EMORY UNIVERSITY

Introduction to Business Data Analytics

Homework #3

Adil Ashish Kumar

(put your name above (incl. any nicknames) 5pt)

Total grade: ________ out of ___150___ points

1) **(20 points) Imagine that you work for an online advertising company that has just been hired to advertise a new local restaurant online. Let's say that it costs $0.015 to present a coupon ad to online consumers. If a consumer cashes in your coupon, you stand to earn $5.**

a) Given this information, what would your cost/benefit matrix be? Explain your reasoning briefly.

[2.5 points for each correct cell in cost/benefit matrix]

Let o1 be event that customer responds

Let o2 be event customer does not respond

V(o1) = 5-0.015 = 4.985$

V(o2) = 0-0.015 = -0.015$

Cost/Benefit matrix would be as below:

	Cashes coupon(p)	Don't cash coupon (n)
Present coupon ad(Y)	$4.985	-$0.015
Don't present coupon ad (N)	0	0

A- coupon presented and customer cashes it
B- coupon presented but customer does not cash it
C- coupon not presented but customer would have cashed it
D- coupon not presented and customer would not cash it

b) Suppose that I build a classifier that provides the following confusion matrix. What is the expected value of that classifier? Justify your answer.

[3 points for expected value formula
 7 points for correct application of expected value formula]

	Positive	Negative
Positive	560	70
Negative	120	450

Expected value = p(Y,p)*b(Y,p) + p(N,p)*b(N,p) + p(N,n)*b(N,n) + p(Y,n)*b(Y,n)

Converting above matrix into expected rates by dividing all the values by the total number of predictions (560+120+70+450), we get the below matrix of probabilities

	Positive	Negative
Positive	560/1200	70/1200
Negative	120/1200	450/1200

	Positive	Negative
Positive	0.47	0.0583
Negative	0.1	0.375

Multiplying the cost benefit matrix and above probability matrix we get the following below:

	Positive	Negative
Positive	0.47 * 4.985	0.0583 *-0.015
Negative	0.1 *0	0.375*0

	Positive	Negative
Positive	2.3263	-0.000875
Negative	0	0

Summing up the above values, we get the expected value as = 2.3263-0.000875 = $2.3245

2) (25 points) You have a fraud detection task (predicting whether a given credit card transaction is "fraud" vs. "non-fraud") and you built a classification model for this purpose. For any credit card transaction, your model estimates the probability that this transaction is "fraud". The following table represents the probabilities that your model estimated for the validation dataset containing 10 records.

Actual Class (from validation data)	**Estimated Probability of Record Belonging to Class "fraud"**	**0.1**	**0.3**	**0.5**	**0.7**	**0.8**	**0.92**
fraud	0.95	F	F	F	F	F	F
fraud	0.91	F	F	F	F	F	N
fraud	0.75	F	F	F	F	N	N
non-fraud	0.67	F	F	F	N	N	N
fraud	0.61	F	F	F	N	N	N
non-fraud	0.46	F	F	N	N	N	N
fraud	0.42	F	F	N	N	N	N
non-fraud	0.25	F	N	N	N	N	N
non-fraud	0.09	N	N	N	N	N	N
non-fraud	0.04	N	N	N	N	N	N

Draw an ROC curve for your model (use at least six different thresholds to draw the ROC curve).

[10 points showing your calculations for each threshold ->i.e., the corresponding metrics you need for ROC curve

5 points for correctly labeling axes in ROC graph
10 points for correctly depicting the ROC graph – you can do this by hand or some using software such as Excel or Python]

In the above table, I have used 6 different thresholds and calculated respective predictions using those thresholds. Based on this, I have calculated TP and FP rate for each threshold value below.

TP rate = TP / (TP+FN)
FP rate = FP / (FP+TN)

For threshold = 0.1

Predicted/Actual	Positive	Negative
Positive	5	3
Negative	0	2

TP rate = 5/(5+0) = 1

FP rate = 3/(3+2) =0.6

For threshold = 0.3

Predicted/Actual	Positive	Negative
Positive	5	2
Negative	0	3

TP rate= 5/(5+0) =1

FP rate = 2/(2+5) = 0.4

For threshold = 0.5

Predicted/Actual	Positive	Negative
Positive	4	1
Negative	1	4

TP rate = 4/(4+1)= 0.8

FP rate = 1/(1+4)=0.2

For threshold = 0.7

Predicted/Actual	Positive	Negative
Positive	3	0
Negative	2	5

TP rate= 3/(3+2)= 0.6

FP rate = 0/(0+5) =0

For threshold = 0.8

Predicted/Actual	Positive	Negative
Positive	2	0
Negative	3	5

TP rate = 2/(2+3) = 0.4

FP rate =0/(0+5) =0

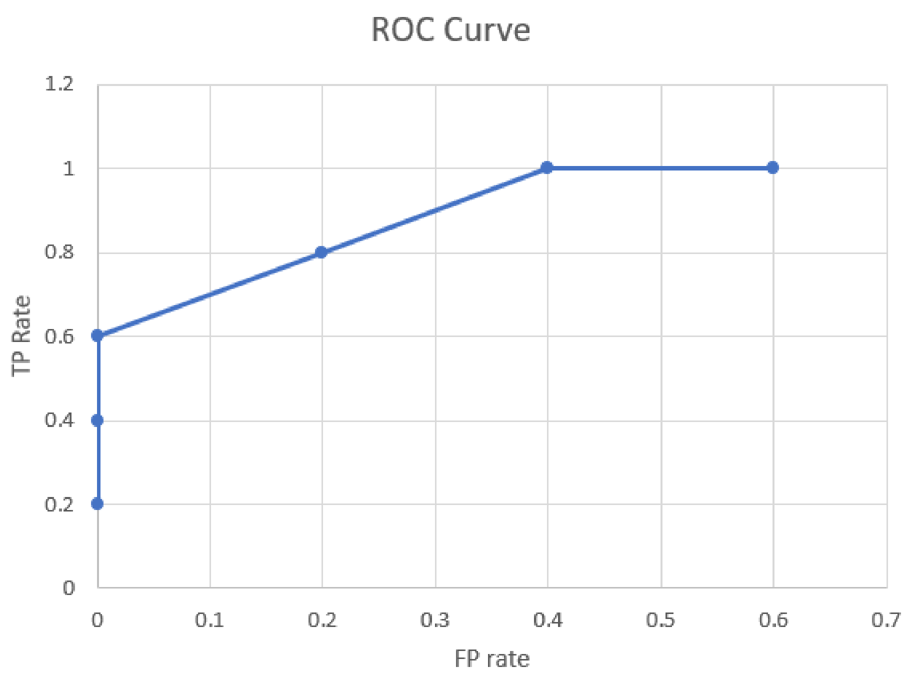
For threshold = 0.92

Predicted/Actual	Positive	Negative
Positive	1	0
Negative	4	5

TP rate = 1/(1+4) = 0.2,

FP rate = 0/(0+5) =0

Using the above TP and FP rates, I have plotted the ROC curve as below:



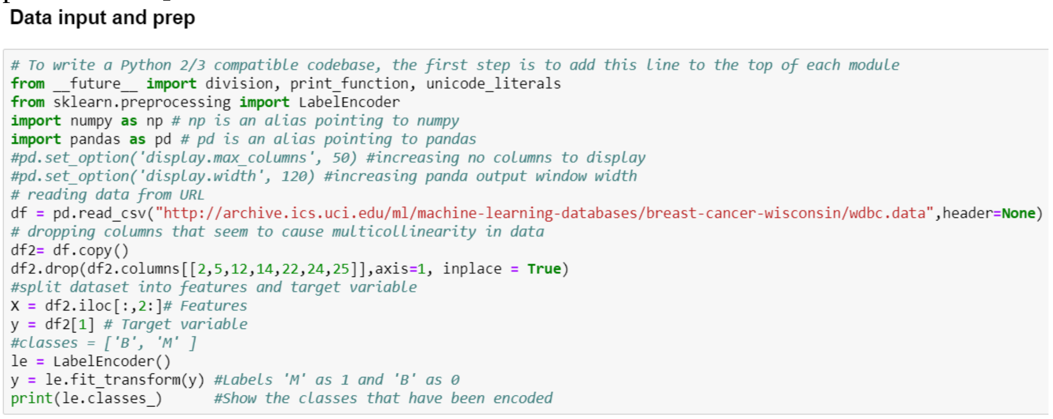
3) (100 points) [Mining publicly available data] Use Python for this Exercise.

Please use the dataset on breast cancer research from this link: http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data We have worked with this dataset in HW2. The description of the data and attributes can be found at this link: http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.names . Each record of the data set represents a different case of breast cancer. Each case is described with 30 real-valued attributes: attribute 1 represents case id, attributes 3-32 represent various physiological characteristics, and attribute 2 represents the type (benign or malignant). If the dataset has records with missing values, you can filter out these records using Python. Alternatively, if the data set has missing values, you could infer the missing values.

[We have seen this data before – No need to explore the data for this exercise]

a) **We would like to perform a predictive modeling analysis on this same dataset using the a) decision tree, b) the k-NN technique and c) the logistic regression technique. Using the nested cross-validation technique, try to optimize the parameters of your classifiers in order to improve the performance of your classifiers (i.e., f1-score) as much as possible. Please make sure to always use a <u>random state of "42"</u> whenever applicable. What are your optimal parameters and what is the corresponding performance of these classifiers? Please provide screenshots of your code and explain the process you have followed.**

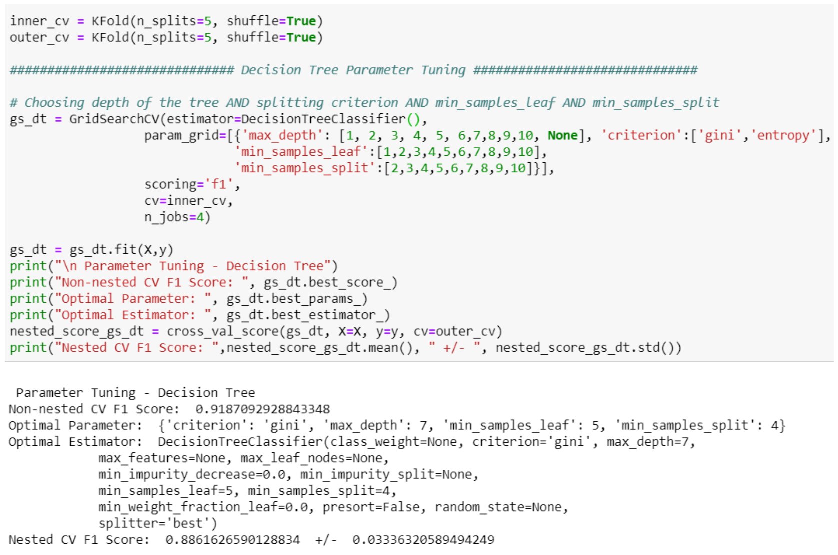
   [part a is worth 25 points in total:
   7 points for correctly optimizing at least two parameters for the Decision Tree and providing screenshots/explaining what you are doing and the corresponding results
   7 points for correctly optimizing at least two parameters for the kNN and providing screenshots/explaining what you are doing and the corresponding results
   7 points for correctly optimizing at least two parameters for the Logistic Regression and providing screenshots/explaining what you are doing and the corresponding results
   4 points for contrasting their performance of all three algorithms and discussing which one would you prefer to use]



Data input and prep

```python
# To write a Python 2/3 compatible codebase, the first step is to add this line to the top of each module
from __future__ import division, print_function, unicode_literals
from sklearn.preprocessing import LabelEncoder
import numpy as np # np is an alias pointing to numpy
import pandas as pd # pd is an alias pointing to pandas
#pd.set_option('display.max_columns', 50) #increasing no columns to display
#pd.set_option('display.width', 120) #increasing panda output window width
# reading data from URL
df = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data",header=None)
# dropping columns that seem to cause multicollinearity in data
df2= df.copy()
df2.drop(df2.columns[[2,5,12,14,22,24,25]],axis=1, inplace = True)
#split dataset into features and target variable
X = df2.iloc[:,2:]# Features
y = df2[1] # Target variable
#classes = ['B', 'M' ]
le = LabelEncoder()
y = le.fit_transform(y) #Labels 'M' as 1 and 'B' as 0
print(le.classes_)      #Show the classes that have been encoded
```

First, I read in the dataset directly from the URL provided. I then drop certain columns that were causing multicollinearity in the dataset. I had identified this earlier in the data exploration phase. I then split the data into X (model features) and y (target variable) for modeling purposes. I have then used a labelencoder() to encode the Malignant class as 1 and Benign class as 0.

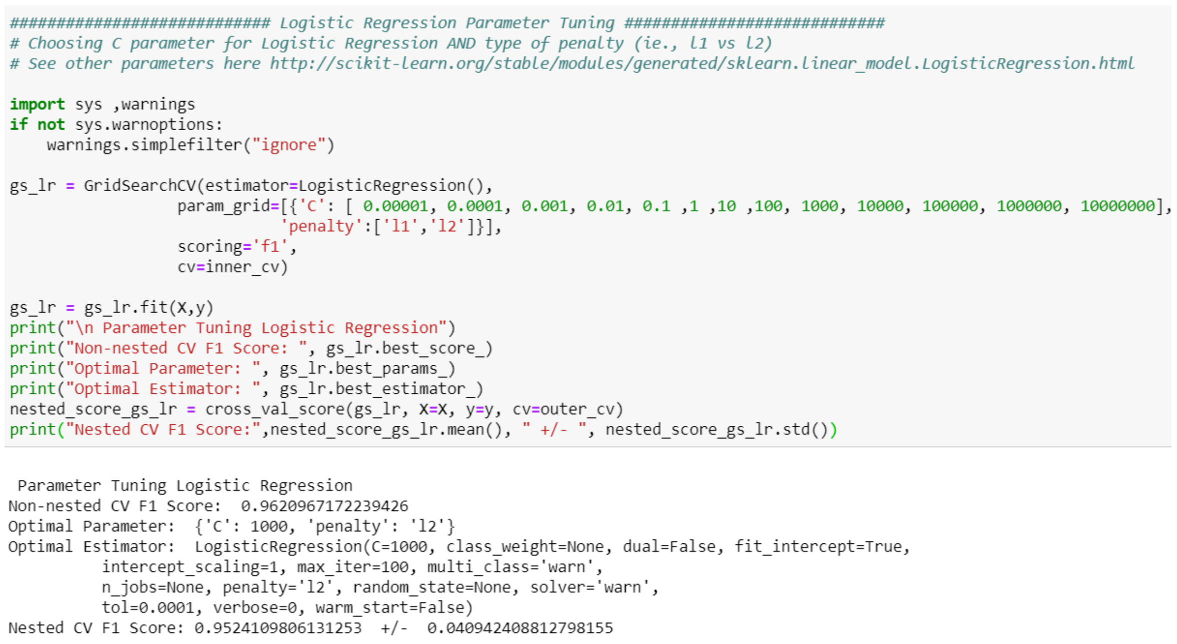


```
inner_cv = KFold(n_splits=5, shuffle=True)
outer_cv = KFold(n_splits=5, shuffle=True)

############################## Decision Tree Parameter Tuning ##############################

# Choosing depth of the tree AND splitting criterion AND min_samples_leaf AND min_samples_split
gs_dt = GridSearchCV(estimator=DecisionTreeClassifier(),
                 param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6,7,8,9,10, None], 'criterion':['gini','entropy'],
                             'min_samples_leaf':[1,2,3,4,5,6,7,8,9,10],
                             'min_samples_split':[2,3,4,5,6,7,8,9,10]}],
                 scoring='f1',
                 cv=inner_cv,
                 n_jobs=4)

gs_dt = gs_dt.fit(X,y)
print("\n Parameter Tuning - Decision Tree")
print("Non-nested CV F1 Score: ", gs_dt.best_score_)
print("Optimal Parameter: ", gs_dt.best_params_)
print("Optimal Estimator: ", gs_dt.best_estimator_)
nested_score_gs_dt = cross_val_score(gs_dt, X=X, y=y, cv=outer_cv)
print("Nested CV F1 Score: ",nested_score_gs_dt.mean(), " +/- ", nested_score_gs_dt.std())
```

```
 Parameter Tuning - Decision Tree
Non-nested CV F1 Score:  0.9187092928843348
Optimal Parameter:  {'criterion': 'gini', 'max_depth': 7, 'min_samples_leaf': 5, 'min_samples_split': 4}
Optimal Estimator:  DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=7,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=4,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
Nested CV F1 Score:  0.8861626590128834  +/-  0.03336320589494249
```

In order to optimize the model hyperparameters and check for generalization performance, I have used the nested cross validation technique. I have defined inner_cv and outer_cv as two K fold cross validation methods with k=5 for both. I use inner_cv in order to find the optimum hyperparameters for each model. Once I find the optimum model hyperparameters, I use outer_cv to calculate the mean F1 score of the optimized models.

In order to optimize the hyperparameters for the decision tree model, I have used the grid search method to find the optimal parameters and used F1 scoring as the criterion for optimization. In the parameters to optimize I have mentioned maximum depth of tree with range of values= (None,1,2,3,4,5,6,7,8,9,10) , splitting criterion with range of values= (gini, entropy) , minimum no of samples to split an internal node with range of values= (1,2,3,4,5,6,7,8,9,10) and minimum no of samples for a leaf node with range of values= (1,2,3,4,5,6,7,8,9,10). All other model parameters for the decision tree model are left unspecified, so the model will consider default values. Nested cross validation method is used here. This means that grid search will first use inner_cv to find the optimum parameters based on f1 score. The highest f1 score obtained by grid search method is 0.919 for the following criterion: splitting criteria= gini, max_depth = 7, min_samples_leaf= 5, min_samples_split= 4. Using this criterion, I checked the mean f1 score on outer_cv to get the average model performance. The mean f1 score on the outer_cv is 0.886 with a standard deviation of 0.033.

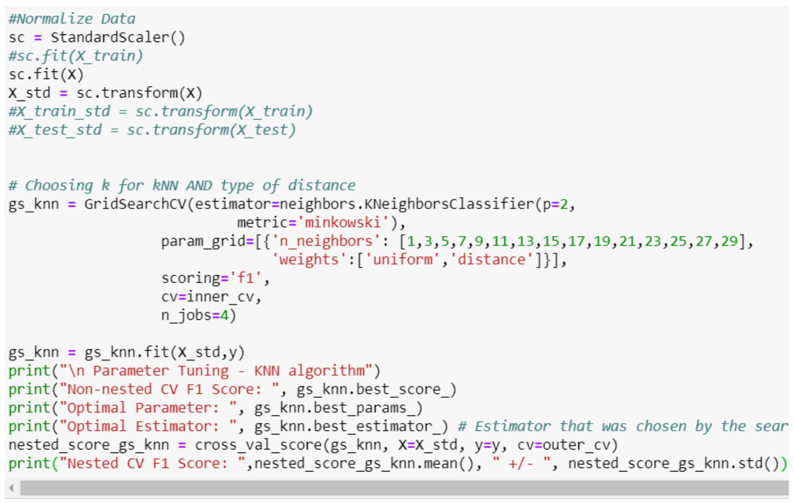


```
############################# Logistic Regression Parameter Tuning ############################
# Choosing C parameter for Logistic Regression AND type of penalty (ie., l1 vs l2)
# See other parameters here http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

import sys ,warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")

gs_lr = GridSearchCV(estimator=LogisticRegression(),
                 param_grid=[{'C': [ 0.00001, 0.0001, 0.001, 0.01, 0.1 ,1 ,10 ,100, 1000, 10000, 100000, 1000000, 10000000],
                             'penalty':['l1','l2']}],
                 scoring='f1',
                 cv=inner_cv)

gs_lr = gs_lr.fit(X,y)
print("\n Parameter Tuning Logistic Regression")
print("Non-nested CV F1 Score: ", gs_lr.best_score_)
print("Optimal Parameter: ", gs_lr.best_params_)
print("Optimal Estimator: ", gs_lr.best_estimator_)
nested_score_gs_lr = cross_val_score(gs_lr, X=X, y=y, cv=outer_cv)
print("Nested CV F1 Score:",nested_score_gs_lr.mean(), " +/- ", nested_score_gs_lr.std())
```

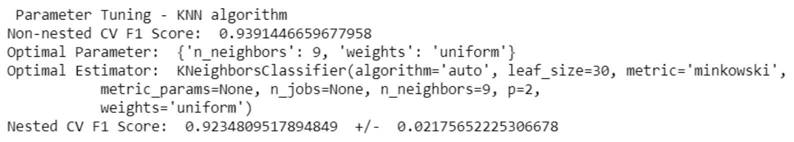
```
 Parameter Tuning Logistic Regression
Non-nested CV F1 Score:  0.9620967172239426
Optimal Parameter:  {'C': 1000, 'penalty': 'l2'}
Optimal Estimator:  LogisticRegression(C=1000, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
Nested CV F1 Score: 0.9524109806131253  +/-  0.040942408812798155
```

To optimize the hyperparameters for the logistic regression model, I have used the grid search method to find the optimal parameters and used F1 scoring as the criterion for optimization. In the parameters to optimize I have mentioned C - the inverse of regularization strength with range of values= (0.00001, 0.0001, 0.001, 0.01, 0.1 ,1 ,10 ,100, 1000, 10000, 100000, 1000000, 10000000) and penalty norm with range of values= (l1, l2). All other model parameters for the logistic regression model are left unspecified, so the model will consider default values. Nested cross validation method is used here. This means that grid search will first use inner_cv to find the optimum parameters based on f1 score. The highest f1 score obtained by grid search method is 0.962 for the following criterion: C= 1000 and penalty= l2. Using this criterion, I checked the mean f1 score on outer_cv to get the average model performance. The mean f1 score on the outer_cv is 0.952 with a standard deviation of 0.041.

In case of the KNN algorithm, the first step is to standardize the data. Since the distance metric in KNN is sensitive to scale of variables, we need to standardize the x variables. I have used the z score scaling method to standardize the x variables. Next, I have used the grid search method to optimize hyperparameters for the KNN model using F1 scoring as the criterion for optimization. I have used the no of neighbors with range of values= (1,3,5,7,9,11,13,15,17,19,21,23,25,27,29) and the weight function with range of values= (uniform, distance) as parameters to optimize in grid search. I have specified p= 2 and metric = minkowski to use the Euclidean distance measure for distance calculation for finding nearest neighbors. Nested cross validation method is used here. This means that grid search will first use inner_cv to find the optimum parameters based on f1 score. The highest f1 score obtained by grid search method is 0.939 for the following criterion: no of neighbors= 9 and weights= uniform. Using this criterion, I checked the mean f1 score on outer_cv to get the average model performance. The mean f1 score on the outer_cv is 0.923 with a standard deviation of 0.022.



```
#Normalize Data
sc = StandardScaler()
#sc.fit(X_train)
sc.fit(X)
X_std = sc.transform(X)
#X_train_std = sc.transform(X_train)
#X_test_std = sc.transform(X_test)


# Choosing k for kNN AND type of distance
gs_knn = GridSearchCV(estimator=neighbors.KNeighborsClassifier(p=2,
                            metric='minkowski'),
                  param_grid=[{'n_neighbors': [1,3,5,7,9,11,13,15,17,19,21,23,25,27,29],
                               'weights':['uniform','distance']}],
                  scoring='f1',
                  cv=inner_cv,
                  n_jobs=4)

gs_knn = gs_knn.fit(X_std,y)
print("\n Parameter Tuning - KNN algorithm")
print("Non-nested CV F1 Score: ", gs_knn.best_score_)
print("Optimal Parameter: ", gs_knn.best_params_)
print("Optimal Estimator: ", gs_knn.best_estimator_) # Estimator that was chosen by the sear
nested_score_gs_knn = cross_val_score(gs_knn, X=X_std, y=y, cv=outer_cv)
print("Nested CV F1 Score: ",nested_score_gs_knn.mean(), " +/- ", nested_score_gs_knn.std())
```

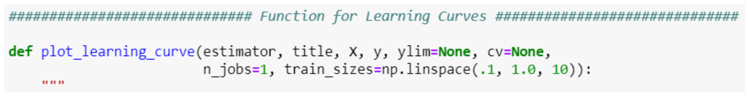


```
 Parameter Tuning - KNN algorithm
Non-nested CV F1 Score:  0.9391446659677958
Optimal Parameter:  {'n_neighbors': 9, 'weights': 'uniform'}
Optimal Estimator:  KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=9, p=2,
           weights='uniform')
Nested CV F1 Score:  0.9234809517894849  +/-  0.02175652225306678
```

Comparing the 3 models with their optimized hyperparameters on the basis of their nested cross validation F1 score, the Logistic regression model has the highest mean F1 score of 0.952. The KNN model comes second with a mean F1 score of 0.923, while the Decision tree model has the least mean F1 score of 0.886. Considering that the F1 score for positive class is for the Malignant class in the data, it is crucial to have the best possible F1 score since we need to have best possible performance in predicting the malignant class. Since this problem involves predicting a life threatening illness like cancer, the cost of wrongly predicting a malignant case as benign is very high and thus I would pick the model that performs the best in predicting the malignant class accurately. Thus, I would choose the logistic regression model in this case, since it has the highest mean nested cross validation F1 score.

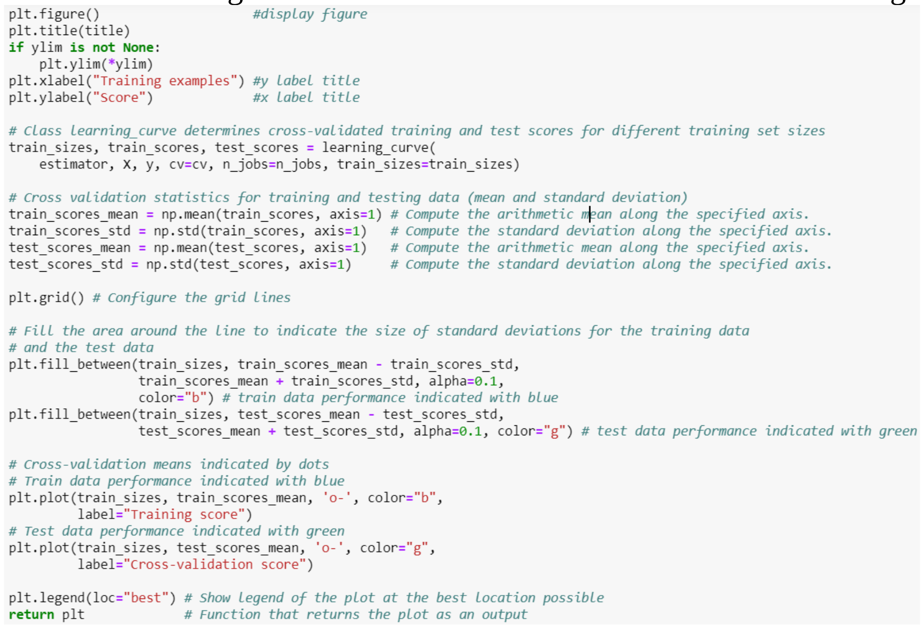
b) Build and visualize a learning curve for the logistic regression technique (visualize the performance for both training and test data in the same plot). Please provide screenshots of your code and explain the process you have followed.

[part b is worth 25 points in total:
8 points for correct visualization of learning curve for in-sample sample performance – show the performance for 10 different sizes - provide screenshots of your code and explain the process you have followed.
8 points for correct visualization of learning curve for out-sample sample performance – show the performance for 10 different sizes - provide screenshots of your code and explain the process you have followed.
9 points for discussing what we can learn from this specific learning curve – what are the insights that can be drawn]



```
############################# Function for Learning Curves #############################

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 10)):
    """
```

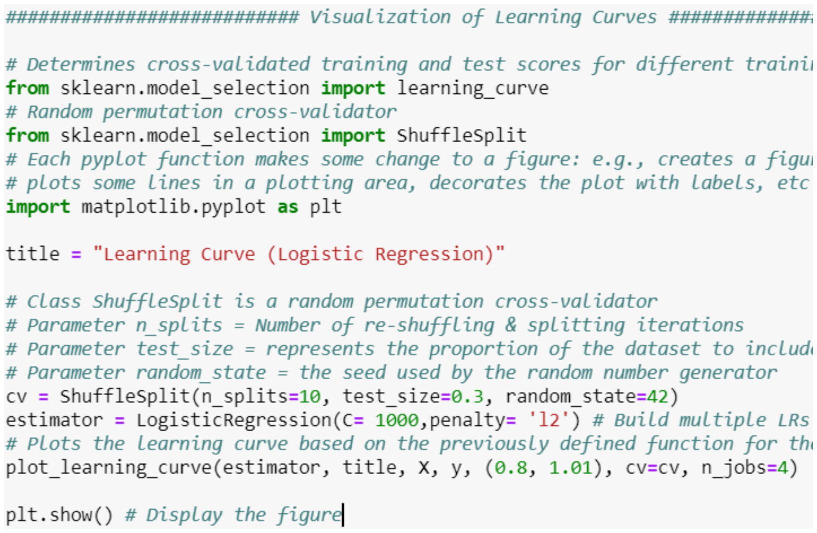
I have used a function in order to plot the learning curve for logistic regression. Since we need 10 different sizes of training instances on the X axis of the learning curve, I have specified the same as a parameter in the above function. The command train_sizes = np.linspace(.1,1.0,10) indicates that there will be a total of 10 sizes of the training data in increments of 10% of the training data. The minimum training data size will be 10% of training data and maximum size would be 100% of training data.



```
plt.figure()                    #display figure
plt.title(title)
if ylim is not None:
    plt.ylim(*ylim)
plt.xlabel("Training examples") #y label title
plt.ylabel("Score")             #x label title

# Class learning_curve determines cross-validated training and test scores for different training set sizes
train_sizes, train_scores, test_scores = learning_curve(
    estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)

# Cross validation statistics for training and testing data (mean and standard deviation)
train_scores_mean = np.mean(train_scores, axis=1) # Compute the arithmetic mean along the specified axis.
train_scores_std = np.std(train_scores, axis=1)   # Compute the standard deviation along the specified axis.
test_scores_mean = np.mean(test_scores, axis=1)   # Compute the arithmetic mean along the specified axis.
test_scores_std = np.std(test_scores, axis=1)     # Compute the standard deviation along the specified axis.

plt.grid() # Configure the grid lines

# Fill the area around the line to indicate the size of standard deviations for the training data
# and the test data
plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1,
                 color="b") # train data performance indicated with blue
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="g") # test data performance indicated with green

# Cross-validation means indicated by dots
# Train data performance indicated with blue
plt.plot(train_sizes, train_scores_mean, 'o-', color="b",
         label="Training score")
# Test data performance indicated with green
plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
         label="Cross-validation score")

plt.legend(loc="best") # Show legend of the plot at the best location possible
return plt             # Function that returns the plot as an output
```

Inside the function, I have used the learning_curve () command to extract the train_sizes, train_scores and test_scores, given the estimator, x variables, y variable, cross validation technique and train_sizes.

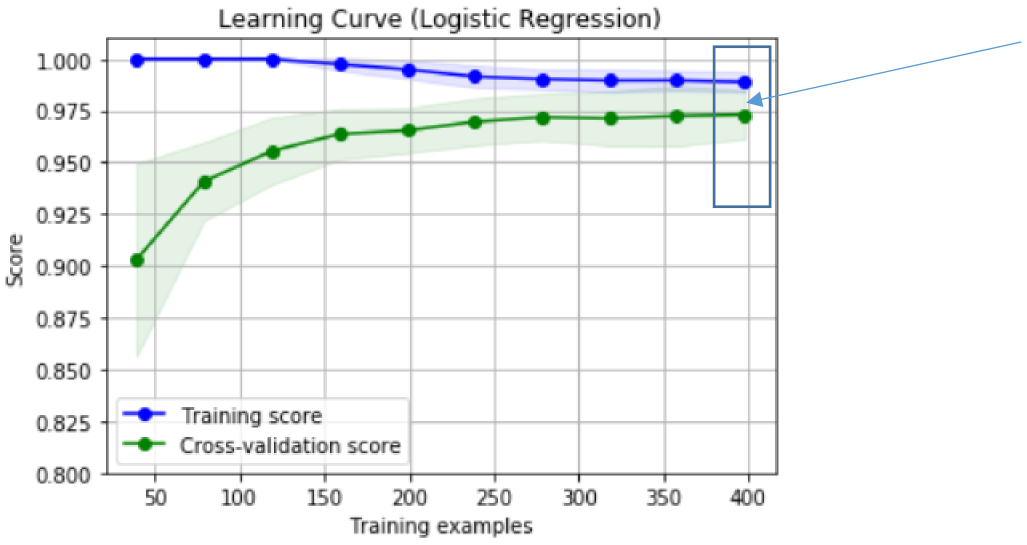
Then I have calculated the mean and standard deviation of train and test scores, which will be helpful in plotting in sample and out of sample performance lines respectively. The mean scores are plotted as lines, while the std deviation values are colored about the mean score lines in graph to show the std deviation of the scores.



```
########################### Visualization of Learning Curves #############

# Determines cross-validated training and test scores for different traini
from sklearn.model_selection import learning_curve
# Random permutation cross-validator
from sklearn.model_selection import ShuffleSplit
# Each pyplot function makes some change to a figure: e.g., creates a figu
# plots some lines in a plotting area, decorates the plot with labels, etc
import matplotlib.pyplot as plt

title = "Learning Curve (Logistic Regression)"

# Class ShuffleSplit is a random permutation cross-validator
# Parameter n_splits = Number of re-shuffling & splitting iterations
# Parameter test_size = represents the proportion of the dataset to includ
# Parameter random_state = the seed used by the random number generator
cv = ShuffleSplit(n_splits=10, test_size=0.3, random_state=42)
estimator = LogisticRegression(C= 1000,penalty= 'l2') # Build multiple LRs
# Plots the learning curve based on the previously defined function for th
plot_learning_curve(estimator, title, X, y, (0.8, 1.01), cv=cv, n_jobs=4)

plt.show() # Display the figure
```

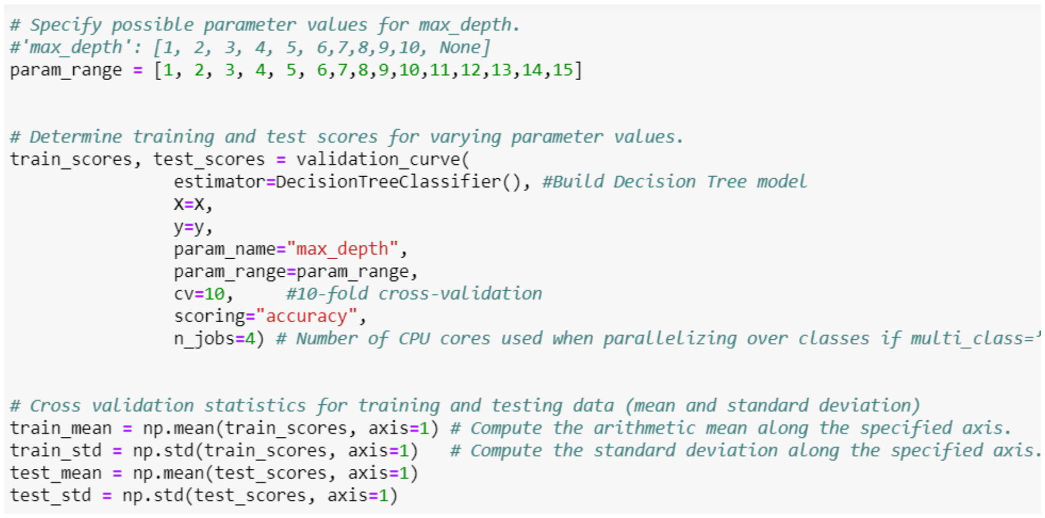
Using the above defined function “plot_learning_curve”, I apply it on an estimator that uses a logistic regression model with optimized hyperparameters as C= 1000, penalty =l2. These hyperparameters are taken from the results of our gridsearch method that we applied earlier. I have used the shufflesplit method for cross validation, specifying n_splits= 10 and test ratio=0.3. This means that 70% of the data is used as train and 30% as test. The way that shuffle split works is that there will be 10 iterations, and in each iteration, the logistic regression model will be trained on the training data and then training and test scores are calculated, which indicate in sample and out of sample performance respectively. We then find the mean training and test scores across the 10 iterations, which is specified inside the plot_learning_curve function.

Below is the learning curve for our logistic regression model. On the extreme left side of the graph, we can see that the no of training instance is about 40, which is 10% of the training data (training data = 70% of total dataset). For that point, we see the plotted mean training score is 1, while the plotted test mean score is about 0.9. As the no of training instances increase, we notice that the test mean score increases while the training mean score does not change significantly. At the point of about 400 training instances (100% of training data), we see that the test mean score has peaked to about 0.975, while the training mean score has fallen to its lowest point. The learning curve shows us the in sample and out of sample performance of the model as the no of training instances have increased. The learning curve gives a good indication of generalization performance of our logistic regression model as the number of training instances increases. In our case, the curve shows that the best out of sample performance is obtained when 100% of the training instances (70% of total dataset) are used to train the model.



c) **Build a fitting graph for different depths of the decision tree (visualize the performance for both training and test data in the same plot). Please provide screenshots of your code and explain the process you have followed.**

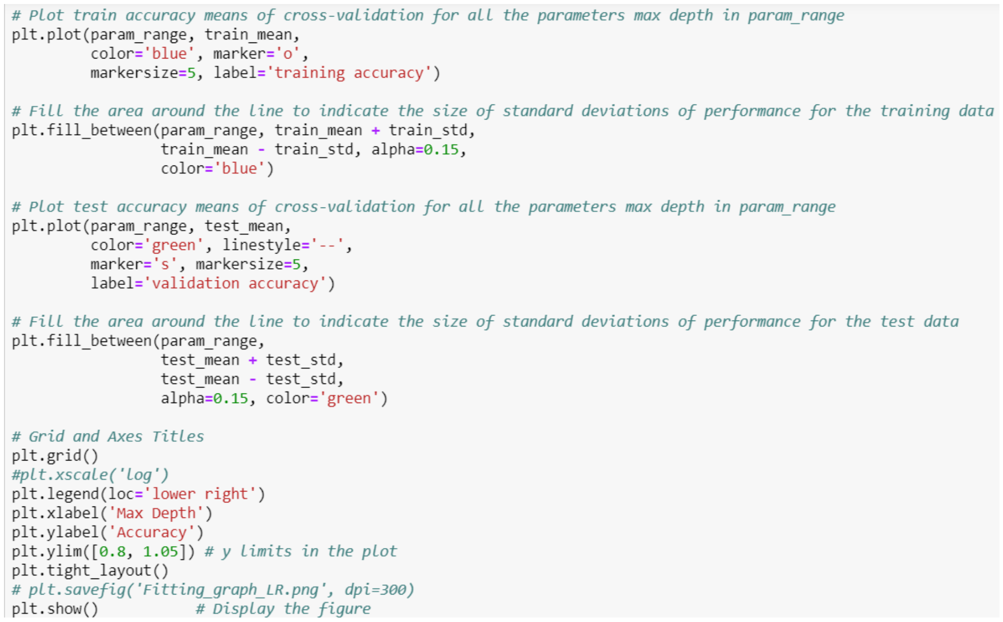
[part c is worth 25 points in total:
8 points for correct visualization of fitting graph for in-sample sample performance – show the performance for 15 different values- provide screenshots of your code and explain the process you have followed
8 points for correct visualization of fitting graph for out-of-sample performance – show the performance for 15 different values- provide screenshots of your code and explain the process you have followed
9 points for discussing what we can learn from this specific fitting graph – what are the insights that can be drawn]



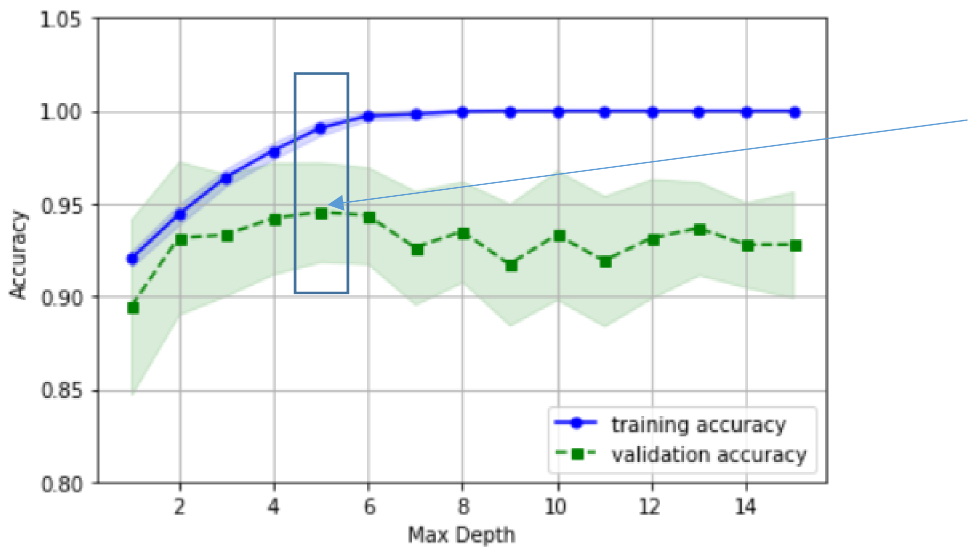
```
# Specify possible parameter values for max_depth.
#'max_depth': [1, 2, 3, 4, 5, 6,7,8,9,10, None]
param_range = [1, 2, 3, 4, 5, 6,7,8,9,10,11,12,13,14,15]


# Determine training and test scores for varying parameter values.
train_scores, test_scores = validation_curve(
                estimator=DecisionTreeClassifier(), #Build Decision Tree model
                X=X,
                y=y,
                param_name="max_depth",
                param_range=param_range,
                cv=10,     #10-fold cross-validation
                scoring="accuracy",
                n_jobs=4) # Number of CPU cores used when parallelizing over classes if multi_class='


# Cross validation statistics for training and testing data (mean and standard deviation)
train_mean = np.mean(train_scores, axis=1) # Compute the arithmetic mean along the specified axis.
train_std = np.std(train_scores, axis=1)   # Compute the standard deviation along the specified axis.
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)
```

First, I have specified the range of max_depth values (param_range) from 1 to 15, so that we have 15 different max depth values on the X axis of our fitting curve.
I have then used the validation_curve command to extract the train and test scores for varying max depth values. I have specified the decision tree classifier with no other modeling parameters defined, so the default values will be applied. I have used the X and y variables to fit the model on the data. I have mentioned the parameter as max depth so that the validation curve will vary the max depth for the specified range of max depth values. I have specified cv= 10 to use a 10 fold cross validation method. The scoring method specified is accuracy since I want to see how the model accuracy of the decision tree varies as max depth varies.
I have then calculated the mean and standard deviation of the training and test scores, which will be plotted as in sample and out of sample performance respectively.

I have used the std deviation values of training and test scores to fill area around the plotted training and test mean scores to indicate the std deviations of the respective performance. I have also specified the line types, colors, mark size for each of the in sample and out of sample performance lines in the graph



```python
# Plot train accuracy means of cross-validation for all the parameters max depth in param_range
plt.plot(param_range, train_mean,
         color='blue', marker='o',
         markersize=5, label='training accuracy')

# Fill the area around the line to indicate the size of standard deviations of performance for the training data
plt.fill_between(param_range, train_mean + train_std,
                 train_mean - train_std, alpha=0.15,
                 color='blue')

# Plot test accuracy means of cross-validation for all the parameters max depth in param_range
plt.plot(param_range, test_mean,
         color='green', linestyle='--',
         marker='s', markersize=5,
         label='validation accuracy')

# Fill the area around the line to indicate the size of standard deviations of performance for the test data
plt.fill_between(param_range,
                 test_mean + test_std,
                 test_mean - test_std,
                 alpha=0.15, color='green')

# Grid and Axes Titles
plt.grid()
#plt.xscale('log')
plt.legend(loc='lower right')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.ylim([0.8, 1.05]) # y limits in the plot
plt.tight_layout()
# plt.savefig('Fitting_graph_LR.png', dpi=300)
plt.show()          # Display the figure
```

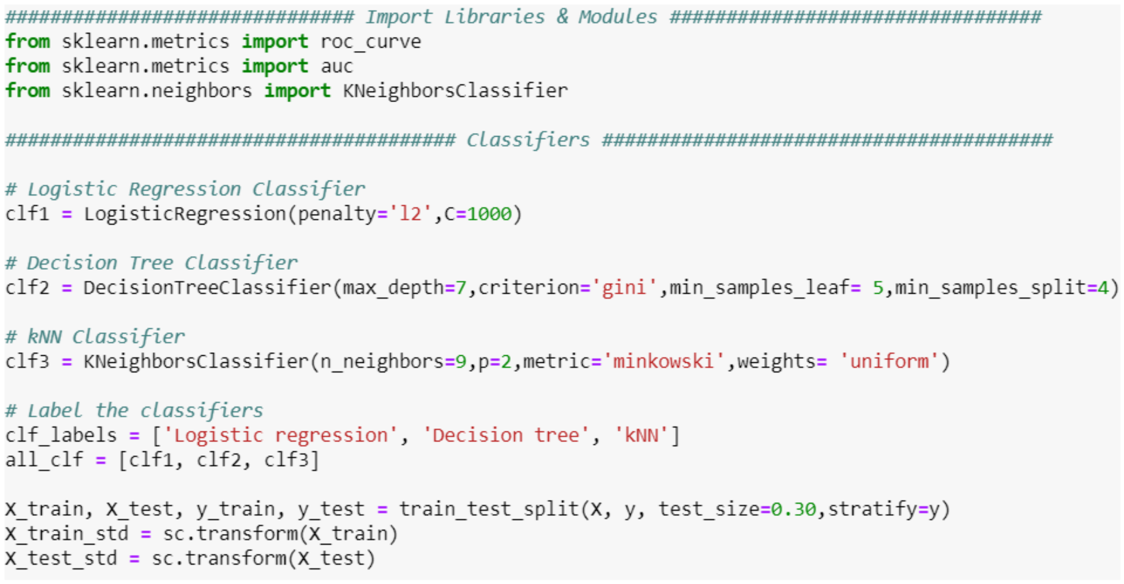


From the above fitting curve, we can see how the decision tree model accuracy varies with varying values of max depth. The curve shows the generalization performance of our decision tree model with increasing complexity. This visual gives us a good indication of when our model is overfitting to the training data. In our graph we can see that validation accuracy (out of sample) peaks when max depth = 5. At this point the training accuracy is not maximum. As max depth increases beyond 5, we observe that training accuracy increases but the validation accuracy falls. Its important to identify such points in the graph as it is a clear indication that the decision tree model is overfitting to the training data beyond max depth =5. Thus, I would limit the model complexity at this point by choosing max depth = 5, as my validation accuracy score is maximum for this level of model complexity.

d) Create an ROC curve for k-NN, decision tree, and logistic regression. Discuss the results. Which classifier would you prefer to choose? Please provide screenshots of your code and explain the process you have followed.

[part d is worth 25 points in total:
5 points for correct visualization of ROC graph for kNN – use optimal kNN from part a
5 points for correct visualization of ROC graph for Decision Tree – use optimal Decision Tree from part a

5 points for correct visualization of ROC graph for Logistic Regression – use optimal Logistic Regression from part a
2 points for showing all the ROC graphs in one single plot
3 points for showing AUC estimators in the ROC graph
5 points for discussing and correctly identifying which classifier you would use]

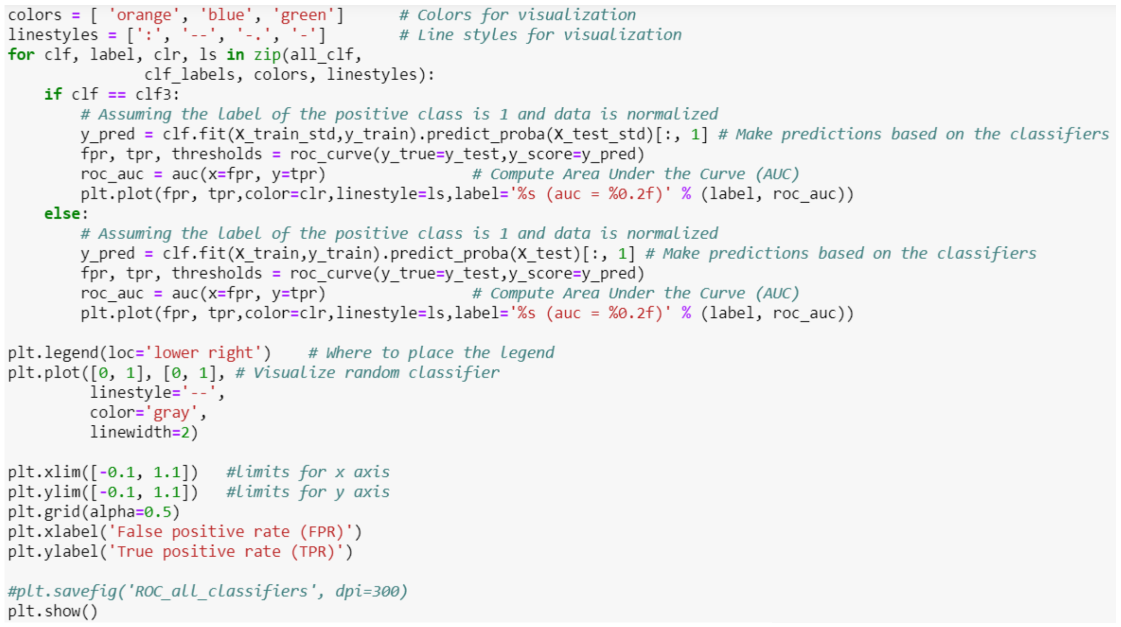


```
############################### Import Libraries & Modules ################################
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.neighbors import KNeighborsClassifier

######################################## Classifiers ########################################

# Logistic Regression Classifier
clf1 = LogisticRegression(penalty='l2',C=1000)

# Decision Tree Classifier
clf2 = DecisionTreeClassifier(max_depth=7,criterion='gini',min_samples_leaf= 5,min_samples_split=4)

# kNN Classifier
clf3 = KNeighborsClassifier(n_neighbors=9,p=2,metric='minkowski',weights= 'uniform')

# Label the classifiers
clf_labels = ['Logistic regression', 'Decision tree', 'kNN']
all_clf = [clf1, clf2, clf3]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,stratify=y)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

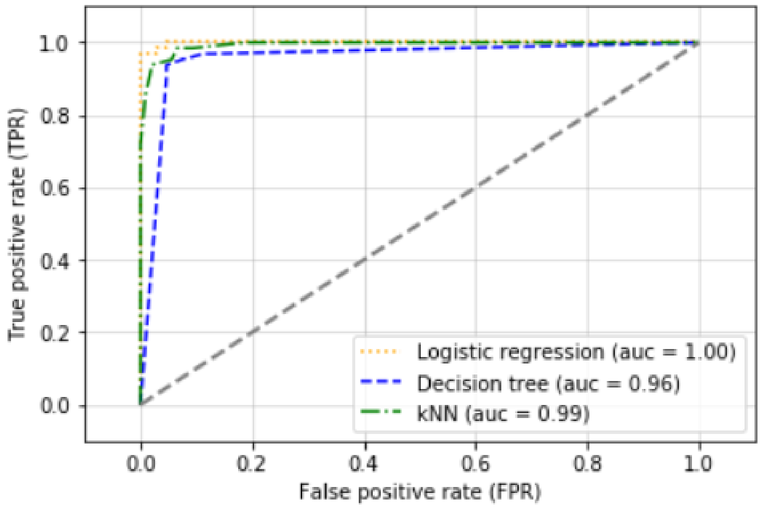
First, I have defined the 3 classifiers – logistic regression, decision tree and KNN. I have used the optimal hyperparameters that I have got from the gridsearch method that I ran earlier. For the Logistic regression model, I have used the optimal hyperparameters as penalty = L2 and inverse of regularization strength C = 1000. For the decision tree model, I have used the following optimal hyperparameters: max depth =7, splitting criterion = gini, minimum samples to split internal node = 4, and minimum samples per leaf node= 5. In the KNN model the optimal hyperparameters stated are no of neighbors = 9 and weights = ‘uniform’. I have used the p=2 and metric = minkowski to use the Euclidean distance measure to find the nearest neighbors.

I have then split the data into train and test with a ratio of 70:30 respectively, using stratified sampling to make sure that test and train have the same ratio of classes in y variable. I have also standardized the x variables in train and test datasets so that the standardized data is used to train the KNN model. It is important to use standardized data as the KNN algorithm is sensitive to scale of x variables while calculating distance between nearest neighbors.

I have then used the roc_curve() method to calculate the fpr, tpr and threshold values that need to plotted in our ROC curve. I have then used the auc() command to calculate the auc value, given the tpr and fpr rates. I have then plotted the various tpr,fpr values for multiple thresholds to give the ROC curve. This whole implementation is done with help of for loop to calculate the same for all 3 classifiers. I have used an if condition to make sure that standardized data is used for KNN model while unstandardized data is used for decision tree and logistic regression models.



```
colors = [ 'orange', 'blue', 'green']      # Colors for visualization
linestyles = [':', '--', '-.', '-']        # Line styles for visualization
for clf, label, clr, ls in zip(all_clf,
               clf_labels, colors, linestyles):
    if clf == clf3:
        # Assuming the label of the positive class is 1 and data is normalized
        y_pred = clf.fit(X_train_std,y_train).predict_proba(X_test_std)[:, 1] # Make predictions based on the classifiers
        fpr, tpr, thresholds = roc_curve(y_true=y_test,y_score=y_pred)
        roc_auc = auc(x=fpr, y=tpr)                # Compute Area Under the Curve (AUC)
        plt.plot(fpr, tpr,color=clr,linestyle=ls,label='%s (auc = %0.2f)' % (label, roc_auc))
    else:
        # Assuming the label of the positive class is 1 and data is normalized
        y_pred = clf.fit(X_train,y_train).predict_proba(X_test)[:, 1] # Make predictions based on the classifiers
        fpr, tpr, thresholds = roc_curve(y_true=y_test,y_score=y_pred)
        roc_auc = auc(x=fpr, y=tpr)                # Compute Area Under the Curve (AUC)
        plt.plot(fpr, tpr,color=clr,linestyle=ls,label='%s (auc = %0.2f)' % (label, roc_auc))

plt.legend(loc='lower right')    # Where to place the legend
plt.plot([0, 1], [0, 1], # Visualize random classifier
         linestyle='--',
         color='gray',
         linewidth=2)

plt.xlim([-0.1, 1.1])   #limits for x axis
plt.ylim([-0.1, 1.1])   #limits for y axis
plt.grid(alpha=0.5)
plt.xlabel('False positive rate (FPR)')
plt.ylabel('True positive rate (TPR)')

#plt.savefig('ROC_all_classifiers', dpi=300)
plt.show()
```



From the above ROC curve, we can see the performance of all 3 classifiers for multiple threshold values. We can see that all 3 classifiers have very high AUC values. The logistic regression model has an AUC of 1, which is the highest among all the 3 classifiers. The KNN model has an AUC of 0.99, while the decision tree model has the least AUC of 0.96. Since the logistic regression has a perfect AUC score of 1, it has the best classification performance for all thresholds among the 3 classifiers. Thus, I would choose logistic regression model based on its AUC score and the fact that our classification task needs to be as accurate as possible, since there are costs associated with predicting malignant classes wrongly as benign. The logistic regression model maximizes TP rate and minimizes the FP rate, which is very much desired outcome in our classification problem.