Introduction to Data Science

Homework 4

Student Name: Eric Niblock

Student Netid: ejn9259

In this assignment we will be looking at data generated by particle physicists to test whether machine learning can help classify whether certain particle decay experiments identify the presence of a Higgs Boson. One does not need to know anything about particle physics to do well here, but if you are curious, full feature and data descriptions can be found here:

- https://www.kaggle.com/c/higgs-boson/data (https://www.kaggle.com/c/higgs-boson/data)
- http://higgsml.lal.in2p3.fr/files/2014/04/documentation_v1.8.pdf (http://higgsml.lal.in2p3.fr/files/2014/04/documentation_v1.8.pdf)

The goal of this assignment is to learn to use cross-validation for model selection as well as bootstrapping for error estimation. We'll also use learning curve analysis to understand how well different algorithms make use of limited data. For more documentation on cross-validation with Python, you can consult the following:

- http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation (http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation)

Part 1: Data preparation (5 Points)

Create a data preparation and cleaning function that does the following:

- Has a single input that is a file name string
- Reads data (the data is comma separated, has a row header and the first column `EventID` is the index) into a pandas `dataframe`
- Cleans the data
  - Convert the feature `Label` to numeric (choose the minority class to be equal to 1)
    - Create a feature `Y` with numeric label
    - Drop the feature `Label`
  - If a feature has missing values (i.e., `-999`):
    - Create a dummy variable for the missing value
      - Call the variable `orig_var_name + _mv` where `orig_var_name` is the name of the actual var with a missing value
      - Give this new variable a 1 if the original variable is missing
    - Replace the missing value with the average of the feature (make sure to compute the mean on records where the value isn't missing). You may find pandas' `.replace()`

function useful.

- After the above is done, rescales the features so that each feature has zero mean and unit variance (hint: look up sklearn.preprocessing)
- Returns the cleaned and rescaled dataset

Grading guideline: if this function is done in more than 30 lines (not including empty lines), we will deduct 2 points.

In [2]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing

def cleanBosonData(infile_name):
    df = pd.read_csv(infile_name, index_col=0)
    df['Y'] = df['Label']
    df['Y'].replace({'s': 1, 'b': 0}, inplace=True)
    df = df.drop(columns = ['Label'])
    precols = df.columns
    for col in df.columns:
        if -999 in np.array(df[col]):
            df[col+'_mv'] = df[col]
            df.loc[df[col+'_mv'] == -999, col+'_mv'] = 1
            df.loc[df[col+'_mv'] != 1, col+'_mv'] = 0
            adj_mean = np.matmul((1-np.array(df[col+'_mv'])).T,np.array(df[col]))
            df.loc[df[col] == -999, col] = adj_mean
    df[list(precols)[0:-1]] = preprocessing.scale(df[list(precols)[0:-1]].to_nump
    data_clean = df[list(precols)]
    return data_clean
```

Part 2: Basic evaluations (5 Points)

In this part you will build an out-of-the box logistic regression (LR) model and support vector machine (SVM). You will then plot ROC for the LR and SVM model.

1. Clean the two data files included in this assignment (`data/boson_training_cut_2000.csv` and `data/boson_testing_cut.csv`) by using the function defined above, and use them as training and testing data sets.

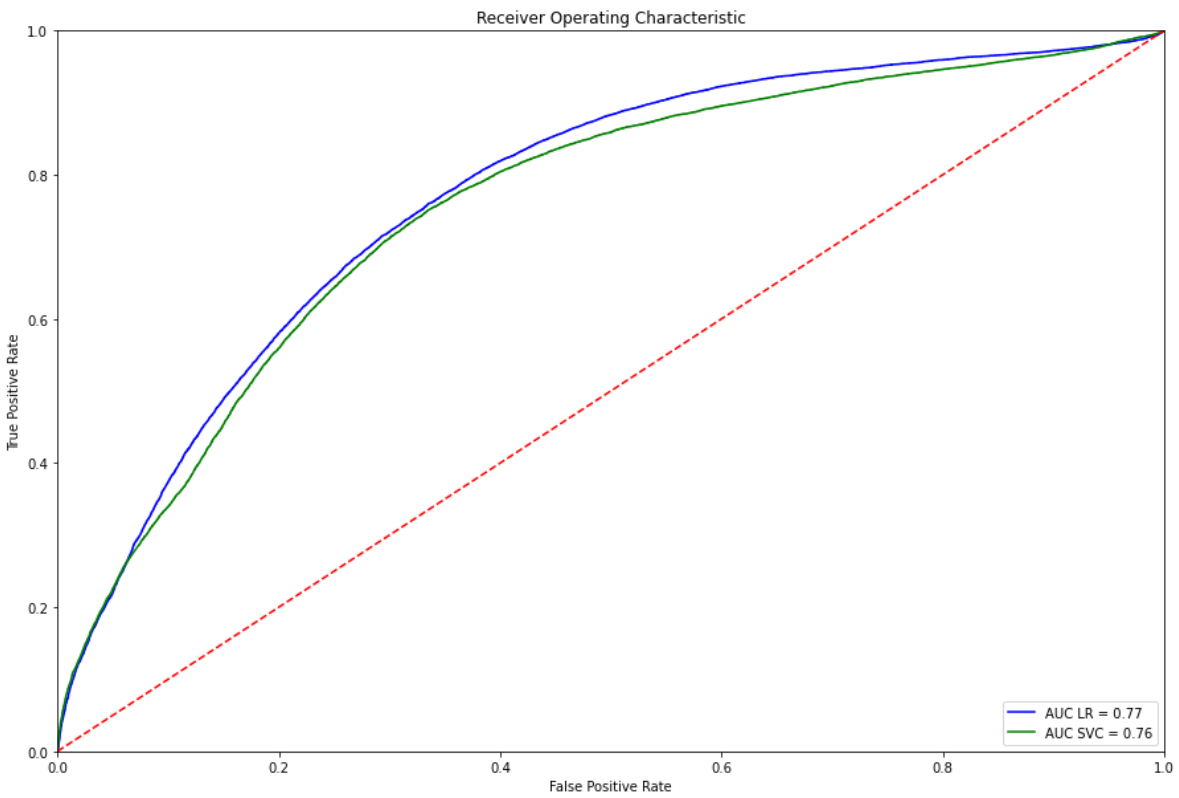
In [3]:
```
data_train = cleanBosonData(r'boson_training_cut_2000.csv')
data_test = cleanBosonData(r'boson_testing_cut.csv')
```

2. On the training set, build the following models:

- A logistic regression using sklearn's `linear_model.LogisticRegression()` . For this model, use `C=1e30` .
- An SVM using sklearn's `svm.svc()` . For this model, specify that `kernel="linear"` .

For each model above, plot the ROC curve of both models on the same plot. Make sure to use the test set for computing and plotting. In the legend, also print out the Area Under the ROC (AUC) for reference.

In [5]:
```
import matplotlib
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.svm import SVC
%matplotlib inline

X_train = data_train.to_numpy()[:,0:-1]
y_train = data_train.to_numpy()[:,-1]

X_test = data_test.to_numpy()[:,0:-1]
y_test = data_test.to_numpy()[:,-1]


clf = LogisticRegression(C=1e30).fit(X_train, y_train)
y_pred_prob = clf.predict_proba(X_test)[:,1]

fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred_prob)
roc_auc = metrics.auc(fpr, tpr)



clf2 = SVC(kernel='linear',probability=True).fit(X_train, y_train)
y_pred_prob2 = clf2.predict_proba(X_test)[:,1]

fpr2, tpr2, threshold2 = metrics.roc_curve(y_test, y_pred_prob2)
roc_auc2 = metrics.auc(fpr2, tpr2)



plt.figure(figsize = (15,10))
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC LR = %0.2f' % roc_auc)
plt.plot(fpr2, tpr2, 'g', label = 'AUC SVC = %0.2f' % roc_auc2)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



3. Which of the two models is generally better at ranking the test set? Are there any classification thresholds where the model identified above as "better" would underperform the other in a classification metric (such as TPR)?

--------ANSWER---------

The linear regression model is generally superior to the SVC model in terms of AUC. Ideally, the curves should push closest to the upper left corner, hence enclosing the most area, and approaching a true positive rate of one while having a false positive rate of zero. The AUC (area under the curve) is a metric which helps describe this event (higher area would mean being pushed closer to that point). Here the linear regression model is above the SVC model at almost every point on the graph, thereby generating this greater AUC.

It is possible that the linear regression model might underpreform in terms of the false negative rate, which is not explicitly shown in an ROC graph.

--------ANSWER---------

Part 3: Model selection with cross-validation (7 Points)

We think we might be able to improve the performance of the SVM if we perform a grid search on the hyper-parameter C. Because we only have 2000 instances, we will have to use cross-validation to find the optimal C.

1. Write a cross-validation function that does the following:

- Takes as inputs a dataset, a label name, # of splits/folds (`k`), a sequence of values for C (`cs`)
- Use sklearn.cross_validation.KFold to map each instance to a fold
- Performs two loops
  - Outer Loop: `for each fold in range(k)` :
    - Splits the data into `cv_train & cv_validate` according to previously defined fold mappings
  - Inner Loop: `for each c in cs` :
    - Trains an SVM on training split with `C=c, kernel="linear"`
    - Computes AUC_c_k on validation data
    - Stores AUC_c_k in a dictionary of values
- Returns a dictionary, where each key-value pair is: `c:[auc_c_1,auc_c_2,..auc_c_k]` (i.e., for each c, we want a list full of auc's from each fold)

Note: Use Sklearn's KFold method, but do not use any other cross-validation convenience function. The goal is to learn how to implement the algorithm yourself!

Grading guideline: if this function is done in more than 30 lines (not including empty lines), we will deduct 2 points.

In [6]:
```
from sklearn.model_selection import KFold

def xValSVM(dataset, label_name, k, cs):
    fold = KFold(k, shuffle=False)
    aucs = {}
    for c in cs:
        aucs[c] = []

    y_train = dataset[label_name].to_numpy()
    X_train = dataset.drop(columns=[label_name]).to_numpy()

    for train_index, val_index in fold.split(X_train):
        Xk_train, yk_train = X_train[train_index], y_train[train_index]
        Xk_val, yk_val = X_train[val_index], y_train[val_index]
        for c in cs:
            clf = SVC(kernel='linear',probability=True, C=c).fit(Xk_train, yk_tra
            y_pred_prob = clf.predict_proba(Xk_val)[:,1]

            fpr, tpr, threshold = metrics.roc_curve(yk_val, y_pred_prob)
            roc_auc = metrics.auc(fpr, tpr)
            aucs[c].append(roc_auc)

    return aucs
```

2. Using the function written above, do the following:

- Generate a sequence of 10 C values in the interval `[10^(-8), ..., 10^1]` (i.e., do all powers of 10 from -8 to 1, inclusive).

2. Call aucs = xValSVM(data_train, 'Y', 10, cs)
3. For each c in cs, get mean(AUC) and StdErr(AUC) (don't forget, standard error of the mean of X is sqrt(Var(X)/N)

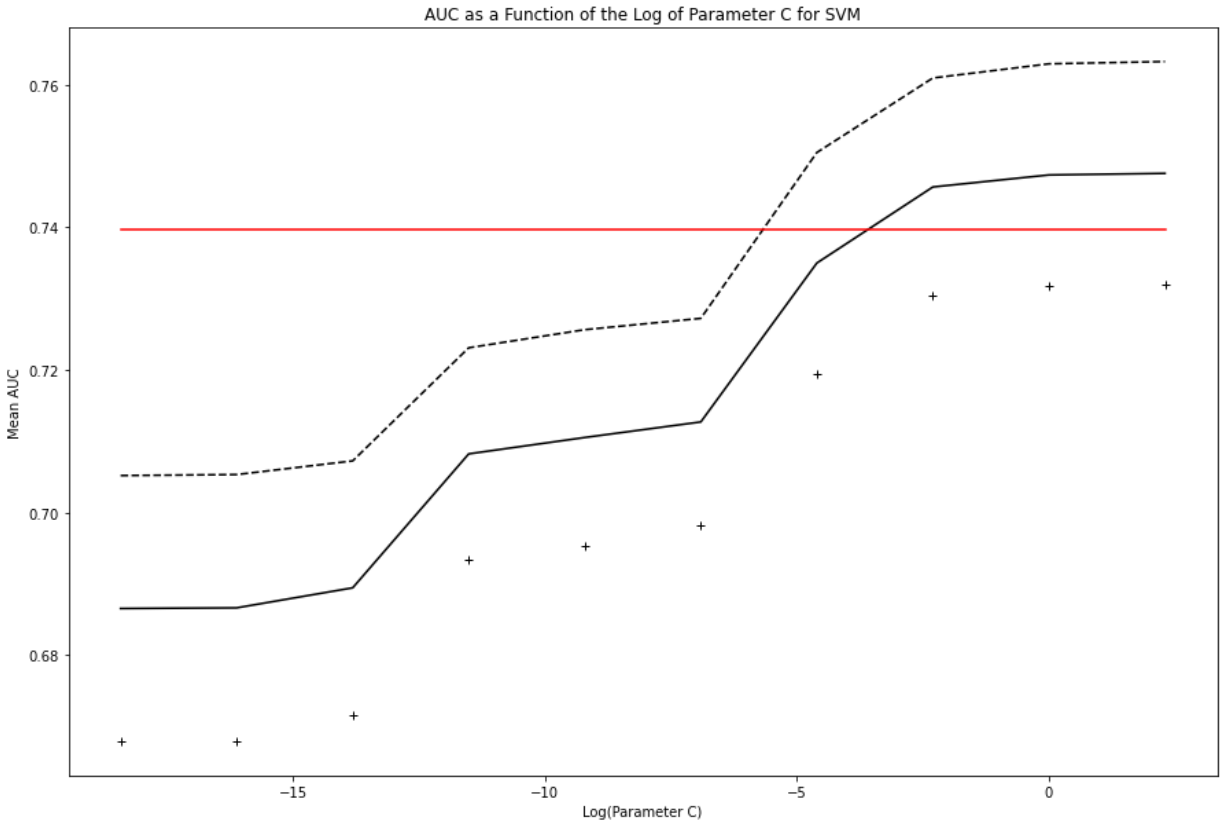
4. Compute the value for max_1std = (mean(AUC)-StdErr(AUC)) associated with the c having max(mean(AUC)). I.e., part of what we have been calling the '1 standard error rule'.
5. Generate a plot with the following:
    - Log10(c) on the x-axis
    - 1 series with mean(AUC) for each c
    - 1 series with mean(AUC)-2*stderr(AUC) for each c (use ‘k+’ as color pattern)
    - 1 series with mean(AUC)+2*stderr(AUC) for each c (use ‘k--‘ as color pattern)
    - a reference line for max_1std (use ‘r’ as color pattern)

Then answer the question: Did the model parameters selected beat the out-of-the-box model for SVM?

In [7]:
```
aucs = xValSVM(data_train, 'Y', 10, [10**(i) for i in range(-8,2)])
```

In [13]:
```
cs = [10**(i) for i in range(-8,2)]
means = [np.mean(aucs[c]) for c in cs]
stderrs = [(np.var(aucs[c])/10)**0.5 for c in cs]
```

In [18]:
```
plt.figure(figsize = (15,10))
plt.title('AUC as a Function of the Log of Parameter C for SVM')
plt.xlabel('Log(Parameter C)')
plt.ylabel('Mean AUC')
plt.plot(np.log(cs), np.array(means)-2*np.array(stderrs), 'k+')
plt.plot(np.log(cs), np.array(means), 'k')
plt.plot(np.log(cs), np.array(means)+2*np.array(stderrs), 'k--')
plt.plot(np.log(cs), [means[-1]-stderrs[-1]]*10, 'r')
plt.show()
```



--------ANSWER---------

Essentially, no. It appears that all of the parameters tested yield the same or worse results than the out-of-the-box model for SVM. If we use the one standard error rule, then we select C=0.1 as the parameter, with an AUC of approximately 0.74

--------ANSWER---------

Part 4: Learning Curve with Bootstrapping (8 Points)

In this HW we are trying to find the best linear model to predict if a record represents the Higgs Boson. One of the drivers of the performance of a model is the sample size of the training set. As a data scientist, sometimes you have to decide if you have enough data or if you should invest in more. We can use learning curve analysis to determine if we have reached a performance plateau. This will inform us on whether or not we should invest in more data (in this case it would be by running more experiments).

Given a training set of size N, we test the performance of a model trained on a subsample of size N_i, where $N_i <= N$. We can plot how performance grows as we move N_i from 0 to N.

Because of the inherent randomness of subsamples of size N_i, we should expect that any single sample of size N_i might not be representative of an algorithm's performance at a given training set size. To quantify this variance and get a better generalization, we will also use bootstrap analysis. In bootstrap analysis, we pull multiple samples of size N_i, build a model, evaluate on a test set, and then take an average and standard error of the results.

An example of using bootstrapping to build a learning curve can be found here: https://github.com/briandalessandro/DataScienceCourse/blob/master/ipython/python35/Lecture_ERM (https://github.com/briandalessandro/DataScienceCourse/blob/master/ipython/python35/Lecture_ER

1. Create a bootstrap function that can do the following:

def modBootstrapper(train, test, nruns, sampsize, lr, c):

- Takes as input:
  - A master training file (train)
  - A master testing file (test)
  - Number of bootstrap iterations (nruns)
  - Size of a bootstrap sample (sampsize)
  - An indicator variable to specific LR or SVM (lr=1)
  - A c option (only applicable to SVM)
- Runs a loop with (nruns) iterations, and within each loop:
  - Sample (sampsize) instances from train, with replacement
  - Fit either an SVM or LR (depending on options specified).
  - Computes AUC on test data using predictions from model in above step
  - Stores the AUC in a list
- Returns the mean(AUC) and Standard Deviation(AUC) across all bootstrap samples. Note: the standard error of the mean AUC is really the standard deviation of the bootstrapped distribution, so just use np.sqrt(np.var(...))

In [15]:
```
# Code here
def modBootstrapper(train, test, nruns, sampsize, lr, c):

    X_test = test.to_numpy()[:,0:-1]
    y_test = test.to_numpy()[:,-1]

    if lr == 1:

        all_train = train.to_numpy()
        aucs_list = []

        for run in range(nruns):

            np.random.shuffle(all_train)
            samp_train = all_train[0:sampsize,:]

            X_train = samp_train[:,0:-1]
            y_train = samp_train[:,-1]

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

            fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred_prob)
            roc_auc = metrics.auc(fpr, tpr)
            aucs_list.append(roc_auc)

        return((np.mean(aucs_list), np.var(aucs_list)**0.5))

    if lr == 0:

        all_train = train.to_numpy()
        aucs_list = []

        for run in range(nruns):

            np.random.shuffle(all_train)
            samp_train = all_train[0:sampsize,:]

            X_train = samp_train[:,0:-1]
            y_train = samp_train[:,-1]

            clf = SVC(kernel='linear',probability=True, C=c).fit(X_train, y_trair
            y_pred_prob = clf.predict_proba(X_test)[:,1]

            fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred_prob)
            roc_auc = metrics.auc(fpr, tpr)
            aucs_list.append(roc_auc)

        return((np.mean(aucs_list), np.var(aucs_list)**0.5))
```

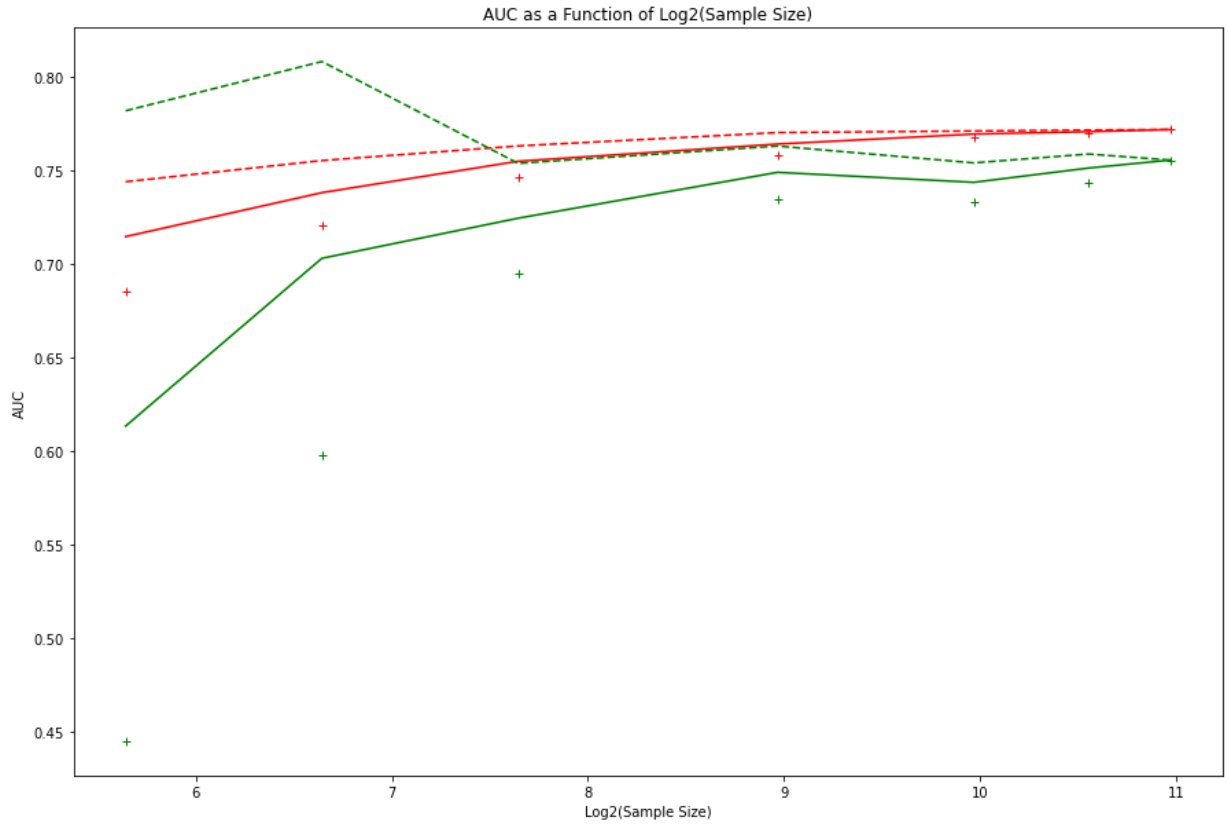
2. For both LR and SVM, run 20 bootstrap samples for each samplesize in the following list: samplesizes = [50, 100, 200, 500, 1000, 1500, 2000]. (Note, this might take 10-15 mins … feel free to go grab a drink or watch Youtube while this runs). For SVM, use the value of C identified using the 1 standard error method from part 3. For LR, use the default C.

Generate a plot with the following:

- Log2(samplesize) on the x-axis
- 2 sets of results lines, one for LR and one for SVM, the set should include
  - 1 series with mean(AUC) for each sampsize (use the color options ‘g’ for svm, ‘r’ for lr)
  - 1 series with mean(AUC)-stderr(AUC) for each samp size (use ‘+’ as color pattern, ‘g’,’r’ for SVM, LR respectively)
  - 1 series with mean(AUC)+stderr(AUC) for each samp size (use ‘--‘ as color pattern ‘g’,’r’ for SVM, LR respectively)

In [16]:
```
lr_auc = []
svm_auc = []

for samp in [50, 100, 200, 500, 1000, 1500, 2000]:
    lr_auc.append(modBootstrapper(data_train, data_test, 20, samp, 1, 0))

for samp in [50, 100, 200, 500, 1000, 1500, 2000]:
    svm_auc.append(modBootstrapper(data_train, data_test, 20, samp, 0, 0.1))
```

In [20]:
```
plt.figure(figsize = (15,10))
plt.title('AUC as a Function of Log2(Sample Size)')
plt.plot(np.log2([50, 100, 200, 500, 1000, 1500, 2000]), [i[0] for i in lr_auc],
plt.plot(np.log2([50, 100, 200, 500, 1000, 1500, 2000]), [i[0]-i[1] for i in lr_a
plt.plot(np.log2([50, 100, 200, 500, 1000, 1500, 2000]), [i[0]+i[1] for i in lr_a
plt.plot(np.log2([50, 100, 200, 500, 1000, 1500, 2000]), [i[0] for i in svm_auc],
plt.plot(np.log2([50, 100, 200, 500, 1000, 1500, 2000]), [i[0]-i[1] for i in svm_
plt.plot(np.log2([50, 100, 200, 500, 1000, 1500, 2000]), [i[0]+i[1] for i in svm_
plt.xlabel('Log2(Sample Size)')
plt.ylabel('AUC')

plt.show()
```



3. Which of the two algorithms are more suitable for smaller sample sizes, given the set of features? If it costs twice the investment to run enough experiments to double the data, do you think it is a worthy investment?

---------ANSWER---------

Lower amount of data corresponds to the region of the graph where log2(Sample Size) is low - lets observe the region below log2(Sample Size) = 8. It is clear that the linear regression model out preforms SVM in this region, since is consistently yields a higher AUC. Furthermore, the standard error of the linear regression model is much lower than that of SVM. This seems to suggest that not only is the linear regression model better preforming in terms of AUC, but that it is also more stable.

In regards to investing in more data, it is unlikely that we would see many improvements in terms of our classification capabilities. The AUC has essentially flat-lined as we approach the greater sample sizes - therefore the accuracy of the model cannot really be improved with more data,

though some small gains may be achieved. However, if we needed to double the investment in order to double the data, this trade off would probably be not worth our while.

---------ANSWER---------

4. Is there a reason why cross-validation might be biased? If so, in what direction is it biased? (Hint: refer to ESL figure 7.8)?

---------ANSWER---------

It is possible that cross-validation might be biased. If the error associated with our model decreases as our sample size increases, then we note that selecting smaller sample sizes tends to bias the model since we underestimate the actual performance of the model. However, as suggested in The Elements of Statistical Learning, we tend to note a diminishing rate of return in terms of accuracy as our sample size increases, so using a sample size that is close to the entirety of the training set would produce only a small amount of bias. Therefore, as an estimate of bias, cross-validation would be biased upward.

---------ANSWER---------