# Prediction models of Decision tree, logistic regression, k-nearest neighbors and SVM that predicts the diagnose result

# Preparation of the data for running model and analyzing the patterns of the data

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from pandas import Series, DataFrame

import pandas as pd
import numpy as np

import warnings
warnings.filterwarnings("ignore")
```

```
# the command below helps to import the dataset into the Jupyter notebook
data1 = pd.read_csv(r"C:\Users\poona\OneDrive\Desktop\Predictive\HW\wdbc.data",header= None)
```

```
# this command helps to check the dataset. If there are any null values or if the datatype is unusual
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
0     569 non-null int64
1     569 non-null object
2     569 non-null float64
3     569 non-null float64
4     569 non-null float64
5     569 non-null float64
6     569 non-null float64
7     569 non-null float64
8     569 non-null float64
9     569 non-null float64
10    569 non-null float64
11    569 non-null float64
12    569 non-null float64
13    569 non-null float64
14    569 non-null float64
15    569 non-null float64
16    569 non-null float64
17    569 non-null float64
18    569 non-null float64
19    569 non-null float64
20    569 non-null float64
21    569 non-null float64
22    569 non-null float64
23    569 non-null float64
24    569 non-null float64
25    569 non-null float64
26    569 non-null float64
27    569 non-null float64
28    569 non-null float64
29    569 non-null float64
30    569 non-null float64
31    569 non-null float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.3+ KB
```

```
# The code below helps to select only the required columns (features). Selecting the 30 columns as features.
And selecting
# the column 1 as target variable.

X = data1.iloc[:,2:31]
y = data1.iloc[:,1]
```

```
# this helps to check the classification of the target variables.
data1[1].value_counts()
```

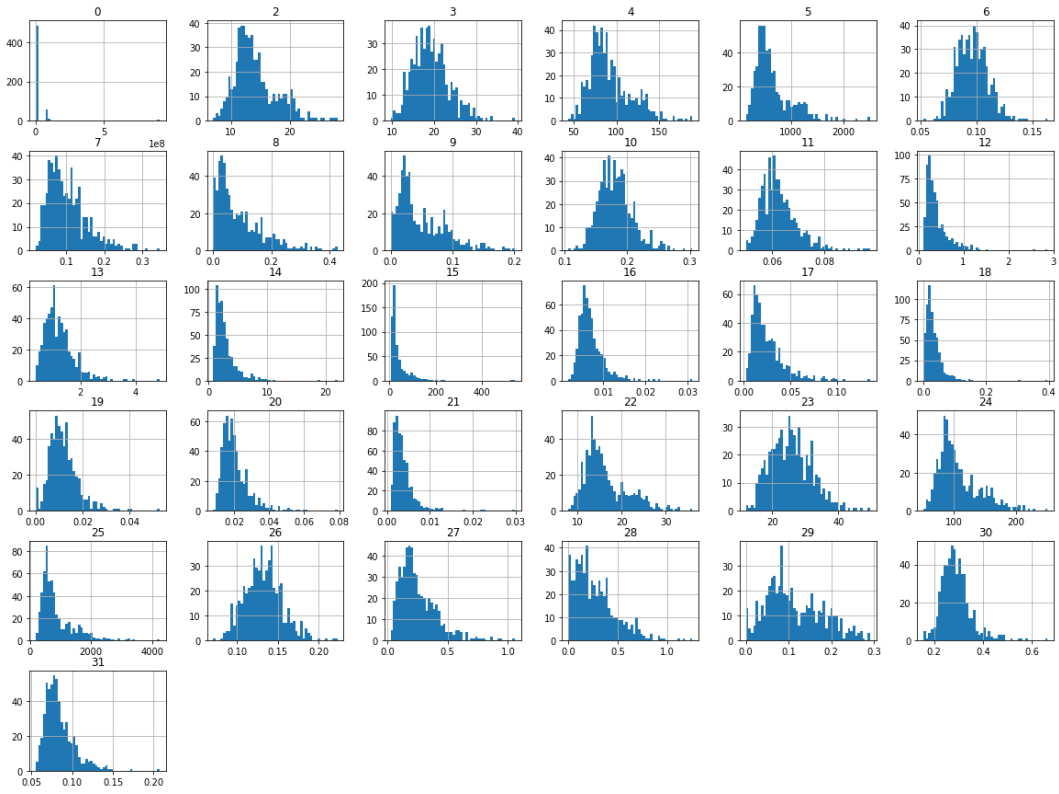
```
B    357
M    212
Name: 1, dtype: int64
```

```
# This code helps to get the idea of the entire dataset. It helps to see the mean, st dev etc.
data1.describe()
```

| | 0 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 5.690000e+02 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000 |
| **mean** | 3.037183e+07 | 14.127292 | 19.289649 | 91.969033 | 654.889104 | 0.096360 | 0.104341 | 0.088799 | 0.04891 |
| **std** | 1.250206e+08 | 3.524049 | 4.301036 | 24.298981 | 351.914129 | 0.014064 | 0.052813 | 0.079720 | 0.03880 |
| **min** | 8.670000e+03 | 6.981000 | 9.710000 | 43.790000 | 143.500000 | 0.052630 | 0.019380 | 0.000000 | 0.00000 |
| **25%** | 8.692180e+05 | 11.700000 | 16.170000 | 75.170000 | 420.300000 | 0.086370 | 0.064920 | 0.029560 | 0.02031 |
| **50%** | 9.060240e+05 | 13.370000 | 18.840000 | 86.240000 | 551.100000 | 0.095870 | 0.092630 | 0.061540 | 0.03350 |
| **75%** | 8.813129e+06 | 15.780000 | 21.800000 | 104.100000 | 782.700000 | 0.105300 | 0.130400 | 0.130700 | 0.07400 |
| **max** | 9.113205e+08 | 28.110000 | 39.280000 | 188.500000 | 2501.000000 | 0.163400 | 0.345400 | 0.426800 | 0.20120 |

8 rows × 31 columns

```
# the code below helps to see the histograms of all the integer columns present in the data. This give the
idea of
# the distribution of the data and to see if the data is normal or not.

%matplotlib inline
import matplotlib.pyplot as plt
data1.hist(bins=50, figsize=(20,15))
plt.show()
```



```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
```

```
# This conversion of different categories is done to find the true positive and true negatives.
# The Malign condition i.e. case of breast cancer is considered as 1. and the case of benign condition is
considered as 0.
y= y.replace("M",1)
y= y.replace("B", 0)
```

```
# the code below helps to split the entire data into train and test. The train data will be further used for
the
# hyperparameter selection. The test data will not be touched during the cross validation and hyperparameter
tuning.


X_train_unscaled, X_test_unscaled, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1) #
80% training and 20% test
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train_unscaled)
X_test = scaler.transform(X_test_unscaled)
```

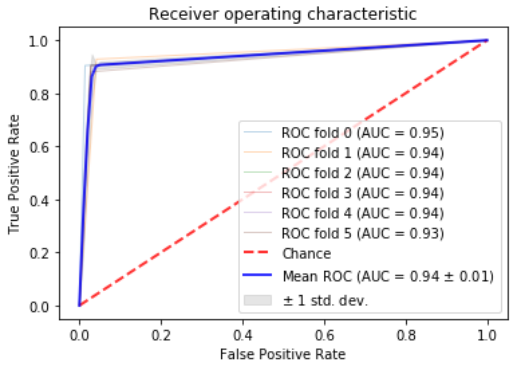
# DECISION TREE MODEL - USING NESTED CROSS VALIDATION

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV, cross_val_score, KFold
import numpy as np

print(__doc__)

# Number of random trials
NUM_TRIALS = 10


# Set up possible values of parameters to optimize over

tree_clf =DecisionTreeClassifier()

criterion_options = ["gini","entropy"]

max_depth_range = list(range(1,35))

p_grid = dict(max_depth = max_depth_range, criterion = criterion_options)


# Arrays to store scores
non_nested_scores = np.zeros(NUM_TRIALS)
nested_scores = np.zeros(NUM_TRIALS)

# Loop for each trial
for i in range(NUM_TRIALS):

    # Choose cross-validation techniques for the inner and outer loops,
    # independently of the dataset.
    # E.g "LabelKFold", "LeaveOneOut", "LeaveOneLabelOut", etc.
    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)

    # Non_nested parameter search and scoring
    clf = GridSearchCV(estimator=tree_clf, param_grid=p_grid, cv=inner_cv)
    clf.fit(X_train, y_train)
    non_nested_scores[i] = clf.best_score_

    # Nested CV with parameter optimization
    nested_score = cross_val_score(clf, X=X_train, y=y_train, cv=outer_cv)
    nested_scores[i] = nested_score.mean()

score_difference = non_nested_scores - nested_scores

print("Average difference of {0:6f} with std. dev. of {1:6f}."
      .format(score_difference.mean(), score_difference.std()))
```

```
Automatically created module for IPython interactive environment
Average difference of 0.015626 with std. dev. of 0.008555.
```

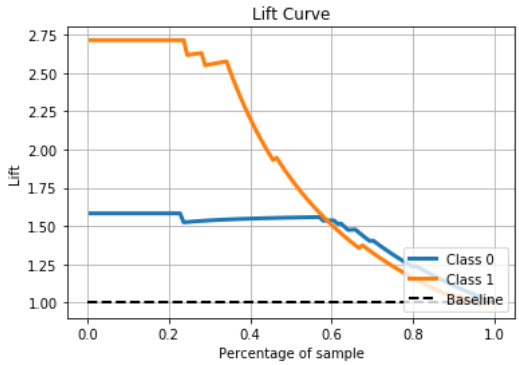
## Model Performance evaluation using ROC curve

```
# ##############################################################################
# Classification and ROC analysis
from sklearn.metrics import confusion_matrix, precision_recall_curve, average_precision_score, roc_curve, auc
from scipy import interp
%matplotlib inline
import matplotlib.pyplot as plt
# Run classifier with cross-validation and plot ROC curves
y_pred = clf.predict(X_test)
cv = StratifiedKFold(n_splits=6)


tprs = []
aucs = []

mean_fpr = np.linspace(0, 1, 100)

i = 0
for train, test in cv.split(X, y):
    probas_ = clf.fit(X_train, y_train).predict_proba(X_test)
    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(y_test, probas_[:, 1])
    tprs.append(interp(mean_fpr, fpr, tpr))
    tprs[-1][0] = 0.0
    roc_auc = auc(fpr, tpr)
    aucs.append(roc_auc)
    plt.plot(fpr, tpr, lw=1, alpha=0.3,
             label='ROC fold %d (AUC = %0.2f)' % (i, roc_auc))

    i += 1
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
         label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
plt.plot(mean_fpr, mean_tpr, color='b',
         label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),
         lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
plt.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                 label=r'$\pm$ 1 std. dev.')

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```




## Model Performance evaluation using Lift curve

```
import scikitplot as skplt
y_score = clf.best_estimator_.predict_proba(X_test)
skplt.metrics.plot_lift_curve(y_test, y_score)
plt.show()
```




```
print (clf.best_score_)
print (clf.best_params_)
print (clf.best_estimator_)
```

```
0.9582417582417583
{'criterion': 'gini', 'max_depth': 20}
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=20,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

```
#Predict the response for test dataset
y_pred = clf.predict(X_test)
ans =confusion_matrix(y_test, y_pred, labels=None, sample_weight=None)
```

```
prec_pos = ans[0,0]/(ans[0,0] +ans[0,1])
prec_neg = ans[1,1]/(ans[1,1] +ans[1,0])
rec_pos = ans[0,0]/(ans[0,0] +ans[1,0])
rec_neg = ans[1,1]/(ans[1,1] +ans[0,1])

f_score = 2*(prec_pos*rec_pos)/(prec_pos+rec_pos)
```

```
print("The confusion Matrix is \n",ans)
```

```
The confusion Matrix is
 [[70  2]
 [ 5 37]]
```

```
print("Prediction Accuracy: ",accuracy_score(y_test, clf.predict(X_test)))
print("Precision Positive: ",prec_pos)
print("Precision Negative: ",prec_neg)
print("Recall Positive: ",rec_pos)
print("Recall Negative: ",rec_neg)
print("F Score:" ,f_score)
```

```
Prediction Accuracy:  0.9385964912280702
Precision Positive:  0.9722222222222222
Precision Negative:  0.8809523809523809
Recall Positive:  0.9333333333333333
Recall Negative:  0.9487179487179487
F Score: 0.9523809523809524
```

## Result Summary for Decision Tree

```
# for Breast Cancer prediction we want the sensitivity to be the maximum. Sensitivity is Recall Positive.
i.e. we want the
# sick person who has the cancer to be diagnosed correctly. Because the chances of wrong diagnosis of a true
cancer patient
# can delay the early treatment and also increase the stage of cancer. We also want the Specificity to be
more.
# Specificity is the Recall Negative. Here  we want the percentage of healthy people who are correctly
identified
#as not having the condition to be more because if we wrongly predict them to have the disease this may lead
# to higher cost of more tests for the people and panic in.
# However the most important factor to determine the best model is to have the highest sensitivity ( Recall
Positive)
```

```
# The Recall positive is 93% for Decision Tree model. We would now compare the Recall positive of this model
with
# other models to choose the best model in this case.
```

# KNN begins here

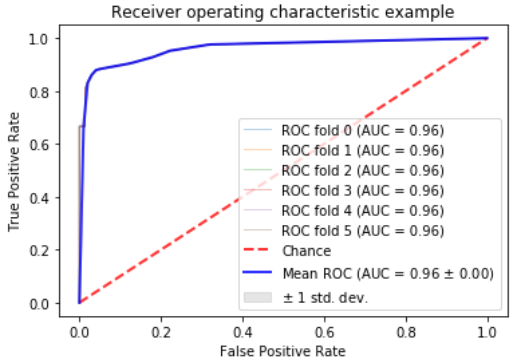
```
from sklearn import neighbors, datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score, KFold
import numpy as np
from sklearn.metrics import accuracy_score

print(__doc__)

# Number of random trials
NUM_TRIALS = 10


# Set up possible values of parameters to optimize over

knn_clf =KNeighborsClassifier()

k_range = list(range(1,31))
weight_options = ["uniform", "distance"]

p_grid = dict(n_neighbors = k_range, weights = weight_options)


# Arrays to store scores
non_nested_scores = np.zeros(NUM_TRIALS)
nested_scores = np.zeros(NUM_TRIALS)

# Loop for each trial
for i in range(NUM_TRIALS):

    # Choose cross-validation techniques for the inner and outer loops,
    # independently of the dataset.
    # E.g "LabelKFold", "LeaveOneOut", "LeaveOneLabelOut", etc.
    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)

    # Non_nested parameter search and scoring
    clf = GridSearchCV(estimator=knn_clf, param_grid=p_grid, cv=inner_cv)
    clf.fit(X_train, y_train)
    non_nested_scores[i] = clf.best_score_

    # Nested CV with parameter optimization
    nested_score = cross_val_score(clf, X=X_train, y=y_train, cv=outer_cv)
    nested_scores[i] = nested_score.mean()

score_difference = non_nested_scores - nested_scores

print("Average difference of {0:6f} with std. dev. of {1:6f}."
      .format(score_difference.mean(), score_difference.std()))
```

```
Automatically created module for IPython interactive environment
Average difference of 0.004181 with std. dev. of 0.003469.
```

## Model Performance evaluation using ROC curve

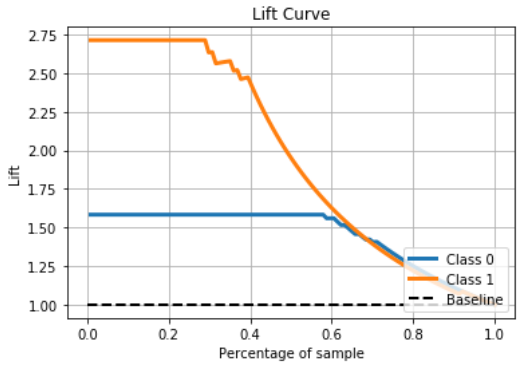
```
# ##############################################################################
# Classification and ROC analysis
from sklearn.metrics import confusion_matrix, precision_recall_curve, average_precision_score, roc_curve, auc
from scipy import interp
```

```
%matplotlib inline
import matplotlib.pyplot as plt
# Run classifier with cross-validation and plot ROC curves
y_pred = clf.predict(X_test)
cv = StratifiedKFold(n_splits=6)


tprs = []
aucs = []

mean_fpr = np.linspace(0, 1, 100)

i = 0
for train, test in cv.split(X, y):
    probas_ = clf.fit(X_train, y_train).predict_proba(X_test)
    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(y_test, probas_[:, 1])
    tprs.append(interp(mean_fpr, fpr, tpr))
    tprs[-1][0] = 0.0
    roc_auc = auc(fpr, tpr)
    aucs.append(roc_auc)
    plt.plot(fpr, tpr, lw=1, alpha=0.3,
             label='ROC fold %d (AUC = %0.2f)' % (i, roc_auc))

    i += 1
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
         label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
plt.plot(mean_fpr, mean_tpr, color='b',
         label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),
         lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
plt.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                 label=r'$\pm$ 1 std. dev.')

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```




## Model Performance evaluation using Lift curve

```
import scikitplot as skplt
y_score = clf.best_estimator_.predict_proba(X_test)
skplt.metrics.plot_lift_curve(y_test, y_score)
plt.show()
```




```
print (clf.best_score_)
print (clf.best_params_)
print (clf.best_estimator_)
```

```
0.9736263736263736
{'n_neighbors': 3, 'weights': 'uniform'}
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

```
#Predict the response for test dataset
y_pred = clf.predict(X_test)
ans =confusion_matrix(y_test, y_pred, labels=None, sample_weight=None)
```

```
prec_pos = ans[0,0]/(ans[0,0] +ans[0,1])
prec_neg = ans[1,1]/(ans[1,1] +ans[1,0])
rec_pos = ans[0,0]/(ans[0,0] +ans[1,0])
rec_neg = ans[1,1]/(ans[1,1] +ans[0,1])

f_score = 2*(prec_pos*rec_pos)/(prec_pos+rec_pos)
```

```
print("The confusion Matrix is \n",ans)
```

```
The confusion Matrix is
 [[69  3]
 [ 4 38]]
```

```
print("Prediction Accuracy: ",accuracy_score(y_test, clf.predict(X_test)))
print("Precision Positive: ",prec_pos)
print("Precision Negative: ",prec_neg)
print("Recall Positive: ",rec_pos)
print("Recall Negative: ",rec_neg)
print("F Score:" ,f_score)
```

```
Prediction Accuracy:  0.9385964912280702
Precision Positive:  0.9583333333333334
Precision Negative:  0.9047619047619048
Recall Positive:  0.9452054794520548
Recall Negative:  0.926829268292683
F Score: 0.9517241379310345
```

## Result Summary for KNN Model

```
# for Breast Cancer prediction we want the sensitivity to be the maximum. Sensitivity is Recall Positive.
i.e. we want the
# sick person who has the cancer to be diagnosed correctly. Because the chances of wrong diagnosis of a true
cancer patient
# can delay the early treatment and also increase the stage of cancer. We also want the Specificity to be
more.
# Specificity is the Recall Negative. Here  we want the percentage of healthy people who are correctly
identified
#as not having the condition to be more because if we wrongly predict them to have the disease this may lead
# to higher cost of more tests for the people and panic in.
# However the most important factor to determine the best model is to have the highest sensitivity ( Recall
Positive)
```

```
# The Recall positive is 95.8% for KNN model. This is more than the Recall positive of Decision Tree Model.
# Thus we would prefer KNN model over Decision Tree model.
```

# Logistic Regression begins here

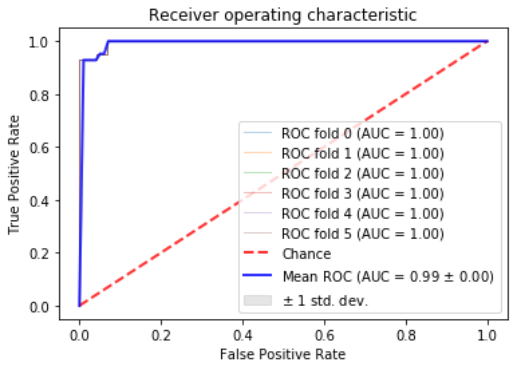
```
from sklearn import neighbors, datasets
from sklearn import linear_model
from sklearn.model_selection import GridSearchCV, cross_val_score, KFold
import numpy as np
from sklearn.metrics import accuracy_score

print(__doc__)

# Number of random trials
NUM_TRIALS = 10


# Set up possible values of parameters to optimize over


logit_clf = linear_model.LogisticRegression()

penalty_options =["l1" or "l2"]
C_lambda = [0.001, 0.01, 0.1, 1, 10]



p_grid = dict(C = C_lambda, penalty = penalty_options)


# Arrays to store scores
non_nested_scores = np.zeros(NUM_TRIALS)
nested_scores = np.zeros(NUM_TRIALS)

# Loop for each trial
for i in range(NUM_TRIALS):

    # Choose cross-validation techniques for the inner and outer loops,
    # independently of the dataset.
    # E.g "LabelKFold", "LeaveOneOut", "LeaveOneLabelOut", etc.
    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)

    # Non_nested parameter search and scoring
    clf = GridSearchCV(estimator=logit_clf, param_grid=p_grid, cv=inner_cv)
    clf.fit(X_train, y_train)
    non_nested_scores[i] = clf.best_score_

    # Nested CV with parameter optimization
    nested_score = cross_val_score(clf, X=X_train, y=y_train, cv=outer_cv)
    nested_scores[i] = nested_score.mean()

score_difference = non_nested_scores - nested_scores

print("Average difference of {0:6f} with std. dev. of {1:6f}."
      .format(score_difference.mean(), score_difference.std()))
```

```
Automatically created module for IPython interactive environment
Average difference of 0.002892 with std. dev. of 0.003284.
```

## Model Performance evaluation using ROC curve

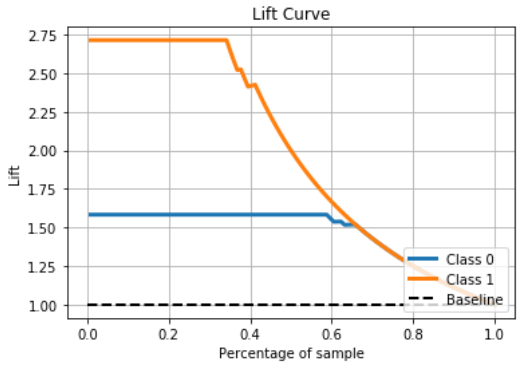
```
# ##############################################################################
# Classification and ROC analysis
from sklearn.metrics import confusion_matrix, precision_recall_curve, average_precision_score, roc_curve, auc
from scipy import interp
```

```
%matplotlib inline
import matplotlib.pyplot as plt
# Run classifier with cross-validation and plot ROC curves
y_pred = clf.predict(X_test)
cv = StratifiedKFold(n_splits=6)


tprs = []
aucs = []

mean_fpr = np.linspace(0, 1, 100)

i = 0
for train, test in cv.split(X, y):
    probas_ = clf.fit(X_train, y_train).predict_proba(X_test)
    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(y_test, probas_[:, 1])
    tprs.append(interp(mean_fpr, fpr, tpr))
    tprs[-1][0] = 0.0
    roc_auc = auc(fpr, tpr)
    aucs.append(roc_auc)
    plt.plot(fpr, tpr, lw=1, alpha=0.3,
             label='ROC fold %d (AUC = %0.2f)' % (i, roc_auc))

    i += 1
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
         label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
plt.plot(mean_fpr, mean_tpr, color='b',
         label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),
         lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
plt.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                 label=r'$\pm$ 1 std. dev.')

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```




## Model Performance evaluation using Lift curve

```
import scikitplot as skplt
y_score = clf.best_estimator_.predict_proba(X_test)
skplt.metrics.plot_lift_curve(y_test, y_score)
plt.show()
```




```
print (clf.best_score_)
print (clf.best_params_)
print (clf.best_estimator_)
```

```
0.9626373626373627
{'C': 10, 'penalty': 'l1'}
LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l1',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
#Predict the response for test dataset
y_pred = clf.predict(X_test)
ans =confusion_matrix(y_test, y_pred, labels=None, sample_weight=None)
```

```
prec_pos = ans[0,0]/(ans[0,0] +ans[0,1])
prec_neg = ans[1,1]/(ans[1,1] +ans[1,0])
rec_pos = ans[0,0]/(ans[0,0] +ans[1,0])
rec_neg = ans[1,1]/(ans[1,1] +ans[0,1])

f_score = 2*(prec_pos*rec_pos)/(prec_pos+rec_pos)
```

```
print("The confusion Matrix is \n",ans)
```

```
The confusion Matrix is
 [[72  0]
 [ 3 39]]
```

```
print("Prediction Accuracy: ",accuracy_score(y_test, clf.predict(X_test)))
print("Precision Positive: ",prec_pos)
print("Precision Negative: ",prec_neg)
print("Recall Positive: ",rec_pos)
print("Recall Negative: ",rec_neg)
print("F Score:" ,f_score)
```

```
Prediction Accuracy:  0.9736842105263158
Precision Positive:  1.0
Precision Negative:  0.9285714285714286
Recall Positive:  0.96
Recall Negative:  1.0
F Score: 0.9795918367346939
```

```
# The Recall positive is 96% for Logistic Regression model. This is more than the Recall positive of Decision Tree Model and KNN.
# Thus we would prefer Logistic Regression model over Decision Tree and KNN model.
```

# SVM Begins here

```
from __future__ import print_function
```

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.svm import SVC



# Set the parameters by cross-validation
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000],'probability':[True]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000],'probability':[True]}]

scores = ['precision', 'recall']



for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print()

    clf = GridSearchCV(SVC(), tuned_parameters, cv=5,
                       scoring='%s_macro' % score)
    clf.fit(X_train, y_train)

    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
    print()
    print("Grid scores on development set:")
    print()
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params))
    print()

    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print()
    y_true, y_pred = y_test, clf.predict(X_test)
    print(classification_report(y_true, y_pred))
    print()

# Note the problem is too easy: the hyperparameter plateau is too flat and the
# output model is the same for precision and recall with ties in quality.
```

```
# Tuning hyper-parameters for precision

Best parameters set found on development set:

{'C': 1000, 'gamma': 0.001, 'kernel': 'rbf', 'probability': True}

Grid scores on development set:

0.313 (+/-0.000) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf', 'probability': True}
0.313 (+/-0.000) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf', 'probability': True}
0.938 (+/-0.017) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf', 'probability': True}
0.313 (+/-0.000) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf', 'probability': True}
0.961 (+/-0.036) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf', 'probability': True}
0.938 (+/-0.017) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf', 'probability': True}
0.976 (+/-0.031) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf', 'probability': True}
0.961 (+/-0.036) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf', 'probability': True}
0.974 (+/-0.032) for {'C': 1, 'kernel': 'linear', 'probability': True}
0.975 (+/-0.033) for {'C': 10, 'kernel': 'linear', 'probability': True}
0.960 (+/-0.037) for {'C': 100, 'kernel': 'linear', 'probability': True}
0.954 (+/-0.054) for {'C': 1000, 'kernel': 'linear', 'probability': True}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       0.96      1.00      0.98        72
           1       1.00      0.93      0.96        42

    accuracy                           0.97       114
   macro avg       0.98      0.96      0.97       114
weighted avg       0.97      0.97      0.97       114
```

# Tuning hyper-parameters for recall

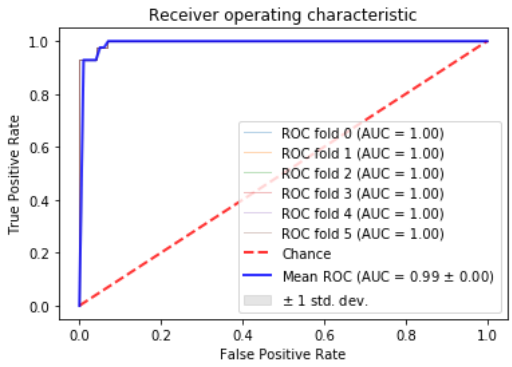
Best parameters set found on development set:

```
{'C': 10, 'kernel': 'linear', 'probability': True}

Grid scores on development set:

0.500 (+/-0.000) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf', 'probability': True}
0.500 (+/-0.000) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf', 'probability': True}
0.886 (+/-0.029) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf', 'probability': True}
0.500 (+/-0.000) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf', 'probability': True}
0.938 (+/-0.036) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf', 'probability': True}
0.886 (+/-0.029) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf', 'probability': True}
0.968 (+/-0.034) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf', 'probability': True}
0.938 (+/-0.036) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf', 'probability': True}
0.961 (+/-0.044) for {'C': 1, 'kernel': 'linear', 'probability': True}
0.969 (+/-0.033) for {'C': 10, 'kernel': 'linear', 'probability': True}
0.957 (+/-0.036) for {'C': 100, 'kernel': 'linear', 'probability': True}
0.953 (+/-0.060) for {'C': 1000, 'kernel': 'linear', 'probability': True}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       0.96      1.00      0.98        72
           1       1.00      0.93      0.96        42

    accuracy                           0.97       114
   macro avg       0.98      0.96      0.97       114
weighted avg       0.97      0.97      0.97       114
```

## Model Performance evaluation using ROC curve

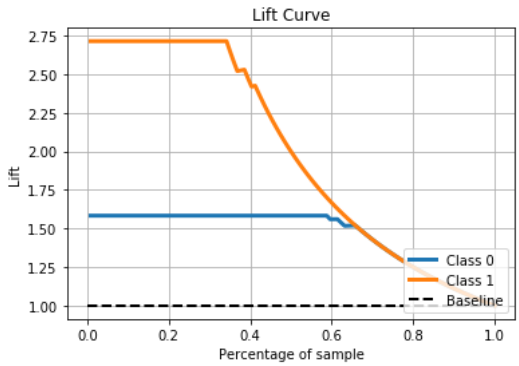
```
# ##############################################################################
# Classification and ROC analysis
from sklearn.metrics import confusion_matrix, precision_recall_curve, average_precision_score, roc_curve, auc
from scipy import interp
%matplotlib inline
import matplotlib.pyplot as plt
# Run classifier with cross-validation and plot ROC curves
y_pred = clf.predict(X_test)
cv = StratifiedKFold(n_splits=6)


tprs = []
aucs = []

mean_fpr = np.linspace(0, 1, 100)

i = 0
for train, test in cv.split(X, y):
    probas_ = clf.fit(X_train, y_train).predict_proba(X_test)
    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(y_test, probas_[:, 1])
    tprs.append(interp(mean_fpr, fpr, tpr))
    tprs[-1][0] = 0.0
    roc_auc = auc(fpr, tpr)
    aucs.append(roc_auc)
    plt.plot(fpr, tpr, lw=1, alpha=0.3,
             label='ROC fold %d (AUC = %0.2f)' % (i, roc_auc))

    i += 1
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
         label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
plt.plot(mean_fpr, mean_tpr, color='b',
         label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),
         lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
plt.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
```

```
                 label=r'$\pm$ 1 std. dev.')

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```




## Model Performance evaluation using Lift curve

```
import scikitplot as skplt
y_score = clf.best_estimator_.predict_proba(X_test)
skplt.metrics.plot_lift_curve(y_test, y_score)
plt.show()
```




```
#Predict the response for test dataset
y_pred = clf.predict(X_test)
ans =confusion_matrix(y_test, y_pred, labels=None, sample_weight=None)
```

```
from sklearn import neighbors, datasets
from sklearn import linear_model
from sklearn.model_selection import GridSearchCV, cross_val_score, KFold
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

```
print (clf.best_score_)
print (clf.best_params_)
print (clf.best_estimator_)
```

```
0.969453044375645
{'C': 10, 'kernel': 'linear', 'probability': True}
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=True, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

```
prec_pos = ans[0,0]/(ans[0,0] +ans[0,1])
prec_neg = ans[1,1]/(ans[1,1] +ans[1,0])
rec_pos = ans[0,0]/(ans[0,0] +ans[1,0])
rec_neg = ans[1,1]/(ans[1,1] +ans[0,1])

f_score = 2*(prec_pos*rec_pos)/(prec_pos+rec_pos)
```

```
print("The confusion Matrix is \n",ans)
```

```
The confusion Matrix is
 [[72  0]
 [ 3 39]]
```

```
print("Prediction Accuracy: ",accuracy_score(y_test, clf.predict(X_test)))
print("Precision Positive: ",prec_pos)
print("Precision Negative: ",prec_neg)
print("Recall Positive: ",rec_pos)
print("Recall Negative: ",rec_neg)
print("F Score:" ,f_score)
```

```
Prediction Accuracy:  0.9736842105263158
Precision Positive:  1.0
Precision Negative:  0.9285714285714286
Recall Positive:  0.96
Recall Negative:  1.0
F Score: 0.9795918367346939
```

## RESULT SUMMARY for SVM

```
# for Breast Cancer prediction we want the sensitivity to be the maximum. Sensitivity is Recall Positive.
i.e. we want the
# sick person who has the cancer to be diagnosed correctly. Because the chances of wrong diagnosis of a true
cancer patient
# can delay the early treatment and also increase the stage of cancer. We also want the Specificity to be
more.
# Specificity is the Recall Negative. Here  we want the percentage of healthy people who are correctly
identified
#as not having the condition to be more because if we wrongly predict them to have the disease this may lead
# to higher cost of more tests for the people and panic in.
# However the most important factor to determine the best model is to have the highest sensitivity ( Recall
Positive)
```

```
# The Recall positive is 93% for Decision Tree model. This is less than the Recall positive of KNN Model
(with 94.5% recall positive). Logistic Regression has 96% recall
# positive. Also important thing to note here is that the Recall negative is 100% for this model. Finally we
see that the SVM also has 96% recall positive along with 100% of recall negative. To choose the best model we
would now look into the ROC curves and compare the AUC value.

# ROC - AUC curve measures the performance of the model at various thresholds settings. ROC is a probability
curve and AUC represents degree or measure of separability. They are plotted between the True Positive Rate
and False Positive Rate. The curve and area measures the performance of the model in distinguishing between
classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. Area of 1 means all 0s and
1s have been predicted correctly by the model.

# The mean AUC for decision tree is 0.94. For KNN, mean AUC is 0.96, for logistic its 0.99 and  for SVM too
the mean AUC is 0.99.
# for a better model the AUC value should be maximum (closer to 1)
# Thus our preferred model would be SVM or logisitc regression as per the best paramter found using grid
search for each model.
# finally we will look at the loft curves to compare all the models.
# for Class 0 the lift is better for Decision Tree compared to KNN. But for Class 1 lift is much better in
KNN compared to decision tree. Thus KNN is better using lift.
# Now, comparing the Lift of KNN with Logistic we see that, KNN has higher lift for most of the prportion of
data for both Class 0 and Class 1.
# Now when we look at the lift of SVM we find that this model has got the best lift amongst all models for
both Class 0 and Class 1. Thus finally we conclude that SVM is the best model

# Note - To choose a better model using lift we must consider a higher value of lift for most of the
proportion of the data for all the possible classes. Higher the lift better the model.

# As the criteria of sensitivity and specificity both are satisfied here. The mean AUC value is highest for
both Logisitc and SVM and the lift is highest for SVM, thus we finally choose SVM as our best model. Also
note if need be, Logistic Regression can also be considered for this analysis because of its satisfaction of
consition of sensitivity and specificity along with higher AUC value.

## If the highest priority is to diagnose sick patients correctly then SVM and logistic regression are
better. Also if the priority is to diagnose sick patient correctly as well  as to find the fit patients
correctly even for this SVM and logistic regression are good.
```

```
#Under fitting
# The under fitting occurs when the accuracy of the model is too bad in the training data itself. This shows
that model is
# not explain the variation in the training data itself.

## Over fitting
# Overfitting occurs when the accuracy of the test data is poor compared to the training data. Because the
model is able to
# generalise the variation for the training data only. And it shows poor results for test data.

# The models made above have a higher accuracy, precision, recall and F-Scores these all results show that
# the models are able to perform well. However for this specific situation the highest Recall Positive is
prefered
# (along with higher Recall Negative).
```

```
# The models are run for different hyperparameter setting in each case. The cross validation is also done on
all
# the three models. Thus cross validation and hyper parameter tuning is done to get the model.

# the final fitting is done on the 20% of the test data. and the performance is measured across different
model
# to decide the best model

# finally the models are compared using the ROC/AUC curve and the lift curve. To analysse the model
performance
```