDevTrainRC, alpha=0.2,
color="r")
plt.plot(parameters_dict['min_samples_split'], meanTrainScoreRC, label="Training score (Min Sample Split)",
             color="g")
plt.fill_between(parameters_dict['min_samples_split'], meanTestScoreRC - stdTestScoreRC, meanTestScoreRC + stdTestScoreRC, alpha=0.2,
color="g")
plt.plot(parameters_dict['min_samples_split'], meanTestScoreRC, label="Cross-validation score (Min Sample Split)",
             color="y")

plt.legend(loc="best")
plt.xticks(parameters_dict['min_samples_split'])
plt.show()

plt.figure(figsize=(22,9))
plt.title("Test-Train Accuracy Varying by min_sample_leaf")
plt.xlabel("min_sample_leaf")
plt.ylabel("Accuracy")
plt.ylim(0.6, 1.0)
plt.fill_between(parameters_dict['min_samples_leaf'], meanTrainScoreLeaf - stdDevTrainLeaf, meanTrainScoreLeaf + stdDevTrainLeaf, alpha=0.2,
color="r")
plt.plot(parameters_dict['min_samples_leaf'], meanTrainScoreLeaf, label="Training score (Min Sample Leaf)",
             color="g")
plt.fill_between(parameters_dict['min_samples_leaf'], meanTestScoreLeaf - stdTestScoreLeaf, meanTestScoreLeaf + stdTestScoreLeaf, alpha=0.2,
color="g")
plt.plot(parameters_dict['min_samples_leaf'], meanTestScoreLeaf, label="Cross-validation score (Min Sample Leaf)",
             color="y")

plt.legend(loc="best")
plt.xticks(parameters_dict['min_samples_leaf'])
plt.show()
plt.figure(figsize=(22,9))
plt.title("Test-Train Accuracy Varying by min_impurity_decrease")
plt.xlabel("min_impurity_decrease")
plt.ylabel("Accuracy")
plt.ylim(0.1, 1.1)
plt.fill_between(parameters_dict['min_impurity_decrease'], meanTrainScorereduc - stdDevTrainreduc, meanTrainScorereduc + stdDevTrainreduc,
alpha=0.2, color="r")
plt.plot(parameters_dict['min_impurity_decrease'], meanTrainScorereduc, label="Training score (min_impurity_decrease)",
             color="b")
plt.fill_between(parameters_dict['min_impurity_decrease'], meanTestScorereduc - stdTestScorereduc, meanTestScorereduc + stdTestScorereduc,
alpha=0.2, color="g")
plt.plot(parameters_dict['min_impurity_decrease'], meanTestScorereduc, label="Cross-validation score (min_impurity_decrease)",
             color="g")

plt.legend(loc="best")
plt.xticks(parameters_dict['min_impurity_decrease'])


plt.show()
```


Test-Train Accuracy Varying by max_depth
Training score (Max Depth)
Cross-validation score (Max Depth)
Accuracy
max_depth


Test-Train Accuracy Varying by min_sample_split
Training score (Min Sample Split)
Cross-validation score (Min Sample Split)
Accuracy
min_sample_split


Test-Train Accuracy Varying by min_sample_leaf
Training score (Min Sample Leaf)
Cross-validation score (Min Sample Leaf)
Accuracy
min_sample_leaf



Decision Tree Performance on Normalized Data

Nested Cross Validation Results

```
parameters_dict = {"max_depth": range(3,10), "min_samples_split" : np.arange(2,10,2), \
                   "min_samples_leaf" : np.arange(2,10,2), "criterion": ["gini"]}
clfTree = GridSearchCV(DecisionTreeClassifier(), parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(clfTree, X=transformedData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Normalized
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.91, Std Deviation: 0.00 For Normalized Data
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.912536	0.012767	0.944405	0.012767	0.928074	0.002017
Not Spam	0.910280	0.017189	0.860457	0.017087	0.884339	0.001960

The best parameter values, Detailed classification report, Confusion Matrix and Feature Importance of Decision Tree

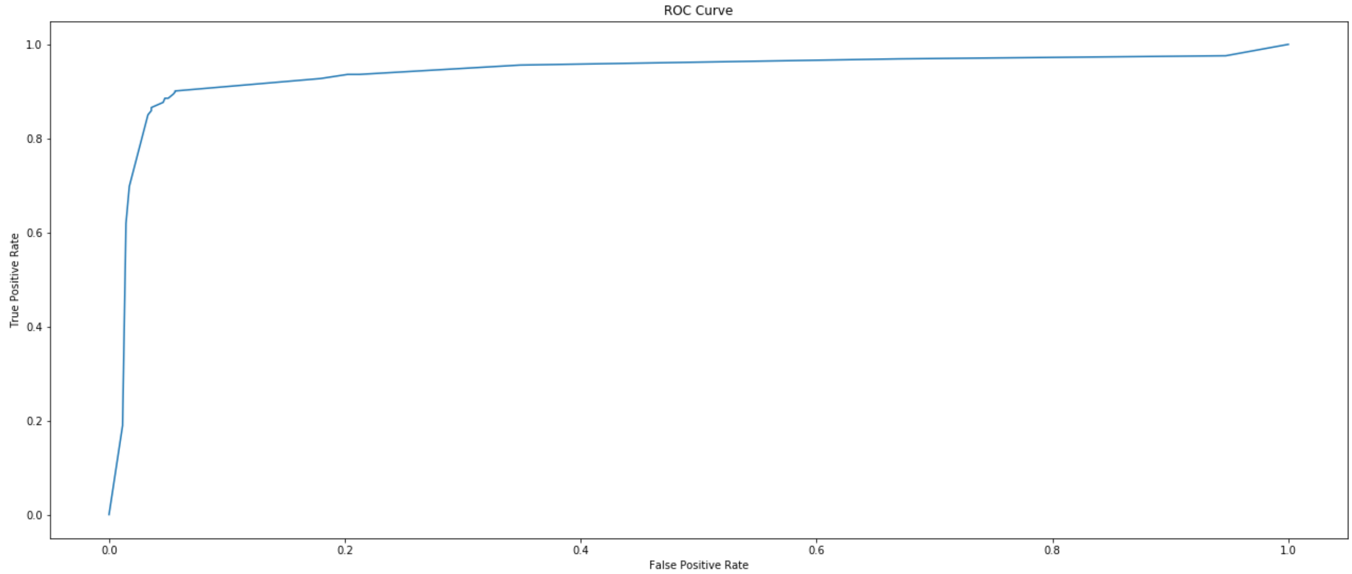
```
clfTree.fit(X_train, y_train)
print('The best parameter values are ',clfTree.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, clfTree.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance (Esitmate of total reduction in entropy brought by a feature)")
print(clfTree.best_estimator_.feature_importances_)
```

```
The best parameter values are  {'criterion': 'gini', 'max_depth': 9, 'min_samples_leaf': 2, 'min_samples_split': 2}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.93      0.95      0.94       697
           1       0.92      0.89      0.90       454

    accuracy                           0.92      1151
   macro avg       0.92      0.92      0.92      1151
weighted avg       0.92      0.92      0.92      1151

Confusion Matrix
[[662  35]
 [ 52 402]]

Feature Importance (Esitmate of total reduction in entropy brought by a feature)
[2.69414596e-03 0.00000000e+00 2.11294039e-03 0.00000000e+00
 1.56206514e-02 5.84561563e-03 1.14879085e-01 3.53909219e-03
 6.71441155e-04 0.00000000e+00 8.01235928e-03 5.55989735e-03
 1.02839534e-03 0.00000000e+00 2.09386894e-03 6.04508817e-02
 5.03961254e-03 1.69900979e-03 6.84208705e-03 0.00000000e+00
```

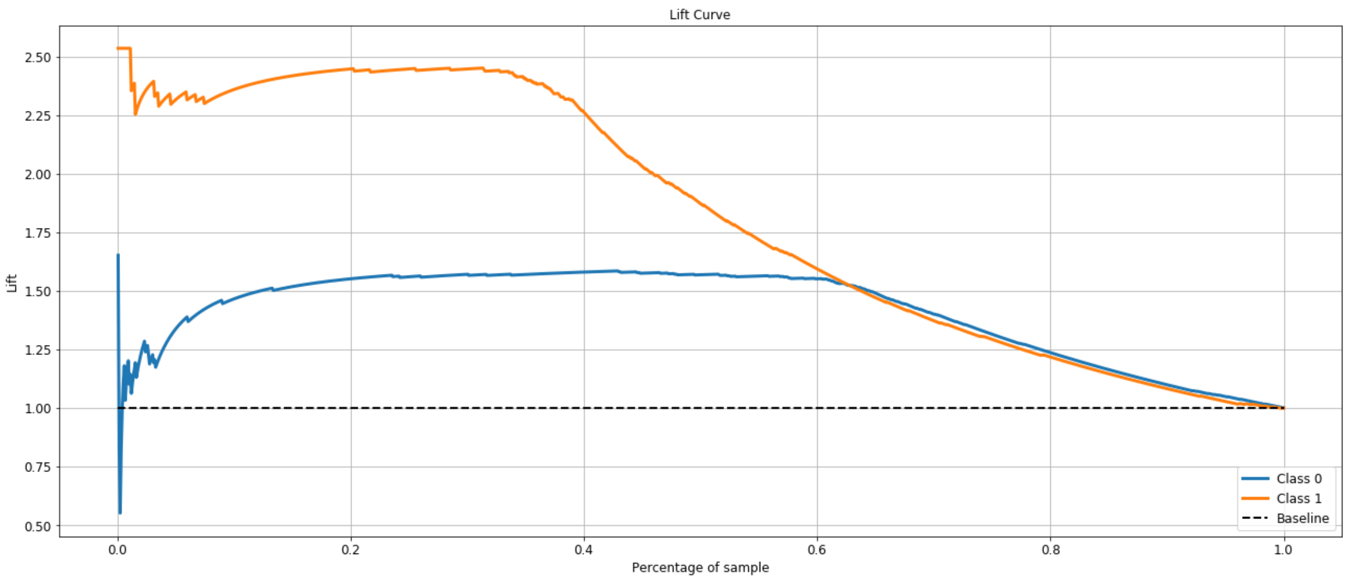
```
 8.44418553e-03 4.23696492e-03 0.00000000e+00 4.80323425e-02
 4.56979800e-02 5.75381788e-03 1.92222131e-02 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 1.56669603e-03 0.00000000e+00 0.00000000e+00
 9.04457855e-03 0.00000000e+00 2.65943036e-03 0.00000000e+00
 0.00000000e+00 5.77970767e-03 0.00000000e+00 0.00000000e+00
 2.06597279e-03 2.36957715e-02 0.00000000e+00 0.00000000e+00
 1.31239816e-03 2.65263012e-03 0.00000000e+00 4.01785074e-01
 3.67149468e-02 3.61545238e-04 2.14109356e-02 1.23473726e-01]
```

ROC and Lift Curves

```
# Probabilites
y_prob = clfTree.best_estimator_.predict_proba(X_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(y_true,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC Curve

```
auc(fpr, tpr)
```

```
0.9400593481187469
```

Decision Tree Performance on MinMax Scaled Data

Nested Cross Validation Results

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
clfTreeMin = GridSearchCV(DecisionTreeClassifier(), parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(clfTree, X=minData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Scaled
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.91, Std Deviation: 0.00 For Scaled Data
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.915528	0.011206	0.945481	0.011206	0.930136	0.001878
Not Spam	0.912240	0.014796	0.865424	0.020179	0.887881	0.004629

The best parameter values, Detailed classification report, Confusion Matrix and Feature Importance of Decision Tree

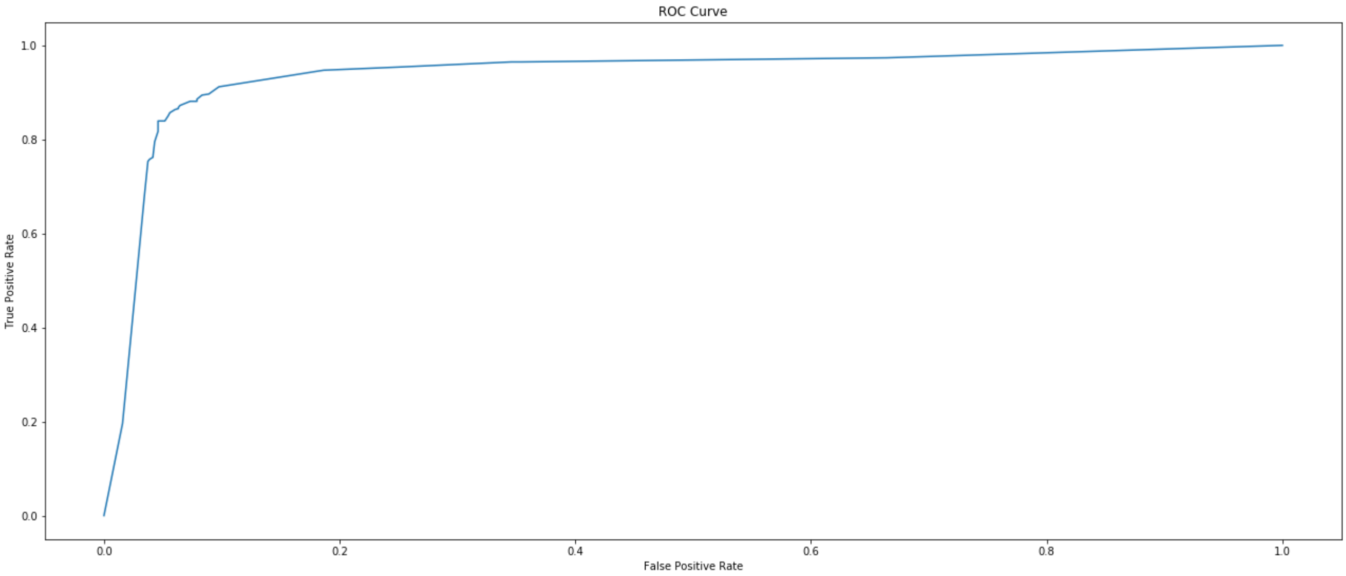
```
clfTreeMin.fit(Xmin_train, ymin_train)
print('The best parameter values are ',clfTreeMin.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = ymin_test, clfTreeMin.best_estimator_.predict(Xmin_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance (Esitmate of total reduction in entropy brought by a feature)")
print(clfTreeMin.best_estimator_.feature_importances_)
```

```
The best parameter values are  {'criterion': 'gini', 'max_depth': 9, 'min_samples_leaf': 6, 'min_samples_split': 8}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.91      0.94      0.93       697
           1       0.91      0.86      0.88       454

    accuracy                           0.91      1151
   macro avg       0.91      0.90      0.90      1151
weighted avg       0.91      0.91      0.91      1151

Confusion Matrix
[[658  39]
 [ 65 389]]

Feature Importance (Esitmate of total reduction in entropy brought by a feature)
[3.77088470e-04 0.00000000e+00 1.98608042e-04 0.00000000e+00
 1.67194906e-02 4.31679691e-03 1.47045846e-01 2.54144890e-02
 1.48508043e-03 2.01033094e-04 9.35436627e-04 5.61043581e-03
 0.00000000e+00 0.00000000e+00 0.00000000e+00 3.88042737e-02
 1.54558741e-02 0.00000000e+00 6.98824074e-03 0.00000000e+00
 7.81372785e-03 0.00000000e+00 0.00000000e+00 8.29970289e-03
 7.08901360e-02 0.00000000e+00 1.19676811e-02 5.86114730e-03
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 9.10454955e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
 1.20283069e-03 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 1.16870950e-02 2.14529169e-04 3.10626429e-05
 7.06072373e-03 1.89709089e-02 0.00000000e+00 0.00000000e+00
 1.78391603e-03 2.97495402e-03 0.00000000e+00 3.98457459e-01
 4.29164950e-02 0.00000000e+00 1.30567370e-01 1.48371126e-02]
```

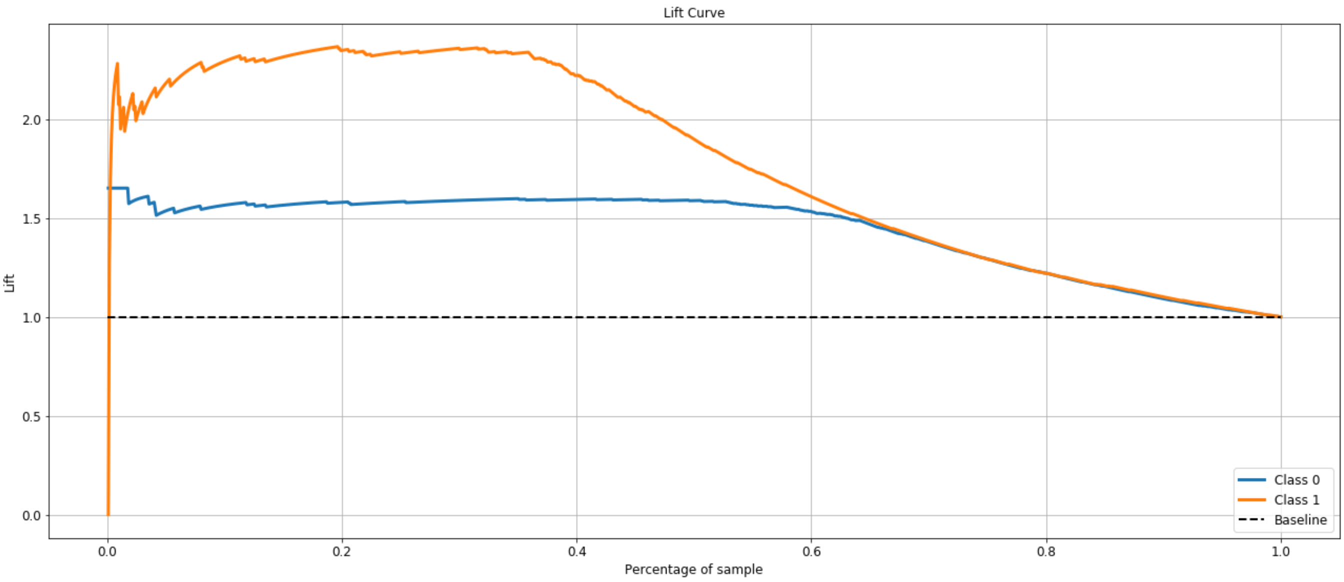
ROC and Lift Curves

```
# Probabilites
y_prob = clfTreeMin.best_estimator_.predict_proba(Xmin_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(ymin_test,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
```

```
plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC Curve

```
auc(fpr, tpr)
```

```
0.9385803854151523
```

Cost Senstitve Training

```
parameters_dict = {"max_depth": range(3,10), "min_samples_split" : np.arange(2,10,2), \
                   "min_samples_leaf" : np.arange(2,10,2), "criterion": ["gini"]}
clfTree = GridSearchCV(DecisionTreeClassifier(), parameters_dict, cv=inner_cv, scoring=cost_scorer, refit=True)
nested_score = cross_validate(clfTree, X=transformedData, y=target, cv=outer_cv, scoring = cost_scorer)
print("Average Missclassification Cost: {0:.2f}, Std Deviation:
{1:.2f}".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
```

```
Average Missclassification Cost: -656.00, Std Deviation: 72.98
```

```
clfTree.fit(X_train, y_train)
print('The best parameter values are ',clfTree.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, clfTree.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance (Esitmate of total reduction in entropy brought by a feature)")
print(clfTree.best_estimator_.feature_importances_)
```

```
The best parameter values are  {'criterion': 'gini', 'max_depth': 7, 'min_samples_leaf': 2, 'min_samples_split': 4}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.93      0.92      0.93       697
           1       0.89      0.90      0.89       454

    accuracy                           0.91      1151
   macro avg       0.91      0.91      0.91      1151
weighted avg       0.91      0.91      0.91      1151

Confusion Matrix
[[644  53]
 [ 46 408]]

Feature Importance (Esitmate of total reduction in entropy brought by a feature)
[0.         0.         0.0008973  0.         0.01159917 0.00335938
 0.12994833 0.01786273 0.00262586 0.00129996 0.         0.
 0.00044865 0.         0.         0.07143521 0.0133324  0.0014955
 0.         0.         0.00731004 0.00336075 0.0008836  0.00850225
 0.05672871 0.01389182 0.01790434 0.         0.         0.00280972
 0.         0.         0.         0.         0.         0.00615509
 0.00526539 0.         0.00411331 0.         0.00276092 0.01351167
 0.         0.00158472 0.         0.02833938 0.         0.
 0.         0.00752068 0.         0.40661193 0.03327586 0.
 0.12097794 0.00418739]
```

Neural Networks

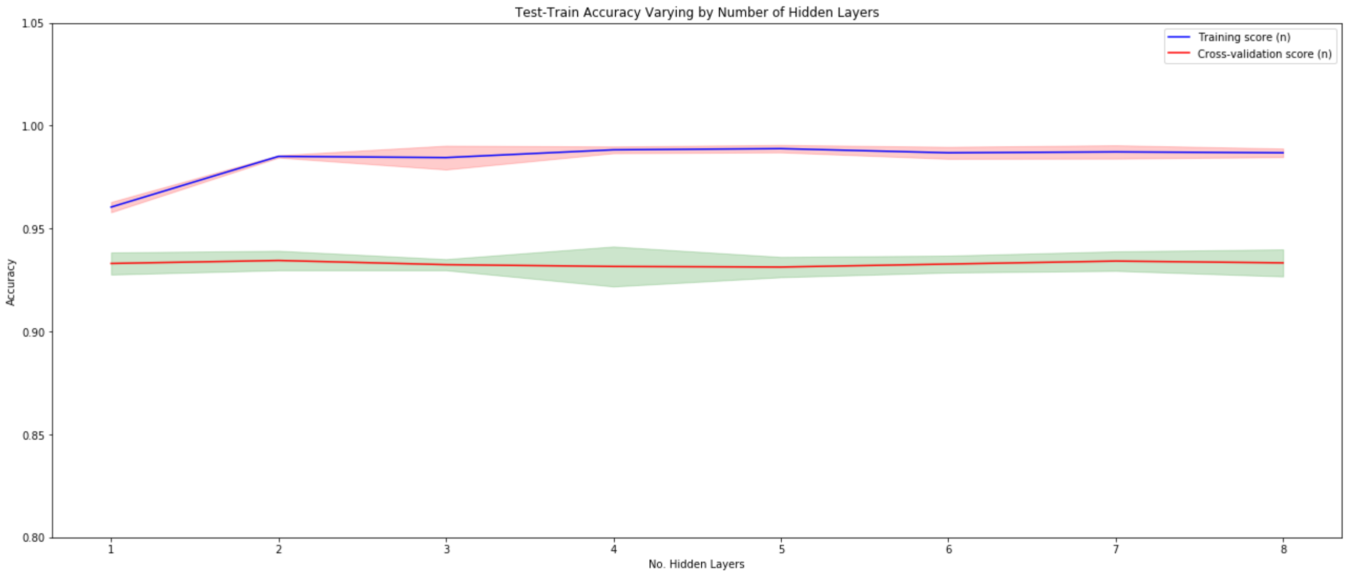
Building Model with different hyperparameters

```
from tensorflow import keras
from tensorflow.keras import layers
from keras.wrappers.scikit_learn import KerasClassifier
def create_model(n = 1,activation='relu',nb_hidden=56,init_mode='uniform',optimizer='adam'):
    #print(n,activation,nb_hidden)
    model = keras.Sequential()
    model.add(layers.Dense(nb_hidden,
                    input_dim=56, kernel_initializer=init_mode,
                    activation=activation))
    if n > 1:
        for i in range(n):
            model.add(layers.Dense(nb_hidden,
                        kernel_initializer=init_mode,
                        activation=activation))

    model.add(layers.Dense(1, kernel_initializer=init_mode, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy',
                 optimizer=optimizer,
                 metrics=['acc'])
    #print(model.summary())
    return model

activations = ['relu']
nb_hiddens = np.arange(56, 64, 2)
ns = np.array([1,2,3,4,5,6,7,8])

parameters_dict = dict(activation=activations, nb_hidden=nb_hiddens, n = ns)
model = KerasClassifier(build_fn=create_model, epochs=30, verbose=0)
```

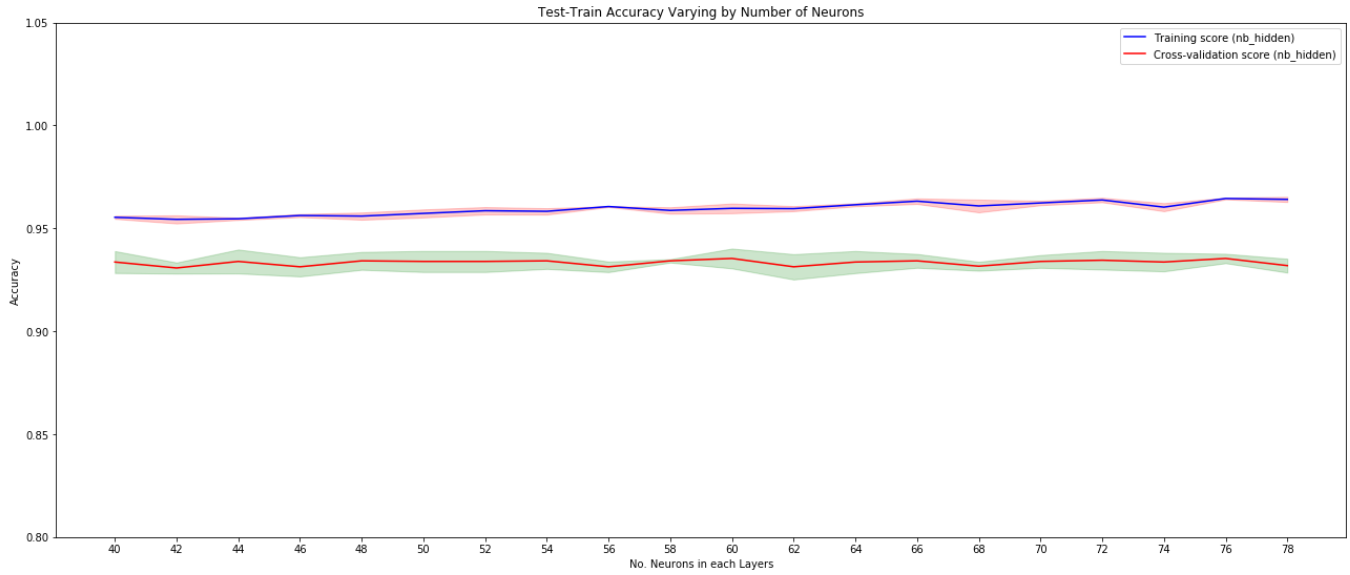
Visualizing Effect of Different Parameters of Neural Network

```
## Dictionary that holds different parameters and their values.
train_scores, test_scores = validation_curve(
    model, X_train, y_train, param_name="n", cv=3,
    param_range=parameters_dict['n'],scoring="accuracy")

meanTrainScore = np.mean(train_scores, axis =1)
```

```
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))

plt.title("Test-Train Accuracy Varying by Number of Hidden Layers")
plt.xlabel("No. Hidden Layers")
plt.ylabel("Accuracy")
plt.ylim(0.8, 1.05)
plt.fill_between(parameters_dict['n'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['n'], meanTrainScore, label="Training score (n)",
             color="b")
plt.fill_between(parameters_dict['n'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['n'], meanTestScore, label="Cross-validation score (n)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['n'])
plt.show()
```




```
activations = ['relu']
nb_hiddens = np.arange(40, 80, 2)
ns = np.array([1,2,3,4,5,6,7,8])
parameters_dict = dict(activation=activations, nb_hidden=nb_hiddens, n = ns)
train_scores, test_scores = validation_curve(
    model, X_train, y_train, param_name="nb_hidden", cv=3,
    param_range=parameters_dict['nb_hidden'],scoring="accuracy")

meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))

plt.title("Test-Train Accuracy Varying by Number of Neurons")
plt.xlabel("No. Neurons in each Layers")
plt.ylabel("Accuracy")
plt.ylim(0.8, 1.05)
plt.fill_between(parameters_dict['nb_hidden'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['nb_hidden'], meanTrainScore, label="Training score (nb_hidden)",
             color="b")
plt.fill_between(parameters_dict['nb_hidden'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['nb_hidden'], meanTestScore, label="Cross-validation score (nb_hidden)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['nb_hidden'])
plt.show()
```



Nested Cross Validated Performance on Normalized Data

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
activations = ['relu']
nb_hiddens = np.arange(45, 75, 5)
ns = np.array([1,2,3,4])
parameters_dict = dict(activation=activations, nb_hidden=nb_hiddens, n = ns)
clf = RandomizedSearchCV(estimator=model, param_distributions=parameters_dict,cv=inner_cv,\
                          n_iter=10, verbose = True)
nested_score = cross_validate(clf, X=transformedData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Normalized
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:  2.6min finished
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:  2.5min finished
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:  2.5min finished
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:  2.6min finished
```

```
Mean Accuracy: 0.94, Std Deviation: 0.00 For Normalized Data
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.945670	0.005404	0.960904	0.005404	0.953209	0.003055
Not Spam	0.938427	0.007907	0.915061	0.007691	0.926556	0.004765

The best parameter values, Detailed classification report, Confusion Matrix

```
clf.fit(X_train, y_train)
print('The best parameter values are ',clf.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, clf.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
```

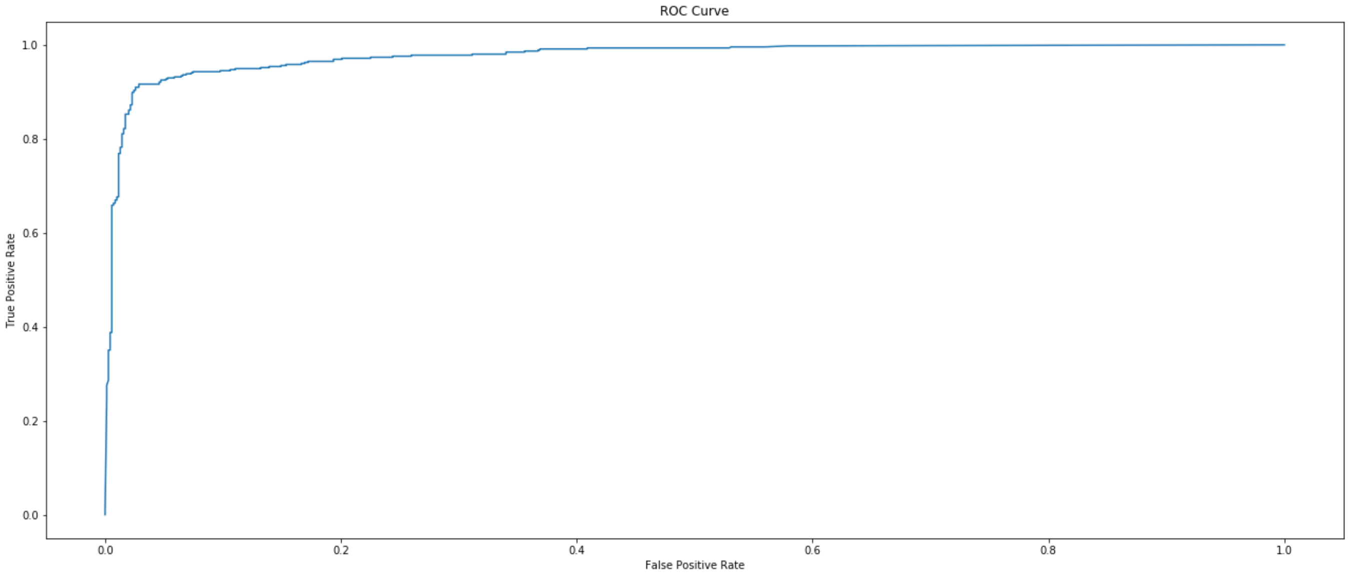
```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:  2.7min finished
```

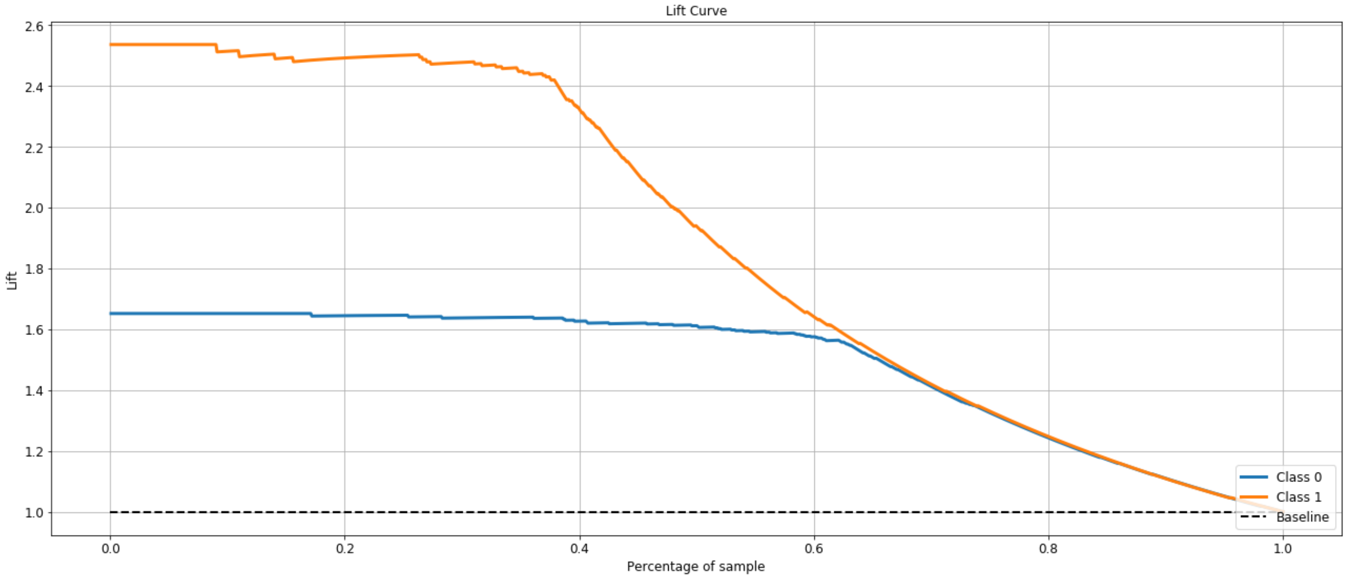
```
The best parameter values are  {'nb_hidden': 55, 'n': 4, 'activation': 'relu'}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.95      0.97      0.96       697
           1       0.95      0.92      0.93       454

    accuracy                           0.95      1151
   macro avg       0.95      0.94      0.94      1151
weighted avg       0.95      0.95      0.95      1151

Confusion Matrix
[[673  24]
 [ 38 416]]
```

ROC and Lift Curves

```
# Probabilites
y_prob = clf.best_estimator_.predict_proba(X_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(y_true,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC Curve

```
auc(fpr, tpr)
```

```
0.976669995386142
```

Nested Cross Validation Performance on MinMaxScaled Data

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
activations = ['relu']
nb_hiddens = np.arange(45, 75, 5)
ns = np.array([1,2,3,4])
parameters_dict = dict(activation=activations, nb_hidden=nb_hiddens, n = ns)
clfNNMin = RandomizedSearchCV(estimator=model, param_distributions=parameters_dict,cv=inner_cv,\
                           n_iter=10, verbose = True)
nested_score = cross_validate(clfNNMin, X=minData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For MinMax Scaled
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:  2.7min finished
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:  2.8min finished
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:  2.7min finished
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:  2.8min finished
```

```
Mean Accuracy: 0.94, Std Deviation: 0.00 For MinMax Scaled Data
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.943395	0.007342	0.963773	0.007342	0.953374	0.003693
Not Spam	0.942647	0.009590	0.910656	0.021606	0.926125	0.007446

Best Parameter Values, Classification Report and Confusion Matrix

```
clfNNMin.fit(Xmin_train, ymin_train)
print('The best parameter values are ',clfNNMin.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = ymin_test, clfNNMin.best_estimator_.predict(Xmin_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
#print("Feature Importance (Esitmate of total reduction in entropy brought by a feature)")
#print(clfNNMin.best_estimator_.feature_importances_)
```

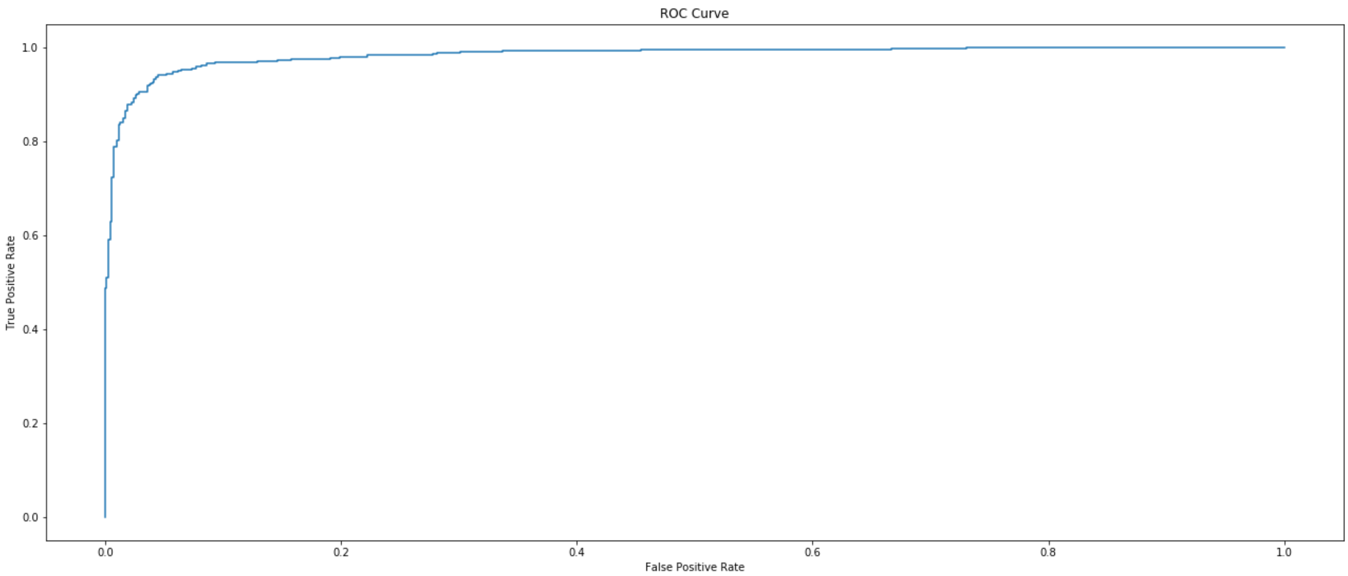
```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:  2.7min finished
```

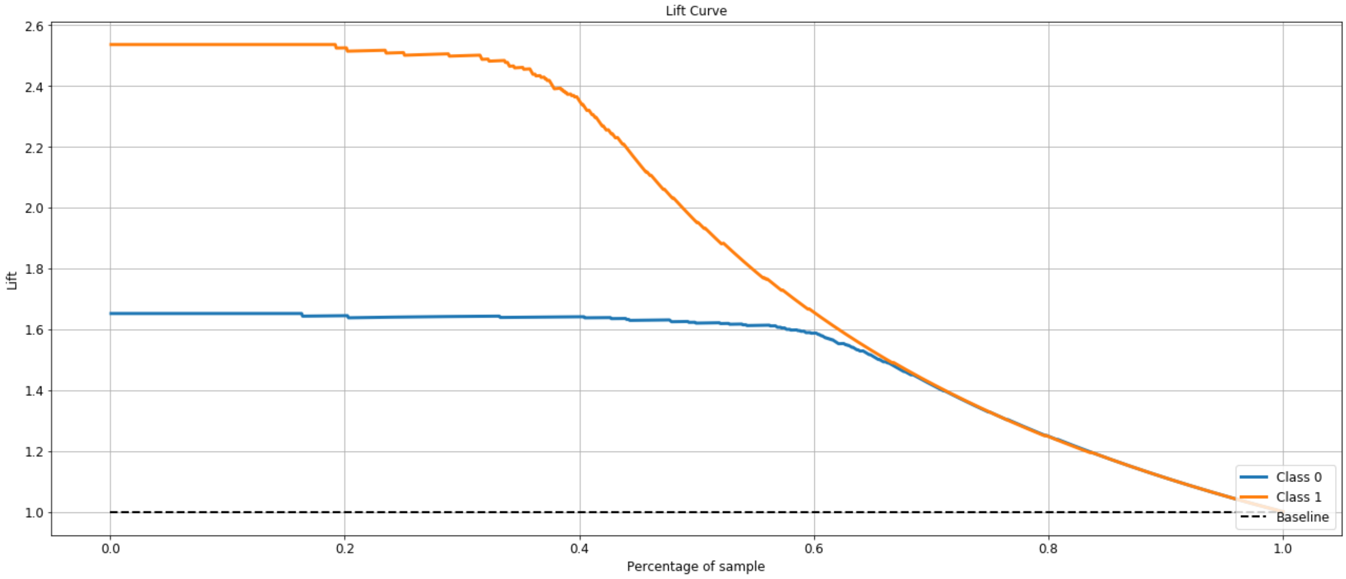
```
The best parameter values are  {'nb_hidden': 50, 'n': 3, 'activation': 'relu'}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.95      0.96      0.96       697
           1       0.94      0.92      0.93       454

    accuracy                           0.95      1151
   macro avg       0.94      0.94      0.94      1151
weighted avg       0.95      0.95      0.95      1151

Confusion Matrix
[[671  26]
 [ 37 417]]
```

ROC and Lift Curves

```
# Probabilites
y_prob = clfNNMin.best_estimator_.predict_proba(Xmin_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(ymin_test,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()

plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC Curve

```
auc(fpr,tpr)
```

```
0.9835354793040026
```

Cost Sensitive Training

```
activations = ['relu']
nb_hiddens = np.arange(45, 75, 5)
ns = np.array([1,2,3,4])
parameters_dict = dict(activation=activations, nb_hidden=nb_hiddens, n = ns)
clf = RandomizedSearchCV(estimator=model, param_distributions=parameters_dict,cv=inner_cv,\
                          n_iter=10, verbose = True)
nested_score = cross_validate(clf, X=transformedData, y=target, cv=outer_cv, scoring = cost_scorer)
print("Mean Missclassification Cost: {0:.2f}, Std Deviation: {1:.2f} For Normalized
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:  2.2min finished
```

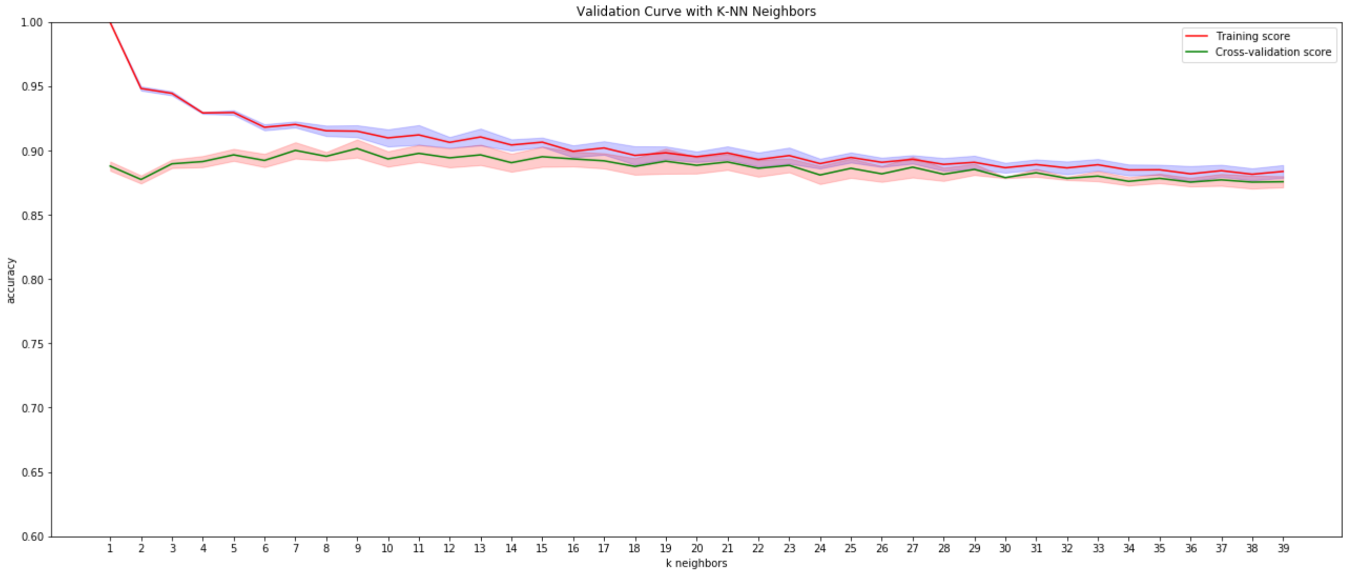
```
Mean Missclassification Cost: -357, Std Deviation: 34.22
```

KNN Classifier

Visualizing Effect of Different Parameters of KNN Classifier

```
# Create range of values for parameter
neighbor = range(1,40,1)

train_scores, test_scores = validation_curve(
    KNeighborsClassifier(), X_train, y_train, param_name="n_neighbors", cv=3,
    param_range=neighbor,
    scoring="accuracy")
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
plt.figure(figsize=(22,9))
plt.title("Validation Curve with K-NN Neighbors")
plt.xlabel("k neighbors")
plt.ylabel("accuracy")
plt.ylim(0.6, 1.0)

# Plot the values for recall with K ranging from 1-20
plt.plot(neighbor, train_scores_mean, label="Training score",
             color="r")
plt.fill_between(neighbor, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha=0.2, color="b")
```

```
plt.plot(neighbor, test_scores_mean, label="Cross-validation score",
            color="g")
plt.fill_between(neighbor, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.2, color="r")
plt.legend(loc="best")
plt.xticks(neighbor)

plt.show()
```



Nested Cross Validation Performance on Normalized Data

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
parameters_dict = {"n_neighbors": range(2,40,1)}
clfKNN = GridSearchCV(KNeighborsClassifier(), param_grid = parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(clfKNN, X=transformedData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Normalized
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.91, Std Deviation: 0.01 For Normalized Data
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.911217	0.003365	0.938307	0.003365	0.924554	0.004244
Not Spam	0.900570	0.005434	0.859340	0.011075	0.879446	0.007527

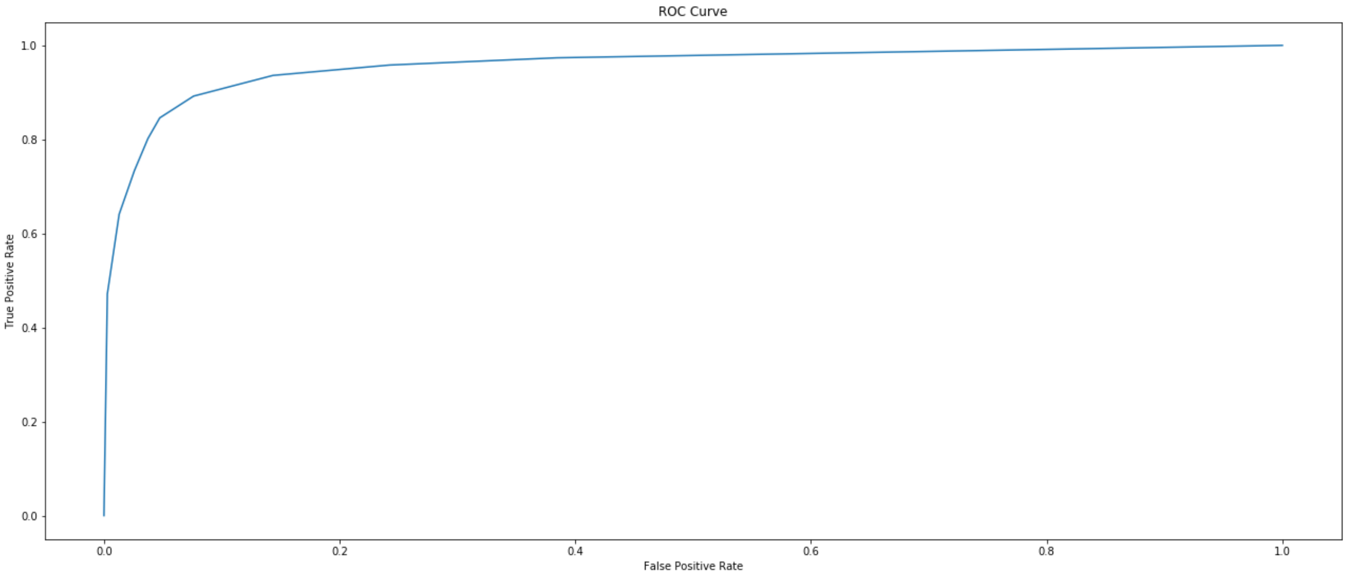
Best Parameter Values, Classification Report and Confusion Matrix

```
clfKNN.fit(X_train, y_train)
print('The best parameter values are ',clfKNN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, clfKNN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
#print("Feature Importance (Esitmate of total reduction in entropy brought by a feature)")
#print(clfTree.best_estimator_.feature_importances_)
```

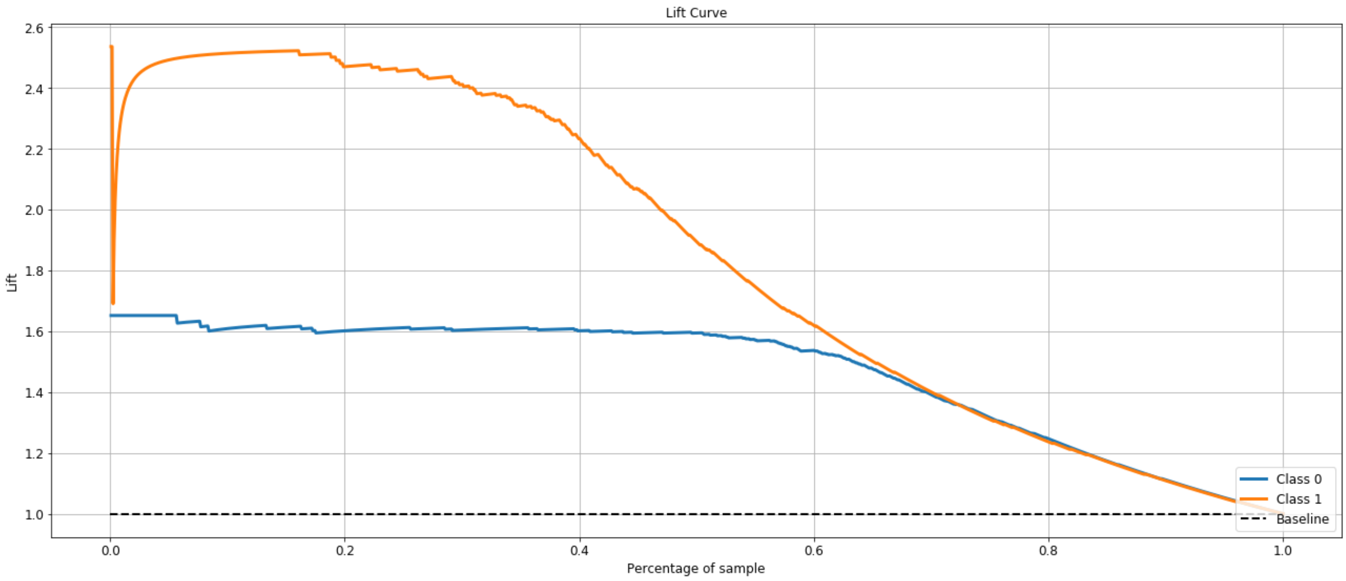
```
The best parameter values are  {'n_neighbors': 9}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.90      0.95      0.93       697
           1       0.92      0.85      0.88       454

    accuracy                           0.91      1151
   macro avg       0.91      0.90      0.90      1151
weighted avg       0.91      0.91      0.91      1151

Confusion Matrix
[[664  33]
 [ 70 384]]
```

ROC and Lift Curves

```
# Probabilites
y_prob = clfKNN.best_estimator_.predict_proba(X_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(y_true,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC curve

```
auc(fpr,tpr)
```

```
0.9571037612423287
```

Performance in MinMax Scaled Data

Nested Cross Validation Performance

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
clfKNMin = GridSearchCV(KNeighborsClassifier(), param_grid = parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(clfTree, X=minData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Scaled
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.91, Std Deviation: 0.00 For Scaled Data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.914916	0.011228	0.946198	0.011228	0.930179	0.002030
Not Spam	0.913187	0.015162	0.864320	0.018515	0.887774	0.004092

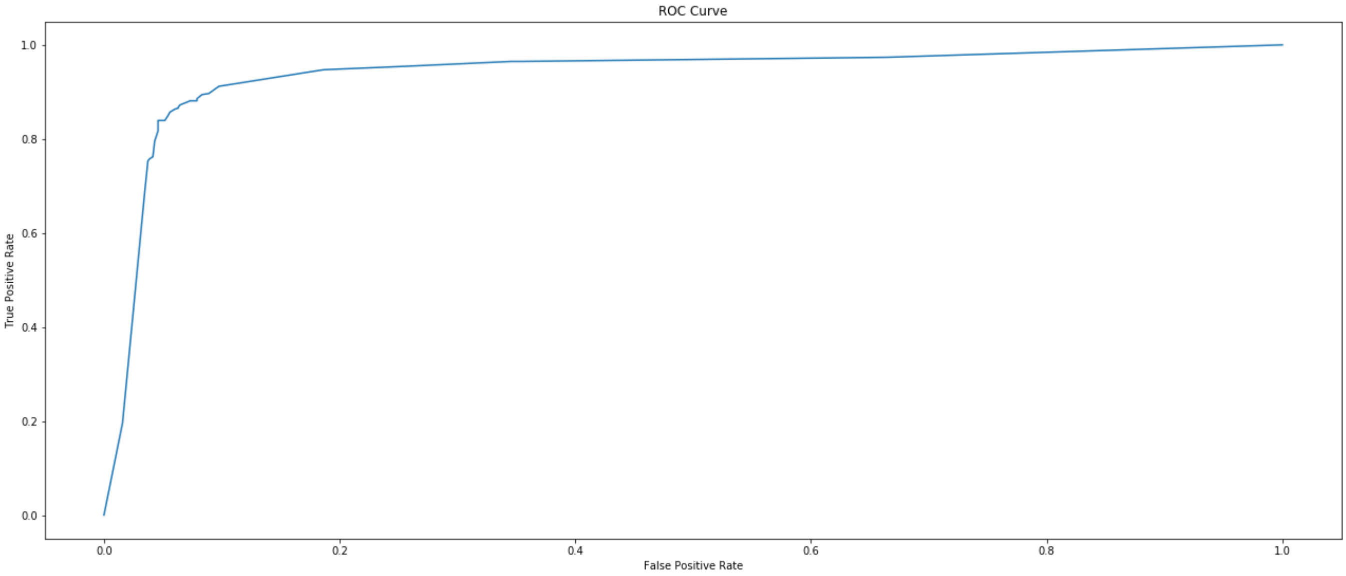
```
#### Classification Report, Best Model Params and Confusion Matrix
```

```
clfKNMin.fit(Xmin_train, ymin_train)
print('The best parameter values are ',clfKNMin.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = ymin_test, clfKNMin.best_estimator_.predict(Xmin_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
#print("Feature Importance (Esitmate of total reduction in entropy brought by a feature)")
#print(clfKNMin.best_estimator_.feature_importances_)
```

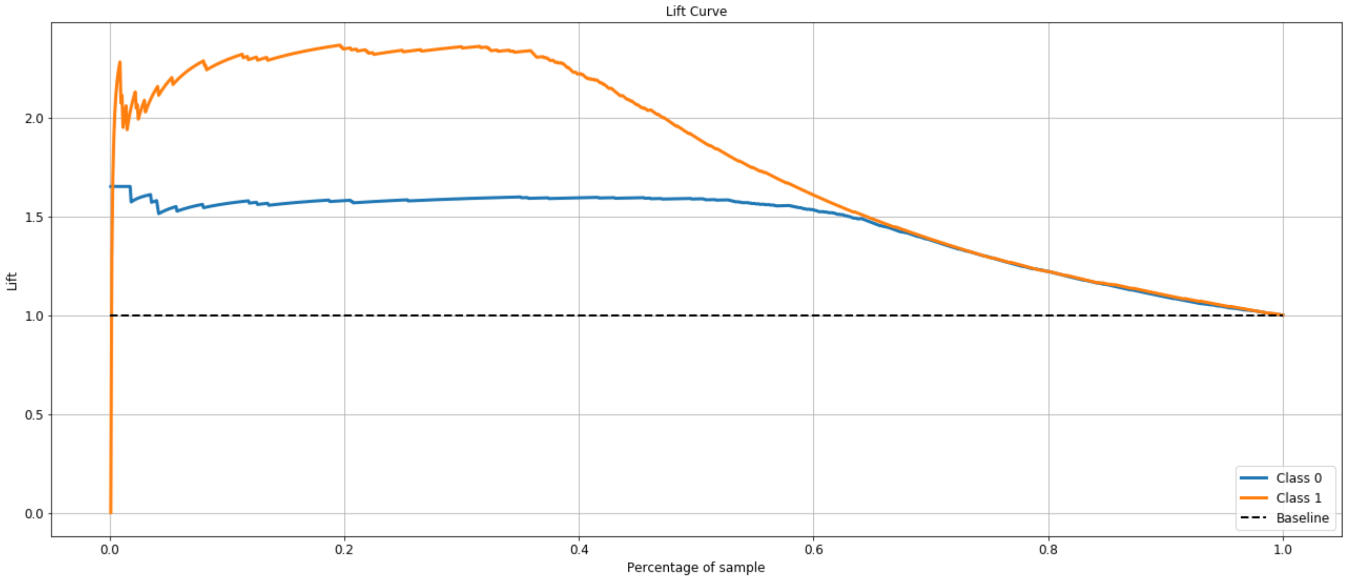
```
The best parameter values are  {'n_neighbors': 7}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.90      0.93      0.92       697
           1       0.89      0.85      0.87       454

    accuracy                           0.90      1151
   macro avg       0.90      0.89      0.89      1151
weighted avg       0.90      0.90      0.90      1151

Confusion Matrix
[[651  46]
 [ 69 385]]
```

ROC and Lift Curves

```
# Probabilites
y_prob = clfTreeMin.best_estimator_.predict_proba(Xmin_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(ymin_test,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC Curve

```
auc(fpr, tpr)
```

```
0.9385803854151523
```

Cost Senstive Training

```
parameters_dict = {"n_neighbors": range(2,24,2)}
clfKNN = GridSearchCV(KNeighborsClassifier(), param_grid = parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(clfKNN, X=transformedData, y=target, cv=outer_cv, scoring = cost_scorer)
print("Mean Missclassification Cost: {0:.2f}, Std Deviation:
{1:.2f}".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
```

```
Mean Missclassification Cost: -823.25, Std Deviation: 62.77
```

```
clfKNN.fit(X_train, y_train)
print('The best parameter values are ',clfKNN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, clfKNN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
#print("Feature Importance (Esitmate of total reduction in entropy brought by a feature)")
#print(clfTree.best_estimator_.feature_importances_)
```

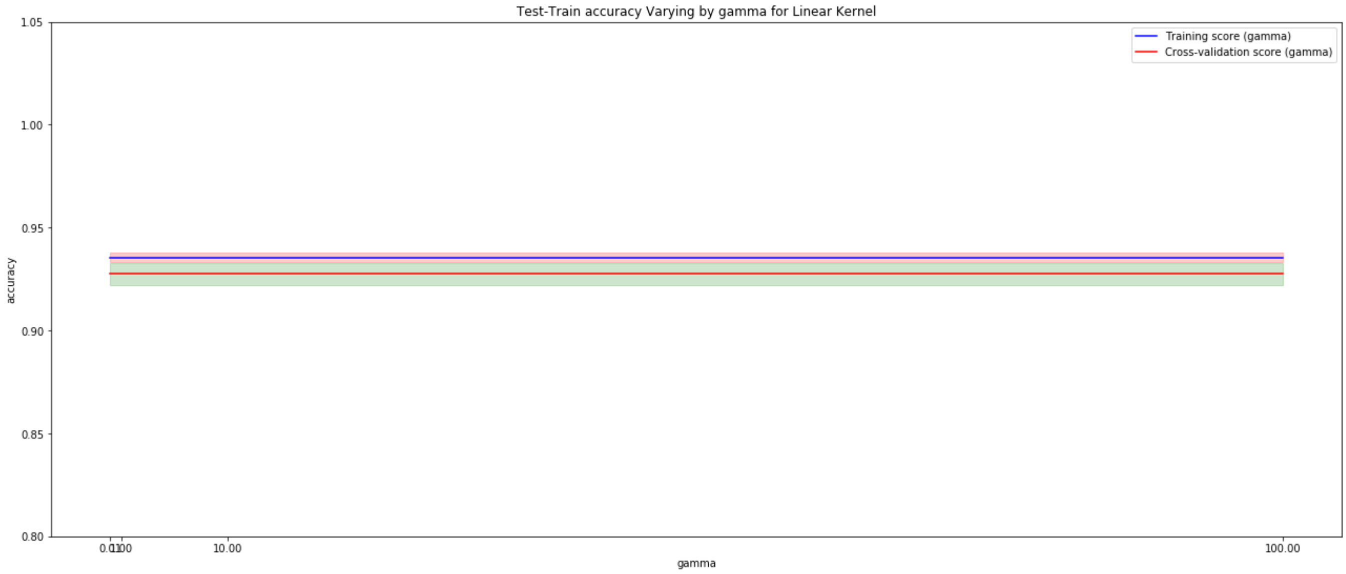
```
The best parameter values are  {'n_neighbors': 4}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.89      0.94      0.91       697
           1       0.90      0.81      0.85       454

    accuracy                           0.89      1151
   macro avg       0.89      0.88      0.88      1151
weighted avg       0.89      0.89      0.89      1151

Confusion Matrix
[[655  42]
 [ 84 370]]
```

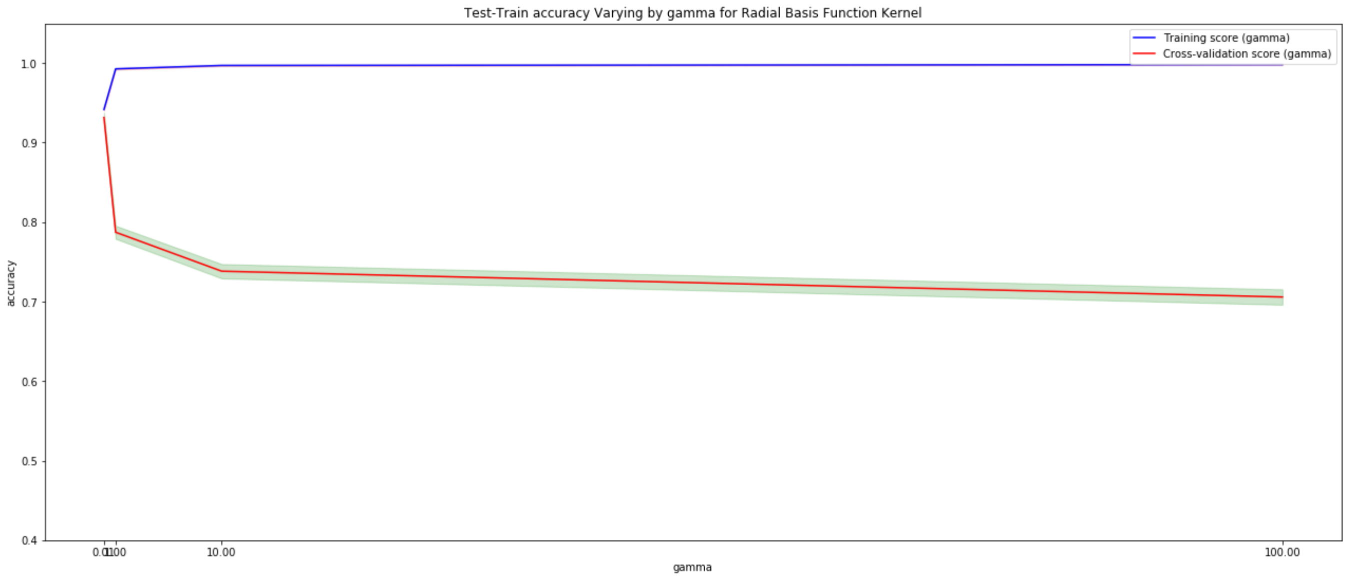
SVM

```
### Creating Visualizations of parameter effect SVM
```

```
parameters_dict={"gamma":[0.01, 1.0,10,100]}
train_scores, test_scores = validation_curve(
    SVC(kernel="linear"),X_train, y_train, param_name="gamma", cv=5,
    param_range=parameters_dict['gamma'],scoring="accuracy")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train accuracy Varying by gamma for Linear Kernel")
plt.xlabel("gamma")
plt.ylabel("accuracy")
plt.ylim(0.8, 1.05)
plt.fill_between(parameters_dict['gamma'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['gamma'], meanTrainScore, label="Training score (gamma)",
             color="b")
plt.fill_between(parameters_dict['gamma'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['gamma'], meanTestScore, label="Cross-validation score (gamma)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['gamma'])
plt.show()

##########################################################################################################

train_scores, test_scores = validation_curve(
    SVC(kernel="rbf"),X_train, y_train, param_name="gamma", cv=5,
    param_range=parameters_dict['gamma'],scoring="accuracy")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train accuracy Varying by gamma for Radial Basis Function Kernel")
plt.xlabel("gamma")
plt.ylabel("accuracy")
plt.ylim(0.4, 1.05)
```

```
plt.fill_between(parameters_dict['gamma'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['gamma'], meanTrainScore, label="Training score (gamma)",
             color="b")
plt.fill_between(parameters_dict['gamma'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['gamma'], meanTestScore, label="Cross-validation score (gamma)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['gamma'])
plt.show()
```





Nested Cross Validation on Normalized Dataset

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
parameters_dict={"gamma":[0.01, 1.0,10,100],"kernel":["linear","rbf"]}
SVMNormal = GridSearchCV(SVC(probability=True), param_grid = parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(SVMNormal, X=transformedData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Normalized
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.93, Std Deviation: 0.00 For Normalized Data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.929707	0.006604	0.953013	0.006604	0.941192	0.003216
Not Spam	0.924978	0.009502	0.889129	0.008378	0.906637	0.004872

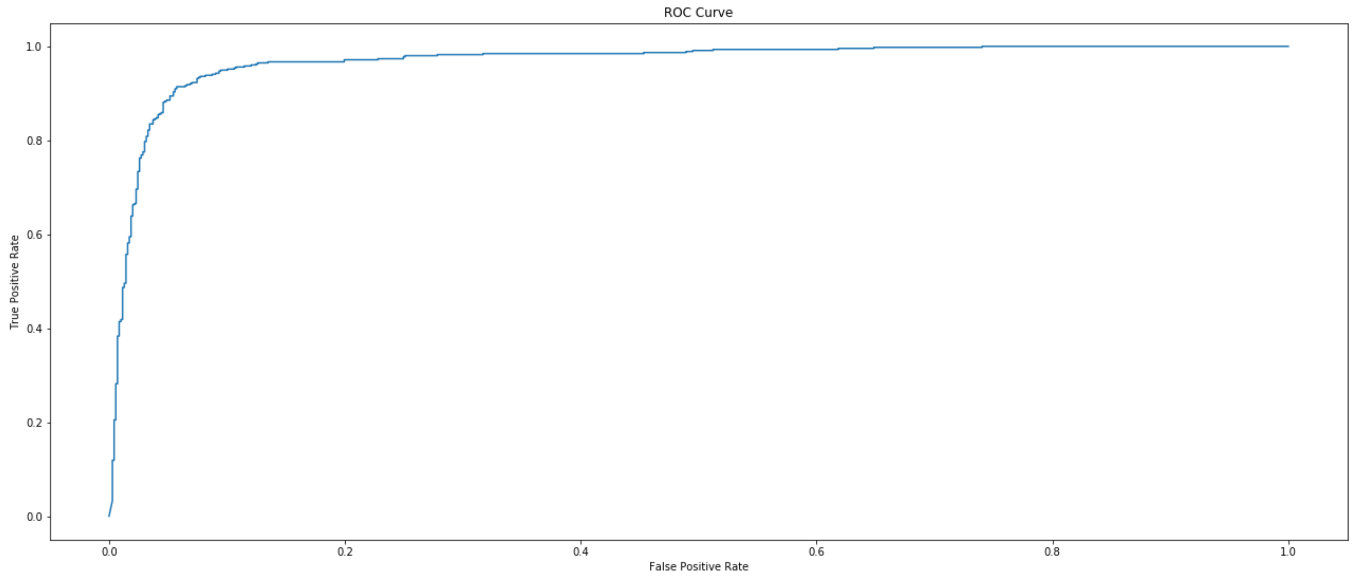
Model Best Params, Classification Report and Feature Importances

```
SVMNormal.fit(X_train, y_train)
print('The best parameter values are ',SVMNormal.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, SVMNormal.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance")
print(SVMNormal.best_estimator_.coef_)
```

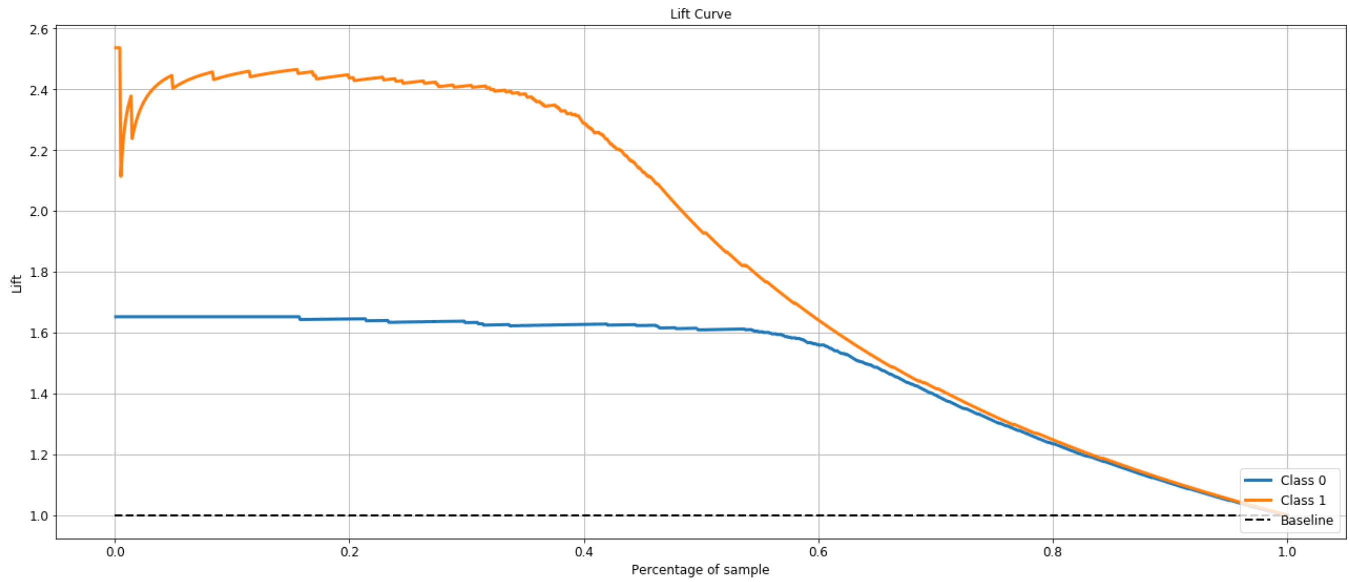
```
The best parameter values are  {'gamma': 0.01, 'kernel': 'linear'}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.93      0.95      0.94       697
           1       0.91      0.89      0.90       454

    accuracy                           0.93      1151
   macro avg       0.92      0.92      0.92      1151
weighted avg       0.93      0.93      0.93      1151

Confusion Matrix
[[659  38]
 [ 48 406]]

Feature Importance
[[-5.68257467e-02  3.27239468e-03  2.21753721e-03  6.73835247e-01
   2.46867520e-01  1.60516372e-01  7.14532680e-01  1.85928998e-01
   1.72636341e-01  1.09783059e-01 -1.07161456e-01 -1.00967276e-01
   8.18257386e-03 -3.93912084e-03  8.86700657e-02  6.35000414e-01
   3.17919718e-01  4.78732029e-02  2.17790928e-02  2.16392933e-01
   1.98007896e-01  1.12144126e-01  4.92392467e-01  1.43835051e-01
  -1.86163191e+00 -5.91950752e-01 -3.36007851e+00  1.33638653e-01
  -5.73527270e-01 -1.02151604e-01 -9.75159170e-01 -5.76875167e-01
  -1.29167711e-01 -4.76198130e-03 -8.39207438e-01  3.31173636e-01
   2.57022246e-02 -8.25943929e-02 -3.04275274e-01 -1.00695284e-01
  -1.24596637e+00 -7.71619749e-01 -3.06847910e-01 -4.79829710e-01
  -5.02886466e-01 -1.23916073e+00  1.34630967e-02 -4.33689644e-01
  -1.38958806e-01 -3.75245795e-02 -4.44285516e-02  5.22762034e-01
   1.00235627e+00  2.99784589e-01  1.41585906e+00  1.46154119e+00]]
```

ROC and Lift Curves

```
# Probabilites
y_prob = SVMNormal.best_estimator_.predict_proba(X_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(y_true,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC Curve

```
auc(fpr, tpr)
```

```
0.9678230806666709
```

Performance on MinMax Scaled Dataset

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
SVMMIn = GridSearchCV(SVC(probability=True), parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(SVMMIn, X=minData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Scaled
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.93, Std Deviation: 0.00 For Scaled Data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.922936	0.003569	0.957317	0.003569	0.939796	0.003209
Not Spam	0.930408	0.005223	0.877007	0.010532	0.902875	0.005767

Model Best Params, Classification Report and Feature Importances

```
SVMMIn.fit(Xmin_train, ymin_train)
print('The best parameter values are ',SVMMIn.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = ymin_test, SVMMIn.best_estimator_.predict(Xmin_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance")
print(SVMMIn.best_estimator_.coef_)
```

ROC and Lift Curves

```
# Probabilites
y_prob = SVMMIn.best_estimator_.predict_proba(Xmin_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(ymin_test,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```

Area Under the ROC Curve

```
auc(fpr, tpr)
```

Cost Sensitive Training

```
parameters_dict={"gamma":[0.01, 1.0,10,100],"kernel":["linear","rbf"]}
SVMNormal = GridSearchCV(SVC(probability=True), param_grid = parameters_dict, cv=inner_cv, scoring=cost_scorer, refit=True)
nested_score = cross_validate(SVMNormal, X=transformedData, y=target, cv=outer_cv, scoring = cost_scorer)
print("Mean Missclassification Cost: {0:.2f}, Std Deviation:
{1:.2f}".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
```

```
Mean Missclassification Cost: -528.25, Std Deviation: 40.36
```

```
SVMNormal.fit(X_train, y_train)
print('The best parameter values are ',SVMNormal.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, SVMNormal.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance")
print(SVMNormal.best_estimator_.coef_)
```

```
The best parameter values are  {'gamma': 0.01, 'kernel': 'linear'}

Detailed classification report:

              precision    recall  f1-score   support
```

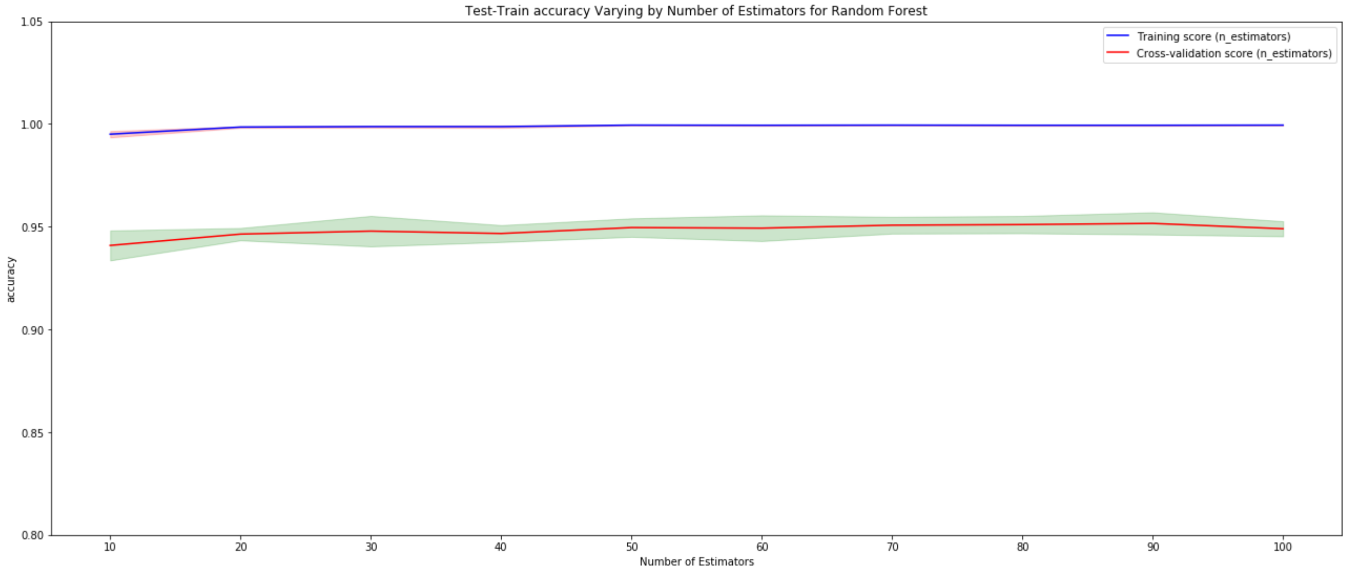
```
           0       0.94      0.94      0.94       697
           1       0.91      0.91      0.91       454

    accuracy                           0.93      1151
   macro avg       0.93      0.92      0.93      1151
weighted avg       0.93      0.93      0.93      1151

Confusion Matrix
[[658  39]
 [ 43 411]]

Feature Importance
[[-1.92566473e-01  4.86780871e-03 -2.18337141e-02  5.74794702e-01
   3.81933333e-01  1.37176382e-01  4.86689145e-01  1.72634956e-01
   1.47082829e-01  6.10851979e-02  5.03604153e-02 -1.14047222e-01
  -1.67823955e-02 -2.14732077e-03  9.50676533e-02  5.28681278e-01
   2.38238266e-01  8.92039207e-02  3.75025338e-02  3.14295471e-01
   1.10322960e-01  1.06007078e-01  6.69244019e-01  3.61719515e-01
  -1.44298099e+00 -5.04727053e-01 -3.06654250e+00  1.19518439e-01
  -3.49464598e-01 -8.90183965e-02 -6.35135764e-01 -2.61364492e-01
  -5.96104350e-02 -4.68424017e-02 -7.70675805e-01  2.21266664e-01
  -5.65484280e-02  2.01840926e-02 -2.08277828e-01 -7.23805778e-02
  -1.24596637e+00 -6.16478378e-01 -3.64153167e-02 -7.21463549e-01
  -4.21166930e-01 -1.12302914e+00 -6.25546447e-02 -3.44576492e-01
  -1.05594907e-01 -1.72482586e-03 -3.23905838e-02  7.61163736e-01
   8.61239683e-01  2.00998069e-01  1.03363073e+00  1.37383467e+00]]
```

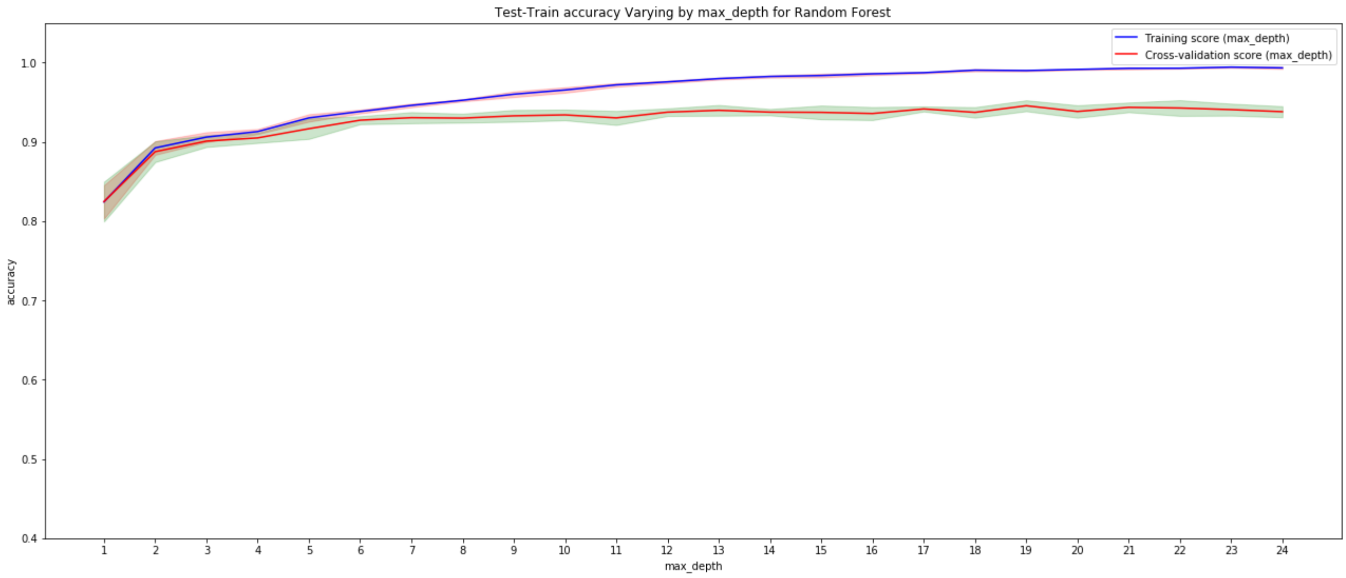
Random Forest

Visualizing effect of Different Parameters on Random Forest

```
parameters_dict = {
    'bootstrap': [True],
    'max_depth': [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [10,20,30,40,50,60,70,80,90,100]

}
train_scores, test_scores = validation_curve(
    RandomForestClassifier(),X_train, y_train, param_name="n_estimators", cv=5,
    param_range=parameters_dict['n_estimators'],scoring="accuracy")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train accuracy Varying by Number of Estimators for Random Forest")
plt.xlabel("Number of Estimators")
plt.ylabel("accuracy")
plt.ylim(0.8, 1.05)
plt.fill_between(parameters_dict['n_estimators'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['n_estimators'], meanTrainScore, label="Training score (n_estimators)",
             color="b")
plt.fill_between(parameters_dict['n_estimators'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['n_estimators'], meanTestScore, label="Cross-validation score (n_estimators)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['n_estimators'])
plt.show()

##########################################################################################################

train_scores, test_scores = validation_curve(
    RandomForestClassifier(),X_train, y_train, param_name="max_depth", cv=5,
    param_range=parameters_dict['max_depth'],scoring="accuracy")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train accuracy Varying by max_depth for Random Forest")
plt.xlabel("max_depth")
plt.ylabel("accuracy")
```

```
plt.ylim(0.4, 1.05)
plt.fill_between(parameters_dict['max_depth'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['max_depth'], meanTrainScore, label="Training score (max_depth)",
             color="b")
plt.fill_between(parameters_dict['max_depth'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['max_depth'], meanTestScore, label="Cross-validation score (max_depth)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['max_depth'])
plt.show()
```





Model Peformance on Normalized Data

```
parameters_dict = {
    'bootstrap': [True],
    'max_depth': [13,14,15],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [10,20,30]}

precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
clfForestN = GridSearchCV(RandomForestClassifier(), param_grid = parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(clfForestN, X=transformedData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Normalized
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.94, Std Deviation: 0.00 For Normalized Data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.937709	0.003268	0.965925	0.003268	0.951598	0.003849
Not Spam	0.945060	0.005217	0.901268	0.010289	0.922617	0.006617

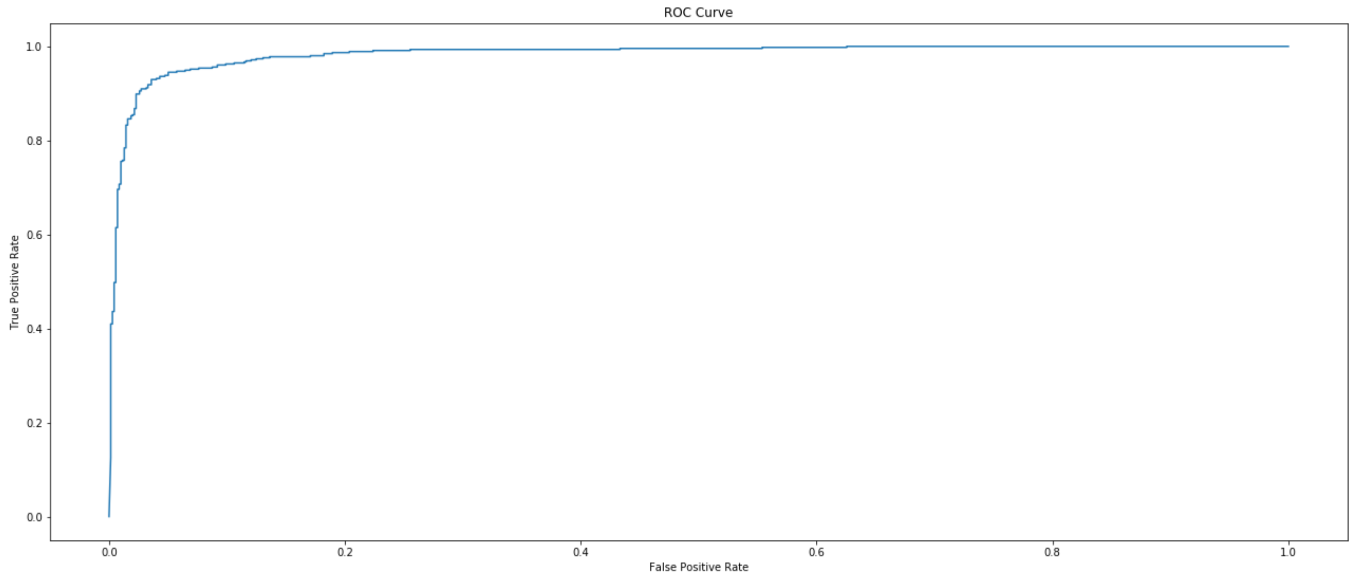
Best Parameters , Classification Report, Confusion Matrix and Feature Importances

```
clfForestN.fit(X_train, y_train)
print('The best parameter values are ',clfForestN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, clfForestN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance (Esitmate of total reduction in entropy brought by a feature)")
print(clfForestN.best_estimator_.feature_importances_)
```

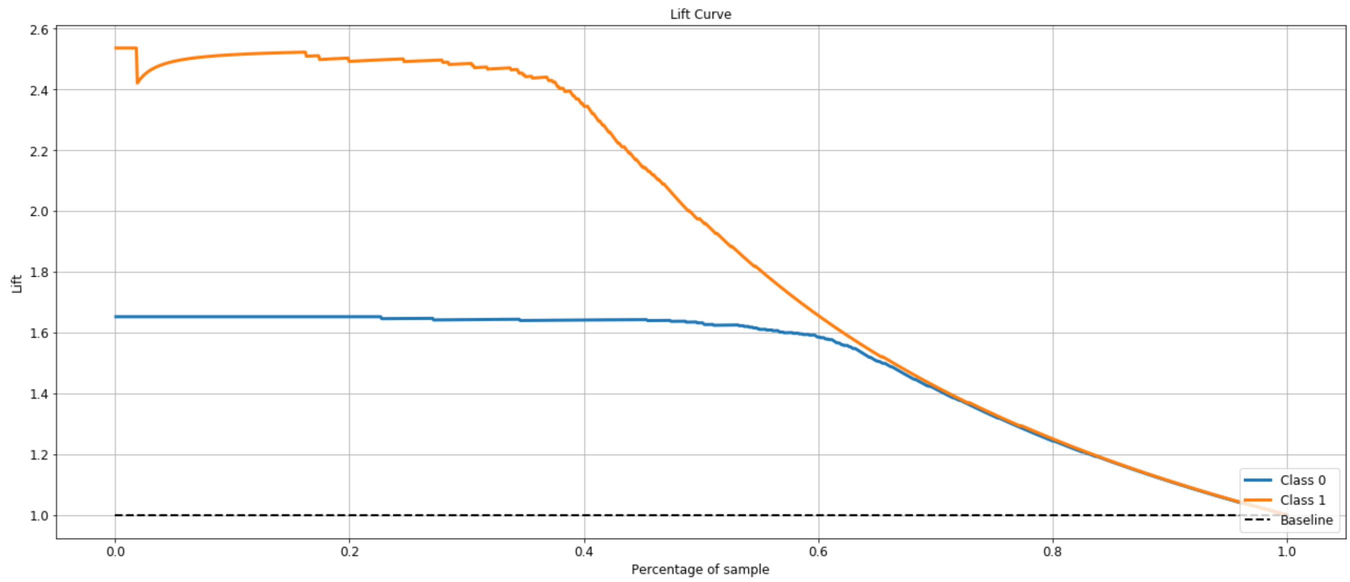
```
The best parameter values are  {'bootstrap': True, 'max_depth': 14, 'min_samples_leaf': 3, 'min_samples_split': 8, 'n_estimators': 30}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.94      0.97      0.96       697
           1       0.96      0.90      0.93       454

    accuracy                           0.95      1151
   macro avg       0.95      0.94      0.94      1151
weighted avg       0.95      0.95      0.94      1151

Confusion Matrix
[[679  18]
 [ 45 409]]

Feature Importance (Esitmate of total reduction in entropy brought by a feature)
[1.22482229e-03 1.14719610e-02 4.81888843e-03 6.16045536e-04
 3.17963647e-02 5.57291518e-03 7.52459360e-02 1.50637546e-02
 2.50766290e-03 8.91954049e-03 4.55208467e-03 8.27832149e-03
 2.37495473e-03 6.66001357e-04 3.88518970e-04 5.63922272e-02
 1.20272771e-02 7.84881158e-03 1.55431564e-02 1.77589813e-03
 8.41190895e-02 2.42652720e-03 3.22448712e-02 4.25649518e-02
 4.96421121e-02 2.05459656e-02 3.20807427e-02 2.95847953e-03
 5.63544019e-04 6.87786002e-03 1.05438475e-03 1.62696378e-04
 3.53073456e-03 6.06049349e-07 2.06157573e-03 2.30948103e-03
 8.72564466e-03 2.46946996e-04 1.50616299e-03 6.47869281e-05
 5.98642597e-04 3.91697068e-03 1.02194870e-03 8.82837181e-04
 8.94635784e-03 1.20430299e-02 0.00000000e+00 1.11195500e-03
 1.14423802e-03 6.61458580e-03 1.49848775e-03 1.64526092e-01
 1.14931802e-01 2.31417686e-03 5.92614828e-02 6.04150864e-02]
```

ROC and Lift Curves

```
# Probabilites
y_prob = clfForestN.best_estimator_.predict_proba(X_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(y_true,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC Curve

```
auc(fpr, tpr)
```

```
0.983276344813202
```

Model Performance on Min Max Scaled Data

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
ForestMin = GridSearchCV(RandomForestClassifier(), parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(ForestMin, X=minData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Scaled
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.94, Std Deviation: 0.01 For Scaled Data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.938112	0.004690	0.966643	0.004690	0.952150	0.006917
Not Spam	0.946134	0.007918	0.901813	0.016369	0.923405	0.011817

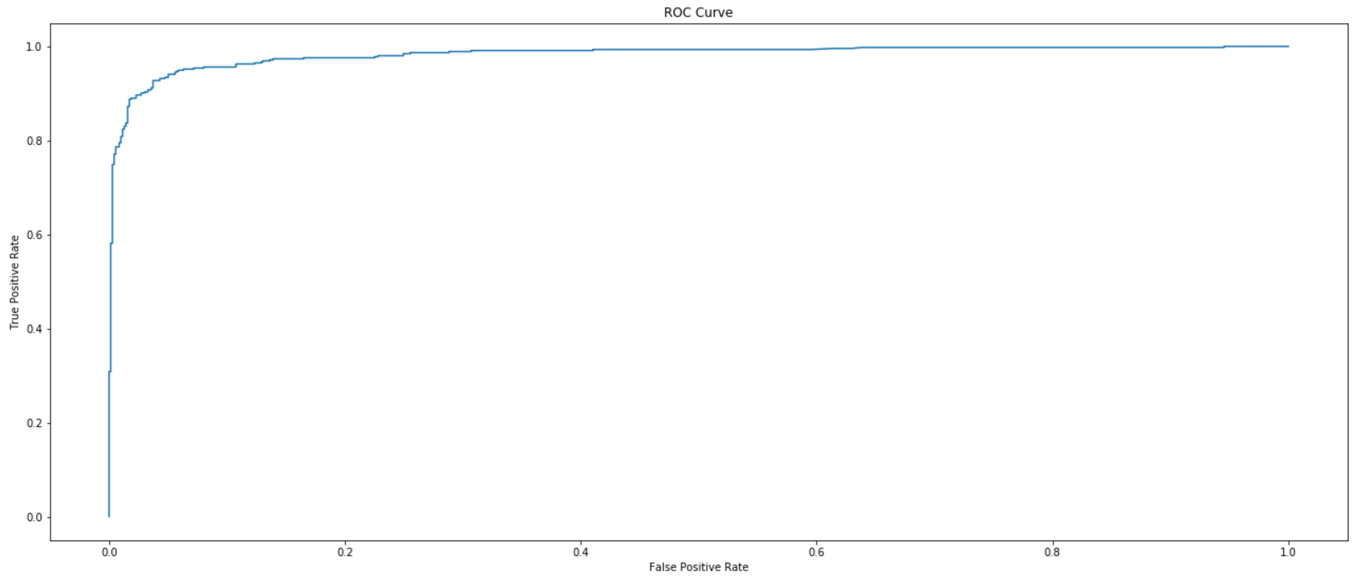
Best Parameters , Classification Report, Confusion Matrix and Feature Importances

```
ForestMin.fit(Xmin_train, ymin_train)
print('The best parameter values are ',ForestMin.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = ymin_test, ForestMin.best_estimator_.predict(Xmin_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance (Esitmate of total reduction in entropy brought by a feature)")
print(ForestMin.best_estimator_.feature_importances_)
```

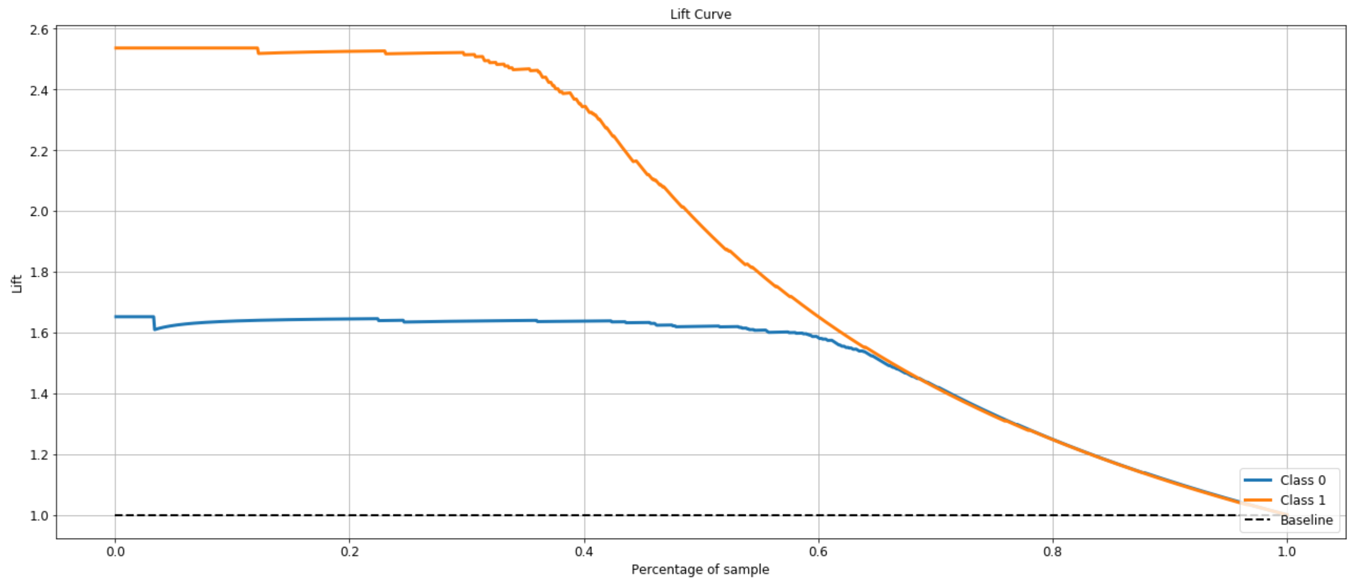
```
The best parameter values are  {'bootstrap': True, 'max_depth': 14, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 30}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.94      0.97      0.95       697
           1       0.95      0.91      0.93       454

    accuracy                           0.94      1151
   macro avg       0.94      0.94      0.94      1151
weighted avg       0.94      0.94      0.94      1151

Confusion Matrix
[[674  23]
 [ 42 412]]

Feature Importance (Esitmate of total reduction in entropy brought by a feature)
[1.96229738e-03 4.24600180e-03 3.63403399e-03 2.02882535e-05
 2.73370166e-02 7.74877990e-03 8.61163343e-02 9.63006315e-03
 1.71456567e-03 5.19712403e-03 1.19443673e-02 8.10728115e-03
 1.43078649e-03 4.54765981e-04 2.16277163e-03 8.99308185e-02
 1.63235186e-02 7.87986478e-03 2.53761662e-02 4.55514348e-03
 3.73958524e-02 2.20356385e-03 4.43128602e-02 3.72172898e-02
 4.11872779e-02 2.21363324e-02 1.53794060e-02 7.09206106e-03
 1.66998815e-03 2.23937190e-03 1.43474167e-03 5.63990291e-04
 1.01569780e-03 3.26635985e-05 1.44715065e-03 2.57379772e-03
 1.19464682e-02 6.12063793e-05 3.96315740e-03 4.84993061e-04
 0.00000000e+00 4.70791046e-03 1.18606342e-03 1.88081958e-03
 5.31290487e-03 1.32744504e-02 0.00000000e+00 2.65069191e-05
 3.86515421e-03 8.45451379e-03 3.05711021e-04 1.46923666e-01
 1.45387759e-01 1.11789146e-03 5.71648947e-02 6.02618945e-02]
```

ROC and Lift Curves

```
# Probabilites
y_prob = ForestMin.best_estimator_.predict_proba(Xmin_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(ymin_test,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()

plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC Curve

```
auc(fpr, tpr)
```

```
0.982484720545573
```

Cost Sensitive Training

```
parameters_dict = {
    'bootstrap': [True],
    'max_depth': [13,14,15],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [10,20,30]}

clfForestN = GridSearchCV(RandomForestClassifier(), param_grid = parameters_dict, cv=inner_cv, scoring=cost_scorer, refit=True)
nested_score = cross_validate(clfForestN, X=transformedData, y=target, cv=outer_cv, scoring = cost_scorer)
print("Mean Missclassification Cost: {0:.2f}, Std Deviation: {1:.2f} For Normalized
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
```

```
Mean Missclassification Cost: -473.00, Std Deviation: 50.26 For Normalized Data
```

```
clfForestN.fit(X_train, y_train)
print('The best parameter values are ',clfForestN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, clfForestN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance (Esitmate of total reduction in entropy brought by a feature)")
print(clfForestN.best_estimator_.feature_importances_)
```

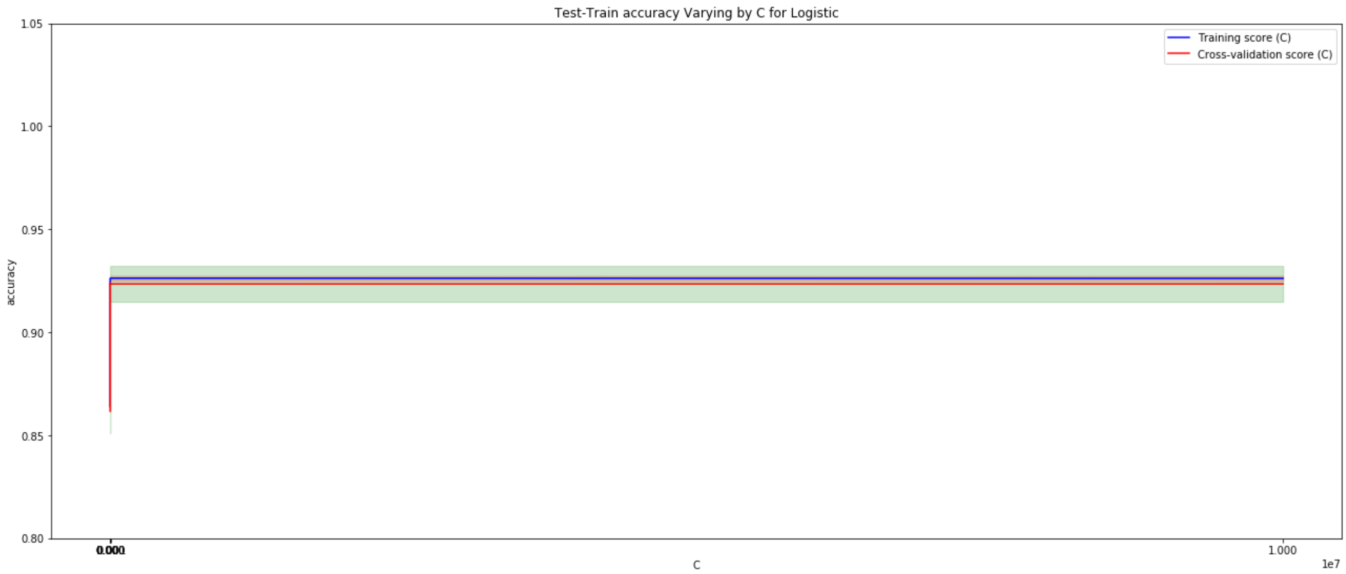
```
The best parameter values are  {'bootstrap': True, 'max_depth': 15, 'min_samples_leaf': 3, 'min_samples_split': 12, 'n_estimators': 20}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.94      0.96      0.95       697
           1       0.94      0.91      0.92       454

    accuracy                           0.94      1151
   macro avg       0.94      0.94      0.94      1151
weighted avg       0.94      0.94      0.94      1151

Confusion Matrix
[[671  26]
 [ 41 413]]

Feature Importance (Esitmate of total reduction in entropy brought by a feature)
[2.43847982e-03 3.00029802e-03 4.82130559e-03 2.10800640e-04
 2.86351108e-02 9.58195299e-03 5.01772595e-02 1.52724026e-02
 2.50341289e-03 4.70053754e-03 1.07247499e-02 5.62556331e-03
 2.57080284e-03 3.89673034e-04 2.30417136e-04 6.38271207e-02
 2.24577722e-02 5.91407973e-03 2.13614865e-02 2.59411996e-03
 1.22912836e-01 1.55619034e-03 2.12687374e-02 1.74029599e-02
 5.82076698e-02 8.34308398e-03 2.24426309e-02 9.02516456e-03
 4.98561253e-04 4.12862897e-03 1.27332005e-03 1.15443541e-05
 9.55434957e-04 5.41053088e-04 9.61333678e-04 2.25836040e-03
 7.85390768e-03 6.99356397e-04 1.27709698e-03 3.94349966e-04
 1.11735679e-03 2.88336834e-03 1.12107381e-03 9.55660089e-04
 8.33770828e-03 1.43206952e-02 0.00000000e+00 5.70421836e-04
 3.98185207e-03 7.96040908e-03 2.49148811e-03 1.92847968e-01
 6.91026968e-02 5.73873578e-03 8.20287018e-02 6.74922984e-02]
```

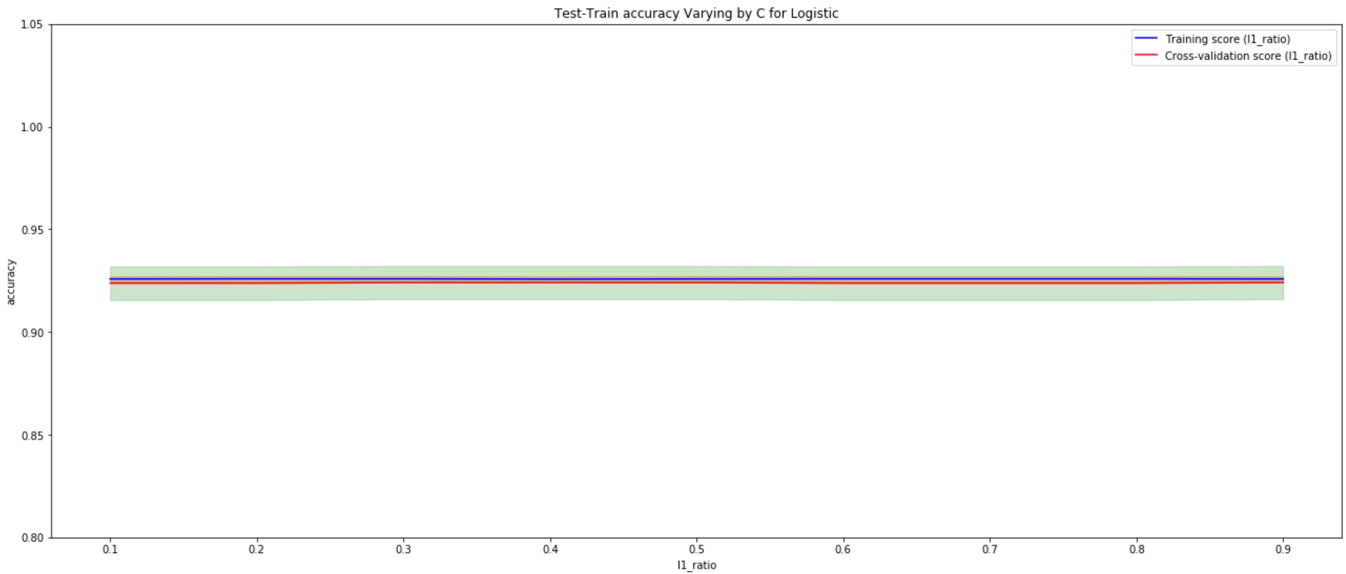
Logistic Regression

Visualizing how different parameters effect Logistic Regression Models

```
parameters_dict = {"C": [0.001,0.01,1,100,10000,1e7], "l1_ratio":np.arange(0.1,1,0.1)}
train_scores, test_scores = validation_curve(
    LogisticRegression(solver = 'saga'),X_train, y_train, param_name="C", cv=5,
    param_range=parameters_dict['C'],scoring="accuracy")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train accuracy Varying by C for Logistic")
plt.xlabel("C")
plt.ylabel("accuracy")
plt.ylim(0.8, 1.05)
plt.fill_between(parameters_dict['C'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['C'], meanTrainScore, label="Training score (C)",
             color="b")
plt.fill_between(parameters_dict['C'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['C'], meanTestScore, label="Cross-validation score (C)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['C'])
plt.show()


train_scores, test_scores = validation_curve(
    LogisticRegression(solver = 'saga'),X_train, y_train, param_name="l1_ratio", cv=5,
    param_range=parameters_dict['l1_ratio'],scoring="accuracy")

#Calculating mean and standard deviations of the scores
```

```
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train accuracy Varying by C for Logistic")
plt.xlabel("l1_ratio")
plt.ylabel("accuracy")
plt.ylim(0.8, 1.05)
plt.fill_between(parameters_dict['l1_ratio'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['l1_ratio'], meanTrainScore, label="Training score (l1_ratio)",
             color="b")
plt.fill_between(parameters_dict['l1_ratio'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['l1_ratio'], meanTestScore, label="Cross-validation score (l1_ratio)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['l1_ratio'])
plt.show()
```





Nested Cross Validation Performance on Normalized Data

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
parameters_dict = {"C": [0.01,100,10000],
                  "penalty":["l1","l2","elasticnet"],
                  "class_weight":["balanced", "None"],
                  "solver":["saga"],"l1_ratio":[0.1,0.5]}
LogitN = GridSearchCV(LogisticRegression(), param_grid = parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(LogitN, X=transformedData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Normalized
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.92, Std Deviation: 0.00 For Normalized Data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.937310	0.005271	0.932568	0.005271	0.934915	0.003305
Not Spam	0.897158	0.007180	0.904027	0.008614	0.900540	0.005207

Best Parameters , Classification Report, Confusion Matrix and Feature Importances

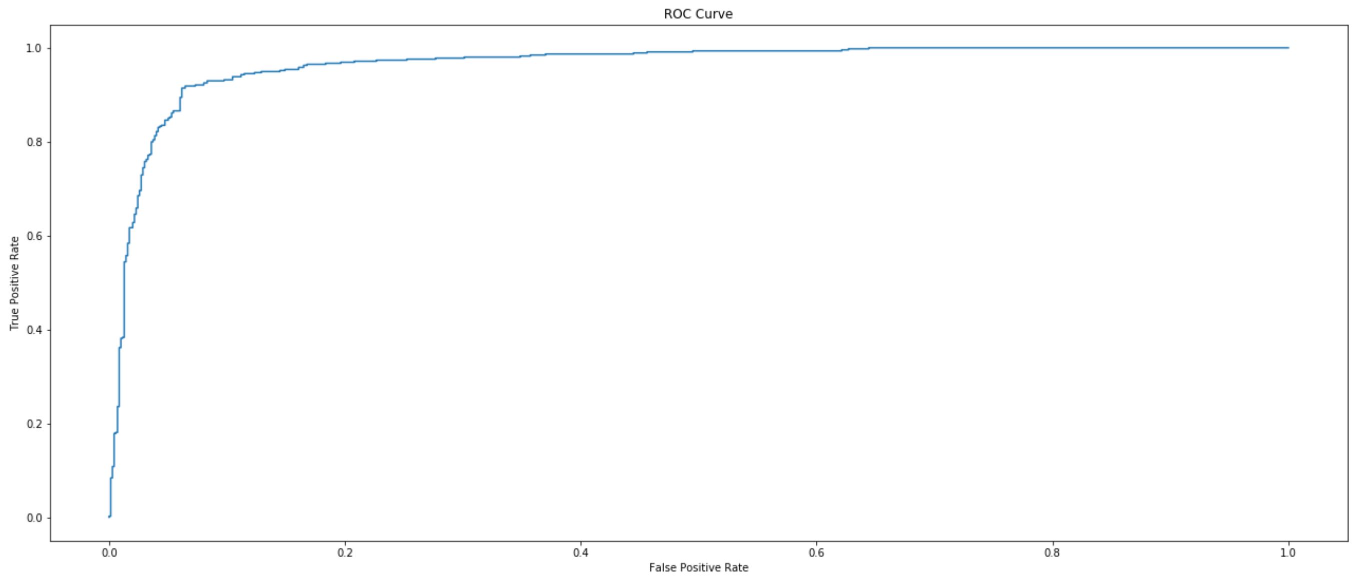
```
LogitN.fit(X_train, y_train)
print('The best parameter values are ',LogitN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, LogitN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance")
print(LogitN.best_estimator_.coef_)
```

```
The best parameter values are  {'C': 100, 'class_weight': 'balanced', 'l1_ratio': 0.5, 'penalty': 'l2', 'solver': 'saga'}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.93      0.94      0.94       697
           1       0.90      0.90      0.90       454

    accuracy                           0.92      1151
   macro avg       0.92      0.92      0.92      1151
weighted avg       0.92      0.92      0.92      1151

Confusion Matrix
[[654  43]
 [ 47 407]]

Feature Importance
[[-0.03830168 -0.19312748  0.06749364  0.38646694  0.33697535  0.2104823
   1.11294452  0.28233126  0.27336953  0.20256493 -0.06193226 -0.18099361
  -0.01260752  0.03692684  0.23054188  0.66957364  0.45578734  0.13870722
   0.114496    0.41833191  0.30122116  0.39378955  0.95483702  0.24184876
  -1.2926243  -0.7903202  -1.14629285  0.08611873 -0.58172971 -0.15927603
  -0.38379646 -0.16165753 -0.42944869 -0.15259894 -0.55175915  0.31789503
   0.02850408 -0.11561462 -0.44498298 -0.19553295 -0.48235027 -0.77360375
  -0.25331967 -0.55264973 -0.65178403 -0.99865174 -0.09723729 -0.45892131
  -0.339352   -0.04084523 -0.19550512  0.75293738  1.27982448  0.39597031
   0.47235082  1.12879055]]
```

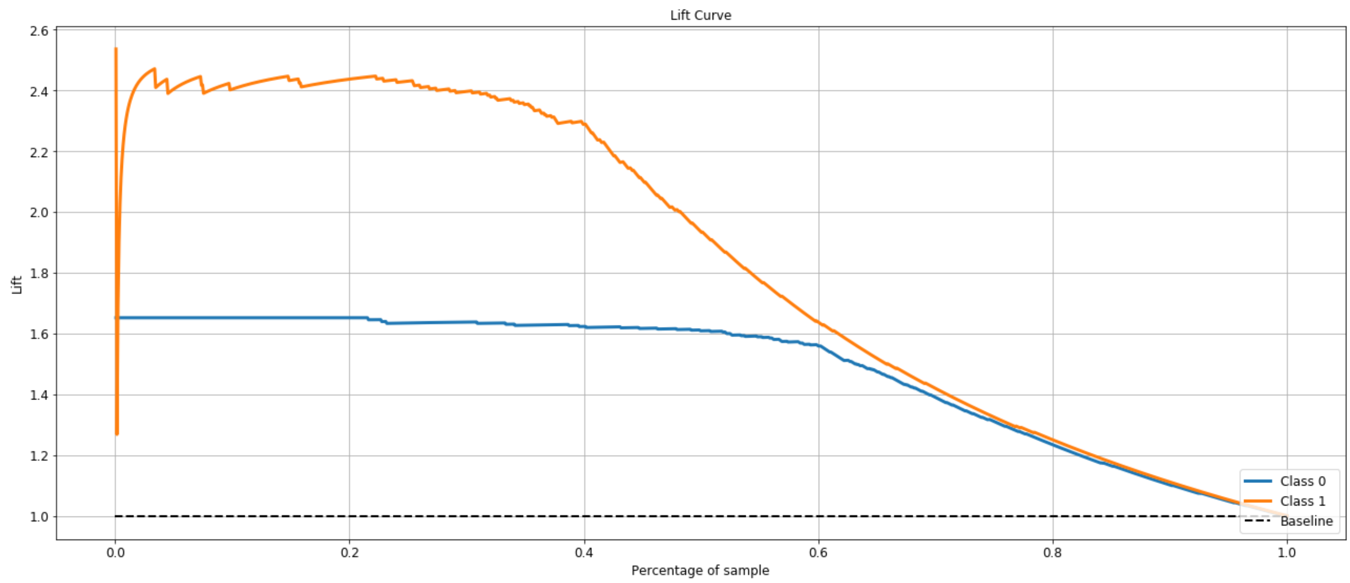
ROC and Lift Curve

```
# Probabilites
y_prob = LogitN.best_estimator_.predict_proba(X_test)
```

```
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(y_true,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
&lt;Figure size 1584x648 with 0 Axes&gt;
```



Area under the ROC curve

```
auc(fpr, tpr)
```

```
0.9645206959973202
```

Nested Cross Validation Performance on MinMax Scaled Data

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
LogitMin = GridSearchCV(LogisticRegression(), parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(LogitMin, X=minData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Scaled
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.93, Std Deviation: 0.00 For Scaled Data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.941275	0.004690	0.936514	0.004690	0.938871	0.002521
Not Spam	0.903177	0.006255	0.910095	0.007999	0.906585	0.004055

Best Parameters , Classification Report, Confusion Matrix and Feature Importances

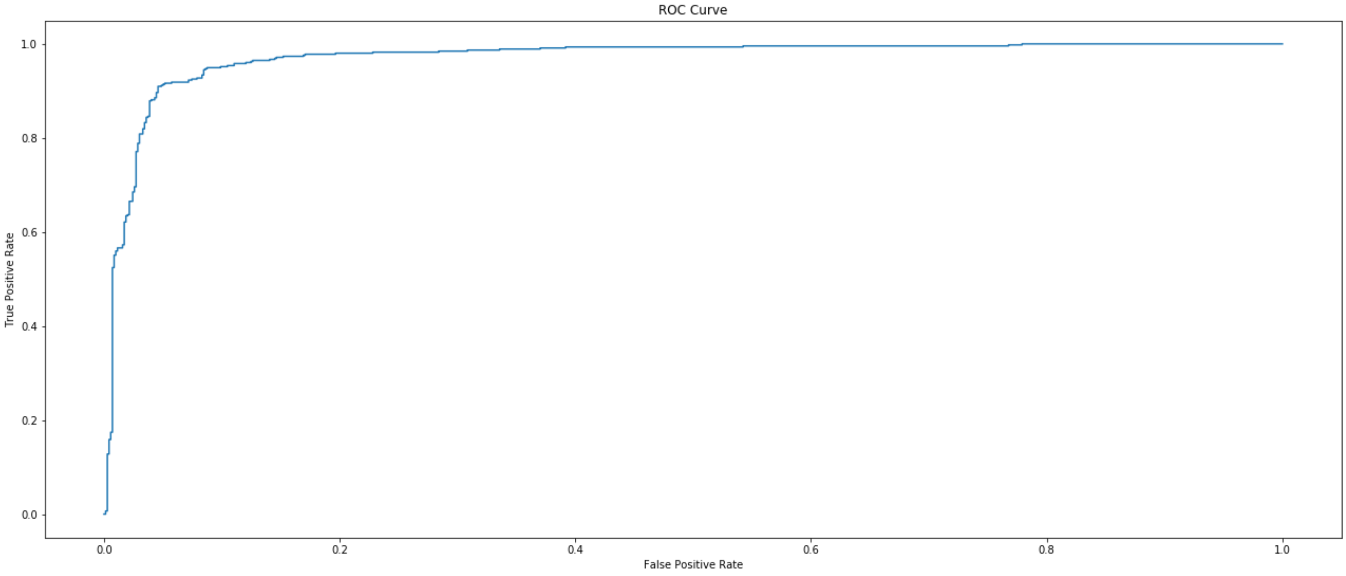
```
LogitMin.fit(Xmin_train, ymin_train)
print('The best parameter values are ',LogitMin.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = ymin_test, LogitMin.best_estimator_.predict(Xmin_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance ")
print(LogitMin.best_estimator_.coef_)
```

```
The best parameter values are  {'C': 10000, 'class_weight': 'balanced', 'l1_ratio': 0.1, 'penalty': 'l1', 'solver': 'saga'}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.94      0.95      0.95       697
           1       0.92      0.91      0.92       454

    accuracy                           0.93      1151
   macro avg       0.93      0.93      0.93      1151
weighted avg       0.93      0.93      0.93      1151

Confusion Matrix
[[661  36]
 [ 39 415]]

Feature Importance
[[ -1.74095759  -2.10819588   0.22311762   9.72749342   6.85632929
    3.37587332  21.13822793   5.35994047   3.48600653   2.66232241
    0.57592553  -1.72575557  -0.91955797   2.23477705  10.08237559
   21.70948366   5.71150826   2.1373666    1.02097704  11.24801724
    2.12204002   8.56832651  14.7287619    8.14517678 -33.95340785
  -17.47134828 -39.09076537  -0.14459451 -11.97116676  -1.44910771
   -3.87614705  -1.89486495 -10.83166604  -2.12867067  -8.31485867
    8.35730322  -0.13000736  -5.09089858  -6.58626709  -1.61940325
  -13.53738266 -19.32469041  -3.50811287 -18.00783261 -13.54918229
  -21.25993873  -3.34102766 -11.95476181  -7.05438095  -4.49423586
   -4.07923817  34.48266714  30.94347274   5.60183489   8.01509014
   17.66329591]]
```

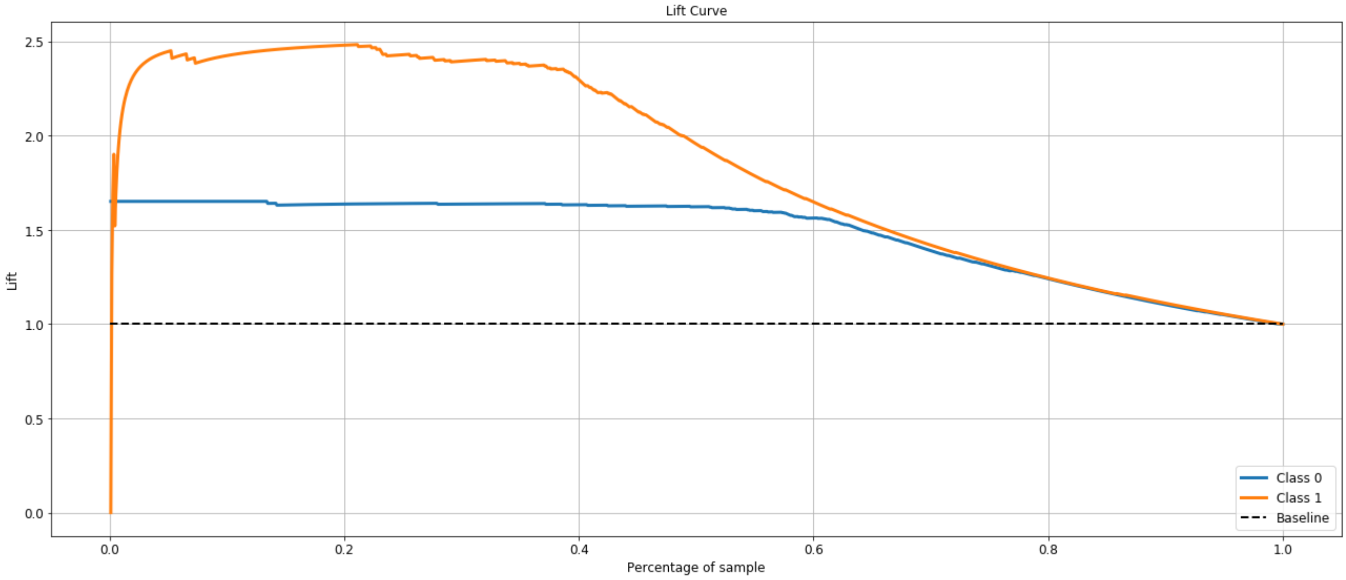
ROC and Lift Curve

```
# Probabilites
y_prob = LogitMin.best_estimator_.predict_proba(Xmin_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(ymin_test,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
```

```
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()

plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC curve

```
auc(fpr, tpr)
```

```
0.9707841662505767
```

Cost Sensitive Training

```
parameters_dict = {"C": [0.01,100,10000],
                  "penalty":["l1","l2","elasticnet"],
                  "class_weight":["balanced", "None"],
                  "solver":["saga"],"l1_ratio":[0.1,0.5]}
LogitN = GridSearchCV(LogisticRegression(), param_grid = parameters_dict, cv=inner_cv, scoring=cost_scorer, refit=True)
nested_score = cross_validate(LogitN, X=transformedData, y=target, cv=outer_cv, scoring = cost_scorer)
print("Mean Missclassification Cost: {0:.2f}, Std Deviation:
{1:.2f}".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
```

```
Mean Missclassification Cost: -482.00, Std Deviation: 38.26
```

```
LogitN.fit(X_train, y_train)
print('The best parameter values are ',LogitN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, LogitN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance")
print(LogitN.best_estimator_.coef_)
```

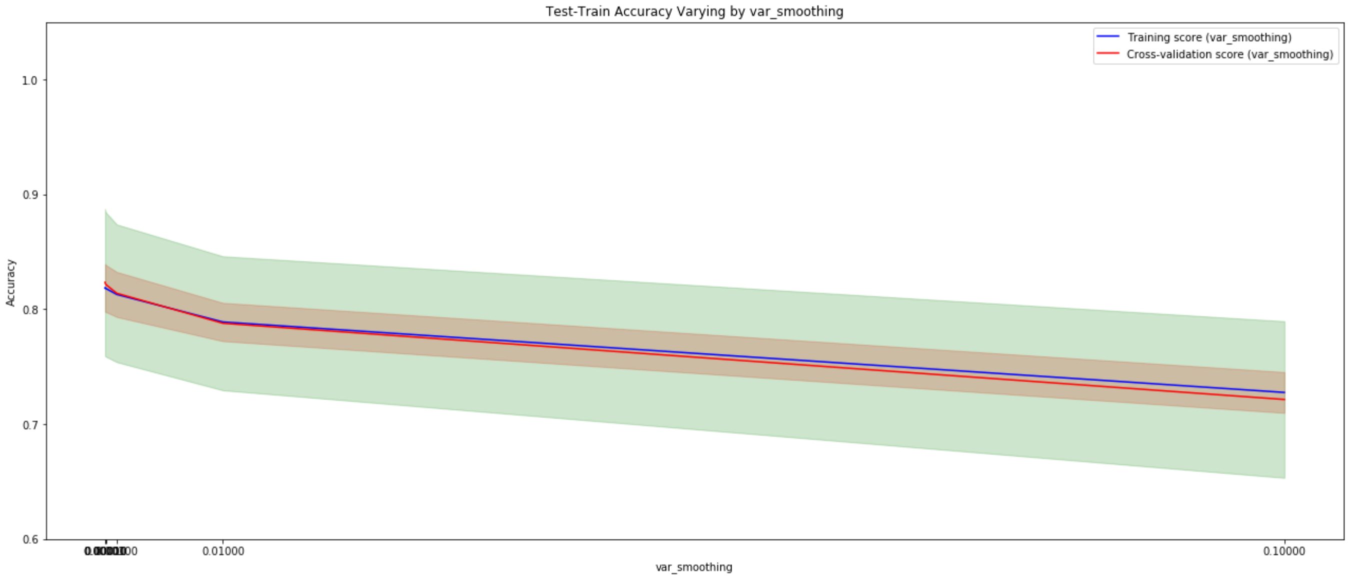
```
The best parameter values are  {'C': 100, 'class_weight': 'balanced', 'l1_ratio': 0.1, 'penalty': 'l1', 'solver': 'saga'}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.94      0.94      0.94       697
           1       0.91      0.91      0.91       454

    accuracy                           0.93      1151
   macro avg       0.92      0.92      0.92      1151
weighted avg       0.93      0.93      0.93      1151

Confusion Matrix
[[654  43]
 [ 41 413]]

Feature Importance
[[-1.43952267e-01 -1.88556207e-01  4.19173055e-02  4.09002235e-01
   5.03158890e-01  1.64860953e-01  8.97518356e-01  1.97353727e-01
   1.70890397e-01  8.07056816e-02  8.46377360e-02 -1.17832236e-01
  -5.66516816e-02 -3.30148110e-04  3.27095125e-01  4.58165518e-01
   3.49792683e-01  1.91453611e-01  1.46980478e-01  4.28370429e-01
   2.00422925e-01  3.82922843e-01  9.58927833e-01  5.27231956e-01
  -1.21151228e+00 -7.68775613e-01 -1.40090378e+00  1.23083391e-01
  -4.41356860e-01 -1.33163288e-01 -2.06884488e-01 -1.99088204e-01
  -2.94222771e-01 -1.96630616e-01 -4.78027730e-01  2.17327258e-01
  -6.92468787e-02  3.92665063e-03 -3.48064566e-01 -1.88513319e-01
  -5.20624234e-01 -8.22642202e-01 -1.70177083e-01 -6.84194870e-01
  -6.93768557e-01 -9.26151645e-01 -1.44773726e-01 -4.54011014e-01
  -2.82054726e-01 -6.16937814e-03 -1.88454454e-01  1.00399571e+00
   1.24740499e+00  3.50058931e-01  4.40746442e-01  1.13367823e+00]]
```

Naive Bayes

Creating Visualizations as to how different parameters affect Naive Baiyes

```
parameters_dict={"var_smoothing":[1e-1,1e-2,1e-3,1e-4,1e-5,1e-6]}
train_scores, test_scores = validation_curve(
    GaussianNB(),transformedData, target, param_name="var_smoothing", cv=5,
    param_range=parameters_dict['var_smoothing'],scoring="accuracy")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train Accuracy Varying by var_smoothing")
plt.xlabel("var_smoothing")
plt.ylabel("Accuracy")
plt.ylim(0.6, 1.05)
plt.fill_between(parameters_dict['var_smoothing'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
```

```
plt.plot(parameters_dict['var_smoothing'], meanTrainScore, label="Training score (var_smoothing)",
             color="b")
plt.fill_between(parameters_dict['var_smoothing'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['var_smoothing'], meanTestScore, label="Cross-validation score (var_smoothing)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['var_smoothing'])
plt.show()
```



Nested Cross Validation Performance on Normalized Data

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
NBN = GridSearchCV(GaussianNB(), param_grid = parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(NBN, X=transformedData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Normalized
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.81, Std Deviation: 0.00 For Normalized Data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.958925	0.004121	0.725251	0.004121	0.825822	0.001986
Not Spam	0.692620	0.001360	0.952026	0.013396	0.801825	0.004814

Best Parameters , Classification Report, Confusion Matrix and Feature Importances

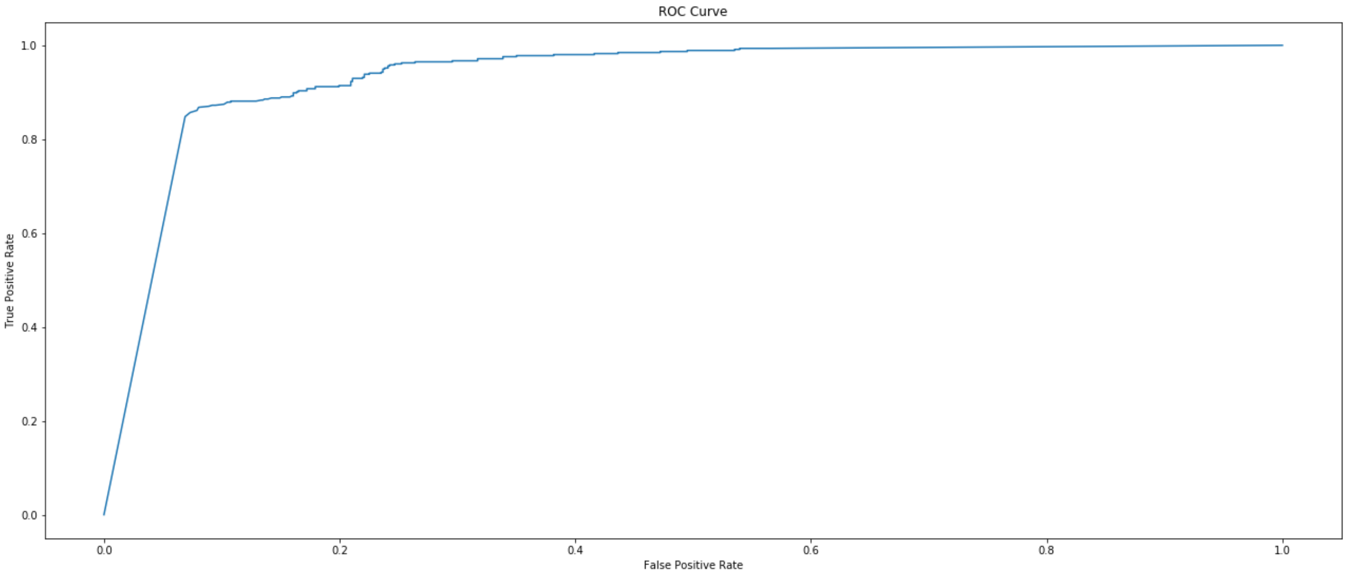
```
NBN.fit(X_train, y_train)
print('The best parameter values are ',NBN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, NBN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
```

```
The best parameter values are  {'var_smoothing': 1e-05}
```

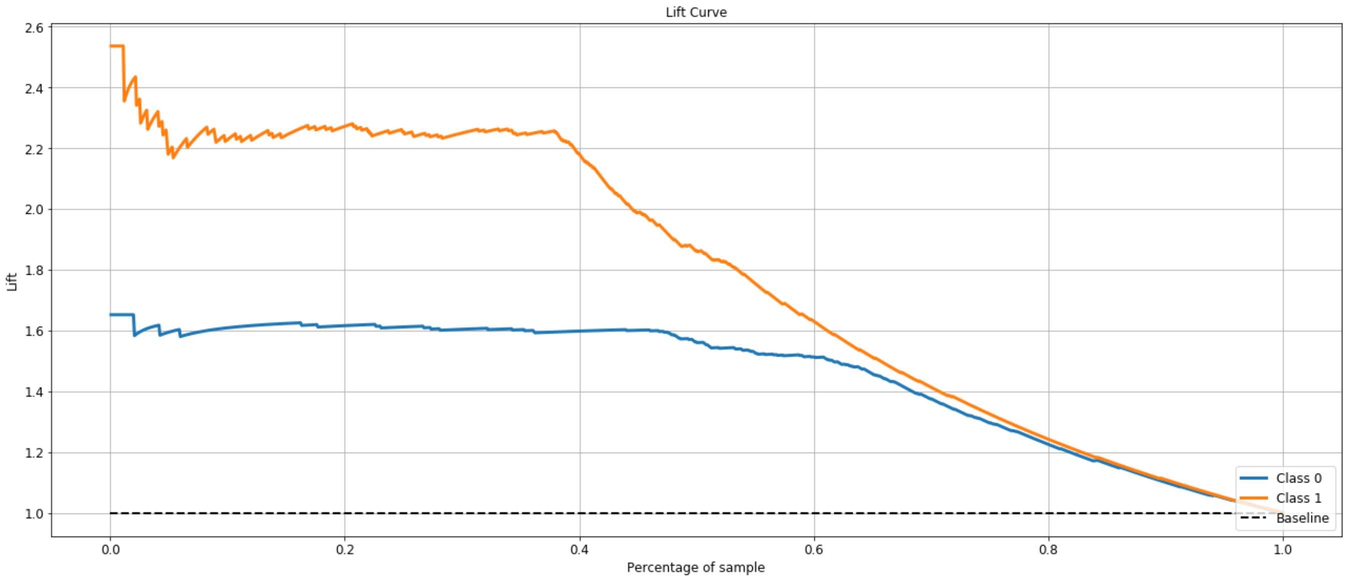
```
Detailed classification report:

              precision    recall  f1-score   support

           0       0.97      0.75      0.84       697
           1       0.71      0.96      0.82       454

    accuracy                           0.83      1151
   macro avg       0.84      0.85      0.83      1151
weighted avg       0.87      0.83      0.83      1151

Confusion Matrix
[[520 177]
 [ 17 437]]
```

ROC and Lift Curve

```
# Probabilites
y_prob = NBN.best_estimator_.predict_proba(X_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(y_true,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC curve

```
auc(fpr, tpr)
```

```
0.9344042118835285
```

Nested Cross Validation Performance on MinMax Scaled Data

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
NBM = GridSearchCV(GaussianNB(), parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(NBM, X=minData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Scaled
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.81, Std Deviation: 0.00 For Scaled Data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.960247	0.004183	0.724534	0.004183	0.825847	0.001354
Not Spam	0.692438	0.000854	0.953680	0.012902	0.802289	0.004306

Best Parameters , Classification Report, Confusion Matrix and Feature Importances

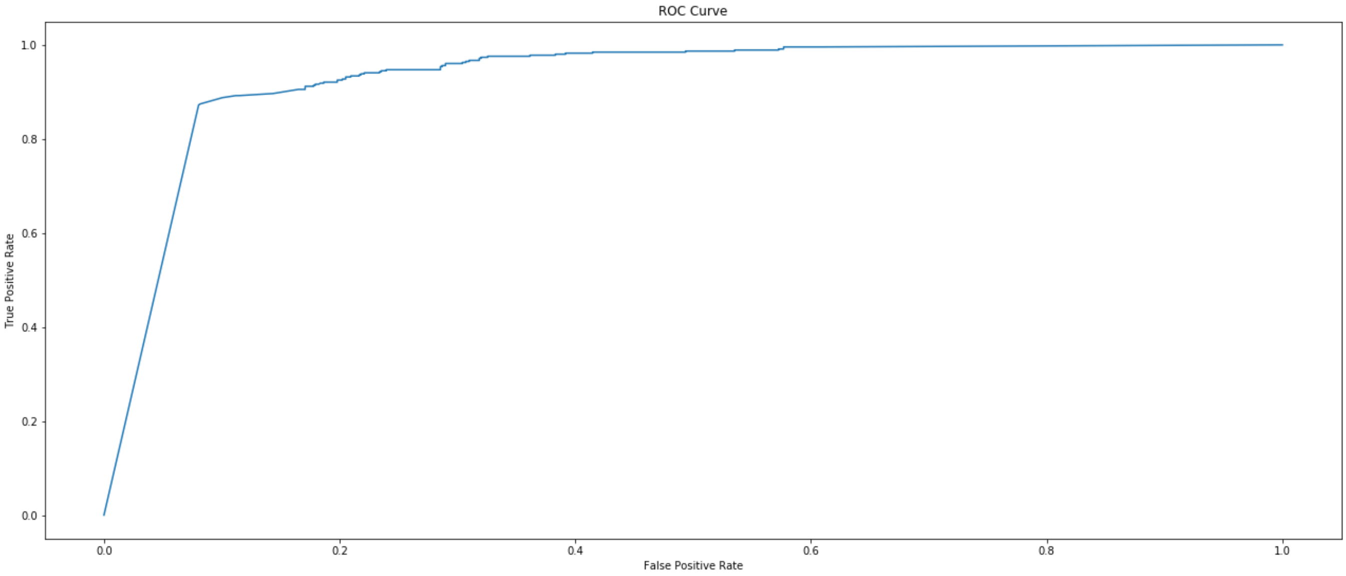
```
NBM.fit(Xmin_train, ymin_train)
print('The best parameter values are ',NBM.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = ymin_test, NBM.best_estimator_.predict(Xmin_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
```

```
The best parameter values are  {'var_smoothing': 0.0001}
```

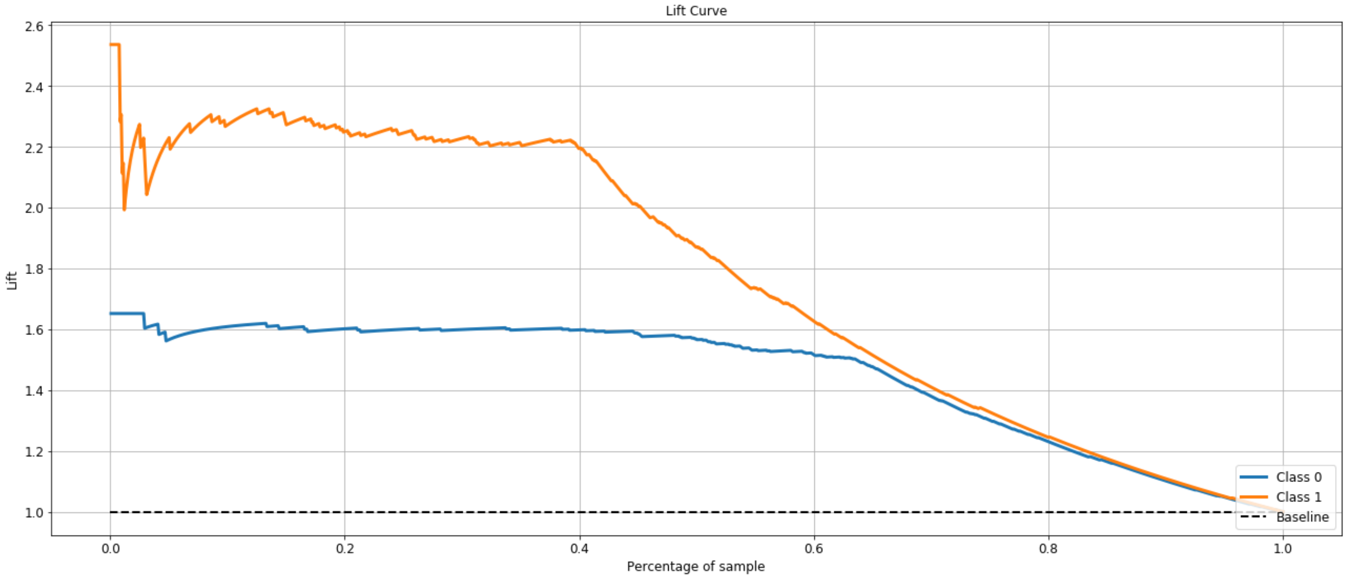
```
Detailed classification report:

              precision    recall  f1-score   support

           0       0.96      0.75      0.84       697
           1       0.71      0.95      0.81       454

    accuracy                           0.83      1151
   macro avg       0.83      0.85      0.83      1151
weighted avg       0.86      0.83      0.83      1151

Confusion Matrix
[[521 176]
 [ 24 430]]
```

ROC and Lift Curve

```
# Probabilites
y_prob = NBM.best_estimator_.predict_proba(Xmin_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(ymin_test,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()

plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC curve

```
auc(fpr, tpr)
```

```
0.9310891865073094
```

Cost Sensitive Training

```
parameters_dict={"var_smoothing":[1e-1,1e-2,1e-3,1e-4,1e-5,1e-6]}
NBN = GridSearchCV(GaussianNB(), param_grid = parameters_dict, cv=inner_cv, scoring=cost_scorer, refit=True)
nested_score = cross_validate(NBN, X=transformedData, y=target, cv=outer_cv, scoring = cost_scorer)
print("Mean Missclassification Cost: {0:.2f}, Std Deviation:
{1:.2f}".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
```

```
Mean Missclassification Cost: -406.25, Std Deviation: 53.93
```

```
NBN.fit(X_train, y_train)
print('The best parameter values are ',NBN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, NBN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
```

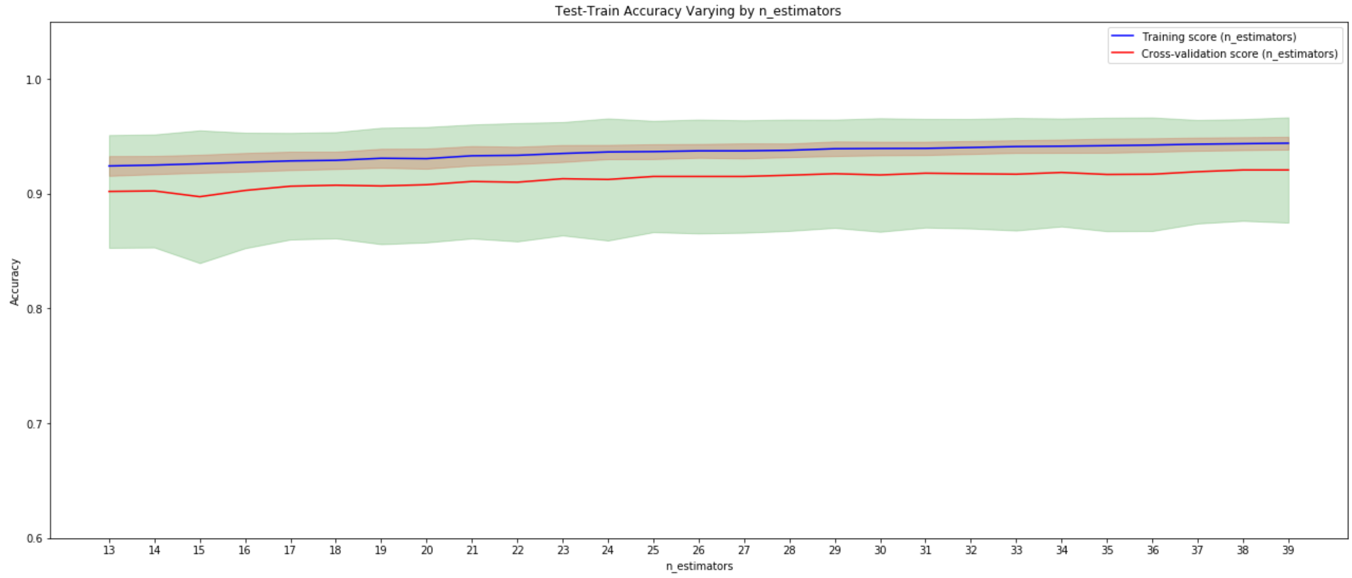
```
The best parameter values are  {'var_smoothing': 0.0001}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.97      0.73      0.83       697
           1       0.70      0.96      0.81       454

    accuracy                           0.82      1151
   macro avg       0.83      0.84      0.82      1151
weighted avg       0.86      0.82      0.82      1151

Confusion Matrix
[[506 191]
 [ 17 437]]
```

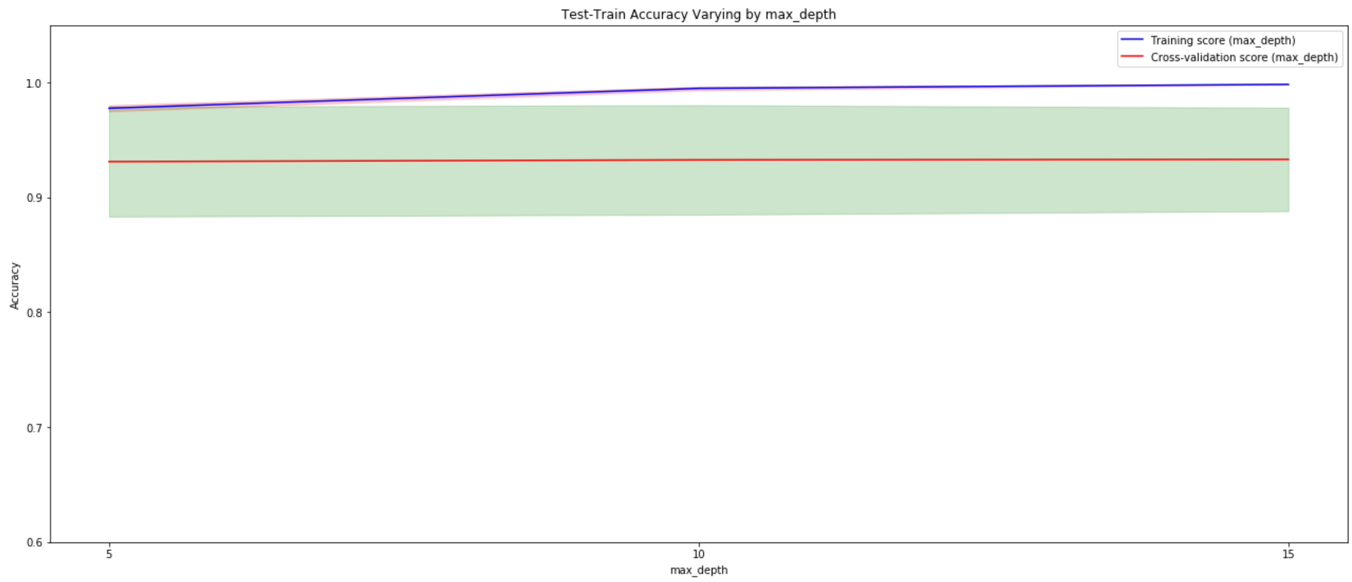
XGBoost

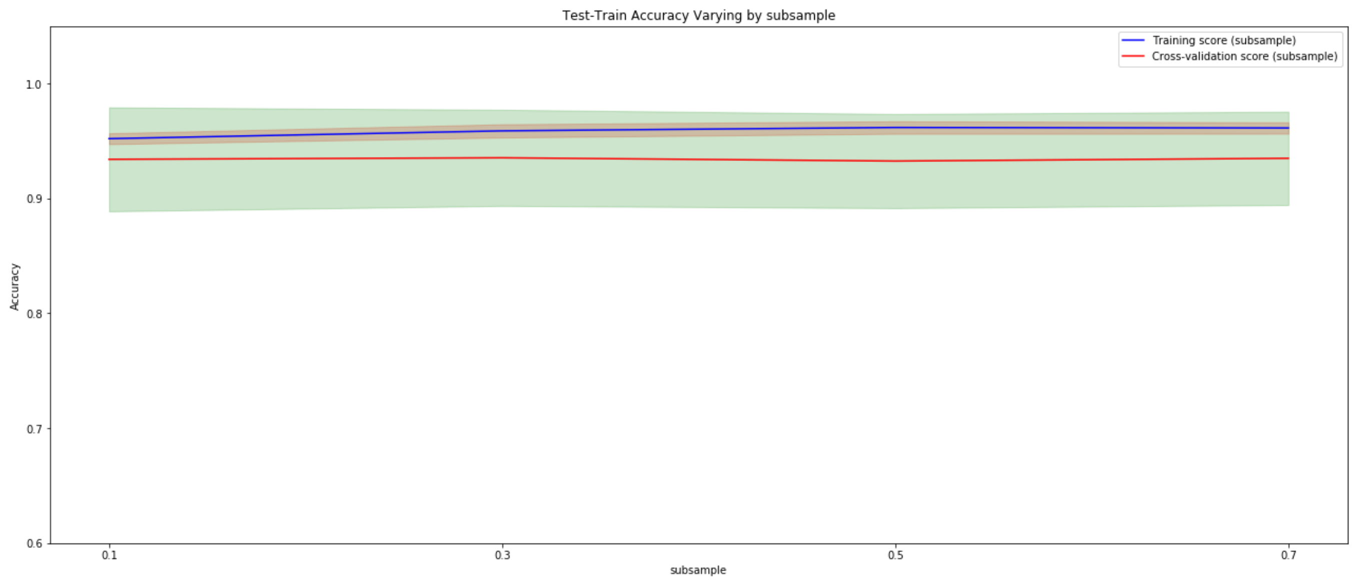
Creating Visualizations as to how parameters affect XGBoost

```python
import xgboost as xgb
parameters_dict = {
    'learning_rate': [0.001,0.01,0.1,1],
    'n_estimators': list(range(13,40)),
    'max_depth' : np.arange(5,20,5),
    'subsample':np.arange(0.1,0.8,0.2),
    'gamma' :[0,1,5]
    }
train_scores, test_scores = validation_curve(
    xgb.XGBClassifier(),transformedData, target, param_name="n_estimators", cv=5,
    param_range=parameters_dict['n_estimators'],scoring="accuracy")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train Accuracy Varying by n_estimators")
plt.xlabel("n_estimators")
plt.ylabel("Accuracy")
plt.ylim(0.6, 1.05)
plt.fill_between(parameters_dict['n_estimators'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['n_estimators'], meanTrainScore, label="Training score (n_estimators)",
             color="b")
plt.fill_between(parameters_dict['n_estimators'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['n_estimators'], meanTestScore, label="Cross-validation score (n_estimators)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['n_estimators'])
plt.show()
######################################################################
train_scores, test_scores = validation_curve(
    xgb.XGBClassifier(),transformedData, target, param_name="max_depth", cv=5,
    param_range=parameters_dict['max_depth'],scoring="accuracy")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train Accuracy Varying by max_depth")
plt.xlabel("max_depth")
plt.ylabel("Accuracy")
plt.ylim(0.6, 1.05)
plt.fill_between(parameters_dict['max_depth'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['max_depth'], meanTrainScore, label="Training score (max_depth)",
             color="b")
plt.fill_between(parameters_dict['max_depth'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['max_depth'], meanTestScore, label="Cross-validation score (max_depth)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['max_depth'])
plt.show()

######################################################################
train_scores, test_scores = validation_curve(
    xgb.XGBClassifier(),transformedData, target, param_name="subsample", cv=5,
    param_range=parameters_dict['subsample'],scoring="accuracy")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train Accuracy Varying by subsample")
plt.xlabel("subsample")
plt.ylabel("Accuracy")
plt.ylim(0.6, 1.05)
plt.fill_between(parameters_dict['subsample'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['subsample'], meanTrainScore, label="Training score (subsample)",
             color="b")
plt.fill_between(parameters_dict['subsample'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['subsample'], meanTestScore, label="Cross-validation score (subsample)",
```

```
            color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['subsample'])
plt.show()
```







Nested Cross Validation Performance on Normalized Data

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
parameters_dict = {
    'learning_rate': [0.001,0.01,0.1,1],
    'n_estimators': list(range(13,16)),
    'max_depth' : [3,4,5],
    'subsample':[0.8,0.9,1],
    'gamma' :[0,1,5]
    }
xgBN = GridSearchCV(xgb.XGBClassifier(), param_grid = parameters_dict, cv=inner_cv, scoring="accuracy", refit=True)
nested_score = cross_validate(xgBN, X=transformedData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Normalized
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.94, Std Deviation: 0.01 For Normalized Data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.948248	0.002958	0.957676	0.002958	0.952909	0.003919
Not Spam	0.933933	0.003930	0.919476	0.014484	0.926576	0.006924

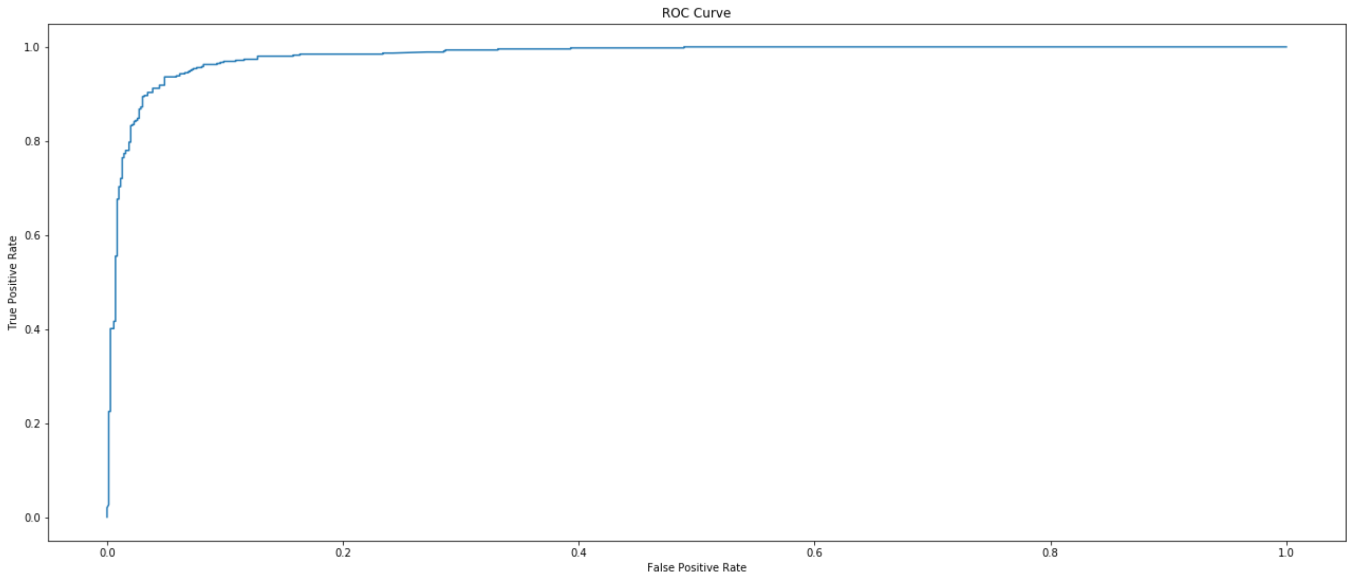
Best Parameters , Classification Report, Confusion Matrix and Feature Importances

```
xgBN.fit(X_train, y_train)
print('The best parameter values are ',xgBN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, xgBN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance")
print(xgBN.best_estimator_.feature_importances_)
```

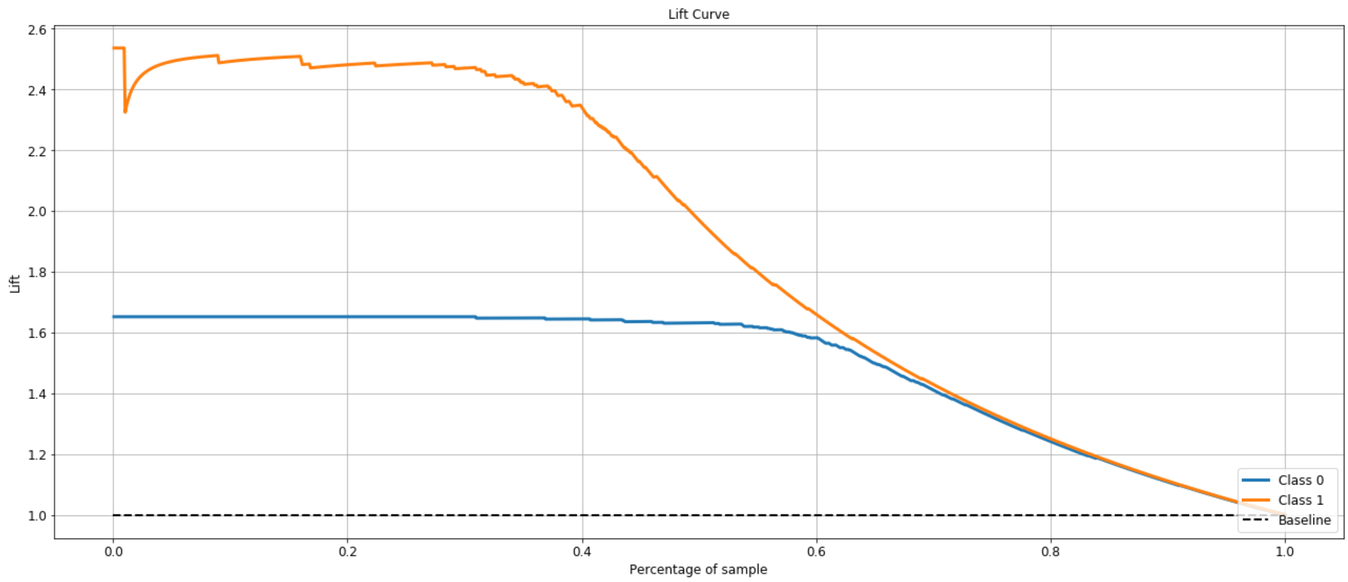
```
The best parameter values are  {'gamma': 1, 'learning_rate': 1, 'max_depth': 3, 'n_estimators': 15, 'subsample': 0.9}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.95      0.95      0.95       697
           1       0.93      0.92      0.92       454

    accuracy                           0.94      1151
   macro avg       0.94      0.94      0.94      1151
weighted avg       0.94      0.94      0.94      1151

Confusion Matrix
[[664  33]
 [ 37 417]]

Feature Importance
[0.00692382 0.         0.00578775 0.         0.03173639 0.01347812
 0.13880989 0.02305574 0.00436854 0.         0.00666837 0.01007587
 0.         0.         0.         0.03413188 0.02465141 0.01105762
 0.0049889  0.         0.00984289 0.         0.00586271 0.09690738
 0.04598042 0.01541704 0.04473614 0.01203421 0.         0.
 0.         0.         0.         0.         0.02555886 0.00794442
 0.02811981 0.         0.         0.         0.         0.01397299
 0.00624196 0.         0.01245473 0.02368092 0.         0.00961246
 0.01054851 0.00643689 0.         0.13804297 0.09498678 0.
 0.01911966 0.05676398]
```

ROC and Lift Curve

```
# Probabilites
y_prob = xgBN.best_estimator_.predict_proba(X_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(y_true,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC curve

```
auc(fpr, tpr)
```

```
0.9817104772498878
```

Cost Sensitive Training

```
parameters_dict = {
    'learning_rate': [0.001,0.01,0.1,1],
    'n_estimators': list(range(13,16)),
    'max_depth' : [3,4,5],
    'subsample':[0.8,0.9,1],
    'gamma' :[0,1,5]
    }
xgBN = GridSearchCV(xgb.XGBClassifier(), param_grid = parameters_dict, cv=inner_cv, scoring=cost_scorer, refit=True)
nested_score = cross_validate(xgBN, X=transformedData, y=target, cv=outer_cv, scoring = cost_scorer)
print("Mean Missclassification Cost: {0:.2f}, Std Deviation:
{1:.2f}".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
```

```
Mean Missclassification Cost: -367.00, Std Deviation: 42.57
```

```
xgBN.fit(X_train, y_train)
print('The best parameter values are ',xgBN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, xgBN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance")
print(xgBN.best_estimator_.feature_importances_)
```

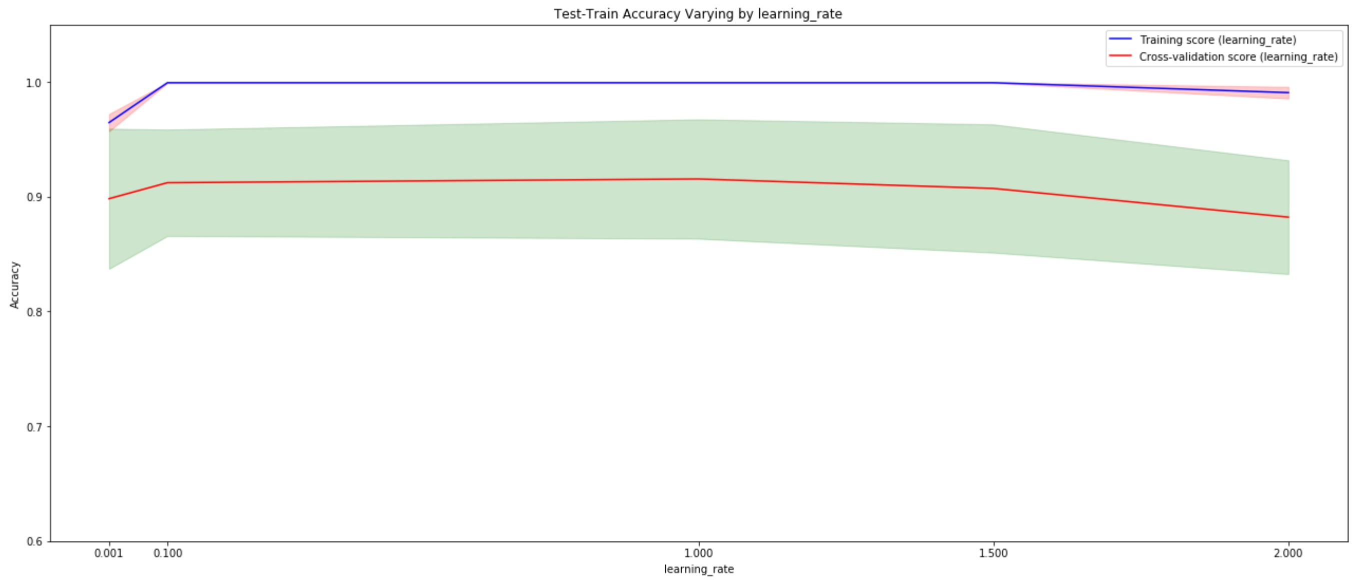
```
The best parameter values are  {'gamma': 0, 'learning_rate': 1, 'max_depth': 5, 'n_estimators': 13, 'subsample': 0.8}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.96      0.94      0.95       697
           1       0.91      0.94      0.92       454

    accuracy                           0.94      1151
   macro avg       0.93      0.94      0.94      1151
weighted avg       0.94      0.94      0.94      1151

Confusion Matrix
[[654  43]
 [ 27 427]]

Feature Importance
[0.00853592 0.0050316  0.01103728 0.00708287 0.03037485 0.01107897
 0.12372988 0.04829952 0.00665891 0.00860901 0.01178051 0.00680773
 0.00470573 0.         0.         0.04624273 0.01088742 0.01224185
 0.00668523 0.         0.04171608 0.00849206 0.026623   0.02200425
 0.05332543 0.01847164 0.03452117 0.03403446 0.         0.00103387
 0.         0.         0.00632686 0.         0.         0.
 0.02775458 0.         0.         0.         0.         0.03608054
 0.01801215 0.0136992  0.02210433 0.04012424 0.         0.
 0.00546882 0.00576184 0.         0.12508725 0.03948465 0.00707475
 0.0401592  0.01284962]
```

AdaBoost

Creating Visualizations as to how different parameters affect AdaBoost

```
parameters_dict = {
    'n_estimators': list(range(13,16)),
    'algorithm': ['SAMME', 'SAMME.R'],
    'learning_rate' : [0.001,0.1,1,1.5,2]
    }
train_scores, test_scores = validation_curve(
    AdaBoostClassifier(DecisionTreeClassifier(max_depth=8)),transformedData, target, param_name="learning_rate", cv=5,
    param_range=parameters_dict['learning_rate'],scoring="accuracy")

#Calculating mean and standard deviations of the scores
meanTrainScore = np.mean(train_scores, axis =1)
stdDevTrain = np.std(train_scores, axis=1)
meanTestScore = np.mean(test_scores, axis=1)
stdTestScore = np.std(test_scores, axis=1)

#Plotting
plt.figure(figsize=(22,9))
#plt.subplot(2,2,1)
plt.title("Test-Train Accuracy Varying by learning_rate")
plt.xlabel("learning_rate")
plt.ylabel("Accuracy")
```

```
plt.ylim(0.6, 1.05)
plt.fill_between(parameters_dict['learning_rate'], meanTrainScore - stdDevTrain, meanTrainScore + stdDevTrain, alpha=0.2, color="r")
plt.plot(parameters_dict['learning_rate'], meanTrainScore, label="Training score (learning_rate)",
             color="b")
plt.fill_between(parameters_dict['learning_rate'], meanTestScore - stdTestScore, meanTestScore + stdTestScore, alpha=0.2, color="g")
plt.plot(parameters_dict['learning_rate'], meanTestScore, label="Cross-validation score (learning_rate)",
             color="r")

plt.legend(loc="best")
plt.xticks(parameters_dict['learning_rate'])
plt.show()
```



Nested Cross Validation Performance on Normalized Data

```
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
adaN = GridSearchCV(AdaBoostClassifier(DecisionTreeClassifier(max_depth=8)), param_grid = parameters_dict, cv=inner_cv, scoring="accuracy",
refit=True)
nested_score = cross_validate(adaN, X=transformedData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Normalized
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.95, Std Deviation: 0.01 For Normalized Data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.952133	0.002352	0.960904	0.002352	0.956469	0.004985
Not Spam	0.939009	0.003416	0.925553	0.016232	0.932160	0.008658

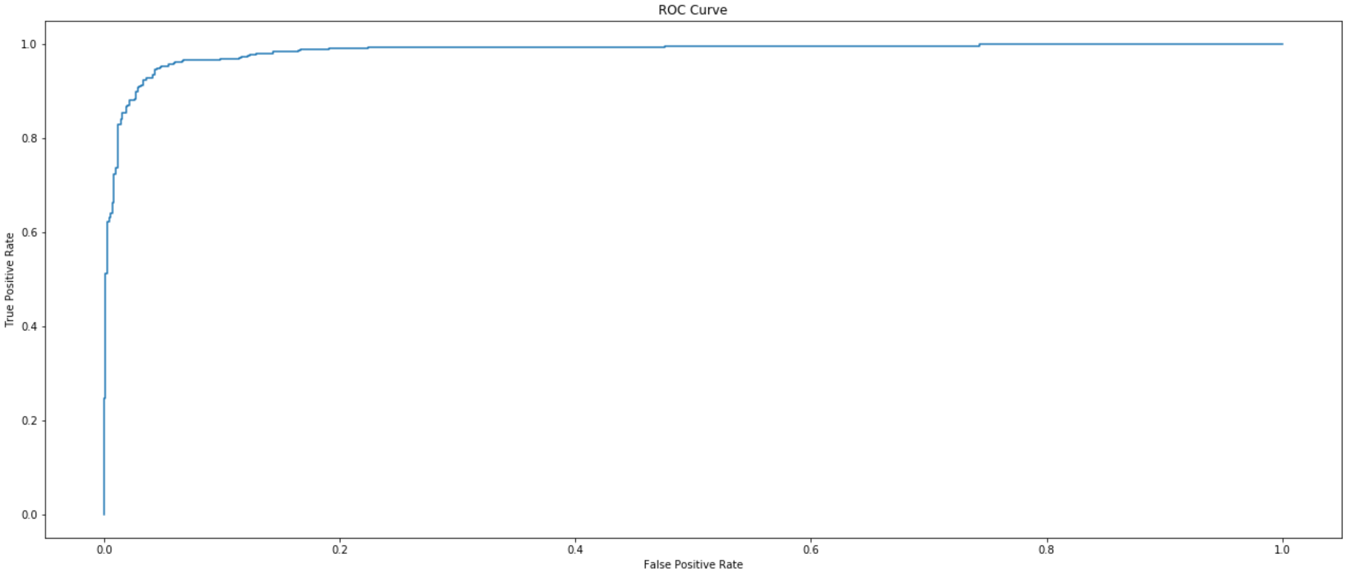
Best Parameters , Classification Report, Confusion Matrix and Feature Importances

```
adaN.fit(X_train, y_train)
print('The best parameter values are ',adaN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, adaN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance")
print(adaN.best_estimator_.feature_importances_)
```

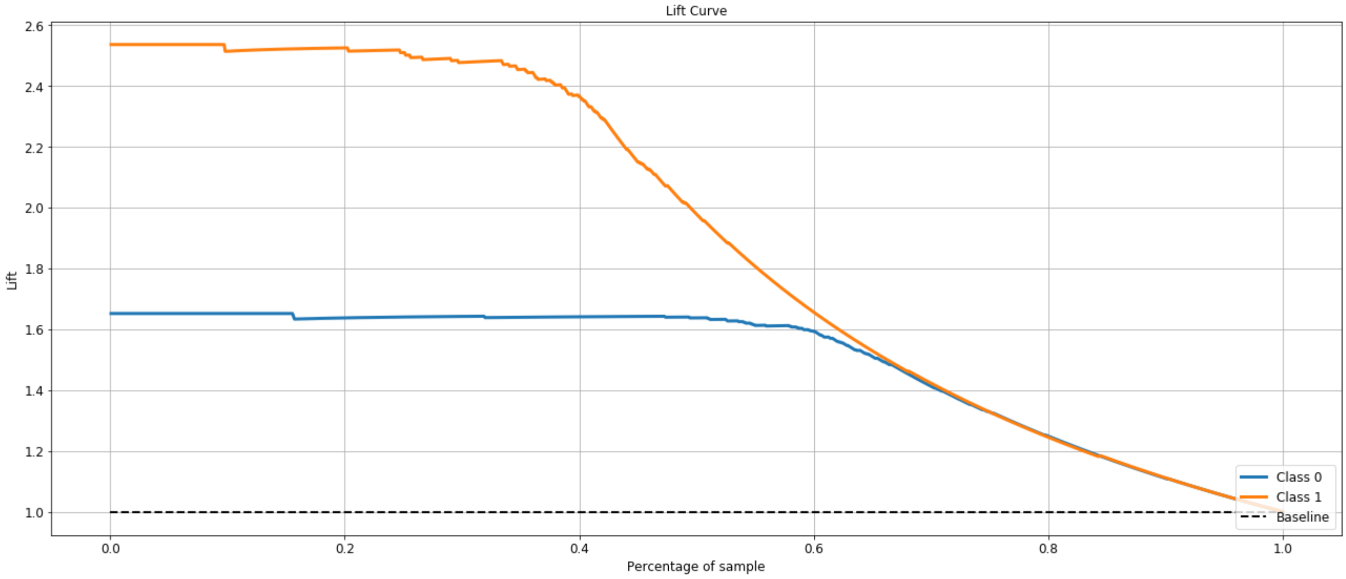
```
The best parameter values are  {'algorithm': 'SAMME', 'learning_rate': 1, 'n_estimators': 14}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.95      0.96      0.96       697
           1       0.94      0.93      0.93       454

    accuracy                           0.95      1151
   macro avg       0.95      0.94      0.95      1151
weighted avg       0.95      0.95      0.95      1151

Confusion Matrix
[[672  25]
 [ 34 420]]

Feature Importance
[1.06238625e-02 7.89342471e-03 2.41140381e-02 2.28323897e-04
 3.59316787e-02 2.23677343e-02 3.15539917e-02 1.21951745e-02
 7.91624385e-03 1.61979192e-02 3.43081781e-03 3.77376280e-02
 2.13595132e-03 3.28675072e-03 1.13276834e-03 3.41600956e-02
 7.84953174e-03 1.37666214e-02 6.67296149e-02 2.89388267e-03
 3.94532631e-02 3.18507471e-03 3.10661041e-03 1.34469470e-02
 3.34894336e-02 4.74474567e-03 3.61656528e-02 1.32560205e-02
 4.05286170e-03 0.00000000e+00 2.95546958e-05 0.00000000e+00
 9.74322991e-03 0.00000000e+00 3.55455266e-03 5.53202306e-03
 6.44152558e-03 3.36540061e-04 3.64772946e-03 7.28786368e-05
 1.57460580e-03 1.03753971e-02 6.51297636e-04 8.29873697e-03
 1.85210352e-02 2.26128400e-02 1.89950750e-03 3.46506417e-03
 1.62124308e-02 4.04555508e-02 6.55940664e-03 9.47989932e-02
 3.74992070e-02 4.04032035e-03 1.16238548e-01 9.43923618e-02]
```

ROC and Lift Curve

```python
# Probabilites
y_prob = adaN.best_estimator_.predict_proba(X_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(y_true,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC curve

```
auc(fpr, tpr)
```

```
0.9847584676935134
```

Cost Sensitive Training

```
parameters_dict = {
    'n_estimators': list(range(13,16)),
    'algorithm': ['SAMME', 'SAMME.R'],
    'learning_rate' : [0.001,0.1,1,1.5,2]}
adaN = GridSearchCV(AdaBoostClassifier(DecisionTreeClassifier(max_depth=8)), param_grid = parameters_dict, cv=inner_cv, scoring=cost_scorer, refit=True)
nested_score = cross_validate(adaN, X=transformedData, y=target, cv=outer_cv, scoring = cost_scorer)
print("Mean Missclassification Cost: {0:.2f}, Std Deviation: {1:.2f}".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
adaN.fit(X_train, y_train)
print('The best parameter values are ',adaN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, adaN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
print("Feature Importance")
print(adaN.best_estimator_.feature_importances_)
```

```
Mean Missclassification Cost: -348.25, Std Deviation: 32.43
The best parameter values are  {'algorithm': 'SAMME', 'learning_rate': 1, 'n_estimators': 14}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.97      0.95      0.96       697
           1       0.92      0.96      0.94       454

    accuracy                           0.95      1151
   macro avg       0.95      0.95      0.95      1151
weighted avg       0.95      0.95      0.95      1151

Confusion Matrix
[[659  38]
 [ 20 434]]

Feature Importance
[1.90088412e-02 1.62653184e-02 1.21051345e-02 7.19142909e-04
 3.44858605e-02 6.57379692e-03 3.96845105e-02 1.31036679e-02
 7.84855145e-03 2.47623652e-02 1.40506786e-02 2.35157635e-02
 2.26613372e-03 1.02320487e-03 2.71376969e-04 3.82843053e-02
 6.79305475e-03 1.22256858e-02 6.85120832e-02 1.84571784e-03
 5.46059687e-02 4.47714925e-04 1.07897976e-02 1.06895917e-02
 3.89275654e-02 1.05600846e-02 3.51905443e-02 1.04040974e-02
```

```
 1.30448919e-03 4.17929093e-04 0.00000000e+00 2.27799206e-05
 1.04489161e-02 4.48917413e-05 7.36796207e-03 5.25960011e-03
 1.35528205e-02 4.29385313e-05 7.93459754e-03 1.18034213e-06
 9.02629436e-04 1.32701089e-02 2.57079289e-03 3.68598369e-03
 2.25447126e-02 2.34212603e-02 9.26964278e-06 4.17176313e-03
 2.28354981e-03 2.39481879e-02 4.71910152e-03 1.11234253e-01
 2.45516026e-02 1.07400315e-02 1.08848406e-01 8.17396831e-02]
```

Stacking

```
# Initializing models

clf1 = KNeighborsClassifier(n_neighbors=26)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
lr = LogisticRegression()
sclf = StackingClassifier(classifiers=[clf1, clf2, clf3],
                          meta_classifier=lr)

params = {'kneighborsclassifier__n_neighbors': [1, 5],
          'randomforestclassifier__n_estimators': [10, 50],
          'meta_classifier__C': [0.1, 10.0]}
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
stackN = GridSearchCV(estimator=sclf,
                    param_grid=params,
                    cv=inner_cv,
                    refit=True)

nested_score = cross_validate(stackN, X=transformedData, y=target, cv=outer_cv, scoring = make_scorer(scoringFunction))
print("Mean Accuracy: {0:.2f}, Std Deviation: {1:.2f} For Normalized
Data".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
createScoreDataFrame(precisionArraySpam,recallArraySpam, f1ArraySpam,precisionArrayNSpam,recallArrayNSpam, f1ArrayNSpam)
```

```
Mean Accuracy: 0.95, Std Deviation: 0.01 For Normalized Data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Mean Precision	Standard Deviation Precision	Mean Recall	Standard Deviation Recall	Mean F1	Standard Deviation F1
Spam	0.951573	0.003569	0.970230	0.003569	0.960779	0.005592
Not Spam	0.952803	0.005400	0.923884	0.017203	0.938039	0.009687

Best Parameters , Classification Report, Confusion Matrix and Feature Importances

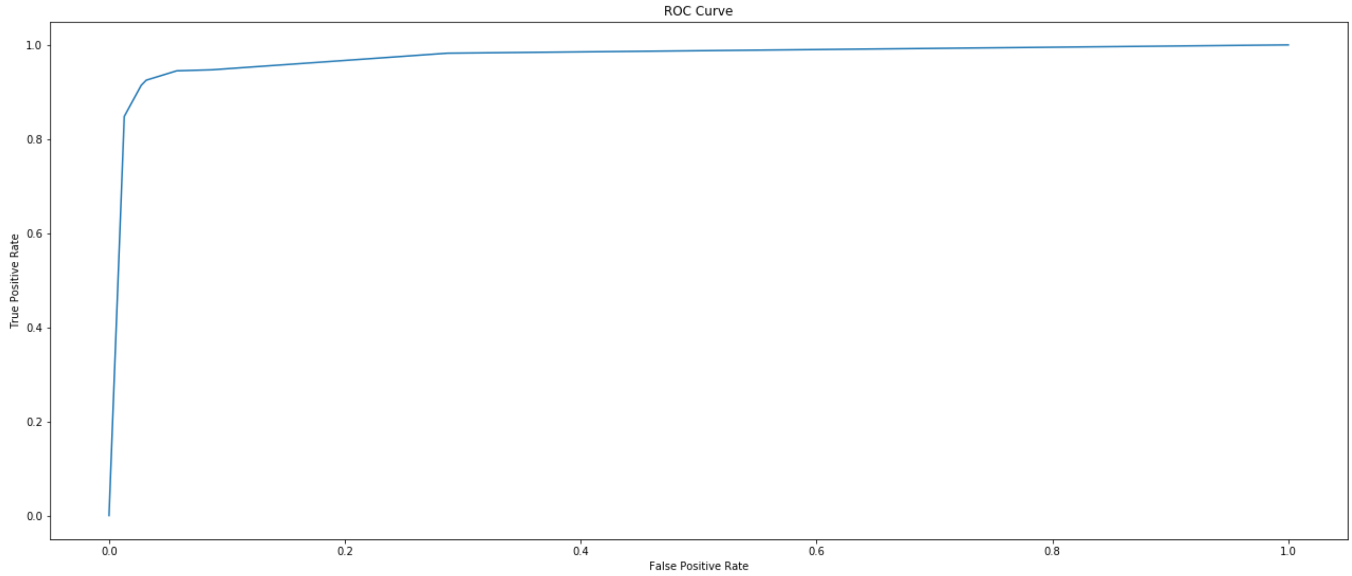
```
stackN.fit(X_train, y_train)
print('The best parameter values are ',stackN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, stackN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
```

```
The best parameter values are  {'kneighborsclassifier__n_neighbors': 5, 'meta_classifier__C': 0.1, 'randomforestclassifier__n_estimators':
50}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.95      0.97      0.96       697
           1       0.95      0.93      0.94       454

    accuracy                           0.95      1151
   macro avg       0.95      0.95      0.95      1151
weighted avg       0.95      0.95      0.95      1151

Confusion Matrix
[[675  22]
```

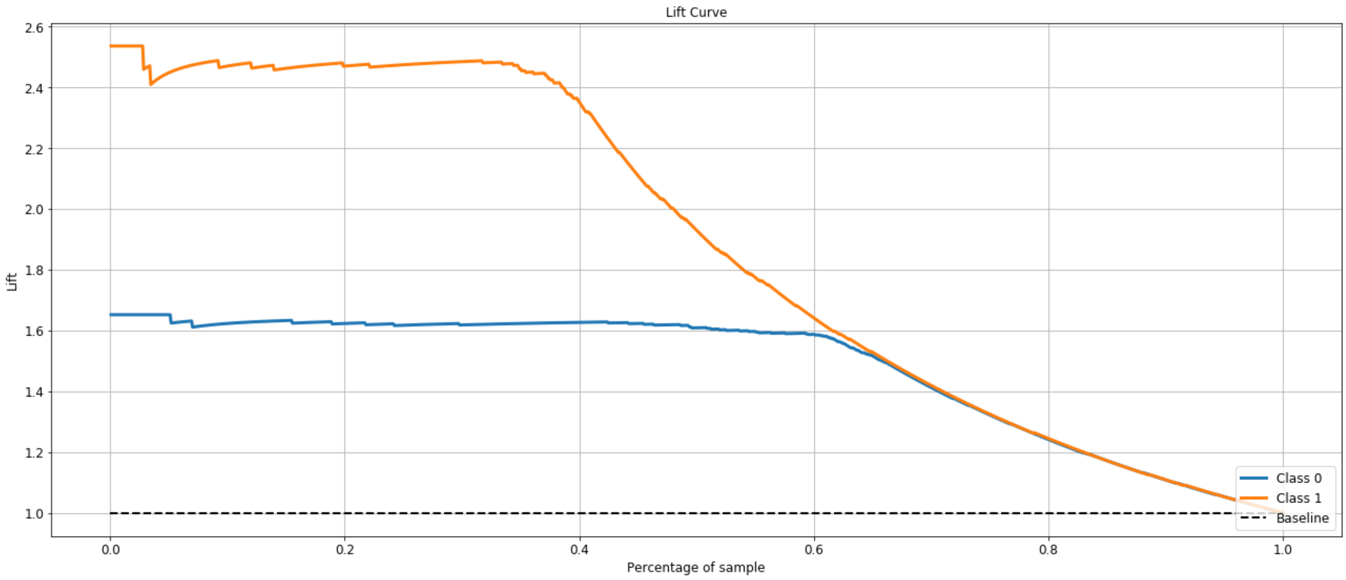
```
 [ 34 420]]
```

ROC and Lift Curve

```
# Probabilites
y_prob = stackN.best_estimator_.predict_proba(X_test)
prob = y_prob[:,1]
fpr, tpr,thresholds = roc_curve(y_true,prob, drop_intermediate=False )
plt.figure(figsize=(22,9))
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
plt.figure(figsize=(22,9))
scikitplot.metrics.plot_lift_curve(y_true,y_prob,title='Lift Curve',figsize=(22,9), title_fontsize='large',text_fontsize="large")
plt.show()
```




```
<Figure size 1584x648 with 0 Axes>
```



Area under the ROC curve

```
auc(fpr, tpr)
```

```
0.9738953602285441
```

Cost Sensitive Training

```
# Initializing models

clf1 = KNeighborsClassifier(n_neighbors=26)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
lr = LogisticRegression()
sclf = StackingClassifier(classifiers=[clf1, clf2, clf3],
                          meta_classifier=lr)

params = {'kneighborsclassifier__n_neighbors': [1, 5],
          'randomforestclassifier__n_estimators': [10, 50],
          'meta_classifier__C': [0.1, 10.0]}
precisionArraySpam,recallArraySpam,f1ArraySpam,precisionArrayNSpam,recallArrayNSpam,f1ArrayNSpam = emptyArrays()
stackN = GridSearchCV(estimator=sclf,
                    param_grid=params,
                    cv=inner_cv,
                    refit=True)

nested_score = cross_validate(stackN, X=transformedData, y=target, cv=outer_cv, scoring = cost_scorer)
print("Mean Missclassification Cost: {0:.2f}, Std Deviation:
{1:.2f}".format(nested_score['test_score'].mean(),nested_score['test_score'].std()))
```

```
Mean Missclassification Cost: -365.75, Std Deviation: 77.97
```

```
stackN.fit(X_train, y_train)
print('The best parameter values are ',stackN.best_params_,'\n')
print("Detailed classification report:")
print()
y_true, y_pred = y_test, stackN.best_estimator_.predict(X_test)
print(classification_report(y_true, y_pred))
print("Confusion Matrix")
print(confusion_matrix(y_true, y_pred))
print()
```

```
The best parameter values are  {'kneighborsclassifier__n_neighbors': 5, 'meta_classifier__C': 0.1, 'randomforestclassifier__n_estimators':
50}

Detailed classification report:

              precision    recall  f1-score   support

           0       0.96      0.96      0.96       697
           1       0.94      0.94      0.94       454

    accuracy                           0.95      1151
   macro avg       0.95      0.95      0.95      1151
weighted avg       0.95      0.95      0.95      1151

Confusion Matrix
[[670  27]
 [ 26 428]]
```

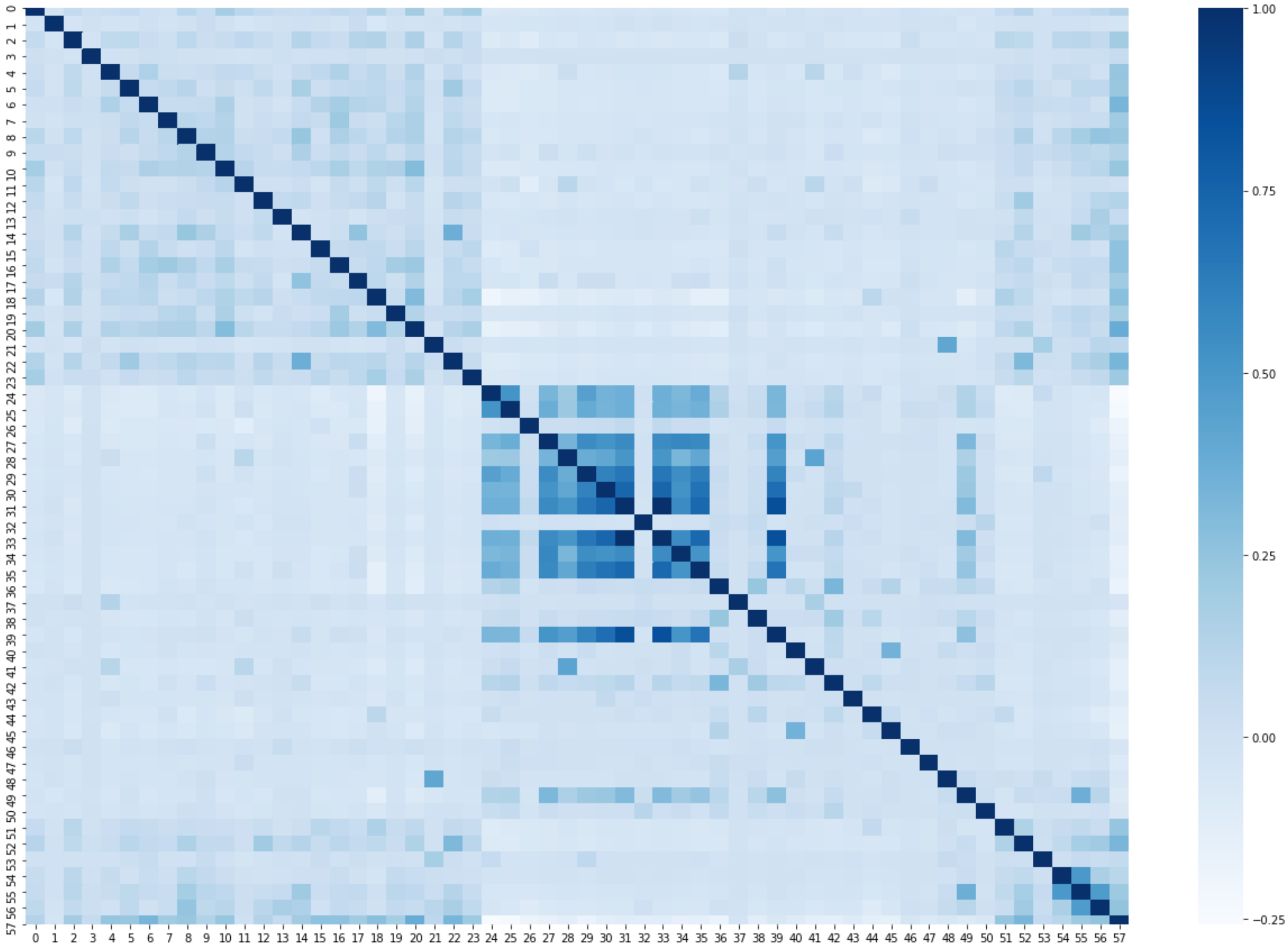
Feature Selection For this Dataset

```
xF = pd.read_csv('spambase.data', header = None, usecols = np.arange(0,58,1))
targetF = pd.read_csv('spambase.data', header = None, usecols = [57], squeeze = True)
xF.head()
```

	0	1	2	3	4	5	6	7	8	9	...	48	49	50	51	52	53	54	55
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.00	0.000	0.0	0.778	0.000	0.000	3.756	61
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.00	0.132	0.0	0.372	0.180	0.048	5.114	101
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.01	0.143	0.0	0.276	0.184	0.010	9.821	485
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.137	0.0	0.137	0.000	0.000	3.537	40
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.135	0.0	0.135	0.000	0.000	3.537	40

5 rows × 58 columns

```
import seaborn as sns
plt.figure(figsize=(22,15))
cor = xF.corr()
sns.heatmap(cor, cmap=plt.cm.Blues)
plt.show()
```



```
#Correlation with output variable
cor_target = abs(cor[57])#Selecting highly correlated features
relevant_features = cor_target[cor_target>0.30]
relevant_features
```

```
6     0.332117
20    0.383234
22    0.334787
52    0.323629
57    1.000000
Name: 57, dtype: float64
```

As part of this assignment we had been asked to do feature selection as part of te data preprocessing. I tried to see co-related variables and i planned to use only one of the correlated varibles
Looks like there is hardly any correlation between any variables.

Performance of Models on this dataset

I have explored all models on the basis of accuracy and also trained them to reduce missclassification cost.
I have used a custom cost scorer to do this.

```
perf = {"Mean Accuracy":[91,94,91,94,92,81,94,95,95]
,"Std Accuracy":[0,0,0.01,0,0,0,0.01,0.01,0.01]
,"Avg Missclassification Cost":[656,357,823.25,473,482,406.25,367,348.25,365.75]
,"Std Missclassification Cost":[72.98,34.22,62.77,50.26,38.26,53.93,42.57,32.43,77.97]}

dfperf = pd.DataFrame(perf)
dfperf.index = ['Decision Tree','Neural Networks','KNN','Random Forest','Logistic Regression','Naïve Bayes','XGBoost','AdaBoost','Stacking'
]
dfperf
```

	Mean Accuracy	Std Accuracy	Avg Missclassification Cost	Std Missclassification Cost
Decision Tree	91	0.00	656.00	72.98
Neural Networks	94	0.00	357.00	34.22
KNN	91	0.01	823.25	62.77
Random Forest	94	0.00	473.00	50.26
Logistic Regression	92	0.00	482.00	38.26
Naïve Bayes	81	0.00	406.25	53.93
XGBoost	94	0.01	367.00	42.57
AdaBoost	95	0.01	348.25	32.43
Stacking	95	0.01	365.75	77.97

Model with Best Accuracy

```
dfperf[dfperf['Mean Accuracy']== dfperf['Mean Accuracy'].max()]
```

	Mean Accuracy	Std Accuracy	Avg Missclassification Cost	Std Missclassification Cost
AdaBoost	95	0.01	348.25	32.43
Stacking	95	0.01	365.75	77.97

Model with Least Missclassifaction Cost

```
dfperf[dfperf['Avg Missclassification Cost']== dfperf['Avg Missclassification Cost'].min()]
```

	Mean Accuracy	Std Accuracy	Avg Missclassification Cost	Std Missclassification Cost
AdaBoost	95	0.01	348.25	32.43

```
Adaboosting worked the best with the best accuracy and the lowest Missclassification Cost across all models. AdaBoost can be used to improve
the performance of machine learning algorithms. It is used best with weak learners and these models achieve high accuracy above random
chance on a classification problem. I have used decision trees as the weak learner in the ADA Boost.
```