EMORY UNIVERSITY

Introduction to Business Data Analytics

Homework #3 **Due: Sunday, Oct 7 at 11.59 pm**

Kaitlin Wallis (Katie)

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

Total grade: _____ out of ___160___ points

1) (8 points) Assume that you built a model for predicting consumer credit ratings and evaluated it on the validation dataset of 5 records. Based the following 5 actual and predicted credit ratings (see table below), calculate the following performance metrics for your model: MAE, MAPE, RMSE, and Average error.

Actual Credit Rating	Predicted Credit Rating	Error	Absolute Error	Squared Error	Absolute Percent Error
670	710	40	40	1600	.0597
680	660	-20	20	400	.029411
550	600	50	50	2500	0.0909
740	800	60	60	3600	0.08108
700	600	-100	100	10000	0.1428
			54	18100	

MAE= mean absolute Error = 54

Mean Absolute Percent Error = Average(Actual – Predicted / Actual) =**0.45147/5 = 8.07%**

RMSE = sqrt(18100) = 60.166

Average Error = 6

2) (12 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.

	Positive	**Negative**
Positive	=$5-0.015 = **$4.985**	**-$0.015**
Negative	**$0**	**$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.

	Positive	**Negative**
Positive	**560**	**70**
Negative	**120**	**450**

$4.985/1200 * 560 – 70/1200* -0.015= $2.32

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

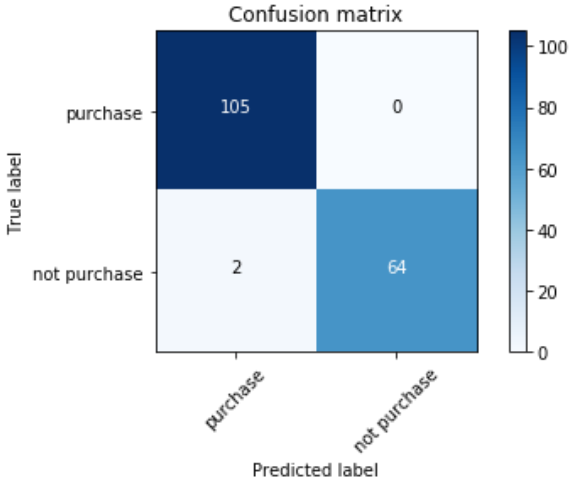
Please use the dataset on breast cancer research from this link: http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data [Note: For Python please read the data directly from the URL without downloading the file on your local disk.**] The description of the data and 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.**

a) **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. Present a brief overview of your predictive modeling process, explorations, and discuss your results. Make sure you present information about the model "goodness" (possible things to think about: confusion matrix, predictive accuracy, precision, recall, f-measure). Please provide screenshots of your code and explain the process you have followed.**

Data Exploration: I started by shuffling the data so that the instances would be in a random order. Then I looked to make sure that the data was appropriately balanced. Because the data was balanced, I did not need to resample the data. Next, I needed to transform the categorical variables to be n-1 dummy variables. In this dataset, the only categorical variable was the target variable, which I switched from M and B to be 0 and 1. Next, I took a look at the summary statistics for the different attributes, seeing nothing unusual, I moved onto splitting the data into training and test sets using Train Test Split from sklearn.

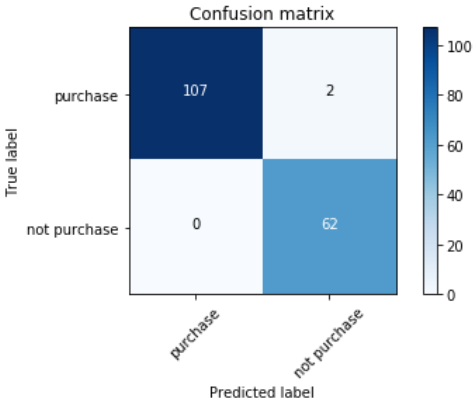
Model Building & Optimization: I wanted to optimize the parameters for all 3 different classifiers. I did this using GridSearchCV. I didn't want to just optimize for accuracy, however. Instead, I ran a for loop that ran the grid search so that it optimized for f1, precision, recall, accuracy, and auc and then examined the different confusion matrices for each result. By optimizing for a variety of different metrics, I knew that I'd be able to see find the best model. (See Code below for Decision Tree. The same process was followed for KNN and Logistic Regression. KNN Parameters: weights and n_neighbors, Logistic Regression Parameters: C and L1 vs L2 Norma penalty). For KNN the x values for both training and test were standardized. The final best method was the Logistic Regression Method. The attributes for the best method depends on the various costs of False Positives and False Negatives. If you run the GridSearchOptimization for accuracy, the model incorrectly classifies two points as purchase when they are not. When you run the optimization for Precision, two points are incorrectly classified as nor purchase when they were purchase. In either case, the model is able to be accurate up to 98.83% of the time on test data.

```
Logistic Regression F1 Optimization Accuracy:  0.9883040935672515
{'C': 100, 'penalty': 'l1'}
```



	precision	recall	f1-score	support
0	0.98	1.00	0.99	105
1	1.00	0.97	0.98	66
avg / total	0.99	0.99	0.99	171

Logistic Regression Precision Optimization Accuracy: 0.9883040935672515
{'C': 10000, 'penalty': 'l1'}



	precision	recall	f1-score	support
0	1.00	0.98	0.99	109
1	0.97	1.00	0.98	62
avg / total	0.99	0.99	0.99	171

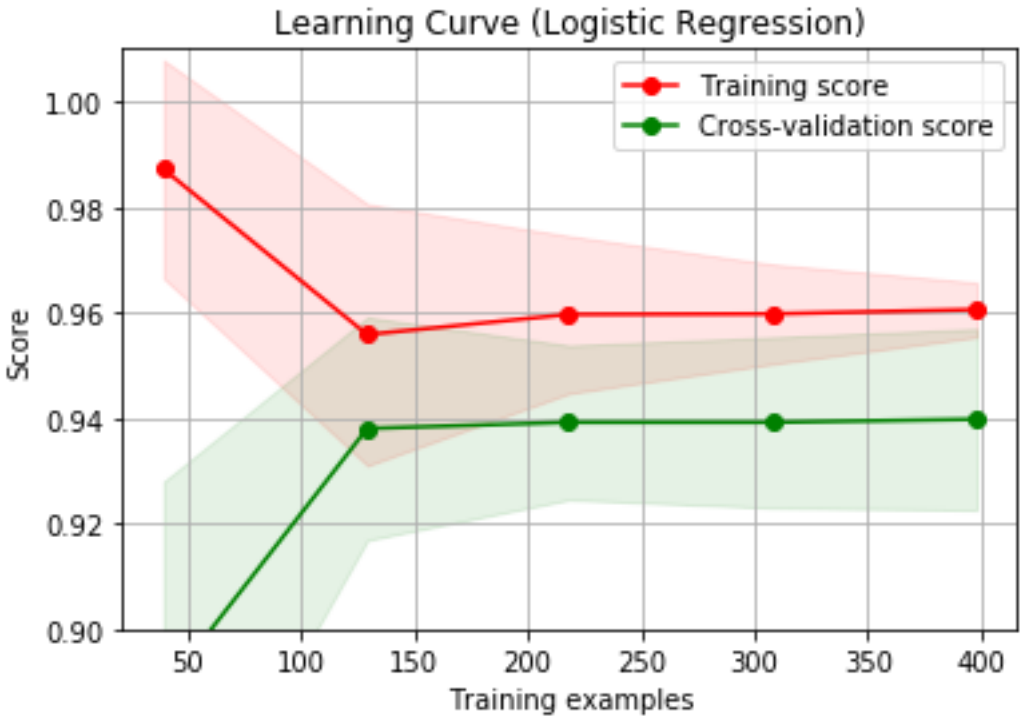
In [241]:

```
#Grid Search For Best Decision Tree
gs_DT_acc = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                  param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None],'min_samples_leaf':[1,2,3,4,5], 'criterion':["gini", "entr
                              'min_samples_split':[2,3,4,5]}],
                  scoring='accuracy', # Specifying multiple metrics for evaluation
                  cv=10)

gs_DT_acc = gs_DT_acc.fit(x_train,y_train)
gs_DT_acc_params = gs_DT_acc.best_params_


#Grid Search For Best Decision Tree for AUC
gs_DT_AUC = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                  param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None],'min_samples_leaf':[1,2,3,4,5],
                              'min_samples_split':[2,3,4,5]}],
                  scoring='roc_auc', # Specifying multiple metrics for evaluation
                  cv=10)

gs_DT_AUC = gs_DT_AUC.fit(x_train,y_train)
gs_DT_AUC_params = gs_DT_AUC.best_params_

#Grid Search For Best Decision Tree for Recall
gs_DT_Recall = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                  param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None],'min_samples_leaf':[1,2,3,4,5],
                              'min_samples_split':[2,3,4,5]}],
                  scoring='recall', # Specifying multiple metrics for evaluation
                  cv=10)

gs_DT_Recall = gs_DT_Recall.fit(x_train,y_train)
gs_DT_Recall_params = gs_DT_Recall.best_params_


#Grid Search For Best Decision Tree for F Measure
gs_DT_F = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                  param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None],'min_samples_leaf':[1,2,3,4,5],
                              'min_samples_split':[2,3,4,5]}],
                  scoring='f1', # Specifying multiple metrics for evaluation
                  cv=10)

gs_DT_F = gs_DT_F.fit(x_train,y_train)
gs_DT_F_params = gs_DT_F.best_params_


#Grid Search For Best Decision Tree for F Measure
gs_DT_P = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                  param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None],'min_samples_leaf':[1,2,3,4,5],
                              'min_samples_split':[2,3,4,5]}],
                  scoring='precision', # Specifying multiple metrics for evaluation
                  cv=10)

gs_DT_P = gs_DT_P.fit(x_train,y_train)
gs_DT_P_params = gs_DT_P.best_params_
```

In [275]:

```
models_DT = [gs_DT_F,gs_DT_acc,gs_DT_AUC,gs_DT_Recall, gs_DT_P]
model_names_DT = ["Decision Tree F1 Optimization","Decision Tree Accuracy Optimization","Decision Tree AUC Optimization","Decisio
y_pred_DT_F = gs_DT_F.predict(x_test)
y_pred_DT_acc = gs_DT_acc.predict(x_test)
y_pred_DT_AUC = gs_DT_AUC.predict(x_test)
y_pred_DT_Recall = gs_DT_Recall.predict(x_test)
y_pred_DT_P = gs_DT_P.predict(x_test)


y_preds_DT = [y_pred_DT_F,y_pred_DT_acc, y_pred_DT_AUC, y_pred_DT_Recall, y_pred_DT_P]

x = 0
for x in [0,1,2,3,4]:
    y_preds_DT[x]=  models_DT[x].predict(x_test)
    print(model_names_DT[x], "Accuracy: ", accuracy_score(y_preds_DT[x], y_test))
    plot_confusion_matrix(confusion_matrix(y_preds_DT[x], y_test), classes=['purchase','not purchase'])
    print(models_DT[x].best_params_)
    plt.show()
    print(classification_report(y_preds_DT[x], y_test))
```

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.



Here you can see that up to about 125 training examples, each additional datapoint has a high return on the accuracy score. After 125, however, gaining more data does not significantly help the accuracy. Based on this, I would not recommend that the company invest in more data.

Code:

```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a simple plot of the test and training learning curve.
```

Parameters

---

estimator : object type that implements the "fit" and "predict" methods

    An object of that type which is cloned for each validation.

title : string

    Title for the chart.

X : array-like, shape (n_samples, n_features)

    Training vector, where n_samples is the number of samples and

    n_features is the number of features.

y : array-like, shape (n_samples) or (n_samples, n_features), optional

    Target relative to X for classification or regression;

    None for unsupervised learning.

ylim : tuple, shape (ymin, ymax), optional

    Defines minimum and maximum yvalues plotted.

cv : int, cross-validation generator or an iterable, optional

    Determines the cross-validation splitting strategy.

    Possible inputs for cv are:

      - None, to use the default 3-fold cross-validation,

      - integer, to specify the number of folds.

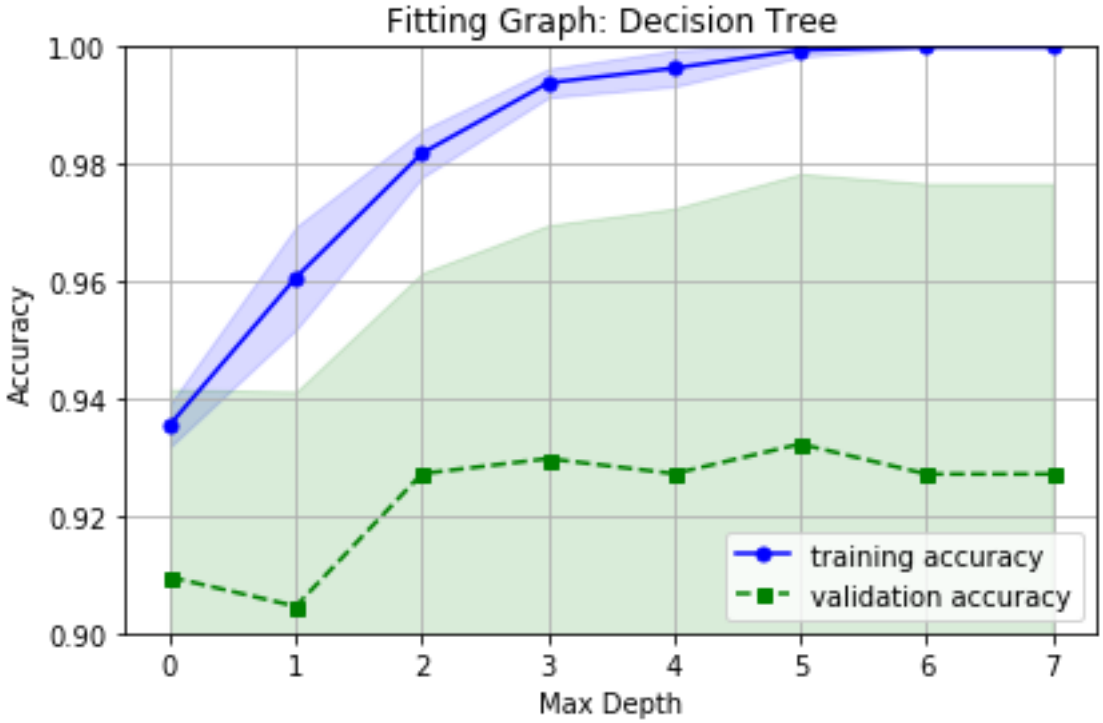
      - An object to be used as a cross-validation generator.

```
      - An iterable yielding train/test splits.

    For integer/None inputs, if ``y`` is binary or multiclass,
    :class:`StratifiedKFold` used. If the estimator is not a classifier
    or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

    Refer :ref:`User Guide <cross_validation>` for the various
    cross-validators that can be used here.

n_jobs : integer, optional
    Number of jobs to run in parallel (default 1).
"""
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="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
```

```
X = cleanedData.loc[:,cleanedData.columns.difference(["M"])]

y = cleanedData.loc[:, "M"]

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 include in the test split (float between 0.0 and 1.0)

# 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() # Build multiple LRs as we increase the size of the traning data

# Plots the learning curve based on the previously defined function for the logistic regression estimator

plot_learning_curve(estimator, title, X, y, (0.9, 1.01), cv=cv, n_jobs=4)


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

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.



Here you can see the fitting graph for Decision Tree. On the X axis, you see the maximum depth and on the y axis there is the accuracy. You can see as complexitiy increases, the training accuracy coninutes to increase but the validation accuracy plateaus with a highest accuracy when the decision tree has a maximum depth of 5.

Code for Fitting Graph:

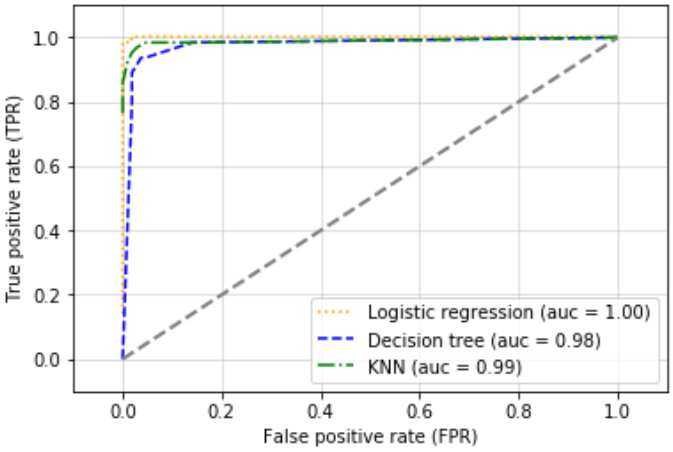
```
# Fitting curve (aka validation curve)
# Determine training and test scores for varying parameter values.
from sklearn.model_selection import validation_curve
# Split validation
from sklearn.model_selection import train_test_split
# Class for Logistic Regression classifier
from sklearn.tree import DecisionTreeClassifier

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

# Specify possible parameter values for C.
# Parameter C: Inverse of regularization strength;
# C must be a positive float; smaller values specify stronger regularization.
param_range = [1,2,3,4,5,6,7, None]

# Compute scores for an estimator with different values of a specified parameter.
# This is similar to grid search with one parameter.
# However, this will also compute training scores and is merely a utility for plotting the results.

# Determine training and test scores for varying parameter values.
train_scores, test_scores = validation_curve(
        estimator=DecisionTreeClassifier(random_state=42), #Build Logistic Regression Models
```

```
            X=x_train,
            y=y_train,
            param_name="max_depth", # parameter C: Inverse of regularization strength; must be a positive float. Smaller values of paramter C specify stronger regularization.
            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='ovr'". This parameter is ignored when the ``solver``is set to 'liblinear' regardless of whether 'multi_class' is specified or not. If given a value of -1, all cores are used.


# 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)

# Plot train accuracy means of cross-validation for all the parameters C in param_range
plt.plot([0,1,2,3,4,5,6,7], 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([0,1,2,3,4,5,6,7], train_mean + train_std,
                 train_mean - train_std, alpha=0.15,
                 color="blue")

# Plot test accuracy means of cross-validation for all the parameters C in param_range
plt.plot([0,1,2,3,4,5,6,7], 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([0,1,2,3,4,5,6,7],
                 test_mean + test_std,
                 test_mean - test_std,
                 alpha=0.15, color='green')

# Grid and Axes Titles
plt.title("Fitting Graph: Decision Tree")
plt.grid()
plt.xscale("linear")
plt.legend(loc='lower right')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.ylim([0.9, 1.0]) # y limits in the plot
plt.tight_layout()
# plt.savefig('Fitting_graph_LR.png', dpi=300)
```

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

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.

When comparing models, I also examined the ROC curve, which also confirmed that logistic regression was working the best at every threshold, making it the best model to use. However, Decision Tree and KNN also had very high scores.



Code for ROC Curve:

```
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

# Logistic Regression Classifier
clf_lr = LogisticRegression(penalty='l1',
                            C=100,
                            random_state=42)

# Decision Tree Classifier
clf_t = DecisionTreeClassifier(max_depth=4, min_samples_leaf = 5, min_samples_split = 2,
                               criterion='gini',
                               random_state=42)

# kNN Classifier
clf_knn = KNeighborsClassifier(n_neighbors=8,
                               weights='uniform')

# Label the classifiers
```

```
clf_labels = ['Logistic regression', 'Decision tree', 'KNN']
all_clf = [clf_lr, clf_t, clf_knn]

for clf, label, clr, ls in zip(all_clf,
            clf_labels, colors, linestyles):
   if clf == clf_knn:
      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, # 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 values
            color=clr,
            linestyle=ls,
            label='%s (auc = %0.2f)' % (label, roc_auc))
   else:
      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, # 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 values
            color=clr,
            linestyle=ls,
            label='%s (auc = %0.2f)' % (label, roc_auc))

#draw random classifier line
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()
```

4) (40 points) Use numeric prediction techniques to build a predictive model for the HW3.xlsx dataset. This dataset is provided on the course website and contains data about whether or not different consumers made a purchase in response to a test mailing of a certain catalog and, in case of a purchase, how much money each consumer spent. The data file has a brief description of all the attributes in a separate worksheet. Note that this dataset has two possible outcome variables: Purchase (0/1 value: whether or not the purchase was made) and Spending (numeric value: amount spent).

Codelist				
Var. #	**Variable Name**	**Description**	**Variable Type**	**Code Description**
1.	US	Is it a US address?	binary	1: yes 0: no
2 - 16	Source_*	Source catalog for the record	binary	1: yes 0: no
		(15 possible sources)		
17.	Freq.	Number of transactions in last year at source catalog	numeric	
18.	last_update_days_ago	How many days ago was last update to cust. record	numeric	
19.	1st_update_days_ago	How many days ago was 1st update to cust. record	numeric	
20.	Web_order	Customer placed at least 1 order via web	binary	1: yes 0: no
21.	Gender=mal	Customer is male	binary	1: yes 0: no
22.	Address_is_res	Address is a residence	binary	1: yes 0: no
23.	Purchase	Person made purchase in test mailing	binary	1: yes 0: no
24.	Spending	Amount spent by customer in test mailing ($)	numeric	

Use Python for this Exercise.

Data Exploration:

Before starting any of the modeling, I started by exploring the data calling "data.describe()" in order to get information on the mins, maxs, and means of each of the attribute looking to see if there were any values that were either missing or didn't make sense. Seeing none, I moved onto looking at the interactions between the attributes themselves. I saw graphed the scatter plots and a correlation matrix of the different attributes looking for different unusual interactions. I noticed that there was a high correlation between 1st_update_days and last_update_days. Not wanting to keep both attributes, I chose to drop "1st_update_days" because it had a higher correlation with the target variable, spending.

Data Splitting: Next, in keeping with best practices for data exploration. I split the data into training, validation and test sets using train test split to create a training and test set and then further split the training set into training and validation using train test split again.

Model Building & Evaluation:

I ran LinearRegression(), KNeighborsRegression(), Lasso() and Ridge() with default parameter values. The results from running the model on the "validation" set were as follows:

Model	**MAE**	**MSE**
Linear Regression	77.309	17189.662
Lasso	77.564	17028.597
Ridge	77.257	17173.073
Decision Tree Regression	86.002	29525.820
KNeighborsRegressor	121.819	38090.717

From there, I used GridSearchCV with scoring = “neg. mean squared error” in order to find the optimial parameters for each algorithm. Here I used cv = 5, a random split of 42, and various parameters for each model. For Linear Regression, I looked at Lasso and Ridge. I optimized alpha, getting an alpha of 0.1 for all. For KNN, I optimized weights and n_neighbors. Weights turned out to be best at “uniform” and n_neighbors was equal to 14. For Decision Tree, I found the best values were a max_depth of 5.

Model	**MAE**	**MSE**
Linear Regression	76.523	14200.716
Lasso	77.170	17139.557
Ridge	77.301	17187.729
Decision Tree Regression	73.284	18806.765
KNeighborsRegressor	114.736	35612.849

Looking at the validation performance of the models, the best model appeared to be the Decision Tree Model with a max depth of 5. The next best models are versions of Linear Regression followed by the worst performing model, KNN. Given this, I ran the Decision Tree model on the test data in order to discover an accurate estimate of generalization performance.

```
MSE train: 17915.847, Test: 20868.244
MAE train: 79.001, Test: 77.182
```

On the test data, the Decision Tree Regressor model provides an MSE score of 20,868.244 and an MAE score of 77.182. Looking at the mean of spending at 102. This is not a great performance. Further exploration could be done with attribute engineering and forward and backward selection to create a better performing model.