Data 558 Final Project: Kaggle Competition

Alyssa Goodrich
June 6, 2017

Section 1: Introduction

In this project I attempt to train a classifier to determine the bird species of each example photo in set of test data. The training set includes 144 different species of birds, each with 30 examples for a total of 4320 training images. The testing set is the same size, and the true labels are unknown to us. Because the given testing data lacks true labels, for this project I will split the "training data" in to two groups, one for training the model and the other for validating. In developing the best model, I will take the approach of beginning with a simplistic model that considers only two classes and build up in complexity to explore different methods of developing a multi-class classification model to ultimately develop the best predictions. As I explore the simpler methods I will rely on my own algorithms for my predictions. This effort will culminate in a logistic classifier using one v rest that will create predictions for the entire test set. After that I will increasingly use Scikit Learn functionality to explore and cross-validate various models. Ultimately the best predictions were found by using a ensemble soft voting classifier that used as inputs the four best models that I found by exploring and cross-validating various model types and parameters.

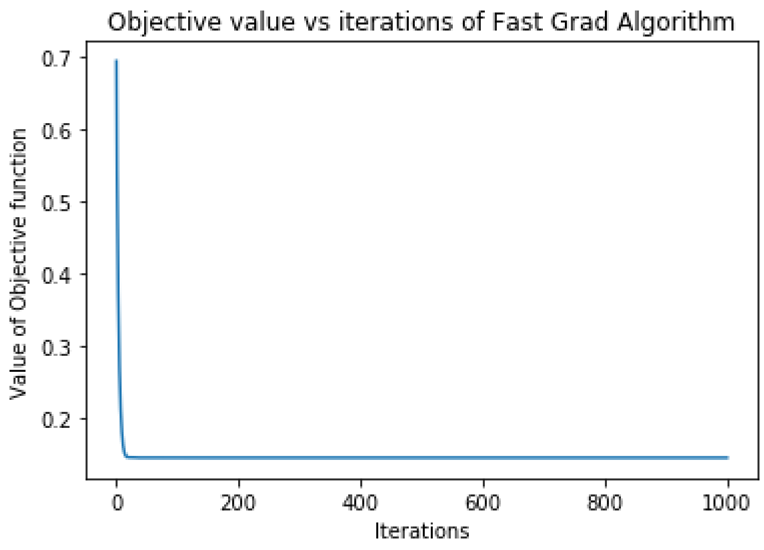
The data features used in this project were extracted from images using Tensor Flow, as developed by Research at Google. I will not discuss the data extraction in this paper, but rather focus on the analysis of the data after it was extracted.

Section 2: Consider only two classes

This section is based on milestones in homework 4
For this section I decided to choose two classes that had some similarity to make the task of classifying them more interesting. From previous analysis I learned that the Bewick Wren and the Caronlina Wren are often confused using logistic classification models and this particular training data, so I chose those two birds for this exercise.
The first exercise to explore classification with two classes was to train an L2 regularized logistic regression classifier using 75% of the data. The logistic regression was appropriate for this exercise because it is effective at determining the probability that a bird belongs to class A (i.e., Bewick Wren), given feature data. I used a decision rule where if the logistic classifier estimated greater than 50% chance of a bird being a Bewick Wren, I predicted it was a that bird. Otherwise, I predicted it was a Carolina Wren.

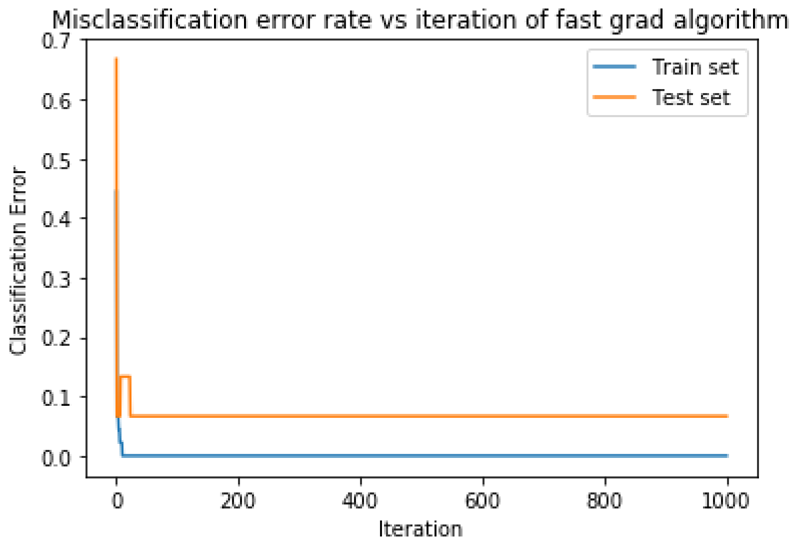
To conduct this analysis I implemented a fast gradient algorithm. When I ran the fast gradient algorithm on the training set of the training data I found that it converged very quickly, see plot below.

```
Out[90]: <matplotlib.text.Text at 0x29c983587f0>
```



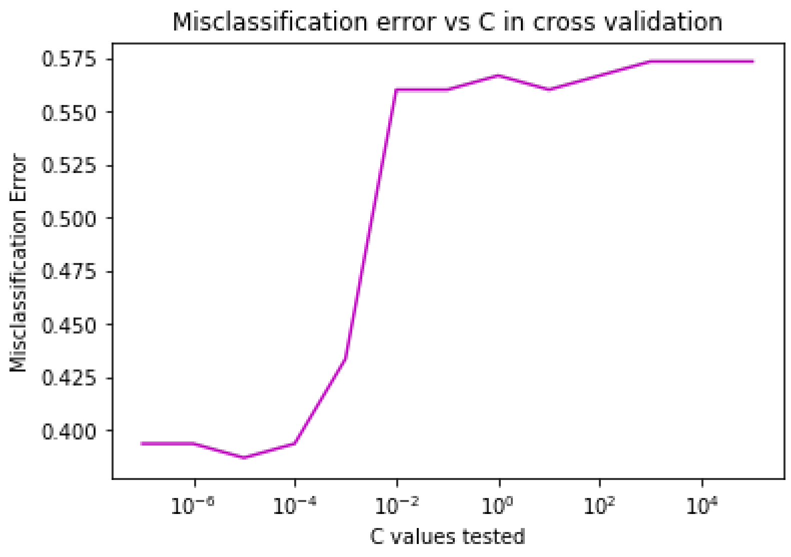
I then used the values of betas found using the fast gradient algorithm as the model to predict classifications on both the training set of training data and the test set of training data. The results are on the below chart. Not surprisingly, the error quickly converged to zero on the training set of data. In this particular data set there are far more features than there are examples so it is easy for the model to over-fit the training data. However, the test data shows that the misclassification error improves initially, but then converges to a minimum of 6.7%. Since I am much more interested in training a model that can predict data that I have not seen before rather than perfectly predicting the training data, I will move on to find the optimal value of lambda that will minimize the error on the test set.

```
Out[13]: <matplotlib.text.Text at 0x19570647e48>
```



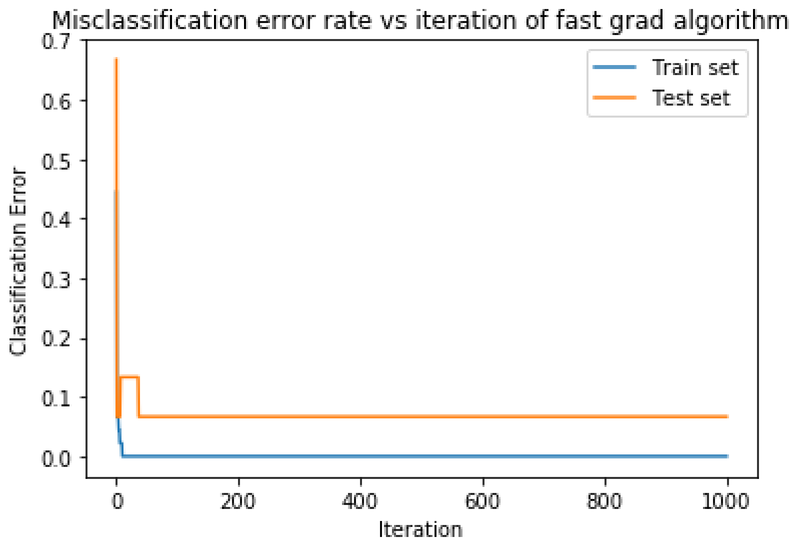
To find a value of lambda that will help minimize the misclassification error on the training set I conducted a cross validation analysis on the training set using Scikit-Learn's LogisticRegressionCV function. I first considered an array of potential values for C between 10^-7 and 10 ^5 and found that the value that minimized the misclassification.

```
The optimal lambda for cross validation is: 0.00075
```



In this chart I can see that the value of c that minimizes the error is 10 e-5. After selecting the the value of C that minimizes the error I need to adjust it so that I can use it in the model. Scikit-Learn uses a slightly different objective function than I do. When I use the new optimal lambda to predict the classes I still get a misclassification error on the test set of 6.7%. See below. However, since the test set includes only 15 observations, this means that I have only one prediction incorrect. Even as I adjust the lambda across an array of values I were unable to find a value of lambda that eliminated that one error.

```
Out[16]: <matplotlib.text.Text at 0x19570617588>
```



Next Steps for two-category logistic regression
Before moving from a simple two category logistic regression to consider a data set with five categories I will discuss some opportunities to improve the two category logistic regression.

1) Implement a stopping rule in my fast gradient algorithm. I currently waste a lot of computational time doing unnecessary iterations that do not reduce the value of the objective function. I should implement a stopping rule that stops the algorithm when the improvement between iterations is

sufficiently small.

2) Consider implementing 1 vs 1 logistical regression for each combination of categories and combine those predictions to create a multi-class classification.

Rather than implement these, I will invest my time in some of the more complicated analyses I will discuss later on in this paper.

Section 3: Five-category logistic regression

This section is based on milestones in homework 5

In this section, I will increase the complexity slightly and consider five of the available 144 categories. The first thing I did was to subset the data to include just five categories. Initially I chose 5 categories at random, which were linked to the numbers 135-139.

To conduct the five category logistic regression I used a one vs rest decision function, meaning I trained five different models, one for each category. I then ran the test set through each of the five models, and assigned each observation to the category that had the highest estimated probability. To create these models I first had to create subsets the data into groups that had 50% observations from the the selected category, and the other 50% of observations from a pool of the other four categories. When I trained and tested these models I found a high degree of accuracy, with only one prediction incorrect. See below for the confusion matrix.

Out[4]:

```
The misclssification error on the all data is: 0.02
```

	White_Breasted_Kingfisher	White_Breasted_Nuthatch	White_Crowned_Sparrow	White_Eyed_Vireo	White_Necked_Ra
White_Breasted_Kingfisher	30.0	0.0	0.0	0.0	0.0
White_Breasted_Nuthatch	0.0	29.0	1.0	0.0	0.0
White_Crowned_Sparrow	0.0	0.0	30.0	0.0	0.0
White_Eyed_Vireo	0.0	2.0	0.0	28.0	0.0
White_Necked_Raven	0.0	0.0	0.0	0.0	30.0

Based on the confusion matrix I can see that the only category that I did not have perfect accuracy was category 137. Although I did achieve very high accuracy with these randomly selected categories, those categories many not accurately represent the challenge of accurately deciphering all categories. So in the next step I will choose categories that may be more difficult to decipher and then develop a process to hone the parameters using cross validation to find the most accurate model.

To do this I first selected five new categories of bird. There are 6 categories of bird that are subspecies of the species wren so I decided that those may be more difficult to decipher since they all belong to the same species. When I create the confusion matrix using a default lambda of 1 I see that these wrens are indeed more difficult to decipher than the previously selected birds, and have a misclassification error of 30% on the training set. I will see if I can find lambdas that will reduce this error.

```
The misclssification error on the all data is: 30%
```

Out[7]:

	Bewick_Wren	Cactus_Wren	Carolina_Wren	Marsh_Wren	Rock_Wren
Bewick_Wren	3.0	0.0	1.0	1.0	1.0
Cactus_Wren	1.0	5.0	0.0	0.0	0.0
Carolina_Wren	1.0	0.0	4.0	0.0	1.0
Marsh_Wren	1.0	0.0	1.0	4.0	0.0
Rock_Wren	1.0	0.0	0.0	0.0	5.0

To hone the lambda parameters I trained a model that considered each potential value of lambda for each category. Then for each potential combination of lambdas I ran through the hold out validation set and found the level of accuracy. I then printed a table of all the combinations of lambda that resulted in the lowest possible error rate. I found 168 different potential combinations of lambda values resulted in an estimated error rate of 21%. While 168 may sound like many different combinations it is only a very small subset of the 16807 possible combinations of lambda. A sample of this table is printed below, and the whole table can be found in appendix 1.

Out[13]:

	C1 Lamda	C2 Lamda	C3 Lamda	C4 Lamda	C5 Lamda	Error
0	100.0	0.001	0.001	0.001	100.0	0.210526
0	100.0	0.001	0.001	0.010	100.0	0.210526
0	100.0	0.001	0.001	0.100	100.0	0.210526
0	100.0	0.001	0.001	1.000	100.0	0.210526
0	100.0	0.001	0.001	10.000	100.0	0.210526
0	100.0	0.001	0.010	0.001	100.0	0.210526

Now that we have lambdas that result in the lowest error, we will use them to create predictions that will have the highest possible accuracy. I created a confusion matrix, this time considering only the hold out validation set. I see that I found a 23% error, which is just above the optimal results I saw from the cross validation.

Out[317]:

	Bewick_Wren	Cactus_Wren	Carolina_Wren	Marsh_Wren	Rock_Wren
Bewick_Wren	5.0	0.0	1.0	0.0	0.0
Cactus_Wren	1.0	5.0	0.0	0.0	0.0
Carolina_Wren	1.0	0.0	5.0	0.0	0.0
Marsh_Wren	1.0	0.0	1.0	4.0	0.0
Rock_Wren	1.0	0.0	1.0	0.0	4.0

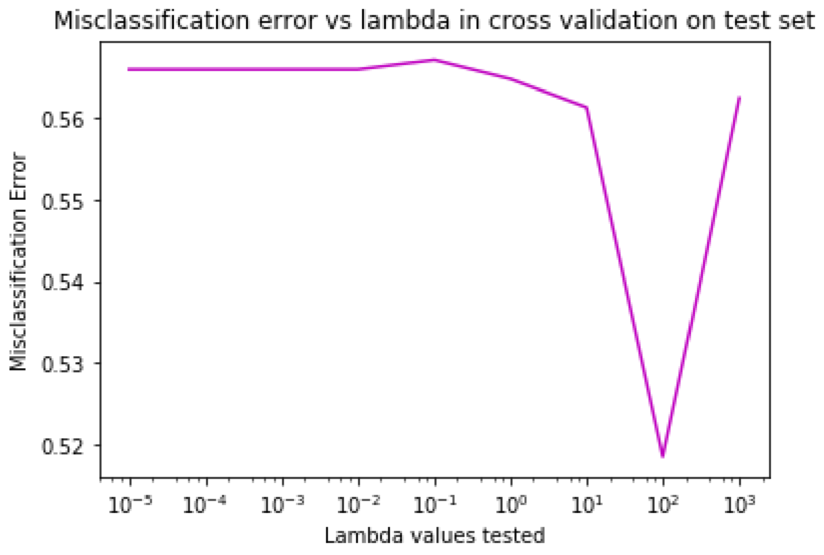
Section 4: All-category logistic regression

I will now extend what I did in the above section to create predictions for the test set using all categories and a one versus rest approach. In this section, I will create predictions for all categories using only algorithms that I wrote, not any Scikit-Learn functionality.

First, I must adopt an approach to sub-setting the data to create the one 144 one vs rest models. The approach I chose is to re-sample the chosen category to create training data set for each category that includes 50% data from the chosen category, and 50% data that includes each of the other 143 categories.

To optimize the model to minimize over-fitting while retaining optimal predictive power, I must tune the parameter lambda. It would be impossible to tune lambda using the approach in the last section because the number of potential combinations approaches infinity. (For fun, this is the number of combinations of lambda for 144 categories with 7 potential lambdas. 7^144 = 4944447410368420611677552723263003650621645445111595111414093793918908668186178
Obviously I cannot calculate the misclassification error for that many combinations so I will have to make some simplifying assumptions.

The first simplifying assumption I tried is to run a cross-validation on a one vs rest logistic regression classifier where each of the 144 models used the same value of lambda. When I did this I found rather high errors on the validation set. The below plot shows, that the errors were above 50% with all of the potential values of lambda that I tested. The error corresponding to lambda = 10 is lower than the others. The sharp dip, followed by an equally sharp increase causes me to wonder if that low value may be in part due to chance. Regardless of whether 10 is the best lambda, it still yields more than a 50% error rate so I will seek to find a better way to tune the lambda parameter.



Perhaps the assumption that all categories have the same lambda was too blunt. So I will try to refine by determining the lambda that results in minimum misclassification error for each category individually. I created a table to show the misclassification error for one vs rest for each value of lambda for each category. I then selected the lowest value of lambda that delivered the lowest misclassification error and used those lambdas to train the model and make the predictions. The below table is a shorter example of the table I created for all categories. See the full table in Appendix 2.

Out[9]:

	0.0001	0.001	0.01	0.1	1.0	10.0	100.0	1000.0	10000.0
American_Crow	0.388889	0.388889	0.388889	0.388889	0.388889	0.388889	0.388889	0.387731	0.388889
American_Goldfinch	0.377315	0.377315	0.377315	0.377315	0.377315	0.377315	0.377315	0.378472	0.378472
American_Pipit	0.396991	0.396991	0.396991	0.396991	0.396991	0.396991	0.396991	0.395833	0.395833
American_Redstart	0.505787	0.505787	0.505787	0.505787	0.505787	0.506944	0.506944	0.509259	0.509259
Anna_Hummingbird	0.348380	0.348380	0.348380	0.348380	0.348380	0.348380	0.348380	0.350694	0.351852
Baltimore_Oriole	0.440972	0.440972	0.440972	0.440972	0.440972	0.440972	0.440972	0.445602	0.447917
Barn_Swallow	0.488426	0.488426	0.488426	0.488426	0.488426	0.490741	0.488426	0.494213	0.496528
Bay_Breasted_Warbler	0.479167	0.479167	0.479167	0.479167	0.479167	0.479167	0.479167	0.479167	0.479167
Belted_Kingfisher	0.460648	0.460648	0.460648	0.460648	0.460648	0.460648	0.460648	0.459491	0.458333
Bewick_Wren	0.467593	0.467593	0.467593	0.467593	0.467593	0.466435	0.467593	0.466435	0.467593
Black_And_White_Warbler	0.521991	0.521991	0.521991	0.521991	0.521991	0.521991	0.521991	0.527778	0.527778
Black_Billed_Cuckoo	0.456019	0.456019	0.456019	0.456019	0.456019	0.454861	0.453704	0.454861	0.454861
Black_Footed_Albatross	0.324074	0.324074	0.324074	0.324074	0.324074	0.324074	0.324074	0.322917	0.321759
Black_Tern	0.420139	0.420139	0.420139	0.420139	0.420139	0.420139	0.417824	0.415509	0.417824

Once I found the optimal lambda for each category individually I used that in training a one vs rest logistic classifier on the training portion of the training data. I then testing it using the hold out validation set of the training data and found a 67% accuracy (woo hoo!). Unfortunately when I trained the model using these lambdas and the entire training set, and then created predictions on the test set and submitted to Kaggle I found a lower level of accuracy. I am curious whether there is an error in my calculations or if I somehow seriously over-fit my model.

Now that I have completed a classifier using my own models, I will begin exploring models available in the Scikit-learn package.

Out[1]: The raw code for this IPython notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.

Section 5: Linear SVM

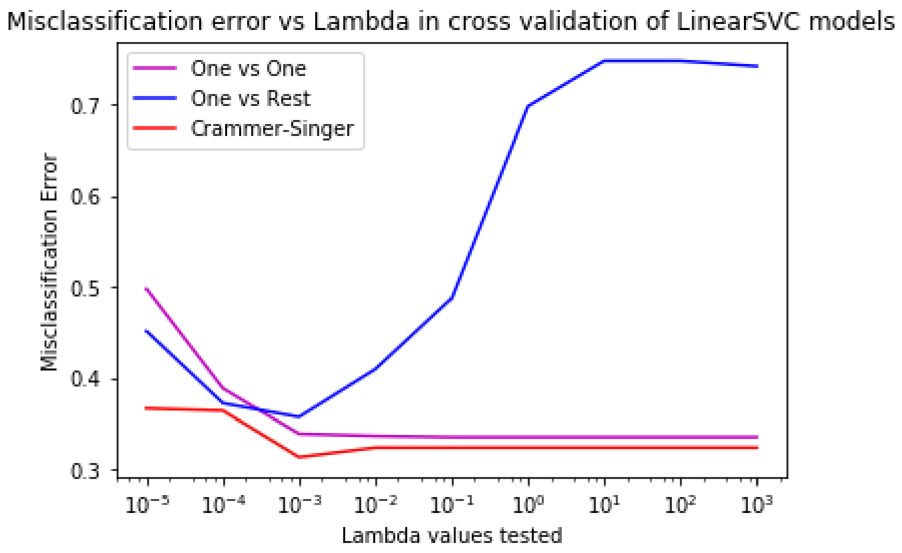
This section is based on the milestones in homework 7

I will start by exploring different a one vs one linear SVC. Although I used the linear SVC to fit the model for each category, I created my own infrastructure to compare predictions from each model for each observation and ultimately make a prediction for every observation. At this early stage I am exploring the various model and using only the default value for lambda. Doing this yielded an misclassification error of 33%, using the default value of lambda. It is interesting how much better this linear SVC model did than my logistic classifier did using a constant value of lambda.

We then considered a one vs rest linear SVC, also using the default lambda value of 1. Similarly to the previous test, I built my own infrastructure to compare the models created for each category. Using this approach I found a misclassification error rate of 71%, which is much higher than using the one v one approach. It was surprising to me that the results were so much worse. I am curious whether tuning the lambda parameter would create results that were more in line with the one v one approach.

Finally I considered the Crammer-Singer method in the linear SVC. Despite Scikit-Learn's warning that Crammer-Singer seldom does better than other methods, for me it did perform slightly better than 1 vs 1, yielding a 32% error. While it was fun to see how these models performed using their default lambda parameter, what I really want to know is how they will perform using their optimal lambda parameter.

To find the answer to that question I ran a cross validation by training the model on a training set and testing it on a hold out validation set. I considered values of lambda on a log scale spanning from 10^-5 to 10^3. The results of those cross validations are below. For each the one v rest, the one v one and the crammer singer, the misclassification error on the hold out validation set was lowest at a value of .001. However at that optimal Lambda Crammer Singer resulted in the lowest misclassification error, followed by one vs one, with one vs rest bringing up the rear. One vs rest misclassification error was about 4 percentage points worse than Crammer-Singer.



Since Crammer Singer was the best on my training set, I then trained a Crammer Singer model using all the test data with a lambda of .001 and tested it by submitting to Kaggle. My results on Kaggle were substantially lower than the the results I predicted with my cross validation set, somewhere in the 40% range.

It seems as though I have sufficiently explored the possibilities with linear SVM so I will now move on to explore Kernel SVM classifiers.

Section 6: Kernel SVM

This section is based on the milestones in homework 8

In this section I will explore kernel SVM. Given the nature of the data, there is no kernel in particular that makes intuitive sense to me so I will explore various options and find if a particular kernel is most effective. I started by exploring the polynomial kernel. I created a table that predicted the accuracy (% total correct estimates) of a model on the hold out validation set considering various degrees of the polynomial kernel and various levels of lambda. This cross validation indicated that the polynomial kernel with degree 2 and a lambda of ten had the best predicted accuracy on the validation set, at 73.7%. This estimated accuracy is the highest I have seen with any model.

Out[59]:

	2	3	4	5	6	7	8	9
0.00001	0.466667	0.216667	0.103333	0.046667	0.026667	0.023333	0.023333	0.0200
0.00010	0.466667	0.216667	0.103333	0.046667	0.026667	0.023333	0.023333	0.0200
0.00100	0.466667	0.216667	0.103333	0.046667	0.026667	0.023333	0.023333	0.0200
0.01000	0.466667	0.216667	0.103333	0.046667	0.026667	0.023333	0.026667	0.0300
0.10000	0.490000	0.260000	0.136667	0.080000	0.070000	0.066667	0.063333	0.0600
1.00000	0.713333	0.526667	0.290000	0.176667	0.116667	0.106667	0.086667	0.0733
10.00000	0.736667	0.653333	0.450000	0.256667	0.180000	0.120000	0.103333	0.0900
100.00000	0.736667	0.660000	0.466667	0.286667	0.180000	0.130000	0.110000	0.0933
1000.00000	0.736667	0.660000	0.466667	0.286667	0.200000	0.136667	0.106667	0.0900

We then conducted a similar analysis on the sigmoid and RBF kernels where I optimized over the parameters Gamma and Lambda. I found the RBF Kernel could achieve accuracy on the hold out validation set of 67.9% using a lambda of 10 and a gamma of .0001. The sigmoid kernel achieved maximum accuracy of 67.6% with lambda of 1000 and gamma of .00001. Both of these models underperformed the polynomial kernel of degree two.

After this, I went to lunch with a friend and lamented that I was not able to find higher accuracy with any model. My friend suggested I try Scikit-Learn's logistic function. Of course I have already tried my own logistic function so why would I need to do Scikit-Learn's as well? Nonetheless I followed my friend's suggestion and train Scikit-Learn's logigisticRegression function on the test data and achieved my best yet score on Kaggle. Having explored the different kernel types and different parameters for individual kernel SVC, I will now move on to consider a classifier that uses and ensemble of models.

Section 7: Ensemble methods

This section is based on the milestones in homework 8

As I explored ensemble methods I began by developing my own hard voting classifier. The hope is that the different models would be incorrect on different observations, and by voting I would increase the accuracy by capturing the best result of all models. I took the five best models developed in in section 6 including:

- Linear SVC one vs one with lambda = .01
- Linear SVC Crammer Singer with Lambmda = .001
- polynomial kernel svc with 2 degrees and lambda = 10
- RBF kernel svc with gamma = .0001 and lambda = 10
- Sigmoid kernel svc with gamma = .00001 and lambda = 1000

I trained these models using their optimal parameters using the entirety of the training data. I stored the predictions for the test set from each model in a table, like the one below, and then I selected the mode of the table.

```
[[114  41  49 ...,  94  93  69]
 [ 22  91  49 ...,  94  93  25]
 [114  91  49 ...,  94  93  69]
 [ 36  36  36 ...,  36  36  36]
 [ 22  41  49 ...,  94  93  78]]
['OneVOneLam01' 'CramSingLam001' 'polyOrder2Lam10' 'RBFLam10Gam0001'
 'SigLam1000Gam00001']
```

This did not yield an improvement over my previous Kaggle score. Perhaps part of the problem was that by training all models on all of the training data I caused some over-fitting. I might have improved upon this by using the bagging method.

After proving in Section 6 that sometimes Scikit-Learn does things a little better than I do, I decided to try Scikit-Learn's voting classifier too rather than rely solely on my own.

I used models with the best performance demonstrated above but dropped the lowest performing models of sigmoid and crammer singer and this time included the logistic regression classifier since its performance was best. I began using hard voting and I weighted the Logistic Regression function slightly more than the other models. I did this so the prediction from the logistic regression function would be selected in the event of a tie because that model had the highest accuracy. I did not cross validate this approach because I had already cross-validated all the models in the voting classifier. I submitted these predictions directly to Kaggle and was still unable to beat my best score.

I then tried Scikit-Learn's voting classifier using soft voting, in the hopes that the model with the highest probability estimates would have the highest chance of being correct. Using this approach yielded the highest accuracy I was able to obtain. I then began throwing some more spaghetti at the wall and tried including additional models, with poorer performance in my soft voting classifier. It turned out that including additional models harmed my performance more than it helped and ultimately my best prediction came from soft voting classifier using the following models.

- LinSVC = SVC(kernel = 'linear',degree=2, gamma='auto', C = .1, probability = True, decision_function_shape = 'ovo')
- SVMPoly2 = SVC(kernel='poly', degree=2, gamma='auto', C = 10, probability = True)
- RBF = SVC(kernel='rbf', C = 10, probability = True)
- LogReg = LogisticRegression()

Section 8: Next steps

While I was able to try many things there were some things I was not able to complete that I would like to explore.

Image preprocessing I expect my results could have been better if I were able to clean up the images by centering and rotating them as demonstrated in a lab earlier in this course. Doing that could have reduced unnecessary variation in the images and could have improved my results.

Feature reduction I would like to explore transforming the features using PCA or Lasso. In the data set I had only 30 examples of each species, and more than 2000 features. Paring down the number of features using PCA or Lasso could have helped eliminate some of the less useful features and reduced potential for over-fitting.

Other classification methods I am curious about how effective some other classifications might be. For example since I know that there are exactly 144 species could a k mean clustering method with 144 means be effective at categorizing the birds. I will also continue to pursue learning about other classification methods mentioned on the last lecture of this course.

Improve programming practices As a fairy novice computer programmer I have learned that I need to improve my modularization methods. I found that I was frequently rewriting code that did very similar things. In the future I will try to write a few functions that perform key tasks that I can wrap within other functions to do more complex analyses. Additionally I will work on functionalizing more of my basic tasks. I found that I was often doing repeat tasks and manually updating parameters. A short function could have reduced effort as well as error.

Out[1]: The raw code for this IPython notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.

Appendix 1:

List of lambda combinations that achieve the lowest misclssification error, for five cateogies

In [54]:
```
#Show all combinations of lambda that minmize the misclassification error
AllLams = findOptLams(xTest = xTest, yTest = yTest, xTrain = xTrain, yTrain = yTr
MinError = AllLams.min(axis=0).iloc[5]
OptLams = AllLams.loc[AllLams['Error'] == MinError]
OptLams
```

```
C:\Users\Alyssa\Anaconda3\lib\site-packages\ipykernel\__main__.py:2: RuntimeWar
ning: overflow encountered in exp
  from ipykernel import kernelapp as app
```

Out[54]:

	C1 Lamda	C2 Lamda	C3 Lamda	C4 Lamda	C5 Lamda	Error
0	100.0	0.001	0.001	0.001	100.0	0.210526
0	100.0	0.001	0.001	0.010	100.0	0.210526
0	100.0	0.001	0.001	0.100	100.0	0.210526
0	100.0	0.001	0.001	1.000	100.0	0.210526
0	100.0	0.001	0.001	10.000	100.0	0.210526
0	100.0	0.001	0.010	0.001	100.0	0.210526
0	100.0	0.001	0.010	0.010	100.0	0.210526
0	100.0	0.001	0.010	0.100	100.0	0.210526
0	100.0	0.001	0.010	1.000	100.0	0.210526
0	100.0	0.001	0.010	10.000	100.0	0.210526
0	100.0	0.001	0.100	0.001	100.0	0.210526
0	100.0	0.001	0.100	0.010	100.0	0.210526
0	100.0	0.001	0.100	0.100	100.0	0.210526
0	100.0	0.001	0.100	1.000	100.0	0.210526
0	100.0	0.001	0.100	10.000	100.0	0.210526
0	100.0	0.001	1.000	0.001	100.0	0.210526
0	100.0	0.001	1.000	0.010	100.0	0.210526
0	100.0	0.001	1.000	0.100	100.0	0.210526
0	100.0	0.001	1.000	1.000	100.0	0.210526
0	100.0	0.001	1.000	10.000	100.0	0.210526
0	100.0	0.001	100.000	0.001	100.0	0.210526
0	100.0	0.001	100.000	0.010	100.0	0.210526
0	100.0	0.001	100.000	0.100	100.0	0.210526
0	100.0	0.001	100.000	1.000	100.0	0.210526
0	100.0	0.001	100.000	10.000	100.0	0.210526
0	100.0	0.001	100.000	100.000	100.0	0.210526

	C1 Lamda	C2 Lamda	C3 Lamda	C4 Lamda	C5 Lamda	Error
0	100.0	0.010	0.001	0.001	100.0	0.210526
0	100.0	0.010	0.001	0.010	100.0	0.210526
0	100.0	0.010	0.001	0.100	100.0	0.210526
0	100.0	0.010	0.001	1.000	100.0	0.210526
...	...	...	...	...	...	...
0	100.0	100.000	0.001	0.100	100.0	0.210526
0	100.0	100.000	0.001	1.000	100.0	0.210526
0	100.0	100.000	0.001	10.000	100.0	0.210526
0	100.0	100.000	0.010	0.001	100.0	0.210526
0	100.0	100.000	0.010	0.010	100.0	0.210526
0	100.0	100.000	0.010	0.100	100.0	0.210526
0	100.0	100.000	0.010	1.000	100.0	0.210526
0	100.0	100.000	0.010	10.000	100.0	0.210526
0	100.0	100.000	0.100	0.001	100.0	0.210526
0	100.0	100.000	0.100	0.010	100.0	0.210526
0	100.0	100.000	0.100	0.100	100.0	0.210526
0	100.0	100.000	0.100	1.000	100.0	0.210526
0	100.0	100.000	0.100	10.000	100.0	0.210526
0	100.0	100.000	1.000	0.001	100.0	0.210526
0	100.0	100.000	1.000	0.010	100.0	0.210526
0	100.0	100.000	1.000	0.100	100.0	0.210526
0	100.0	100.000	1.000	1.000	100.0	0.210526
0	100.0	100.000	1.000	10.000	100.0	0.210526
0	100.0	100.000	10.000	0.001	100.0	0.210526
0	100.0	100.000	10.000	0.010	100.0	0.210526
0	100.0	100.000	10.000	0.100	100.0	0.210526
0	100.0	100.000	10.000	1.000	100.0	0.210526
0	100.0	100.000	10.000	10.000	100.0	0.210526
0	100.0	100.000	10.000	100.000	100.0	0.210526
0	100.0	100.000	100.000	0.001	100.0	0.210526
0	100.0	100.000	100.000	0.010	100.0	0.210526
0	100.0	100.000	100.000	0.100	100.0	0.210526

	C1 Lamda	C2 Lamda	C3 Lamda	C4 Lamda	C5 Lamda	Error
0	100.0	100.000	100.000	1.000	100.0	0.210526
0	100.0	100.000	100.000	10.000	100.0	0.210526
0	100.0	100.000	100.000	100.000	100.0	0.210526

168 rows × 6 columns

Appendix 2:

List of misclssification error associated with each category in one v rest logistic regression

In [71]:

```
lams = np.array((.0001,.001, .01, .1, 1, 10, 100, 1000,10000))
LambdaMatrix( xTrain = xTrain, yTrain = yTrain, xTest = xTest, yTest = yTest, lam
```

```
C:\Users\Alyssa\Anaconda3\lib\site-packages\ipykernel\__main__.py:2: RuntimeWar
ning: overflow encountered in exp
  from ipykernel import kernelapp as app
```

Out[71]:

	0.0001	0.001	0.01	0.1	1.0	10.0
American_Crow	0.388889	0.388889	0.388889	0.388889	0.388889	0.388889
American_Goldfinch	0.377315	0.377315	0.377315	0.377315	0.377315	0.377315
American_Pipit	0.396991	0.396991	0.396991	0.396991	0.396991	0.396991
American_Redstart	0.505787	0.505787	0.505787	0.505787	0.505787	0.506944
Anna_Hummingbird	0.348380	0.348380	0.348380	0.348380	0.348380	0.348380
Baltimore_Oriole	0.440972	0.440972	0.440972	0.440972	0.440972	0.440972
Barn_Swallow	0.488426	0.488426	0.488426	0.488426	0.488426	0.490741
Bay_Breasted_Warbler	0.479167	0.479167	0.479167	0.479167	0.479167	0.479167
Belted_Kingfisher	0.460648	0.460648	0.460648	0.460648	0.460648	0.460648
Bewick_Wren	0.467593	0.467593	0.467593	0.467593	0.467593	0.466435
Black_And_White_Warbler	0.521991	0.521991	0.521991	0.521991	0.521991	0.521991
Black_Billed_Cuckoo	0.456019	0.456019	0.456019	0.456019	0.456019	0.454861
Black_Footed_Albatross	0.324074	0.324074	0.324074	0.324074	0.324074	0.324074
Black_Tern	0.420139	0.420139	0.420139	0.420139	0.420139	0.420139
Black_Throated_Sparrow	0.465278	0.465278	0.465278	0.465278	0.465278	0.465278
Blue_Grosbeak	0.384259	0.384259	0.384259	0.384259	0.384259	0.386574
Blue_Headed_Vireo	0.515046	0.515046	0.515046	0.515046	0.515046	0.515046
Blue_Jay	0.399306	0.399306	0.399306	0.399306	0.399306	0.399306
Blue_Winged_Warbler	0.371528	0.371528	0.371528	0.371528	0.371528	0.371528
Boat_Tailed_Grackle	0.370370	0.370370	0.370370	0.370370	0.370370	0.370370
Bobolink	0.453704	0.453704	0.453704	0.453704	0.452546	0.452546
Bohemian_Waxwing	0.473380	0.473380	0.473380	0.473380	0.473380	0.473380
Bronzed_Cowbird	0.399306	0.399306	0.399306	0.399306	0.399306	0.399306
Brown_Pelican	0.369213	0.369213	0.369213	0.369213	0.369213	0.369213
Cactus_Wren	0.449074	0.449074	0.449074	0.449074	0.449074	0.449074
California_Gull	0.313657	0.313657	0.313657	0.313657	0.313657	0.312500
Canada_Warbler	0.423611	0.423611	0.423611	0.423611	0.423611	0.423611

	0.0001	**0.001**	**0.01**	**0.1**	**1.0**	**10.0**	**1**
Cape_Glossy_Starling	0.363426	0.363426	0.363426	0.363426	0.363426	0.363426	0
Cape_May_Warbler	0.486111	0.486111	0.486111	0.486111	0.486111	0.486111	0
Carolina_Wren	0.549769	0.549769	0.549769	0.549769	0.549769	0.549769	0
...	...	...	...	...	...	...	.
Rusty_Blackbird	0.496528	0.496528	0.496528	0.496528	0.496528	0.496528	0
Sage_Thrasher	0.478009	0.478009	0.478009	0.478009	0.478009	0.478009	0
Savannah_Sparrow	0.392361	0.392361	0.392361	0.392361	0.393519	0.393519	0
Sayornis	0.487269	0.487269	0.487269	0.487269	0.487269	0.487269	0
Scarlet_Tanager	0.453704	0.453704	0.453704	0.453704	0.453704	0.453704	0
Scissor_Tailed_Flycatcher	0.554398	0.554398	0.554398	0.554398	0.554398	0.553241	0
Scott_Oriole	0.445602	0.445602	0.445602	0.445602	0.445602	0.445602	0
Seaside_Sparrow	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000	0
Shiny_Cowbird	0.370370	0.370370	0.370370	0.370370	0.370370	0.370370	0
Song_Sparrow	0.390046	0.390046	0.390046	0.390046	0.390046	0.390046	0
Summer_Tanager	0.452546	0.452546	0.452546	0.452546	0.452546	0.452546	0
Tree_Sparrow	0.466435	0.466435	0.466435	0.466435	0.466435	0.466435	0
Tree_Swallow	0.413194	0.413194	0.413194	0.413194	0.413194	0.413194	0
Tropical_Kingbird	0.459491	0.459491	0.459491	0.459491	0.459491	0.457176	0
Vermilion_Flycatcher	0.447917	0.447917	0.447917	0.447917	0.447917	0.447917	0
Vesper_Sparrow	0.388889	0.388889	0.388889	0.388889	0.388889	0.388889	0
Warbling_Vireo	0.502315	0.502315	0.502315	0.502315	0.502315	0.502315	0
Western_Grebe	0.298611	0.298611	0.298611	0.298611	0.298611	0.298611	0
Western_Gull	0.300926	0.300926	0.300926	0.300926	0.300926	0.300926	0
Western_Meadowlark	0.464120	0.464120	0.464120	0.464120	0.464120	0.464120	0
Western_Wood_Pewee	0.524306	0.524306	0.524306	0.524306	0.524306	0.524306	0
White_Breasted_Kingfisher	0.447917	0.447917	0.447917	0.447917	0.447917	0.447917	0
White_Breasted_Nuthatch	0.400463	0.400463	0.400463	0.400463	0.400463	0.400463	0
White_Crowned_Sparrow	0.459491	0.459491	0.459491	0.459491	0.459491	0.459491	0
White_Eyed_Vireo	0.513889	0.513889	0.513889	0.513889	0.513889	0.513889	0
White_Necked_Raven	0.409722	0.409722	0.409722	0.409722	0.409722	0.409722	0
White_Throated_Sparrow	0.449074	0.449074	0.449074	0.449074	0.449074	0.449074	0
Wilson_Warbler	0.365741	0.365741	0.365741	0.365741	0.365741	0.365741	0

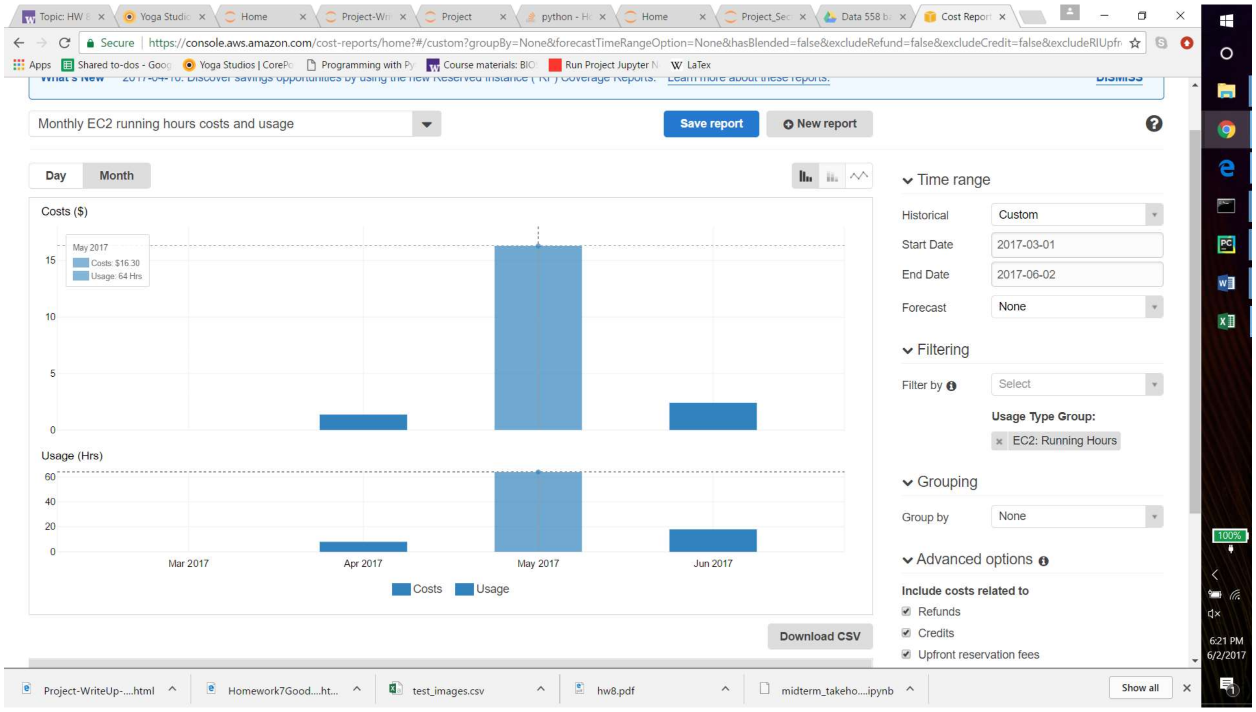
	0.0001	0.001	0.01	0.1	1.0	10.0	1
Winter_Wren	0.422454	0.422454	0.422454	0.422454	0.422454	0.422454	0
Yellow_Warbler	0.406250	0.406250	0.406250	0.406250	0.406250	0.407407	0

144 rows × 9 columns

Appendix 3: AWS Screenshots

Out[1]: The raw code for this IPython notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.

I ran many of my analyses before I was aware we should take screen shots. As evidence that I used AWS for my analyses I am proving a screenshot of my EC2 use (hours and cost), as you can see I used it a lot in May and some in June.



https://console.aws.amazon.com/cost-reports/home?#/custom?groupBy=None&forecastTimeRangeOption=None&hasBlended=false&excludeRefund=false&excludeCredit=false&excludeRIUpfr

Time range

Historical: Custom
Start Date: 2017-03-01
End Date: 2017-06-02
Forecast: None

Filtering

Filter by: Select
Usage Type Group: EC2: Running Hours

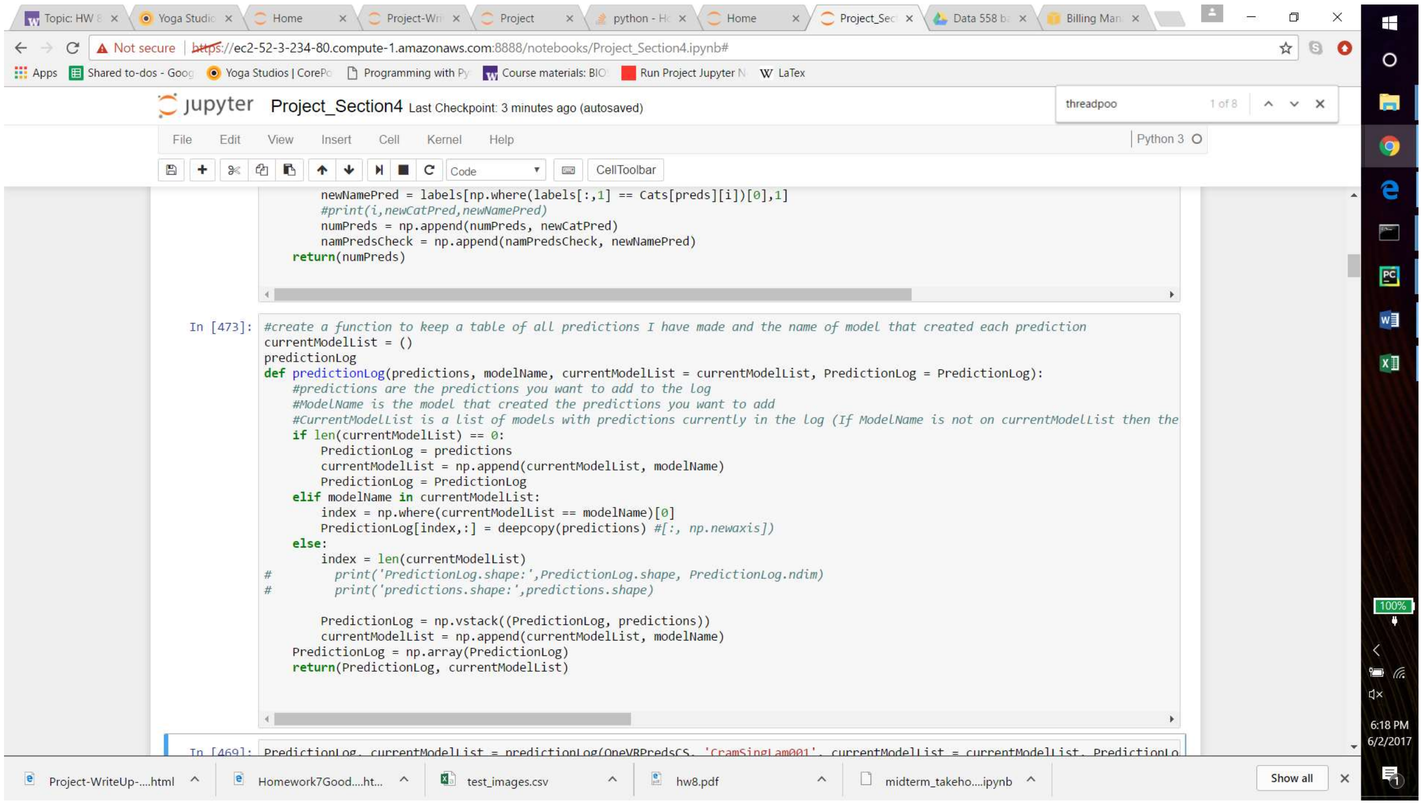
Grouping

Group by: None

Advanced options

Include costs related to
- [✓] Refunds
- [✓] Credits
- [✓] Upfront reservation fees

Here is a screenshot of running one of my jupyter notebooks on AWS in June.

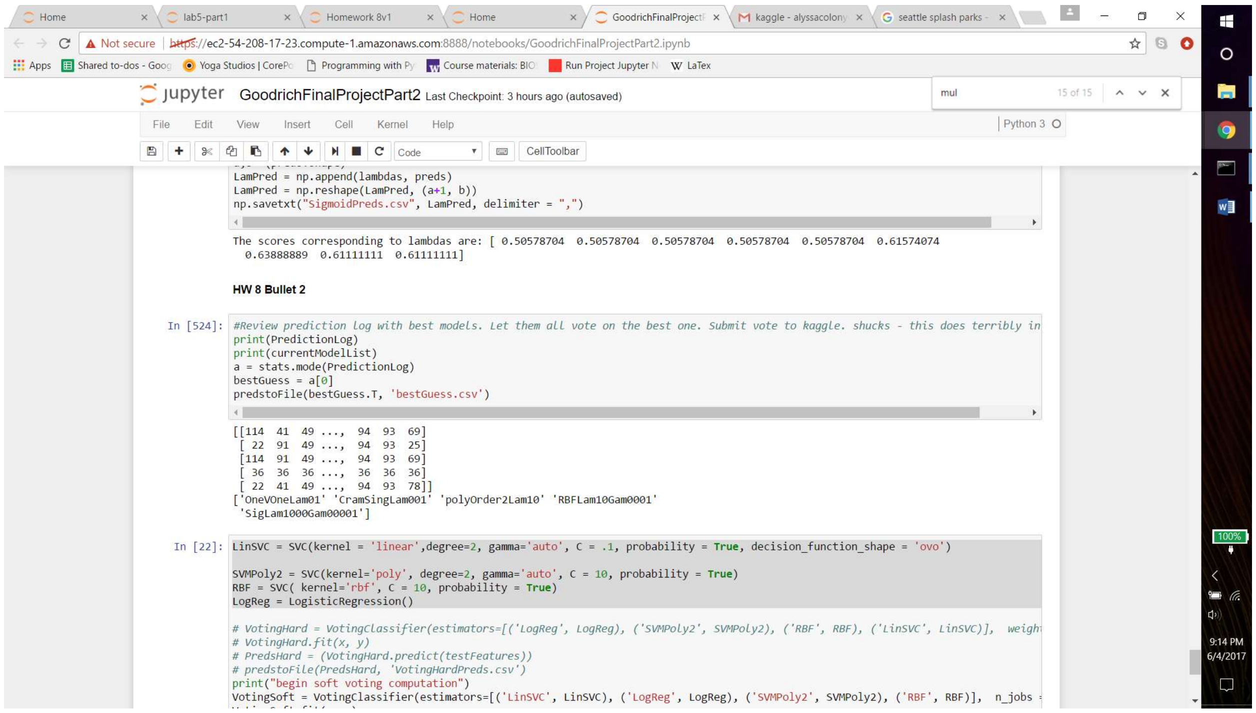

Not secure
~~https~~://ec2-52-3-234-80.compute-1.amazonaws.com:8888/notebooks/Project_Section4.ipynb#
jupyter
Project_Section4 Last Checkpoint: 3 minutes ago (autosaved)
threadpoo
1 of 8
File
Edit
View
Insert
Cell
Kernel
Help
Python 3
Code
CellToolbar

```
        newNamePred = labels[np.where(labels[:,1] == Cats[preds][i])[0],1]
        #print(i,newCatPred,newNamePred)
        numPreds = np.append(numPreds, newCatPred)
        namPredsCheck = np.append(namPredsCheck, newNamePred)
    return(numPreds)
```

```
In [473]: #create a function to keep a table of all predictions I have made and the name of model that created each prediction
currentModelList = ()
predictionLog
def predictionLog(predictions, modelName, currentModelList = currentModelList, PredictionLog = PredictionLog):
    #predictions are the predictions you want to add to the log
    #ModelName is the model that created the predictions you want to add
    #CurrentModelList is a list of models with predictions currently in the log (If ModelName is not on currentModelList then the
    if len(currentModelList) == 0:
        PredictionLog = predictions
        currentModelList = np.append(currentModelList, modelName)
        PredictionLog = PredictionLog
    elif modelName in currentModelList:
        index = np.where(currentModelList == modelName)[0]
        PredictionLog[index,:] = deepcopy(predictions) #[:, np.newaxis])
    else:
        index = len(currentModelList)
#         print('PredictionLog.shape:',PredictionLog.shape, PredictionLog.ndim)
#         print('predictions.shape:',predictions.shape)

        PredictionLog = np.vstack((PredictionLog, predictions))
        currentModelList = np.append(currentModelList, modelName)
    PredictionLog = np.array(PredictionLog)
    return(PredictionLog, currentModelList)
```

```
In [469]: PredictionLog, currentModelList = predictionLog(OneVRPredsCS, 'CramSingLam001', currentModelList = currentModelList, PredictionLo
```



```
LamPred = np.append(lambdas, preds)
LamPred = np.reshape(LamPred, (a+1, b))
np.savetxt("SigmoidPreds.csv", LamPred, delimiter = ",")
```

```
The scores corresponding to lambdas are: [ 0.50578704  0.50578704  0.50578704  0.50578704  0.50578704  0.61574074
  0.63888889  0.61111111  0.61111111]
```

HW 8 Bullet 2

In [524]:
```
#Review prediction log with best models. Let them all vote on the best one. Submit vote to kaggle. shucks - this does terribly in
print(PredictionLog)
print(currentModelList)
a = stats.mode(PredictionLog)
bestGuess = a[0]
predstoFile(bestGuess.T, 'bestGuess.csv')
```

```
[[114  41  49 ...,  94  93  69]
 [ 22  91  49 ...,  94  93  25]
 [114  91  49 ...,  94  93  69]
 [ 36  36  36 ...,  36  36  36]
 [ 22  41  49 ...,  94  93  78]]
['OneVOneLam01' 'CramSingLam001' 'polyOrder2Lam10' 'RBFLam10Gam0001'
 'SigLam1000Gam00001']
```

In [22]:
```
LinSVC = SVC(kernel = 'linear',degree=2, gamma='auto', C = .1, probability = True, decision_function_shape = 'ovo')

SVMPoly2 = SVC(kernel='poly', degree=2, gamma='auto', C = 10, probability = True)
RBF = SVC( kernel='rbf', C = 10, probability = True)
LogReg = LogisticRegression()

# VotingHard = VotingClassifier(estimators=[('LogReg', LogReg), ('SVMPoly2', SVMPoly2), ('RBF', RBF), ('LinSVC', LinSVC)],  weigh
# VotingHard.fit(x, y)
# PredsHard = (VotingHard.predict(testFeatures))
# predstoFile(PredsHard, 'VotingHardPreds.csv')
print("begin soft voting computation")
VotingSoft = VotingClassifier(estimators=[('LinSVC', LinSVC), ('LogReg', LogReg), ('SVMPoly2', SVMPoly2), ('RBF', RBF)],  n_jobs
```