Homework #3 Part 2

Dola Qiu, Seth Abayo, Lanston Chen, Yizhou Sun, Yihua Wang

(put your names above (incl. any nicknames))

Note: This is a team homework assignment. Discussing this homework with your classmates outside your MSBA team is a **violation** of the Honor Code. If you **borrow code** from somewhere else, please add a comment in your code to **make it clear** what the source of the code is (e.g., a URL would sufficient). If you borrow code and you don't provide the source, it is a violation of the Honor Code.

Total grade: _____ out of ___100___ points

ATTENTION: HW3 has two parts. Please first complete the Quiz "HW3_Part1" on Canvas. Then, proceed with Part 2 in the following page. You will need to submit (a) a PDF file with your answers and screenshots of Python code snippets and (b) the Python code.

(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 random state of "42" 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.**

   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

4.1 Decision Tree

```
[ ] gs_dt = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                         param_grid=[{'max_depth': range(1,20),'criterion':['gini','entropy']}],
                         scoring='f1',
                         cv=inner_cv)
    gs_dt = gs_dt.fit(X_train,y_train)
    print("Inner CV F1: ", gs_dt.best_score_)
    print("Optimal Parameter: ", gs_dt.best_params_)
    print("Optimal Estimator: ", gs_dt.best_estimator_)
```

```
Inner CV F1:  0.9214790585758328
Optimal Parameter:  {'criterion': 'gini', 'max_depth': 7}
Optimal Estimator:  DecisionTreeClassifier(max_depth=7, random_state=42)
```

```
[ ] best_dt=DecisionTreeClassifier(random_state=42,criterion= 'gini', max_depth= 7)

    nested_score_gs_dt = cross_val_score(best_dt, X=X, y=y, cv=outer_cv,scoring='f1')
    print("Nested CV F1: ",nested_score_gs_dt.mean(), " +/- ", nested_score_gs_dt.std())
```

```
Nested CV F1:  0.9103676851259814  +/-  0.02211607718435726
```

```
[ ] from sklearn.metrics import confusion_matrix
    best_dt = DecisionTreeClassifier(random_state=42, criterion='gini', max_depth=7)
    best_dt.fit(X_train, y_train)
    y_pred = best_dt.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:")
    print(cm)
```

```
Confusion Matrix:
[[70  2]
 [ 5 37]]
```

We optimized the parameters of ‘max_depth’ and ‘criterion’ in the decision tree model using nested cross-validation to get the highest F1 score. After the process of GridSearchCV, we identify the optimal parameter: {'criterion': 'gini', 'max_depth': 7}. Then, we apply the best model on the whole dataset to check the generalization performance, and the result is that the F1 score is near 0.910. Moreover, we also calculate the confusion matrix.

7 points for correctly optimizing at least two parameters for the kNN and providing screenshots/explaining what you are doing and the corresponding results

4.3 KNN

```
[ ] from sklearn.pipeline import Pipeline
    from sklearn.preprocessing import StandardScaler
    from sklearn.neighbors import KNeighborsClassifier
    import random
    import numpy as np
    np.random.seed(42)

    pipe = Pipeline([
        ('sc', StandardScaler()),
        ('knn', KNeighborsClassifier(p=2, metric='minkowski')) ])
    params = {  'knn__n_neighbors': range(1,20),
            'knn__weights': ['uniform', 'distance'] }
    gs_knn = GridSearchCV(estimator=pipe,param_grid=params,scoring='f1',cv=inner_cv,)

    gs_knn = gs_knn.fit(X_train,y_train)
    print("Inner CV F1: ", gs_knn.best_score_)
    print("Optimal Parameter: ", gs_knn.best_params_)
    print("Optimal Estimator: ", gs_knn.best_estimator_)
```

```
Inner CV F1:  0.9567639694417907
Optimal Parameter:  {'knn__n_neighbors': 3, 'knn__weights': 'uniform'}
Optimal Estimator:  Pipeline(steps=[('sc', StandardScaler()),
                ('knn', KNeighborsClassifier(n_neighbors=3))])
```

```
[ ] best_knn =Pipeline([('sc', StandardScaler()),
        ('knn', KNeighborsClassifier(n_neighbors=3,weights='uniform',p=2, metric='minkowski')) ])
    nested_score_gs_knn= cross_val_score(best_knn, X=X, y=y, cv=outer_cv,scoring='f1')
    print("Nested CV F1: ",nested_score_gs_knn.mean(), " +/- ", nested_score_gs_knn.std())
```

```
Nested CV F1:  0.9470833861351012  +/-  0.01778525540609691
```

```
[ ] from sklearn.metrics import confusion_matrix
    best_knn.fit(X_train, y_train)
    y_pred = best_knn.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:")
    print(cm)
```

```
Confusion Matrix:
[[71  1]
 [ 6 36]]
```

First of all, we tried to optimize the parameters of the number of k and 'weights' in the KNN model using nested cross-validation to get the highest F1 score. In order to standardize the scale of different variables, we utilized the StandardScaler in the Pipeline. After the process of GridSearchCV, we identify the optimal parameter: {'knn__n_neighbors': 3, 'knn__weights': 'uniform'. Then, we apply the best model on the whole dataset to check the generalization performance, and the result is that the F1 score is near 0.947. Eventually, we also calculate the confusion matrix.

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

4.2 Logistic Regression

```
[ ] gs_lr = GridSearchCV(estimator=LogisticRegression(random_state=42, solver='liblinear'),
                         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_train,y_train)

    print("Inner CV F1: ", gs_lr.best_score_)
    print("Optimal Parameter: ", gs_lr.best_params_)
    print("Optimal Estimator: ", gs_lr.best_estimator_)
```

```
Inner CV F1:  0.9506778093986104
Optimal Parameter:  {'C': 1000, 'penalty': 'l1'}
Optimal Estimator:  LogisticRegression(C=1000, penalty='l1', random_state=42, solver='liblinear')
```

```
[ ] best_lr=LogisticRegression(random_state=42,C=1000, penalty='l1',solver='liblinear')
    nested_score_gs_lr = cross_val_score(best_lr, X=X, y=y, cv=outer_cv,scoring='f1')
    print("Nested CV F1:",nested_score_gs_lr.mean(), " +/- ", nested_score_gs_lr.std())
```

```
Nested CV F1: 0.9543454715219422  +/-  0.03002585198790486
```

```
[ ] from sklearn.metrics import confusion_matrix
    best_lr.fit(X_train, y_train)
    y_pred = best_lr.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:")
    print(cm)
```

```
Confusion Matrix:
[[70  2]
 [ 4 38]]
```

To begin with, we optimized the parameter C and the way of penalty in the logistic regression model using nested cross-validation to get the highest F1 score. After the process of GridSearchCV, we identify the optimal parameter: {'C': 1000, 'penalty': 'l1'}. Then, we apply the best model on the whole dataset to check the generalization performance, and the result is that the F1 score is near 0.954. Lastly, we tried to find out the confusion matrix.

4 points for contrasting their performance of all three algorithms and discussing which one would you prefer to use

The F1 score of decision tree model, KNN model, and logistic regression model are respectively 0.910, 0.947, and 0.95. Meanwhile, the number of false negatives are respectively 5, 6, 4.

I prefer to use the logistic regression model. Here are my reasons.

Above all, it is plain to see that the logistic regression model has the highest F1 score, which means that it can predict better in terms of both precision and recall.

What is more, when referring to the confusion matrix, we can easily find that the logistic regression model has the smallest number of false negatives. In this scenario, a false negative is significantly more costly than a false positive, so we tend to pay more attention to false negatives, and the logistic regression model behaves the best on that.

Lastly, logistic regression has its advantages in simplicity and interpretability. It is based on linear combinations of features, making it easy to understand how each feature influences the classification outcome. Moreover, logistic regression is computationally efficient, and it doesn't require substantial

computational resources or large datasets for training. This efficiency allows it to provide real-time results, making it a practical choice for this application.

To sum up, based on both the specific dataset metrics and general considerations, I prefer to make use of logistic regression for cancer detection

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]

Import libraries

```
%matplotlib inline
# Import necessary libraries and specify that graphs should be plotted inline.
from sklearn.datasets import load_iris # import Iris Data Set
from sklearn import linear_model       # the sklearn.linear_model module implements generalized linear models
import numpy as np                     # NumPy is the package for scientific computing with Python
from sklearn.model_selection import learning_curve # determines cross-validated training and test scores for different training set sizes
```

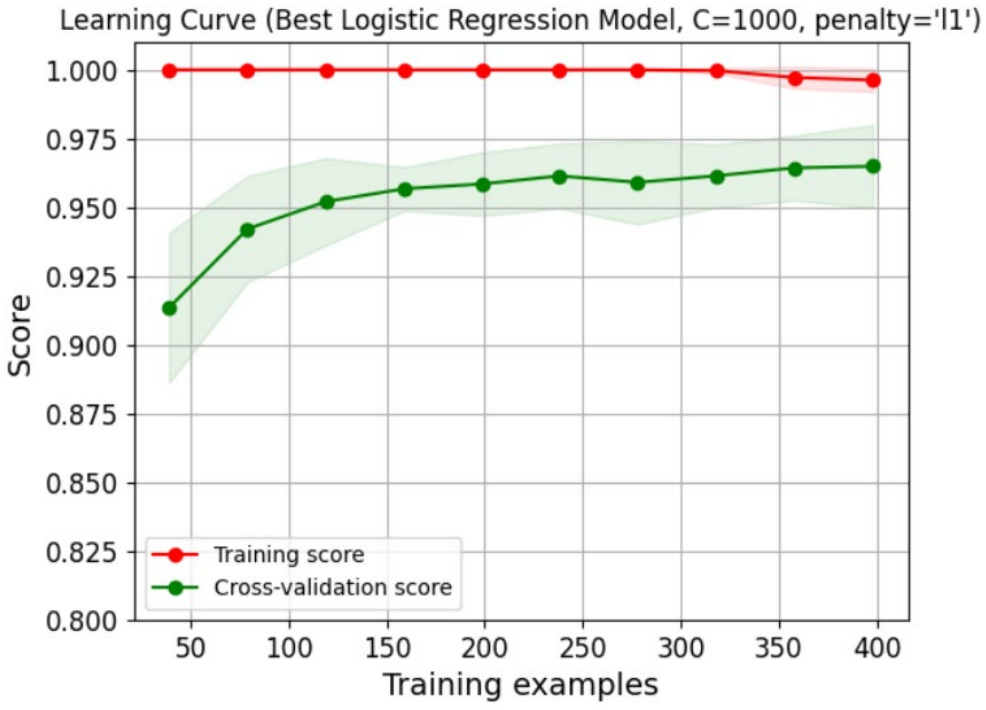
Define learning curve function

```
def plot_learning_curve(estimator,       # data science algorithm
                        title,           # title of the plot
                        X, y,            # data (features and target variable)
                        ylim=None,       # minimum and maximum y values plotted
                        cv=None,         # cross validation splits
                        n_jobs=1,        # parallell estimation using multiple processors
                        train_sizes=np.linspace(.1, 1.0, 10)): #linspace returns evenly spaced numbers over a specified interval (start, stop, num)

    # Initialization of Figure
    plt.figure()                  # display figure

    # Titles/labels for the plot are set
    plt.title(title)              # specify title based on parameter provided as input
    if ylim is not None:          # if ylim was specified as an input, make sure the plots use these limits
        plt.ylim(*ylim)
    plt.xlabel("Training examples") # y label title
    plt.ylabel("Score")           # x label title

    # Learning Curve Calculation
    # This helps to understand how the model's performance varies as more data is used for training.
    # Class learning_curve determines cross-validated training and test scores for different training set sizes
    # Detailed documentation: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.learning_curve.html
    # train_sizes = numbers of training examples that has been used to generate the learning curve
    # train_scores = scores on training sets (array)
    # test_scores = scores on test set (array)
    train_sizes, train_scores, test_scores = learning_curve(estimator, # data science algorithm
                                                            X, y,      # data (features and target variable)
                                                            cv=cv,     # cross-validation folds
                                                            n_jobs=n_jobs, # number of jobs to run in parallel using multiple processors
                                                            train_sizes=train_sizes) # relative or absolute numbers of training examples
                                                                                     # that will be used to generate the learning curve

    # Score Calculations
    # 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.

    # Visualization of Learning Curve
    plt.grid()                                        # configure the grid lines in the plot
                                                      # adds grid lines to the plot for better visualization and understanding of the data points

    # Fill Between Scores to Indicate Standard Deviation
    # Fill the area around the line to indicate the size of standard deviations for the training data
    # and the test data
    # The area filled represents one standard deviation above and below the mean of the training/test scores
    plt.fill_between(train_sizes,                           # the x coordinates of the nodes defining the curves
                     train_scores_mean - train_scores_std,  # the y coordinates of the nodes defining the first curve
                     train_scores_mean + train_scores_std,  # the y coordinates of the nodes defining the second curve
                     alpha=0.1,                             # level of transparency in the color fill
                     color="r")                             # train data performance indicated with red
    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

    # Plotting the Mean Scores
    # Cross-validation mean scores indicated by dots
    # Train data performance indicated with red
    plt.plot(train_sizes,                                   # the horizontal coordinates of the data points
             train_scores_mean,                             # the vertical coordinates of the data points
             'o-',                                          # o- will produce a small circle and a solid line to connect the markers
             color="r",                                     # line of red color
             label="Training score")                        # specify label title for this plot

    # Test data performance indicated with green
    plt.plot(train_sizes,
             test_scores_mean,
             'o-',
             color="g",                                     # line of green color
             label="Cross-validation score")

    plt.legend(loc="best")                  # show legend of the plot at the best location possible
                                            # placing it in the "best" location based on where matplotlib thinks it will least overlap with other elements
    return plt                              # function that returns the plot as an output
```

Plotting a learning curve using LogisticRegression as the estimator, utilizing the ShuffleSplit method for cross-validation.

```python
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
import matplotlib.pyplot as plt

# Previously, the title was set for Logistic Regression.
# If you know the best parameters, you might want to include them in the title for clarity.
title = "Learning Curve (Best Logistic Regression Model, C=1000, penalty='l1')"

# The cross-validator remains the same; no change here.
cv = ShuffleSplit(n_splits=10, test_size=0.3, random_state=42)

# Previously, an instance of a default Logistic Regression model was created.
# Now, we create an instance of the best Logistic Regression model that you tuned earlier.
# We use the best parameters found by GridSearchCV for C and penalty and specify the solver.
best_lr = LogisticRegression(random_state=42, C=1000, penalty='l1', solver='liblinear')

# Now, call the plot_learning_curve function with the best_lr estimator.
# The ylim, cv, and n_jobs parameters remain the same.
plot_learning_curve(best_lr, title, X, y, (0.8, 1.01), cv=cv, n_jobs=4)

# Finally, display the figure. This remains unchanged.
plt.show()
```

Visualization for learning curve



Insights

With our best model, the training score is high, and the cross-validation score improves as the training example increases. The validation score is between 0.95 and 0.975 which indicates that the model performs well on unseen data, preventing overfitting from occurring.

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]

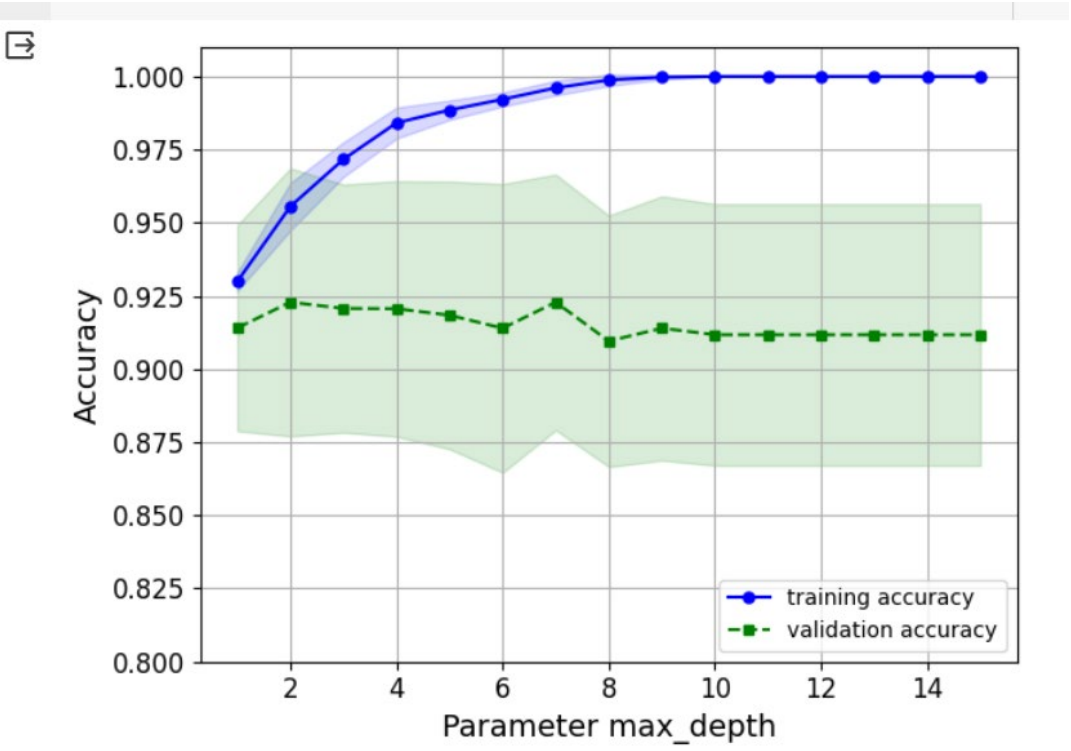
Import libraries

```
[21] from sklearn.model_selection import validation_curve
     np.random.seed(42)
     param_range =range(1,16)
     X_train,X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,stratify=y, random_state=42)

     train_scores, test_scores = validation_curve(
                    estimator=DecisionTreeClassifier(random_state=42),
                    X=X_train,
                    y=y_train,
                    param_name="max_depth",
                    param_range=param_range,
                    cv=10,
                    scoring="accuracy",                          )
```

The code calculates the mean and standard deviation of accuracies across a parameter range and plots these for both training and validation datasets, filling the area around the lines to indicate the uncertainty or variability in performance.

```
train_mean = np.mean(train_scores, axis=1) # compute the arithmetic mean along the specified axis (train data)
train_std = np.std(train_scores, axis=1)   # compute the standard deviation along the specified axis (train data)
test_mean = np.mean(test_scores, axis=1)   # compute the arithmetic mean along the specified axis (test data)
test_std = np.std(test_scores, axis=1)     # compute the standard deviation along the specified axis (test data)

############################## Visualization - Fitting Graph ##############################

# Plot train accuracy means of cross-validation for all the parameters C in param_range
plt.plot(param_range,                    # the horizontal coordinates of the data points
         train_mean,                     # the vertical coordinates of the data points
         color='blue',                   # aesthetic parameter - color
         marker='o',                     # aesthetic parameter - marker
         markersize=5,                   # aesthetic parameter - size of marker
         label='training accuracy')      # specify label title

# Fill the area around the line to indicate the size of standard deviations of performance for the training data
plt.fill_between(param_range,            # the x coordinates of the nodes defining the curves
                 train_mean + train_std, # the y coordinates of the nodes defining the first curve
                 train_mean - train_std, # the y coordinates of the nodes defining the second curve
                 alpha=0.15,             # level of transparency in the color fill
                 color='blue')           # aesthetic parameter - color

# Plot test accuracy means of cross-validation for all the parameters C 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.legend(loc='lower right')
plt.xlabel('Parameter max_depth')
plt.ylabel('Accuracy')
plt.ylim([0.8, 1.01]) # y limits in the plot
plt.tight_layout()

plt.show()
```

Visualization for fitting graph



Insights

In a decision tree model, as the maximum depth increases, the model becomes more complex and tends to fit the training data more closely, which explains the increase in training accuracy to 1. However, this increased complexity can lead to overfitting, where the model captures the noise and outliers in the training data as if they were true patterns. Consequently, the model's generalization to unseen data, represented by validation accuracy, can degrade, resulting in a slight decrease.

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 library

```
[52] import numpy as np
     from sklearn.linear_model import LogisticRegression
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import roc_curve
     from sklearn.metrics import auc
```

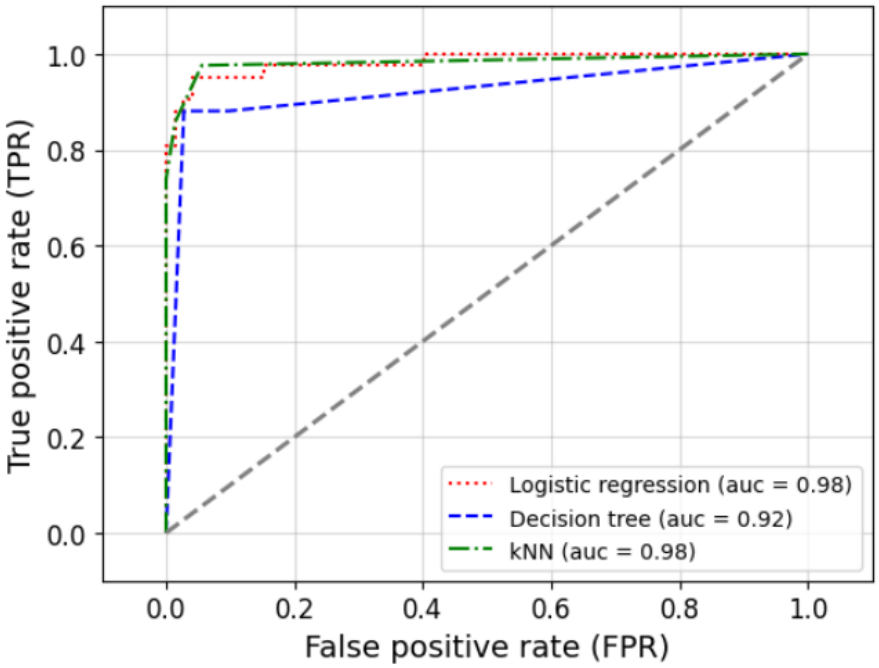
Define classifier

```
# LR Classifier
clf1 = LogisticRegression(random_state=42,C=1000, penalty='l1',solver='liblinear')

# Decision Tree Classifier
clf2 = DecisionTreeClassifier(random_state=42,criterion= 'gini', max_depth= 7)

# kNN Classifier

clf3 = Pipeline([
    ('sc', StandardScaler()),
    ('knn', KNeighborsClassifier(n_neighbors=3,weights='uniform',p=2, metric='minkowski')) ])

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

Plot ROC graph for three models

```
colors = [ 'red', 'blue', 'green']
linestyles = [':', '--', '-.', '-']
for clf, label, clr, ls in zip(all_clf,
                clf_labels, colors, linestyles):


    y_pred = clf.fit(X_train, y_train).predict_proba(X_test)[:, 1]

    fpr, tpr, thresholds = roc_curve(y_true=y_test,y_score=y_pred)
    roc_auc = auc(x=fpr, y=tpr)
    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='--',         # aesthetic parameters
         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.show()
```

Visualization of ROC graph



Interpretation of the graph

While both Logistic Regression and k-Nearest Neighbors (kNN) models exhibit high AUC values of 0.98, suggesting strong performance, the choice between them necessitates a consideration of various factors beyond AUC. Logistic Regression is typically more interpretable and computationally efficient for predictions, especially for large datasets, but it makes certain assumptions such as linearity and can be sensitive to outliers. In contrast, kNN, though assumption-free and potentially more robust to noisy training data, might struggle with high-dimensional data and is computationally intensive, particularly with large datasets. Additionally, kNN requires careful hyperparameter tuning, specifically choosing an appropriate 'k' and a distance metric.