1a. In this problem, we consider 5 different classifiers on three different data sets. Based on the data we can tell that in dataset A all 5 classifiers perform at a roughly similar accuracy in the 0.95-0.97 range. For dataset B 4 of the 5 classifiers perform well. Logistic regression struggles to classify this data with a mere accuracy of 0.46 whereas all of the other classifiers lie in a range of 0.93-0.97. For dataset C all of the classifiers perform fairly well with Naive Bayes and Logistic Regression performing a bit worse(0.88) than the other 3 classifiers where we are between 0.94 and 0.97.

1b. The various classifiers work differently on different types of data. Logistic regression works by classifying data with a single decision surface and works well when the data is easily linearly separable by some arbitrary decision boundary. Naive Bayes works by treating each feature in the dataset independently and using bayes theorem to calculate the probability of classifying a data point and then making a binary decision based on that. SVM works by creating a hyper-plane(decision boundary) with a margin(connecting support vectors above and below the hyper-plane) and classifying the data based on the margin and the distance away from it. A decision tree works by creating a tree that selects the path that gains the most info/entropy until it has classified fully and the leaves are pure unless otherwise specified by the max depth of the tree. Random forest works by creating several different decision trees(specifically set in a parameter) that work on subsets of data; classification is done based on the averages of the classification of each of these subsets. For example, in dataset A we have a very high accuracy for all 5 classifiers. This makes intuitive sense because the data is linearly separable meaning that the classes of patterns can be separated with a single decision surface which in this case is a line. Each of the classifiers can do this fairly easily and create a decision boundary to classify the data separately given all of the methods they use listed above. In data B this data is non-linearly separable because it cannot be separated by a single decision surface(in this dimension anyways) so logistic regression will perform the worst and the rest will perform well because they use a method other than simple linear/logistic separability to classify the data. The other classification methods are still able to achieve high accuracy because they don't just try to separate the data based on a line/log equation like logistic regression does. They use independent probabilities, info scores, margins, or several different decision trees sub-classifications to classify the data. In data c, this data is still fairly separable when you use a logistic function to separate the data. This explains why the accuracy scores are so high for each of the classifiers. Each of these classification methods works extremely well when the data is easily separable by a decision boundary. Note that for logistic regression we have a lower accuracy score than most other classifiers because a logistic function doesn't fully split the data well enough in comparison to other classification methods. All 3 data-sets are still separable by a linear/logistic function however we need to consider that for dataset b we would need to perform a kernel trick to map this onto a third dimension to separate the data.

2a1.

Classifier	Mean	Std.dev
Logistic Regression	0.852	0.06
Naive Bayes	0.782	0.052
SVM(SVC)	0.542	0.004
Decision Tree	0.786	0.051
Random Forest	0.852	0.056
MLPClassifier	0.692	0.046
Extra Trees Classifier	0.836	0.048

2a2. Given the following datasets, I would argue that Random Forest would be the best model given that it has the highest mean along with a smaller standard deviation than other options. I started off with the main 3 contenders, Logistic Regression, Random Forest, and Extra Trees Classifier. We can already eliminate Extra Trees Classifier because it has a lower mean than both Logistic Regression and Random forest. Even though it has a lower standard deviation than both it still would not quite reach the higher mean auroc score that both the logistic and random forest have when doing mean +- standard deviation on each of the models. Last, we have to compare the logistic regression with the random forest. Both of these classifiers have identical mean values although Random forest has a marginally lower Standard Deviation making it the best model. Standard deviation is defined is a measure of the amount of variance in a data-set. A good classifier will minimize the amount of variance that exists with respect to our model which is why Random Forest wins.

2a3. The first classifier I used was the MLP classifier. This classifier works by using a feedforward artificial neural network which maps input data(features) onto outputs(labels). It is made up of more than one perceptron which allows us to have multiple layers/features. Each feature would be considered a layer and in each layer we have a number to neurons that fire when we consider the impact of a feature on the label(prediction). Rather than just have one perceptron(feature) make the prediction we use several each with their own set of neurons just like a neural network. The second classifier I used was the Extra Trees Classifier. This classifier works by using the same decision tree classifiers that the random forest uses except now we test random splits on “extra trees” over a few features allowing us to test all possible splits for the “extra trees”. The extra trees classifier can sometimes be better than a random forest classification since it can generalize better but this is heavily dependent on parameters such as n_estimators, max_features, and min_samples_split.

2b1. For hyperparameter selection I used the library Grid-Search to find my optimal parameters in both Random Forest and SVM. Grid search is a method that allows us to quickly iterate through several hyperparameter combinations to find our optimal ones. We can perform Grid search with k fold cross validation by specifying the number of k folds we desire. We use this method because rather than performing an exhaustive search of messing with one parameter and another in an attempt to find a higher mean validation score/lower standard deviation we can simply loop through every possible parameter choice with the different values WE PROVIDE each parameter. At the end of grid search we are given the parameter choices that give us the highest mean validation score along with the lowest standard deviation score for that specific mean score. In Grid Search, there are many functions we can use to fiddle with the hyper-parameters whether that is grid_search.best_params_ which provides us with the best parameters for our classifier when we give it values for each parameter to choose from and loop through to find the best combination or grid_search.best_estimators_ which simply takes the data and the classifier and gives us the best possible combinations of ALL parameters for that classifier. For the latter function we do not need to provide any values as the function will identify these values with its own looping and search mechanisms. In my code, I used grid_search.best_params_ as I provided several different parameter values for c, gamma for SVC and max_depth, n_estimators for Random Forest. The Grid search looped through all my possible parameter values provided in each classifier to find the best combination for each classifier.

2b2. Our first parameter is the c parameter in SVM. C allows us to set a margin based on how much you want to avoid misclassifying a training example. When we set c to a really low value we will have a larger margin with respect to our hyper-plane on a SVM which would in turn misclassify points. On the other hand, if we have a large c value, we will have a smaller margin with respect to our hyper-plane which would in turn classify more points correctly since the margin is quite small. See the below diagram for a visual example of how c affects the size of our margin with respect to the hyper-plane. A small c can underfit the data not classifying well enough whereas a large c can overfit the data and classify everything separately. Our next parameter is gamma in SVM. Gamma is defined as how far the influence of a training example reaches. In other words, we are trying to weight points based on the value of gamma and our decision boundary. If we have a high hamma value, our decision boundary(hyperplane) gives higher weight to nearer points causing a curvy decision boundary. In this case we will have a high bias because we are being biased(weighted towards only the points nearby but a much lower variance because we are only considering mostly the nearby points as they are given a higher weight. If we have a low gamma value we give the points that are further away more weight which causes a more straight line decision boundary. In this case we will have a low bias because we are considering all data points both near and far from the decision boundary(though further away points have higher weight) but high variance since we have more data to deal with along with weighting to further away point. All in all we want to make both gamma and c optimal so we don't overfit or underfit along with normal variance/bias. Now let's move onto Random Forest. The first parameter is max_depth which is defined as the maximum depth of the tree with respect to the initial forest. When we do random forest classifier we use several different

decision trees(n_estimators) which we will talk about in a second. Each of these decision trees has a depth at which they have done a full classification of the data. With our max_depth parameter we can simply set this depth to a certain value or to none(which allows the tree to expand until there are no more expansions able to be done and we have fully classified everything as well as possible). Most of the time our optimal max_depth_value should be none so that we can expand the tree to full evaluation. Our next parameter is n_estimators which is defined as the number of decision trees in our random forest classification. When we perform random forest classification, we are essentially taking sub-samples of our data and making several different decision trees(Number defined by n_estimators) classify these subsets of data and then average the accuracy of classification. As we increase n_estimators, we create more subsamples of the data and also create more decision trees making it harder to classify data as n_estimators increases more and more. On the other hand, if we decrease n_estimators we make it more difficult to classify all the data correctly because we just have one or a few big sub samples making it not much better than just using a decision tree classifier.

2b3. Yes, I was able to improve on my best model from 2a2 with hyper-parameter tuning. Initially, for my best model of Random Forest Classification I had a mean auroc score of the 10 folds of 0.852 with a standard deviation of 0.056. After messing with the n_estimators and max_depth using grid_search I was able to increase my mean validation (auroc) score to 0.872. Although the difference may seem somewhat marginal, I decided to see if my other parameter tuning on SVC did anything. Initially I had a mean auroc score of 0.542. After performing grid-search on both c and gamma for svc I received a mean auroc score of 0.802 with a standard deviation of 0.063. This is a significant increase from the initial mean auroc score showing that this hyper-parameter tuning is definitely doing something.

2c1.

N = 500	Actual Yes	Actual No
Predicted yes	271	0
Predicted No	1	228

Precision: 0.996
Recall: 1.0
Accuracy:0.998
Note: I read the addendum on piazza that said that we must use the full data-set to train the model and we predict on that same dataset which accounts for my highly accurate confusion matrix. If this is not the case in my code commented out there is use of cross_val_predict to find y_pred for the confusion matrix when using cross validation. I did not include this as we are training and testing on the same full dataset but if needed can also be done with cross validation. Just uncomment the lines that have the cross_val_predict function and y_pred and substitute y_pred into the second argument of conf_mat arguments.

2c2. Given that we are training and testing on the full data-set I cannot say much about the accuracy of my model without performing cross validation on the data. Right now, it may appear that my model is highly accurate, however, that is simply because we are using the same training data as testing data. Since the model has already been exposed to this it is more likely to classify all of the points correctly when in actuality if we gave it an arbitrary data set it may have an accuracy similar to the mean auroc score we found above after hyper-parameter tuning of 0.872 for the random forest classifier(my current best model).

2d1.Best model: RandomForest(n_estimators = 50 , max_depth= None)