Problem 1

Problem 1 (15 points) Consider the AdaBoost algorithm we discussed in the class [1]. AdaBoost is an example of ensemble classifiers where the weights in next round are decided based on the training error of the weak classifier learned on the current weighted training set. We wish to run the AdaBoost on the dataset provided in Table 1.

Instance	**Color**	**Size**	**Shape**	**Edible?**
D1	Yellow	Small	Round	Yes
D2	Yellow	Small	Round	No
D3	Green	Small	Irregular	Yes
D4	Green	Large	Irregular	No
D5	Yellow	Large	Round	Yes
D6	Yellow	Small	Round	Yes
D7	Yellow	Small	Round	Yes
D8	Yellow	Small	Round	Yes
D9	Green	Small	Round	No
D10	Yellow	Large	Round	No
D11	Yellow	Large	Round	Yes
D12	Yellow	Large	Round	No
D13	Yellow	Large	Round	No
D14	Yellow	Large	Round	No
D15	Yellow	Small	Irregular	Yes
D16	Yellow	Large	Irregular	Yes

Table 1: Mushroom data with 16 instances, three categorical features, and binary labels.

a) Assume we choose the following decision stump f_1 (a shallow tree with a single decision node), as the first predictor (i.e., when training instances are weighted uniformly):

```
if(Color is Yellow):
      predict  Eadible = Yes
else:
      predict Eadible = No
```

What would be the weight of f_1 in final ensemble classifier (i.e., α_1 in $f(\boldsymbol{x}) = \sum_{i=1}^{K} \alpha_i f_i(\boldsymbol{x})$)?

First step is to set the weight vector equal to:

$$w^{(0)} = \left[\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right] = \left[\frac{1}{16}, \frac{1}{16}, \ldots, \frac{1}{16}\right]$$

The formula to calculate the error is:

$$\epsilon_k = \sum_{i=1}^{n} w_i^{(k-1)} \mathbb{I}[y_i \neq f_k(\boldsymbol{x}_i)]$$

Which means that where ever there is an error (if green = yes or yellow = no), add it to the error.

Since there are 6 errors out of the 16 instances,

$$\epsilon_1 = \frac{6}{16}$$

The weight in final ensemble classifier is given by:

$$\alpha_1 = \frac{1}{2}\log\left(\frac{1-\epsilon_1}{\epsilon_1}\right)$$

$$= \frac{1}{2}\log\left(\frac{\frac{10}{16}}{\frac{6}{16}}\right) = \frac{1}{2}\log\left(\frac{10}{6}\right) = 0.2554128119$$

b) After computing f_1, we proceed to next round of AdaBoost. We begin by recomputing data weights depending on the error of f_1 and whether a point was (mis)classified by f_1. What is the weight of each instance in second boosting iteration, i.e., after the points have been re-weighted? Please note that the weights across the training set are to be uniformly

Now we have to adjust the weights.

$$w_i^{(k)} = \frac{1}{Z} w_i^{(k-1)} \exp\left(-\alpha_k y_i f_k(x_i)\right)$$

If the data is classified, then it would be:

$$w_i^{(k)} = \frac{1}{Z}\frac{1}{16}\exp(-0.2554128119) = 0.4841229183$$

If the data is misclassified, then it would be:

$$w_i^{(k)} = \frac{1}{Z}\frac{1}{16}\exp(0.2554128119) = 0.8068715305$$

c)

You should stop if the error becomes zero because the strength becomes zero too. The new weights become the previous iterations' weights which means if we don't stop, the algorithm would loop infinitely.

$$\epsilon_k = \sum_{i=1}^{n} w_i^{(k-1)} \mathbb{I}[y_i \neq f_k(\boldsymbol{x}_i)] = 0$$

$$\alpha_i = \frac{1}{2}\log\left(\frac{1-\epsilon_i}{\epsilon_i}\right) = 0$$

$$w_i^{(k)} = \frac{1}{Z} w_i^{(k-1)} \exp\left(-\alpha_k y_i f_k(x_i)\right) = w_i^{(k-1)}$$

Problem 2:

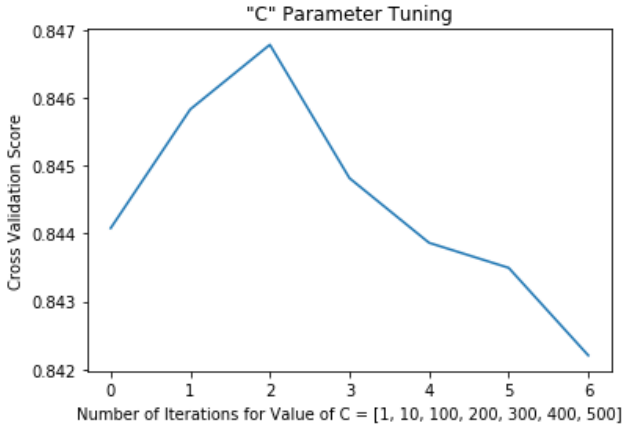
Support Vector Machines

I use K-fold cross-validation to adjust at least some of the parameters to get good performance on the entire data. Then I compute the test error rates after I settled on hyperparameter settings and trained your two final classifiers.

Following are the steps I used in order to get the best parameters for C, Kernel and Gamma for SVM.

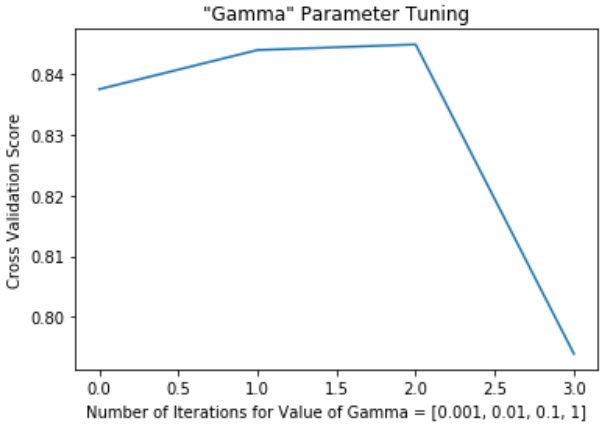
Using CV = 2 and Kernel = 'rbf' I get the following results:

1. First, I train the entire data with different C values with a simple for loop. The following graph depicts the Cross-Validation Score against the different values of C.



   According to the graph above, the best value of **C = 100.**

2. Second, I train the data with just the different values of gamma using a simple for loop. The following graph depicts the Cross-Validation Score against the different values of gamma.



   According to the graph above, the best value of **Gamma = 0.1.**

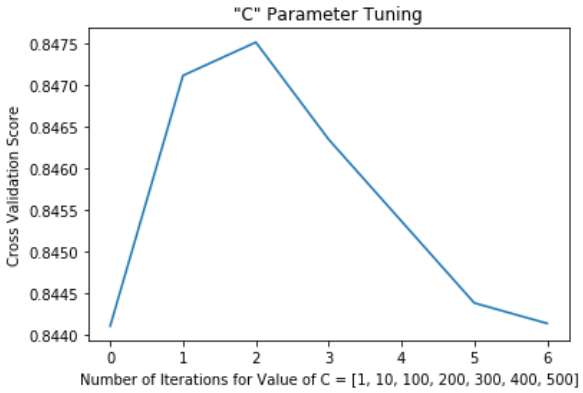
3. Then I used all the values of Gamma and C that I used earlier, and trained the data with different combinations of C and Gamma by using a simple double for loop. The following table depicts all the Cross-validation scores given the Gamma and C value:

gamma\C	1	10	100	200	300	400	500
0.001	0.837505	0.844415	0.846350	**0.846995**	0.846719	0.846381	0.846411
0.01	0.843954	0.845644	0.845613	0.843801	0.842511	0.841405	0.839931
0.1	0.844876	0.833451	0.806056	0.802525	0.801204	0.799975	0.799484
1	0.793956	0.796628	0.796628	0.796628	0.796628	0.796628	0.796628

According to the table the highest score occurred when **Gamma = 0.001, and C = 200**

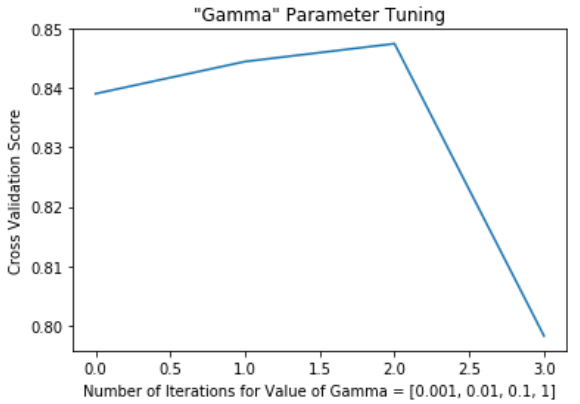
I do the same thing using CV = 3 and Kernel = 'rbf'. I get the following results:

1. First, I train the entire data with different C values with a simple for loop. The following graph depicts the Cross-Validation Score against the different values of C.



According to the graph above, the best value of **C = 100. Still the same.**

2. Second, I train the data with just the different values of gamma using a simple for loop. The following graph depicts the Cross-Validation Score against the different values of gamma.



According to the graph above, the best value of **Gamma = 0.1. Still the same.**

3. Then I used all the values of Gamma and C that I used earlier, and trained the data with different combinations of C and Gamma by using a simple double for loop. The following table depicts all the Cross-validation scores given the Gamma and C value:

gamma\C	1	10	100	200	300	400	500
0.001	0.839010	0.845122	0.847026	0.847118	0.846780	0.846872	0.847087
0.01	0.844415	**0.847486**	0.847026	0.845552	0.844200	0.843924	0.843647
0.1	0.847425	0.838211	0.808329	0.804705	0.803968	0.803323	0.803477
1	0.798286	0.801419	0.801419	0.801450	0.801450	0.801450	0.801450

According to the table the highest score occurred when **Gamma = 0.01, and C = 10.** Since the validation scores were higher when we chose cv=3, I used CV = 3 for the future tests.

Then I split the training data in the ratio of 60:40. The accuracy I achieve using Gamma = 0.01, and C = 10 is 84.87%. I then notice that I did not check the values between C = 10 and C = 100. I arbitrarily choose C = 35 (still using Gamma = 0.01) and I achieve the accuracy of 85.01%. Just using the training data, I looped using the C values between 30 and 40. It turns out that C = 35 was the best and I just got lucky. Following were the results for **hold out cross validation**:

C: 30
Validation Score: 0.845516
Accuracy: 84.94%

C: 31
Validation Score: 0.845721
Accuracy: 84.95%

C: 32
Validation Score: 0.845823
Accuracy: 84.98%

C: 33
Validation Score: 0.846028
Accuracy: 85.00%

C: 34
Validation Score: 0.846028
Accuracy: 84.99%

C: 35
Validation Score: 0.846182
Accuracy: 85.01%

C: 36
Validation Score: 0.845977
Accuracy: 85.00%

C: 37
Validation Score: 0.845874
Accuracy: 84.98%

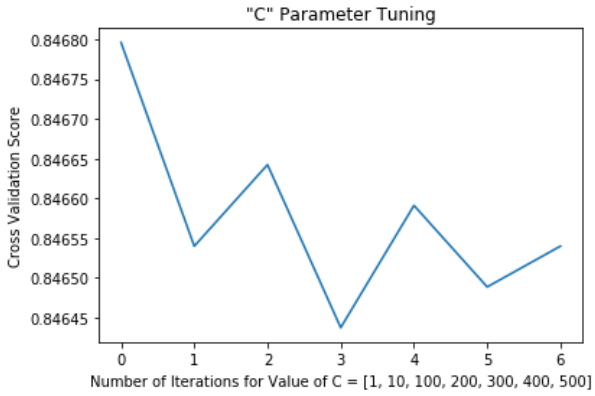
C: 38
Validation Score: 0.845874
Accuracy: 85.01%

C: 39
Validation Score: 0.845618
Accuracy: 85.01%

C: 40
Validation Score: 0.845618
Accuracy: 85.01%

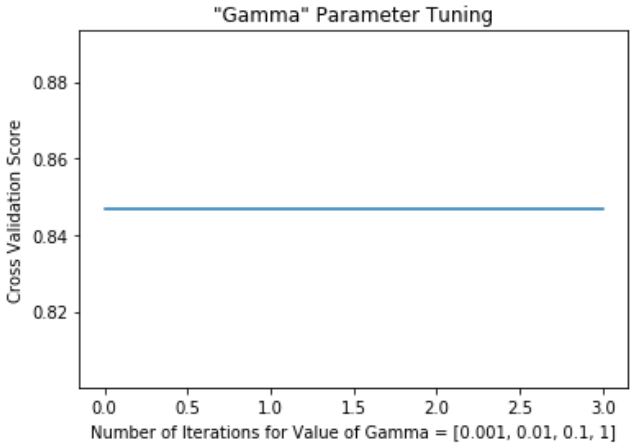
Now I change the kernel to linear and do the same thing as above: (using the entire training data to train the model)

1. First, I train the entire data with different C values with a simple for loop. The following graph depicts the Cross-Validation Score against the different values of C.



   According to the graph above, the best value of **C = 1.**

2. Second, I train the data with just the different values of gamma using a simple for loop. The following graph depicts the Cross-Validation Score against the different values of gamma.



   According to the graph above, the gamma value did not effect the score.

3. Then I used all the values of Gamma and C that I used earlier, and trained the data with different combinations of C and Gamma by using a simple double for loop. The following table depicts all the Cross-validation scores given the Gamma and C value:

gamma\C	1	10	100	200	300	400	500
0.001	0.846796	0.846540	0.846642	0.846437	0.846591	0.846489	0.846540
0.01	0.846796	0.846540	0.846642	0.846437	0.846591	0.846489	0.846540
0.1	0.846796	0.846540	0.846642	0.846437	0.846591	0.846489	0.846540
1	0.846796	0.846540	0.846642	0.846437	0.846591	0.846489	0.846540

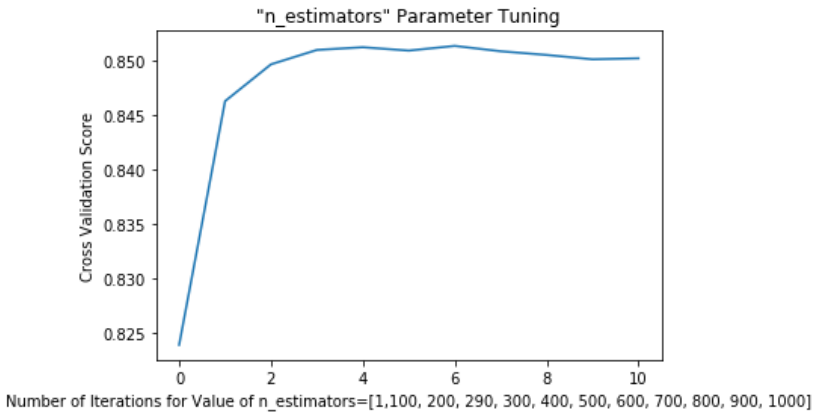
According to the table the highest score occurred when **Gamma = 0.001, and C = 1.** Since the validation scores were higher when Kernel was 'rbf' I decided to use Kernel to 'rbf'.

When I test C = 35, gamma = 0.01 and Kernel = 'rbf' using the actual test data (using training data entirely just for training) the accuracy I achieve is 85.06%.

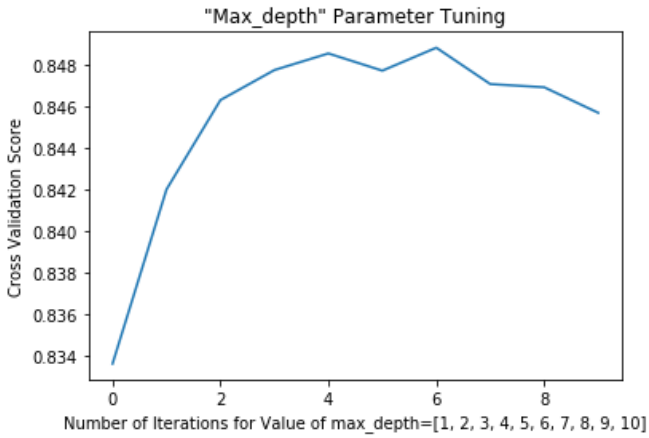
FINAL ANSWER: c = 35, gamma = 0.01, Kernel = 'rbf'

XGBoost

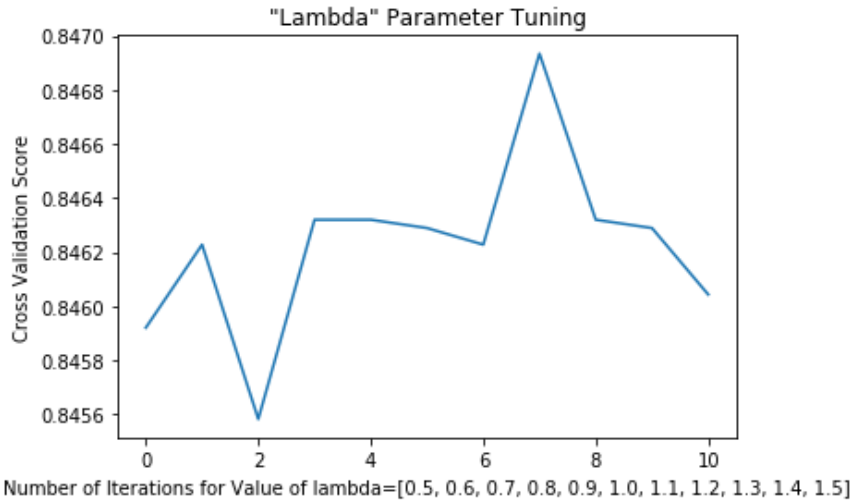
For tuning the parameters, of XG boost, I tuned the parameters one by one. I get the following results. **Please note for every result, I am using default values of the other parameters.**



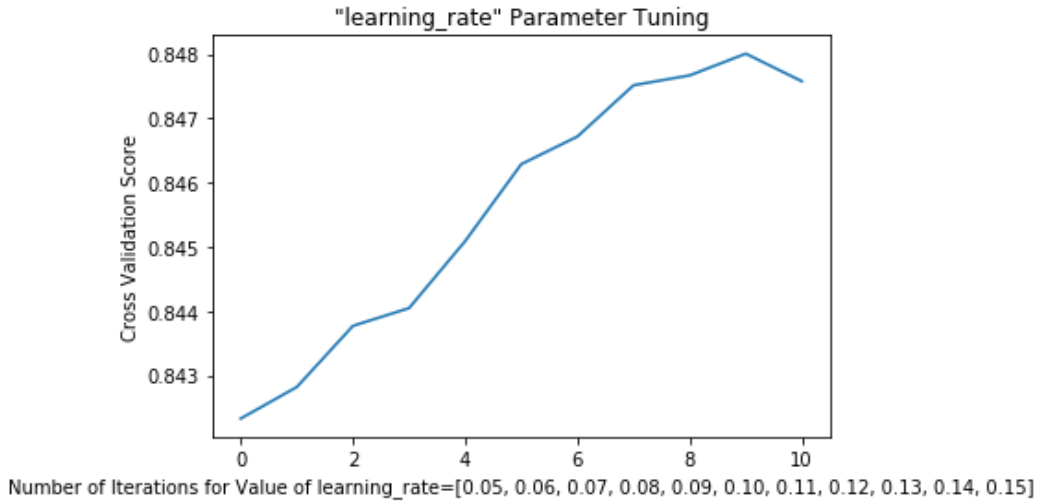
Best n_estimators value: 600



Best Max depth value: 7.0



Best lambda value: 1.2

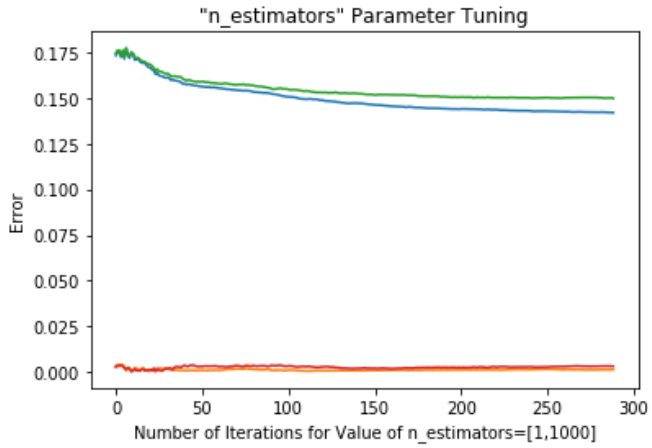


Best learning_rate value: 0.140000

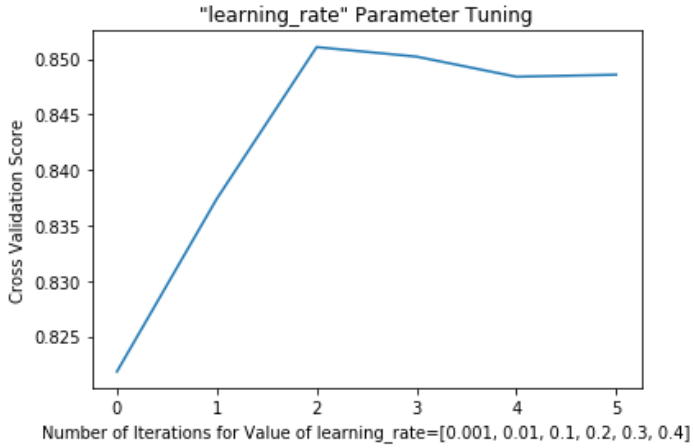
The accuracy I achieve using the above parameters and splitting the training data to the ratio of 60:40 (train:test) is: **Accuracy: 84.08%**

Then I think to myself, if I updated the previously tuned parameter for the next parameter that was being tuned I may get more accuracy. The following data is what I achieve when

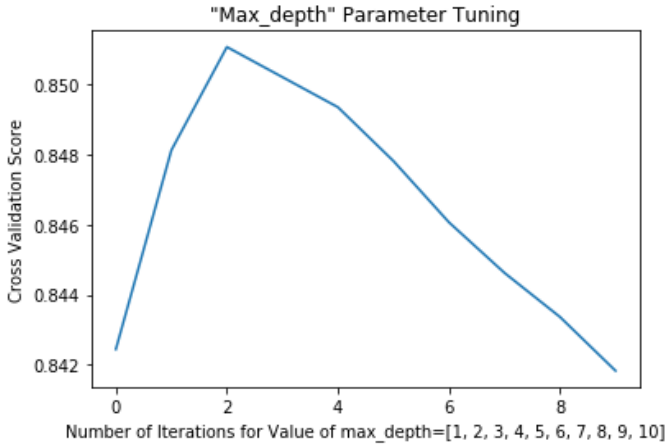
To calculate the optimum value of n_estimators, xgb.cv will be preferred as the computation time is very less compared to grid search and also the results from xgb.cv are more interpretable as it gives the cross-validation score at each iterations. Therefore, I use xgb.cv on the next set of results in order to tune n_estimators. I also made sure that I used a reasonable stopping point so that I do not over fit the data. Over fitting could reduce the testing accuracy.



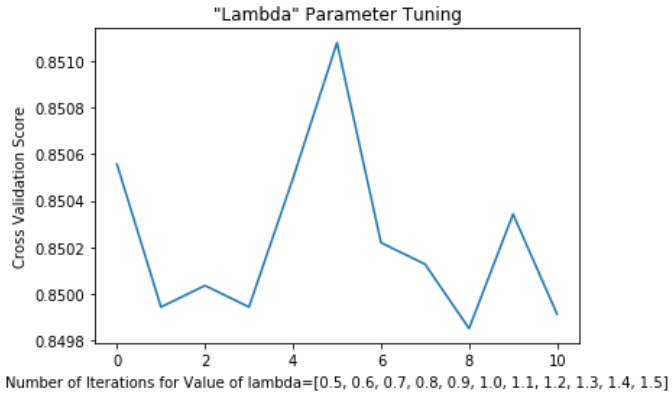
Best n_estimators value: 289.000000



Best learning rate value: 0.100000



Best Max depth value: 3.000000



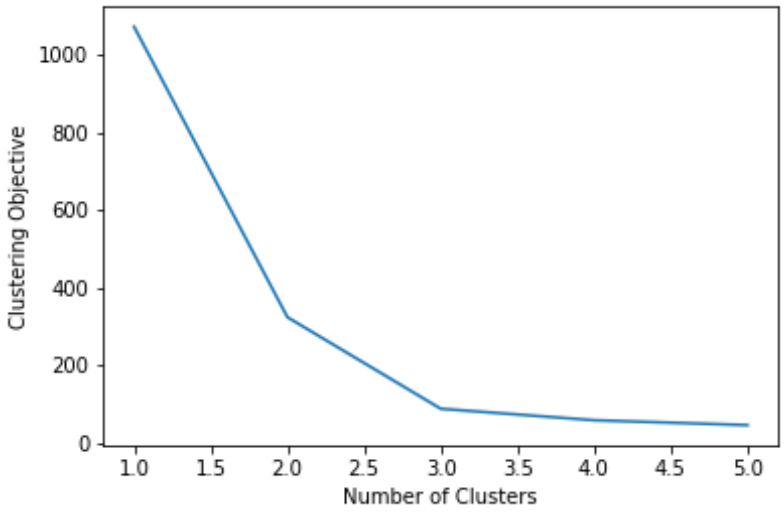
Best lambda value: 1.000000

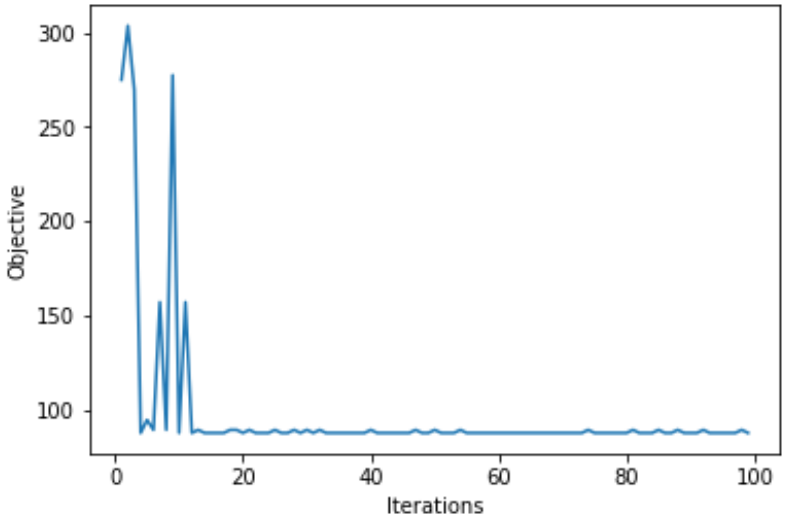
The accuracy that I achieve is 85.39%. I stopped at this point because the accuracy I achieved for XGBoost was higher than the accuracy for SVM.

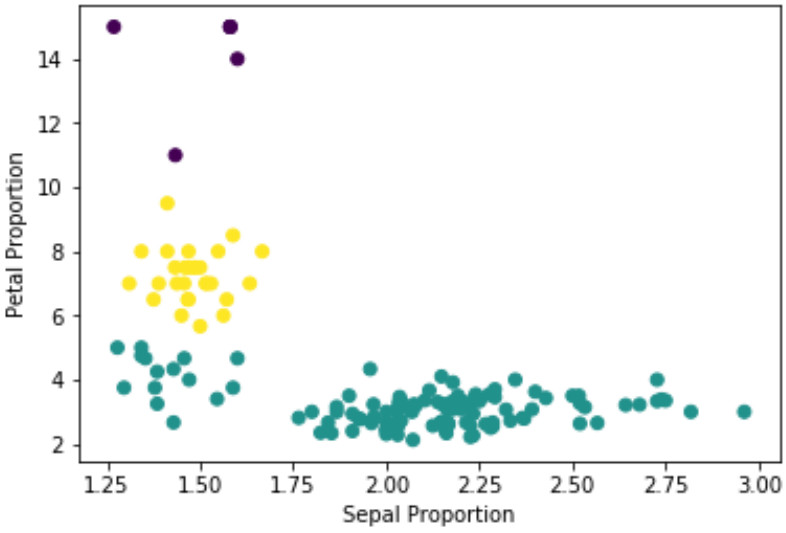
Final Answer:

Name of Parameters	**Original Value**	**Tuned Value**
n_estimators	100	289
learning_rate	0.1	0.1
max_depth	3	3
Reg_lambda	1.0	1.0

Problem 3:


Clustering Objective
1000
800
600
400
200
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Number of Clusters


Objective
300
250
200
150
100
0
20
40
60
80
100
Iterations


Petal Proportion
14
12
10
8
6
4
2
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Sepal Proportion