Homework #3

_____Jie Zhu_____

(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]

	Positive	**Negative**
Yes	**4.985**	**-0.015**
No	**0**	**0**

If a coupon is presented and a consumer cashes it, the value is $5 - $0.015 = $4.985. If a coupon is presented but nobody cashes it, the value is $0 - $0.015 = $-0.015. If a coupon isn't presented but a consumer would have cashed it if offered, the value is $0. If a coupon isn't presented and a consumer wouldn't have cashed it if offered, the value is $0.

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**

p(Y,p) = 560 / (560+70+120+450) = 560 / 1200 = 0.4667

p(Y,n) = 70 / 1200 = 0.0583

p(N,p) = 120 / 1200 = 0.1

p(N,n) = 450 / 1200 = 0.375

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) = 0.4667*4.985 + 0.0583*(-0.015) + 0.1*0 + 0.375*0 = $2.33

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.

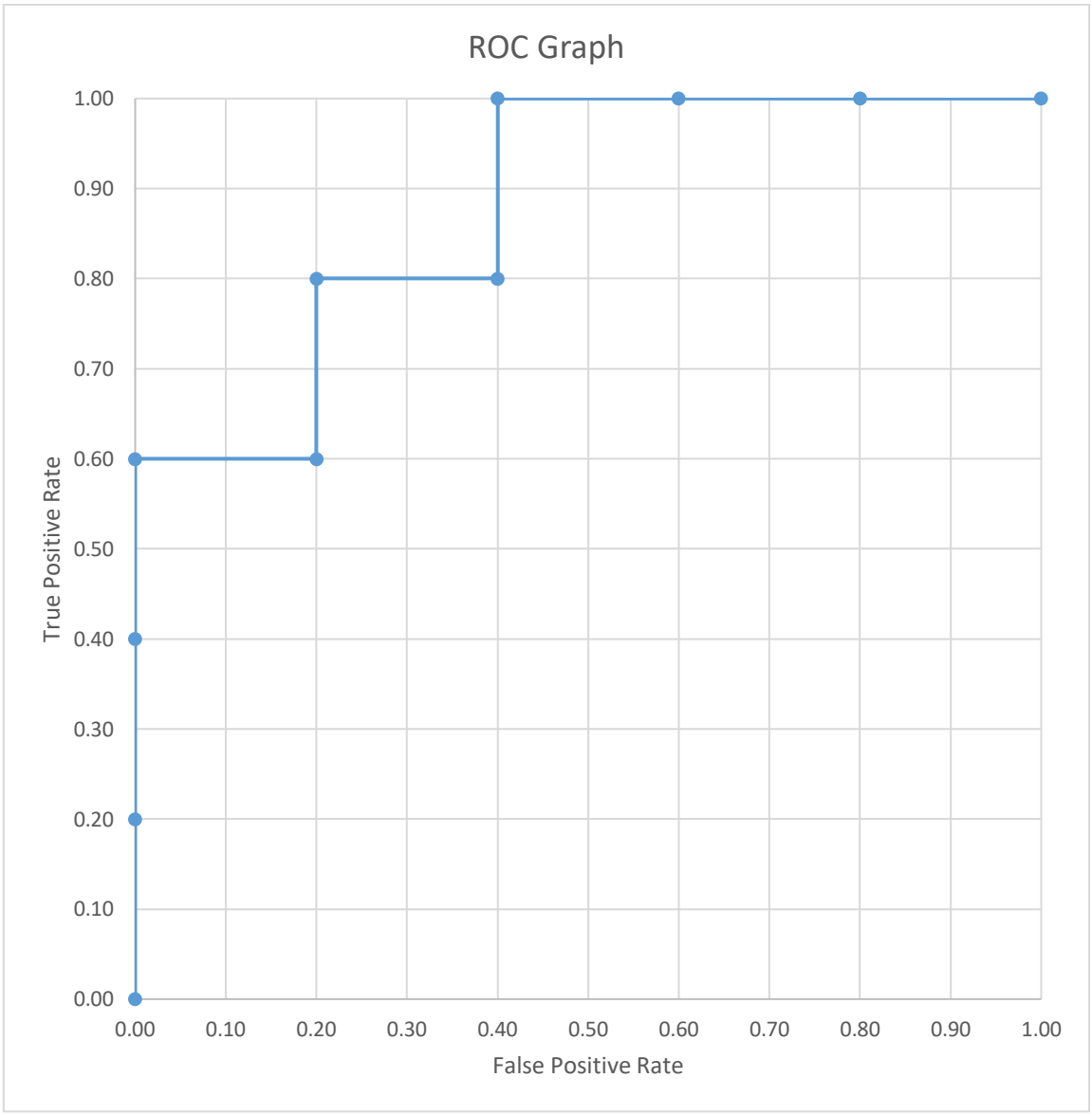
Actual Class (from validation data)	Estimated Probability of Record Belonging to Class "fraud"
fraud	0.95
fraud	0.91
fraud	0.75
non-fraud	0.67
fraud	0.61
non-fraud	0.46
fraud	0.42
non-fraud	0.25
non-fraud	0.09
non-fraud	0.04

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]

Positive(Fraud) = 5; Negative(Fraud) = 5

Threshold	FP	FP Rate (FP/N)	TP	TP Rate (TP/P)
0.96	**0**	**0**	**0**	**0**
0.92	**0**	**0**	**1**	**0.2**
0.76	**0**	**0**	**2**	**0.4**
0.68	**0**	**0**	**3**	**0.6**
0.62	**1**	**0.2**	**3**	**0.6**
0.47	**1**	**0.2**	**4**	**0.8**
0.43	**2**	**0.4**	**4**	**0.8**
0.26	**2**	**0.4**	**5**	**1**
0.10	**3**	**0.6**	**5**	**1**
0.05	**4**	**0.8**	**5**	**1**
0.03	**5**	**1**	**5**	**1**


ROC Graph
True Positive Rate
False Positive Rate
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00
0.00
0.10
0.20
0.30
0.40
0.50
0.60
0.70
0.80
0.90
1.00

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 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.**

[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]

For this problem, since there are only 569 observations, I use 5 folds for nested cross-validation purpose. To find the best model, I use GridSearchCV() function. In the decision tree model, I optimize 4 parameters, which are max_depth, criterion, min_samples_leaf, and min_samples_split.

```
inner_cv = KFold(n_splits=5, shuffle=True, random_state=42)
outer_cv = KFold(n_splits=5, shuffle=True, random_state=42)

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

# Choosing depth of the tree AND splitting criterion AND min_samples_leaf AND min_samples_split
gs_dt2 = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                      param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None], 'criterion':['gini','entropy'],
                                   'min_samples_leaf':[1,2,3,4,5],
                                   'min_samples_split':[2,3,4,5]}],
                      scoring='f1',
                      cv=inner_cv,
                      n_jobs=4)

gs_dt2 = gs_dt2.fit(X,y)
print("\n Parameter Tuning #1")
print("Non-nested CV F1 Score: ", gs_dt2.best_score_)
print("Optimal Parameter: ", gs_dt2.best_params_)
print("Optimal Estimator: ", gs_dt2.best_estimator_)
nested_score_gs_dt2 = cross_val_score(gs_dt2, X=X, y=y, cv=outer_cv)
print("Nested CV F1 Score: ",nested_score_gs_dt2.mean(), " +/- ", nested_score_gs_dt2.std())
```

The best decision tree model is one with max_depth = 5, criterion = "gini", min_samples_leaf = 5, min_samples_split = 2. The best F1 score for this model is 0.935, and the nested mean CV F1 score is 0.927. The standard-deviation of nested CV F1 is 0.014.

```
 Parameter Tuning #1
Non-nested CV F1 Score:  0.934657984394364
Optimal Parameter:  {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 5, 'min_samples_split': 2}
Optimal Estimator:  DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=42,
            splitter='best')
Nested CV F1 Score:  0.9265708104408736  +/-  0.014452515617888772
```

In the logistic regression model, I optimize 2 parameters, which are C (from 0.00001 to 10000000) and penalty (L1 or L2).

```
gs_lr2 = GridSearchCV(estimator=LogisticRegression(random_state=42),
                 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_lr2 = gs_lr2.fit(X,y)
print("\n Parameter Tuning #2")
print("Non-nested CV F1 Score: ", gs_lr2.best_score_)
print("Optimal Parameter: ", gs_lr2.best_params_)
print("Optimal Estimator: ", gs_lr2.best_estimator_)
nested_score_gs_lr2 = cross_val_score(gs_lr2, X=X, y=y, cv=outer_cv)
print("Nested CV F1 Score:",nested_score_gs_lr2.mean(), " +/- ", nested_score_gs_lr2.std())
```

The best logistic regression model is one with C = 100 and penalty = "L1". The best F1 score for this model is 0.960, and the nested mean CV F1 score is 0.944. The standard-deviation of nested CV F1 is 0.029.

```
 Parameter Tuning #2
Non-nested CV F1 Score:  0.9600639979772475
Optimal Parameter:  {'C': 100, 'penalty': 'l1'}
Optimal Estimator:  LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=42, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
Nested CV F1 Score: 0.9444853129388919  +/-  0.028926982011796845
```

In the KNN model, I optimize 3 parameters, which are the number of neighbors, p, and the type of weights (uniform or distance).

```
# Choosing k for kNN AND type of distance
gs_knn2 = GridSearchCV(estimator=neighbors.KNeighborsClassifier(
                           metric='minkowski'),
                 param_grid=[{'n_neighbors': [1,3,5,7,9,11,13,15,17,19,21],
                              'weights':['uniform','distance'],
                              'p': [1,2,3,4,5]}],
                 scoring='f1',
                 cv=inner_cv,
                 n_jobs=4)

gs_knn2 = gs_knn2.fit(X,y)
print("\n Parameter Tuning #3")
print("Non-nested CV F1 Score: ", gs_knn2.best_score_)
print("Optimal Parameter: ", gs_knn2.best_params_)
print("Optimal Estimator: ", gs_knn2.best_estimator_) # Estimator that was chosen by the search, i.e. estimator which gave highest score
nested_score_gs_knn2 = cross_val_score(gs_knn2, X=X, y=y, cv=outer_cv)
print("Nested CV F1 Score: ",nested_score_gs_knn2.mean(), " +/- ", nested_score_gs_knn2.std())
```

The best KNN model I have is one with n_neighbors = 5, p = 1, weights = "uniform". The best F1 score for this model is 0.921, and the nested mean CV F1 score is 0.914. The standard-deviation of nested CV F1 is 0.040.

```
 Parameter Tuning #3
Non-nested CV F1 Score:  0.9207223798751314
Optimal Parameter:  {'n_neighbors': 5, 'p': 1, 'weights': 'uniform'}
Optimal Estimator:  KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=1,
           weights='uniform')
Nested CV F1 Score:  0.9138948051874017  +/-  0.04025188646924722
```

The best among these 3 models is the logistic regression. It has higher value in both the best F1 score (0.960 vs. 0.935 & 0.921 for the other two models) and the mean nested F1 score (0.944 vs. 0.927 & 0.914 for the other two models). Meanwhile, the standard deviation for the logistic regression is moderate and acceptable, showing that this model has stable performance for this dataset.

```
 Parameter Tuning #1
Non-nested CV F1 Score:  0.934657984394364
Optimal Parameter:  {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 5, 'min_samples_split': 2}
Optimal Estimator:  DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=42,
            splitter='best')
Nested CV F1 Score:  0.9265708104408736  +/-  0.014452515617888772

 Parameter Tuning #2
Non-nested CV F1 Score:  0.9600639979772475
Optimal Parameter:  {'C': 100, 'penalty': 'l1'}
Optimal Estimator:  LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=42, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
Nested CV F1 Score: 0.9444853129388919  +/-  0.028926982011796845

 Parameter Tuning #3
Non-nested CV F1 Score:  0.9207223798751314
Optimal Parameter:  {'n_neighbors': 5, 'p': 1, 'weights': 'uniform'}
Optimal Estimator:  KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=1,
           weights='uniform')
Nested CV F1 Score:  0.9138948051874017  +/-  0.04025188646924722
```

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]

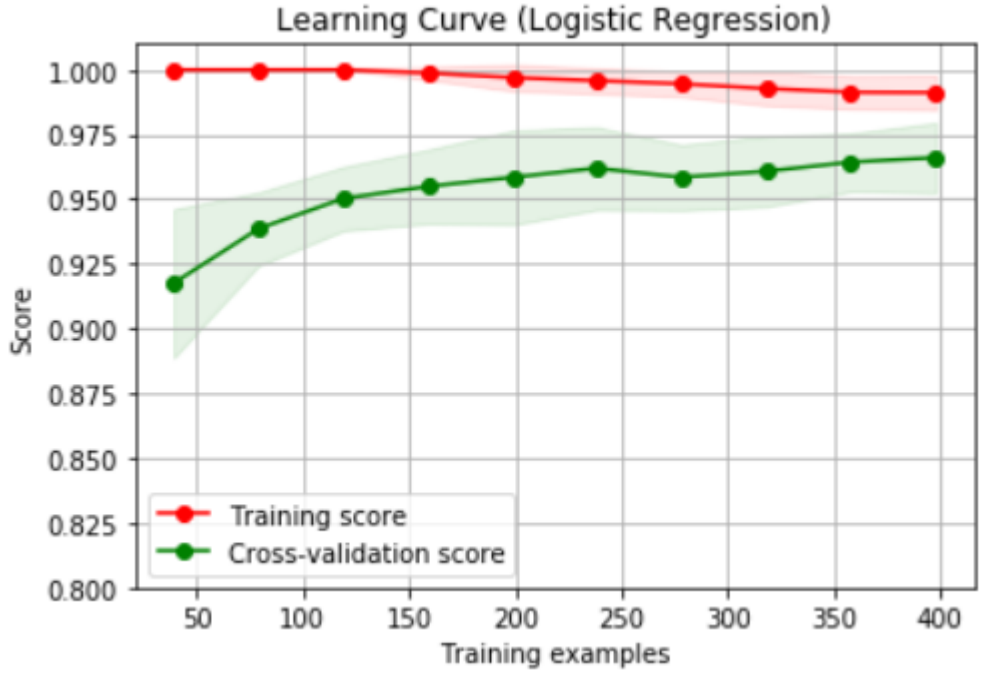
In order to plot the learning curve for both test data performance and train data performance in the same plot, I define a function called plot_learning_curve(). First, I determine cross-validated training and test scores for different training set sizes (min=0.1, max=1.0, 10 different sizes). Then, I calculate mean and standard deviation for train and test scores in order to fill the area around the line to indicate the size of standard deviations for training data and test data. Finally, I color the two lines with two different colors, red and green.

```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 10)):
    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
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     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

    # Cross-validation means indicated by dots
    # Train data performance indicated with red
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             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
```

After defining this function, I shuffle and split the original data set into 70% train and 30% test. I apply the function into the best logistic regression model I get from previous part, which specifies C= 100 and penalty = "L1".

```
title = "Learning Curve (Logistic Regression)"

cv = ShuffleSplit(n_splits=10, test_size=0.3, random_state=42)
estimator = LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=42, solver='warn',
          tol=0.0001, verbose=0, warm_start=False) # Build multiple LRs as we increase th
# Plots the learning curve based on the previously defined function for the logistic regre
plot_learning_curve(estimator, title, X, y, (0.8, 1.01), cv=cv, n_jobs=4)

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

The learning curve graph has two lines, the red line is the one for training data scores, while the green line is for test data scores. The green line climbs up until about 240 samples. It slides down for about 40 samples and slowly goes up after. The standard deviations for training data become larger as number of samples increases. As the gap between red line and green line shrinks, the model turns from underfitting to neutral. Due to the fact that the green curve becomes stable after 350 samples of training examples, more data at this point won't improve this logistic regression model much.



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]

In order to plot the fitting graph for different depths of the previous decision tree model, I use the validation_curve() function. I pass the previous optimal parameters for decision tree into the estimator. In this function, I use X=X & y=y instead of X=X_train & y=y_train. This is because I am already using 5-fold cross-validation (cv = 5) in the function. The parameter I am changing in this case is max_depth. I set 16 different values for max_depth, from 1 to 16.

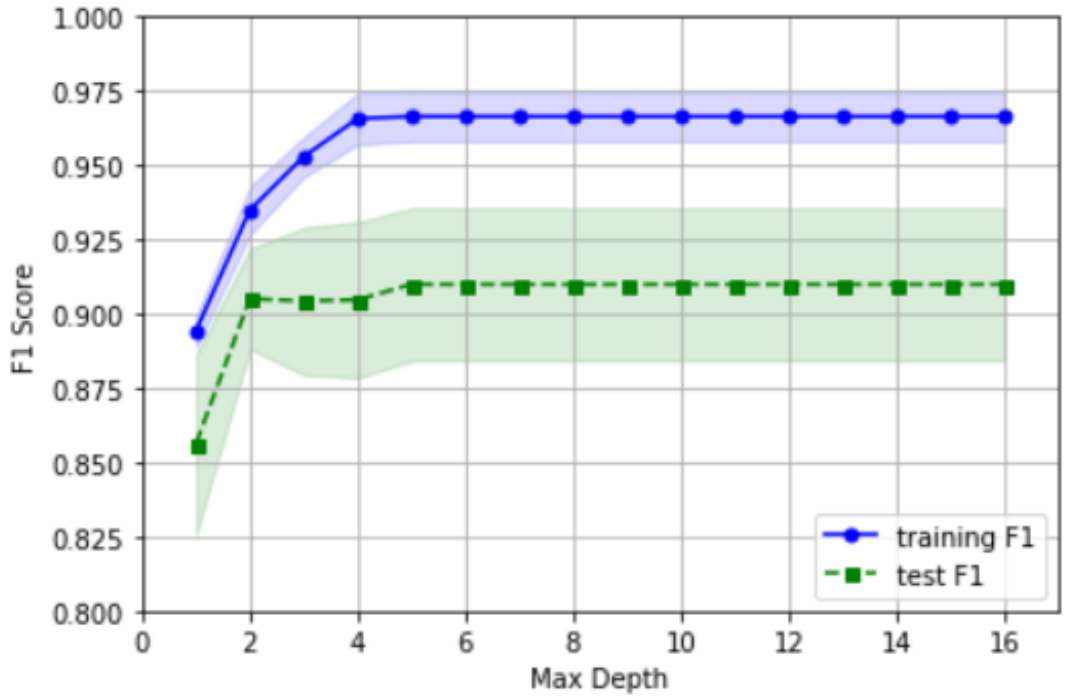
```
np.random.seed(42) #the seed used by the random number generator for np

################################ Parameters - Varying Complexity ################################

param_range = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

# Determine training and test scores for varying parameter values.
train_scores, test_scores = validation_curve(
                estimator=DecisionTreeClassifier(class_weight=None, criterion='gini',
                max_features=None, max_leaf_nodes=None,
                min_impurity_decrease=0.0, min_impurity_split=None,
                min_samples_leaf=5, min_samples_split=2,
                min_weight_fraction_leaf=0.0, presort=False, random_state=42,
                splitter='best'),
                X=X,
                y=y,
                param_name="max_depth",
                param_range=param_range,
                cv=5,
                scoring="f1",
                n_jobs=4)
```

Then, I plot the f1 score means of cross-validation for all the max_depth in the param_range and fill the area to indicate standard deviations.

```
# Cross validation statistics for training and testing
train_mean = np.mean(train_scores, axis=1) # Compute t
train_std = np.std(train_scores, axis=1)   # Compute t
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

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

# Plot train accuracy means of cross-validation for al
plt.plot(param_range, train_mean,
         color='blue', marker='o',
         markersize=5, label='training F1')

# Fill the area around the line to indicate the size o
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
plt.plot(param_range, test_mean,
         color='green', linestyle='--',
         marker='s', markersize=5,
         label='test F1')

# Fill the area around the line to indicate the size o
plt.fill_between(param_range,
                 test_mean + test_std,
                 test_mean - test_std,
                 alpha=0.15, color='green')
```

To better show the standard deviation area, use [0.8, 1.0] for ylim.

```
# Grid and Axes Titles
plt.grid()
plt.legend(loc='lower right')
plt.xscale('linear')
plt.xlabel('Max Depth')
plt.ylabel('F1 Score')
plt.ylim([0.8, 1.0])
plt.xlim([0,17])# y limits in
plt.tight_layout()
# plt.savefig('Fitting_graph_L
plt.show()            # Display
```



The final results agree with the previous GridSearchCV() result. The best performing max_depth for this model is 5. From max_depth = 1 to 2, the F1 scores for test data are low due to underfitting problem. As max_depth increases, the model gets more and more complex and peaks at max_depth = 5. However, since there aren't many nodes with high depth, the increase of max_depth after max_depth = 6 doesn't change the F1 score much. Meanwhile, the F1 scores for training data increases when max_depth moves from 1 to 5, and then remain stable for values bigger than 5. It can be concluded that holding other parameters the same, the best max_depth value for this decision tree model is 5 for its best test data performance.

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

The original dataset is not standardized. I use StandardScaler() to standardize the attributes and X_train & X_test. I then paste the optimal parameters of three previous best models to form a classifier list. After that, I estimate the AUC based on cross validation and finally plot ROC curve and calculate the AUC.

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

# Logistic Regression Classifier
clf1 = LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=42, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)

# Decision Tree Classifier
clf2 = DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=42,
            splitter='best')

# kNN Classifier
clf3 = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=1,
           weights='uniform')

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

```
print('5-fold cross validation:\n')
# Note: We are assuming here that the data is standardized. For the homework, you need to m
for clf, label in zip([clf1, clf2, clf3], clf_labels): #For all classifiers
    scores = cross_val_score(estimator=clf,   #Estimate AUC based on cross validation
                             X=X_std,
                             y=y,
                             cv=5,
                             scoring='roc_auc')
    print("ROC AUC: %0.2f (+/- %0.2f) [%s]" #Print peformance statistics based on cross-val
          % (scores.mean(), scores.std(), label))

#################################### Visualization #####################################

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):

    # 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 t
    fpr, tpr, thresholds = roc_curve(y_true=y_test, # Build ROC curve
                                     y_score=y_pred)
    roc_auc = auc(x=fpr, y=tpr)                # Compute Area Under the Curve (AUC)
    plt.plot(fpr, tpr,                         # Plot ROC Curve and create label with AUC v
             color=clr,
             linestyle=ls,
             label='%s (auc = %0.2f)' % (label, roc_auc))
```

The logistic regression model has AUC of 0.99. The decision tree model has AUC of 0.95. The KNN model has AUC of 0.97. From the graph and the 5-fold cross-va lidation, it can be concluded that logistic regression is better than the other two in this case. From the graph, logistic regression model is the best, because its curve captured the most space. Thus, using the information in the graph and 5-fold CV stats, logistic regression is the best model.

5-fold cross validation:

```
ROC AUC: 0.99 (+/- 0.01) [Logistic regression]
ROC AUC: 0.95 (+/- 0.02) [Decision tree]
ROC AUC: 0.97 (+/- 0.01) [kNN]
```