

# Kaggle Data Competition

Dane Jordan



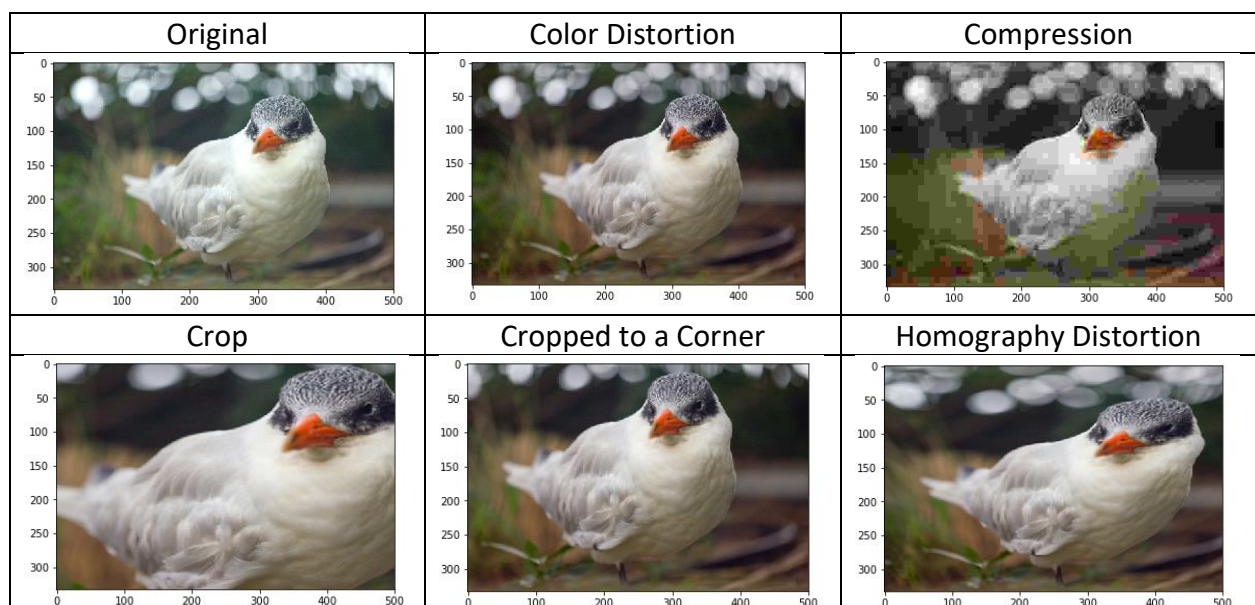
## Introduction

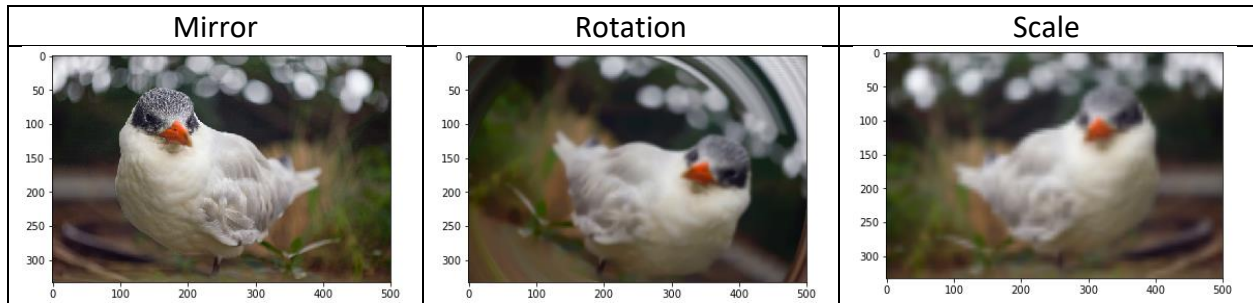
The goal of this Kaggle data competition is, “to learn how to use machine learning algorithms, and gain understanding about them...<through> bird species classification.” In this paper, I describe the progression from the very beginning, extracting features using tensorflow on Amazon Web Services (AWS), through my explorations into several various models and data manipulation, eventually ending with the understanding gained through the data competition and further research that could be performed to potentially improve classification using images.

## Feature Extraction using tensorflow on AWS and First Milestone

Before using any algorithms and models the data had to first be obtained. This process was done on AWS using a convolutional neural network called tensorflow, that utilizes the computational power of graphics processing units (GPUs). The pre-trained deep learning model that was used to extract feature vectors for the images of birds was Inception-V3, developed by Google.

After spending the better part of twelve hours on AWS using the Kernix article, “Image classification with a pre-trained deep neural network,” I obtained feature data for the test and train bird images. I also manipulated the training images with help from Corinne’s lab4-part2.ipynb, including color distortion, compression, cropping, cropping to a corner, homography distortion, mirroring, slight rotation, and scaling. My notes and code for extracting features can be found in the file *Notes\_FeatureExtraction.html*.





Although it was frustrating at times, I did learn how to use AWS effectively and enhanced my command line skills through AWS CLI. I see the benefit of using virtual machines, although in the future I would most likely use ones in a more stable environment; continually getting the message “connection reset by peer” and having to log back into an EC2 instance was not as efficient as using my local machine.

After obtaining the feature data for the train and test images I could begin exploring the data using logistic classification, specifically my own fast gradient algorithm from the homework assignments. This was performed on a small subset of the train data using two classes rather than all 144. After choosing a regularization parameter of 1 and training my algorithm, I used scikit-learn’s `LogisticRegressionCV()` to find the optimal regularization parameter and retrained my fast gradient algorithm, plotting the misclassification error for both versions (optimal regularization parameter, and when it was equal to 1). The python code for this first exploration into the feature data is included in the attached files *MilestoneHW4.html* and *MilestoneHW4.py*.

## Exploration in Logistic Classification and the Second Milestone

The next logical progression was to scale my logistic classification model to a multiclass situation as described in the next milestone. Rather than scaling it specifically for five classes, I chose to scale it in such a way that it could be applied to any number of classes. I then applied it to the first five classes in the data set. Up until this point I had only worked with binary classification in supervised learning situations. I learned that it was important to stratify my data when splitting between train and test, especially when dealing with many classes that do not include a significant number of observations. 144 classes with only 30 images each, falls into such a category of classification problems.

Scaling from two classes up to five classes was not difficult. I focused on scaling my own fast gradient algorithm by inserting it into another function that handled a one-vs-rest classification problem. While doing this I continuously compared my own algorithm’s coefficients to sci-kit learn’s coefficients obtained using `LogisticRegression()`. This gave me confidence that I was correctly scaling up my own algorithm. After verifying that my algorithm was matching sci-kit

learn's `LogisticRegression()`, I predicted the labels on the test data set that I had previously split off, and found that depending on the split it was somewhere between 85 – 100% accurate. I also wrote a function to display the misclassification overall as well as by class to see which classes were underperforming with my algorithm. Writing the code to perform cross-validation with five classes was a very informative exercise. I learned that while it is very possible to perform cross-validation using a parameter grid, it is not reasonable to try to scale such a cross-validation to a large multiclass problem. Even testing five different regularization parameters creates a problem that is five to the fifth power (3,125 combinations), not including running multiple folds. The python code can be found in the attached files *MilestoneHW5.html* and *MilestoneHW5.py*. Command line output from AWS of the multiclass misclassification error and confusion matrix can be seen in *Appendix B* under *Milestone Homework 5 (part 1)* and *Milestone Homework 5 (part 2)*.

Since I had planned to scale my algorithm for any number of classes, the next step was to apply my scaled algorithm to the full data set and possibly make my first submission in the Kaggle competition with my own algorithm. I began by splitting the data into a train and test set and standardizing both according to the 75% training set. For sake of efficiency, I also reduced the dimensionality from 2,048 features to approximately 600 using principal component analysis (PCA). Using sci-kit learn's `LogisticRegressionCV()` to find the optimal regularization parameter, I used this value within my own algorithm and trained it for all 144 classes. Simultaneously, I also trained a model using sci-kit learn to verify that the results were like that of my own algorithm. The results of the coefficients and predictions were nearly identical giving me confidence that I had correctly scaled up my algorithm.

I submitted sci-kit learn's results and achieved approximately 56% correct on Kaggle. My next submission ended up being a 0% as the file I uploaded was in float format rather than integers. When I did manage to upload my own logistic classification using my fast gradient algorithm, my score was much lower than predicted. It turns out that I had forgotten to standardize the actual test set that is used to submit predictions on Kaggle. Once this was fixed, I reran my algorithm and achieved approximately 58% correct. This was a beneficial learning experience that lead to the creation of a function that both split and standardized all necessary data as well as another function that exported a prediction in the correct format to a csv. At the same time, I also decided to create a PCA decomposition function that applied to all three data sets: train, test, and the true test set. My own algorithm can be seen in the file *Exploration\_Logistic.html* and reproduced using the file *Exploration\_Logistic.py*.

After successfully submitting my own algorithm, I began experimenting using sci-kit learn's and choosing between one-vs-one, one-vs-rest, and multinomial. Both one-vs-one and multinomial performed quite well, and one-vs-rest fell somewhat short of the other two. This makes sense as one-vs-one is performing several more comparisons, however, the one-vs-one method is not nearly as efficient as the other two and computationally takes a significant amount of time (note that I always ran PCA prior to fitting any model while experimenting to improve efficiency

in the number of models I was running). Finally, after experimenting with sci-kit learn's LogisticRegressionCV() I tried introducing added features, stacking them one at a time on top of the base features (i.e. base features + mirror features, or base features + homography features). At first, I thought it was going to be great as I achieved very high accuracy scores against the test set which was split off after stacking the added features. After making a submission to Kaggle with mirror features stacked, I realized that the accuracy scores I was receiving were inflated. My assumption is that I was overfitting the data and needed to have performed image transformations on the true test set as well. Still, looking at the various features stacked on top of the base features one at a time provided insight into which features were more beneficial achieving a higher prediction accuracy.

My exploration with logistic classification can be seen in the file *Exploration\_Logistic.html*, and can be rerun in the file *Exploration\_Logistic.py*. It will utilize my saved models on S3 and should run much more quickly than the original code.

## Exploration in Linear SVMs and the Third Milestone

Feeling as though I had a good grasp of how the logistic classification models worked and were performing, I proceeded onto the next milestone dealing with support vector machines (SVMs) as well as my own exploration into them. More specifically I familiarized myself with linear SVMs. As I did with logistic classification, I first worked with my own fast gradient algorithm to implement my own version of a linear SVM. Running in parallel with sci-kit learn's LinearSVC(), I could utilize my own algorithm to achieve the same coefficient estimates and similar accuracy predictions as that of LinearSVC(). This can be seen in the file *Exploration\_LinearSVC.html* and reproduced using the file *Exploration\_LinearSVC.py*. Again, I made two submissions using both my own implementation of a linear SVM as well as sci-kit learn's LinearSVC(). Both models achieved an accuracy score of approximately 65%.

Like my exploration using logistic classification, I also explored the linear SVMs using sci-kit learn after successfully running my own algorithm. I once again noticed that the one-vs-one and multiclass (Crammer-Singer) outperformed the one-vs-rest enough to be significant. Also, the one-vs-one approach again took a long time before fitting the model—nearly six or seven hours! Subsequently, after completing the homework milestones, I immediately began using PCA again to increase efficiency as 95% of the variance was accounted for in approximately 600 component vectors.

By this point I felt comfortable in my methodology. I was successfully splitting and standardizing my data throughout all models, and implementing cross-validation and PCA correctly such that I could make submissions representative of what I expected using my train/test split and 'prediction' function. I also experimented with different split sizes of my data, thinking that



increasing the number of observations would yield higher results against the Kaggle set. While this was true to some degree, it appeared marginal at best.

Linear SVMs performed nearly the same as logistic classification. The one-vs-one method may have been slightly better using linear SVMs, however, the multinomial logistic and multiclass (Crammer-Singer) SVM were on par with each other and significantly more efficient than running one-vs-one models.

After exploring with the various multiclass classification methods, I again moved on to using the added feature data and found similar results to that of logistic classification. I believe that my data was overfitting once again, and that I should have done the same image transformations to the test set of data as well. Still, the feedback from the image transformations being added in one-at-a-time was useful to determine which transformation could be useful in aiding predictions. I did try experimenting with stacking multiple transformations (mirror and homography as they were the two highest performing on an individual level). The results did slightly increase my prediction accuracy, but as I found with adjusting the split sizes, the results were marginal at best.

My exploration with linear SVMs can be seen in the file *Exploration\_LinearSVC.html*, and can be rerun in the file *Exploration\_LinearSVC.py*. It will utilize my saved models on S3 and should run much more quickly than the original code. Other related files are the homework milestone files *MilestoneHW7.html* and *MilestoneHW7.py*.

## Exploration in Kernel SVMs

When I finally began looking at the kernel SVMs I had a solid methodology down for testing the performance of various models. I again split and standardized my data, ran PCA and cross-validation to find my optimal regularization parameter and subsequently fitted my models. For the radial basis function (RBF) kernel I tried varying the gamma parameter and running cross-validation over and over for the regularization parameter, however, I didn't feel as though I could get the RBF kernel to perform very well. The results I achieved were slightly better than that of one-vs-rest for logistic classification and linear SVMs, but fell short of the one-vs-one/multinomial logistic classification and one-vs-one/multiclass (Crammer-Singer) linear SVMs.

I then proceeded to investigate the polynomial kernel. Performing the same routine (split, standardize, PCA, and cross-validate), I attempted using different orders for the polynomial kernel including orders 1 thru 7. While higher order polynomials appeared to fit the data very well, they fit it 'too' well. When compared to the hold out set from the split, I noticed it was severely overfitting my training set. With order 1 being a decent score (essentially linear), and order 2 increasing in prediction accuracy, order 3 began to decline, with order 7 eventually predicting only 10% of my hold out set accurately.

Once again, I tried adding in the additional feature data, but to no avail except for marginal increases. My explorations with kernel SVMs can be seen in the file *Exploration\_SVC.html*, and can be rerun in the file *Exploration\_SVC.py*. As with the logistic classification and linear SVM .py files, it will utilize my saved models on S3 to run more quickly than the original code.

## Exploration in Subsampling/Resampling and Ensembles of Models

After spending days on the three previous types of models I began asking myself how I could improve my score. Yes, there were literally days (and nights) worth of time spent ‘playing’ with several possibilities. Despite it being stressful at times, I did learn a great deal simply by adjusting the models using some common sense and much of what I learned throughout the quarter. To improve my score, I considered taking a majority vote of multiple models; this was before the lecture regarding ensemble models. Since I had been saving all my model predictions (Kaggle submissions), I decided to just take each of the files and take a majority vote across each image. Although it seemed surprisingly simple, it did end up working and adding an additional 2% to my highest accuracy score so far.

I then wondered if running all my ‘additional feature’ model predictions in conjunction with the base models across logistic classification, linear SVMs, and kernel SVMs would increase my prediction accuracy. I tried this, and it took quite some time to run 72 models and take a majority vote even after saving them as fitted pickle files. While it performed quite well, I was surprised to find that the accuracy on Kaggle was slightly lower than my highest score achieved already which was the majority vote of my prior submissions.

Up to this point I had been using random states in my splits to be able to save the models as pickle files and guarantee that running PCA with 95% explained variance would yield the same number of component vectors. This also means that I had been performing PCA on all models to be more efficient. Reproducibility was important when some models took nearly an hour to fit with cross-validation.

I felt as though I had done significant exploration and gained an immense understanding of the models I was working with, but now I wanted to see if I could push my accuracy any higher. I determined that the best performing models from each ‘group’ were multinomial logistic classification, multiclass (Crammer-Singer) linear SVMs, and polynomial (order 2) kernel SVMs. No longer performing PCA or including a random state in my split, I wrapped these three models with cross-validation into an iterative loop that took a subsample of the training data differently each loop, but still with the possibility to select the same image across multiple loops. Looping through ten times, I ultimately received 30 sets of predictions that I took a majority vote from. This again improved my accuracy score on Kaggle by approximately 2%.

I attempted adding the additional feature data I had into my new methodology, and it took quite a bit of time to run. Once it completed and I had the majority vote, I made the submission

on Kaggle which ultimately feel slightly short of my highest score. While polishing this code and trying to make it more efficient for submission purposes, I ended up running it three more times to make sure I had not done anything to corrupt its integrity. Out of curiosity, I compiled these three additional runs to my 30 other predictions from the first run, and took the majority vote across all 120 model predictions. This was ultimately my highest score that is out there. The code that achieved this score is in the files *Competition\_Submission.py* and *start.py*. These files are effectively identical, the latter being included due to the instructions on Kaggle. Keep in mind that my highest score was achieved based on 40 iterations through the loop rather than the 10 that are currently included in the code.

## Further Explorations and Conclusions

I have several more ideas that I would have enjoyed trying out and more exploration with ensemble methods that could have been done, however, with the time available I am unable to test everything I would like to. It appears that a majority vote method across multiple models works quite well. I believe that to increase the performance of this I should choose models that perform well against a held-out set, but a larger variance between the models being used. If the models have similar variance among them, then specific classes may suffer. Having increased variance with high prediction accuracy among models would be beneficial for these classes.

Although I spent a significant amount of time on AWS transforming images and resaving copies of all transformed images to extract completely new feature data sets from, I do not believe I fully grasped an understanding of the use of these images. Had I also transformed the test images in the same way as the train images, I may have noticed different results.

To try and increase prediction accuracy I would have also liked to further explore specific classes that were underperforming when looking at my train/test split and the results. By identifying the underperforming classes, I may have been able to find methods or models that increased the prediction accuracy of these specific classes, and then including them in my ensemble methodology.

I have come a long way since the beginning of the quarter, and even more so since exploring this data competition. I recognize that one-vs-rest performs fine on smaller data sets, but is outperformed by one-vs-one and multiclass on larger data sets. Also, one-vs-one runs quite quickly on smaller data sets but is significantly hindered by computational time when it is scaled to larger data sets. The differences in performance between logistic classification, linear SVMs and kernel SVMs is highly dependent on the data, but they all can perform quite well given the right situation. Ensemble methods may further increase the prediction accuracy, but also risk overfitting the data.

Despite some very late nights and stressful situations, I find machine learning to be quite fun. The exposure to Python, AWS, machine learning methods, and the interactions between each



of these has been extremely beneficial. I hope to participate in more Kaggle competitions in the future.

# Appendix A

## Listing of Attached Files

Please see the additional homework milestone files that were included in the submission on canvas:

- Notes\_FeatureExtraction.html -  
[https://s3.amazonaws.com/stat558drjordankaggle/Competition/Notes\\_FeatureExtraction.html](https://s3.amazonaws.com/stat558drjordankaggle/Competition/Notes_FeatureExtraction.html)
- MilestoneHW4.html -  
<https://s3.amazonaws.com/stat558drjordankaggle/Competition/MilestoneHW4.html>
- MilestoneHW5.html -  
<https://s3.amazonaws.com/stat558drjordankaggle/Competition/MilestoneHW5.html>
- MilestoneHW7.html -  
<https://s3.amazonaws.com/stat558drjordankaggle/Competition/MilestoneHW7.html>
- MilestoneHW8.html -  
<https://s3.amazonaws.com/stat558drjordankaggle/Competition/MilestoneHW8.html>

Please see the additional exploration files that were included in the submission on canvas for logistic classification, linear support vector machines, kernel support vector machines, and subsampling/resampling/ensembles of models:

- Exploration\_LinearSVC.html -  
[https://s3.amazonaws.com/stat558drjordankaggle/Competition/Exploration\\_LinearSVC.html](https://s3.amazonaws.com/stat558drjordankaggle/Competition/Exploration_LinearSVC.html)
- Exploration\_Logistic.html -  
[https://s3.amazonaws.com/stat558drjordankaggle/Competition/Exploration\\_Logistic.html](https://s3.amazonaws.com/stat558drjordankaggle/Competition/Exploration_Logistic.html)
- Exploration\_SVC.html -  
[https://s3.amazonaws.com/stat558drjordankaggle/Competition/Exploration\\_SVC.html](https://s3.amazonaws.com/stat558drjordankaggle/Competition/Exploration_SVC.html)

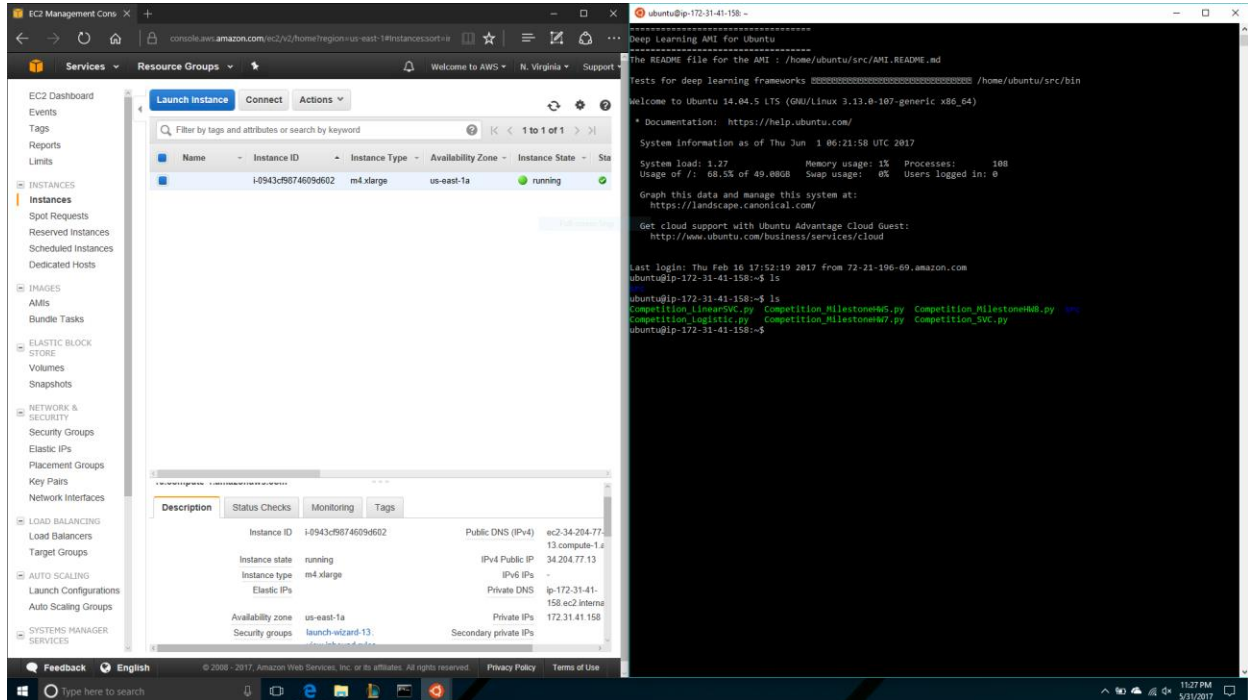
Please see the additional .py files in the submission on canvas. **NOTE** that these .py files run using saved models on AWS's S3 storage for efficiency purposes. The final two, *Competition\_Submission.py* and *start.py* do not run using S3 pickle files:

- Homework Milestones
  - MilestoneHW4.py -  
<https://s3.amazonaws.com/stat558drjordankaggle/Competition/MilestoneHW4.py>
  - MilestoneHW5.py -  
<https://s3.amazonaws.com/stat558drjordankaggle/Competition/MilestoneHW5.py>
  - MilestoneHW7.py -  
<https://s3.amazonaws.com/stat558drjordankaggle/Competition/MilestoneHW7.py>
  - MilestoneHW8.py -  
<https://s3.amazonaws.com/stat558drjordankaggle/Competition/MilestoneHW8.py>
- Explorations into Models
  - Exploration\_Logistic.py -  
[https://s3.amazonaws.com/stat558drjordankaggle/Competition/Exploration\\_Logistic.py](https://s3.amazonaws.com/stat558drjordankaggle/Competition/Exploration_Logistic.py)
  - Exploration\_LinearSVC.py -  
[https://s3.amazonaws.com/stat558drjordankaggle/Competition/Exploration\\_LinearSVC.py](https://s3.amazonaws.com/stat558drjordankaggle/Competition/Exploration_LinearSVC.py)
  - Exploration\_SVC.py -  
[https://s3.amazonaws.com/stat558drjordankaggle/Competition/Exploration\\_SVC.py](https://s3.amazonaws.com/stat558drjordankaggle/Competition/Exploration_SVC.py)
- Example of Later Submission File (Subsampling/Resampling/Ensembles)
  - Competition\_Submission.py -  
[https://s3.amazonaws.com/stat558drjordankaggle/Competition/Competition\\_Submission.py](https://s3.amazonaws.com/stat558drjordankaggle/Competition/Competition_Submission.py)
  - start.py (same as file above) -  
<https://s3.amazonaws.com/stat558drjordankaggle/Competition/start.py>

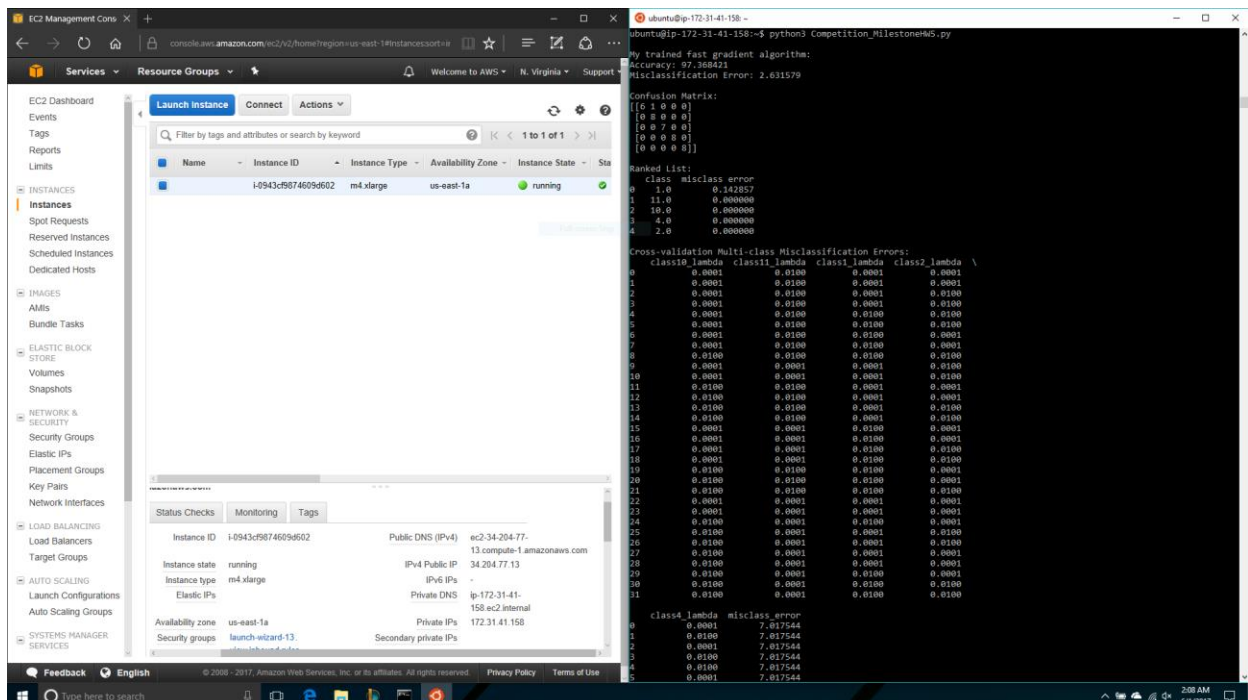
## Appendix B

### Amazon Web Services (AWS) Screenshots

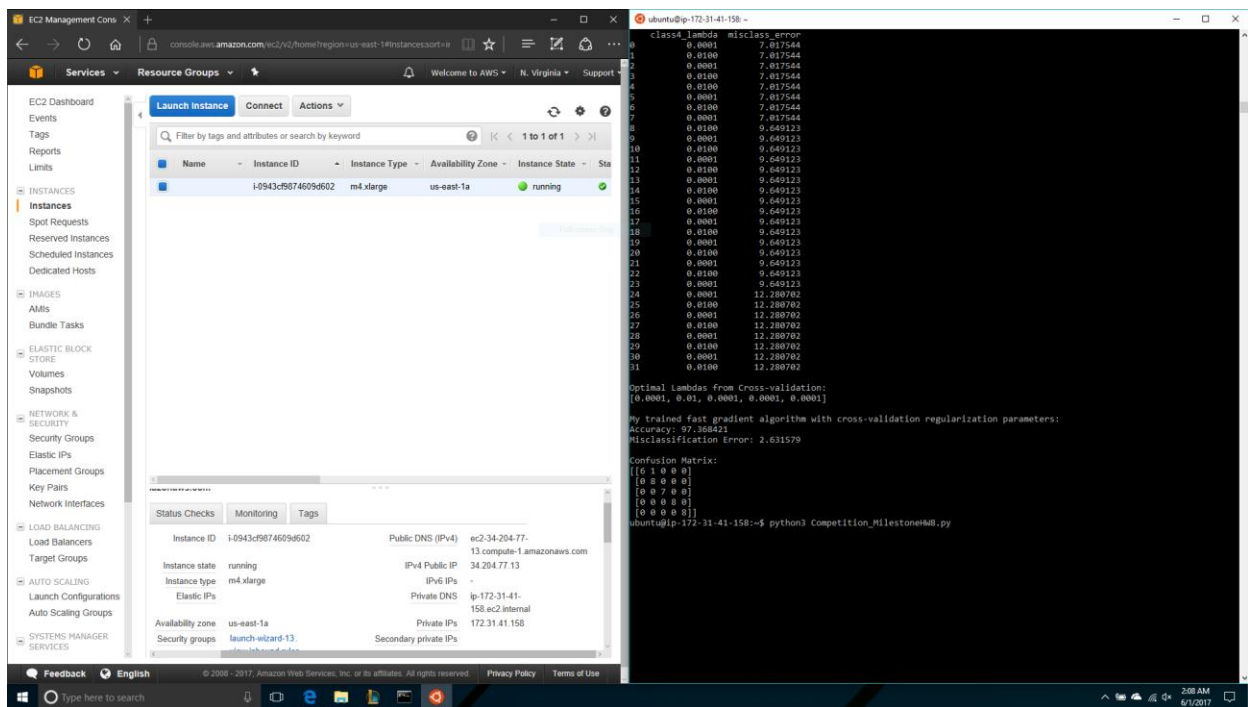
AWS login and transfer of .py files from local machine to EC2:



MilestoneHW5 (part 1):



## MilestoneHW5 (part 2):



The screenshot shows the AWS Management Console on the left and a terminal window on the right. The console displays the EC2 instance details for instance `i-0943c9874609d602`, which is a `m4.xlarge` instance in the `us-east-1a` availability zone, currently in a `running` state. The terminal window shows the output of a Python script, including a confusion matrix and classification results.

**EC2 Instance Details:**

Instance ID	Instance Type	Availability Zone	Instance State
i-0943c9874609d602	m4.xlarge	us-east-1a	running

**Terminal Output:**

```
classf_lambda misclass_error
0 0.0001 7.817544
1 0.0100 7.817544
2 0.0001 7.817544
3 0.0100 7.817544
4 0.0100 7.817544
5 0.0001 7.817544
6 0.0100 7.817544
7 0.0001 7.817544
8 0.0100 9.649123
9 0.0001 9.649123
10 0.0100 9.649123
11 0.0001 9.649123
12 0.0100 9.649123
13 0.0001 9.649123
14 0.0100 9.649123
15 0.0001 9.649123
16 0.0100 9.649123
17 0.0001 9.649123
18 0.0100 9.649123
19 0.0001 9.649123
20 0.0100 9.649123
21 0.0001 9.649123
22 0.0100 9.649123
23 0.0001 9.649123
24 0.0001 12.280702
25 0.0100 12.280702
26 0.0001 12.280702
27 0.0100 12.280702
28 0.0001 12.280702
29 0.0100 12.280702
30 0.0001 12.280702
31 0.0100 12.280702

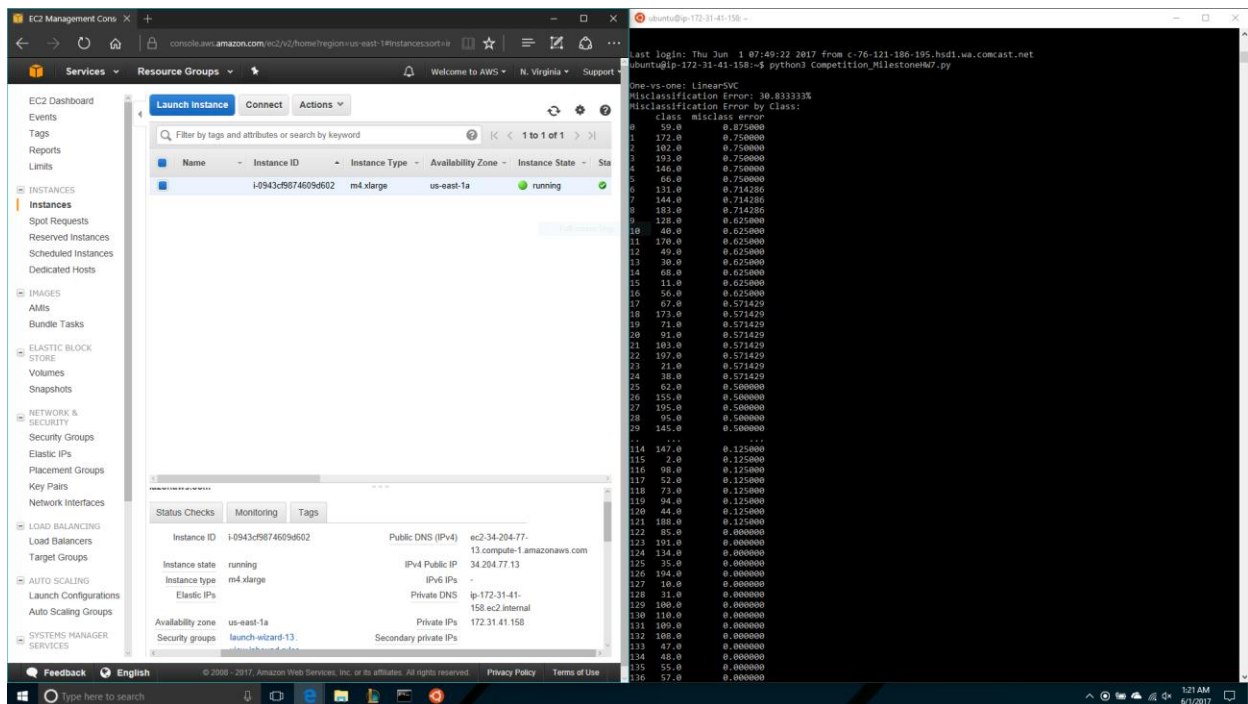
Optimal lambdas from Cross-validation:
[0.0001, 0.01, 0.0001, 0.0001, 0.0001]

My trained fast gradient algorithm with cross-validation regularization parameters:
Accuracy: 97.368421
Misclassification Error: 2.631579

Confusion Matrix:
[[0 1 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]

ubuntu@ip-172-31-41-158:~$ python3 Competition_MilestoneHW5.py
```

## MilestoneHW7 (part 1):



The screenshot shows the AWS Management Console on the left and a terminal window on the right. The console displays the EC2 instance details for instance `i-0943c9874609d602`, which is a `m4.xlarge` instance in the `us-east-1a` availability zone, currently in a `running` state. The terminal window shows the output of a Python script, including a confusion matrix and classification results.

**EC2 Instance Details:**

Instance ID	Instance Type	Availability Zone	Instance State
i-0943c9874609d602	m4.xlarge	us-east-1a	running

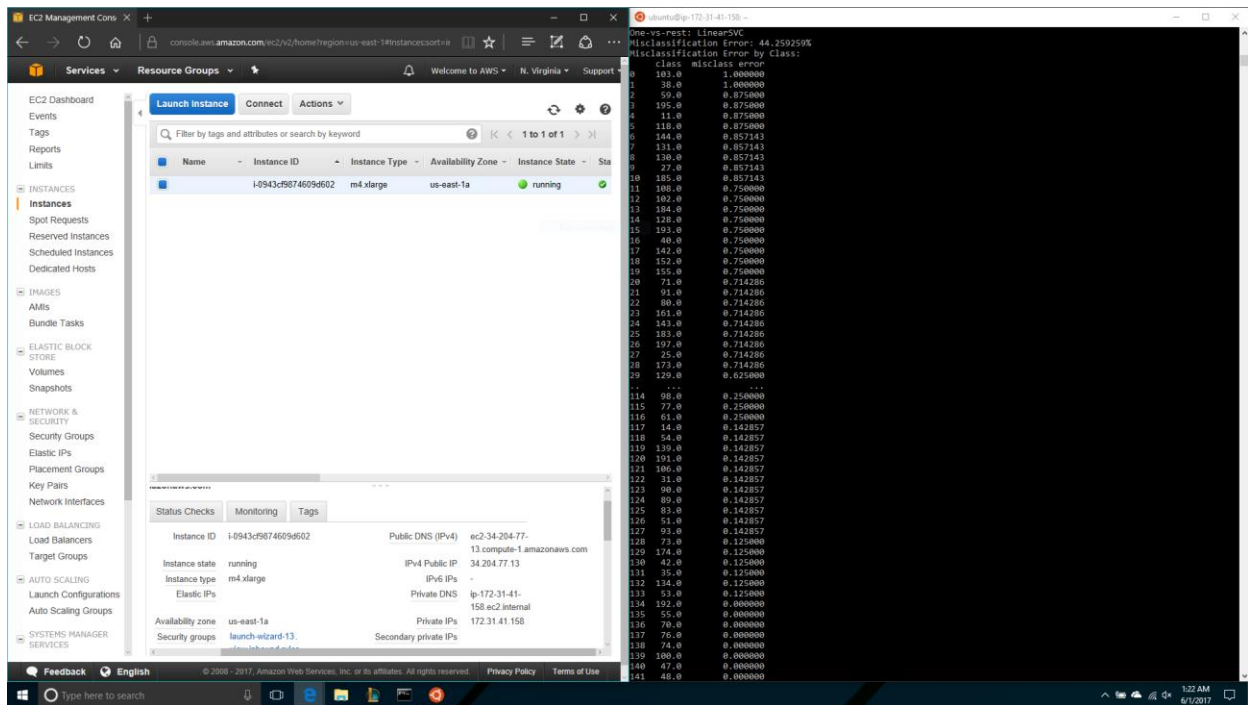
**Terminal Output:**

```
last login: Thu Jun 1 07:49:22 2017 from c-76-121-186-195.hsd1.wa.comcast.net
ubuntu@ip-172-31-41-158:~$ python3 Competition_MilestoneHW7.py

One-vs-one: LinearSVC
Misclassification Error: 30.833333%
Misclassification Error by Class:

class misclass_error
0 59.0 0.875000
1 172.0 0.750000
2 102.0 0.750000
3 193.0 0.750000
4 146.0 0.750000
5 66.0 0.750000
6 131.0 0.714286
7 144.0 0.714286
8 103.0 0.714286
9 128.0 0.625000
10 40.0 0.625000
11 170.0 0.625000
12 49.0 0.625000
13 30.0 0.625000
14 68.0 0.625000
15 11.0 0.625000
16 56.0 0.625000
17 67.0 0.571429
18 173.0 0.571429
19 71.0 0.571429
20 91.0 0.571429
21 103.0 0.571429
22 107.0 0.571429
23 21.0 0.571429
24 18.0 0.571429
25 62.0 0.500000
26 155.0 0.500000
27 195.0 0.500000
28 95.0 0.500000
29 145.0 0.500000
...
114 147.0 0.125000
115 2.0 0.125000
116 98.0 0.125000
117 52.0 0.125000
118 73.0 0.125000
119 94.0 0.125000
120 44.0 0.125000
121 188.0 0.125000
122 65.0 0.000000
123 191.0 0.000000
124 134.0 0.000000
125 35.0 0.000000
126 194.0 0.000000
127 10.0 0.000000
128 11.0 0.000000
129 180.0 0.000000
130 110.0 0.000000
131 109.0 0.000000
132 188.0 0.000000
133 47.0 0.000000
134 48.0 0.000000
135 55.0 0.000000
136 57.0 0.000000
```

## MilestoneHW7 (part 2):



The screenshot shows the AWS Management Console on the left and a terminal window on the right. The console displays the EC2 instance details for instance ID i-0943c9874609d602, which is a running m4.xlarge instance in the us-east-1a availability zone. The terminal window shows the output of a command, likely a classification task, displaying a list of misclassification errors by class.

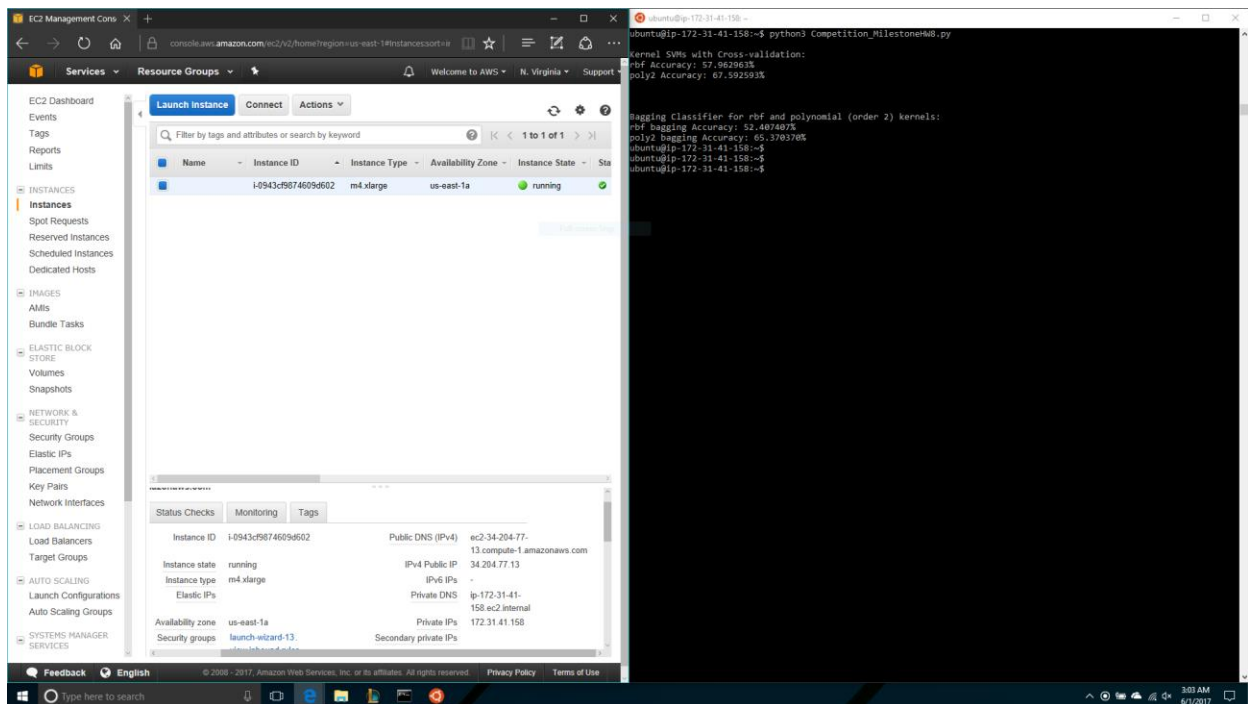
**EC2 Instance Details:**

Instance ID	Instance Type	Availability Zone	Instance State
i-0943c9874609d602	m4.xlarge	us-east-1a	running

**Terminal Output:**

```
One-vs-rest: LinearSVC
Misclassification Error: 44.259259%
Misclassification Error by Class:
class misclass error
0 103.0 1.000000
1 38.0 1.000000
2 59.0 0.875000
3 195.0 0.875000
4 11.0 0.875000
5 118.0 0.875000
6 144.0 0.857143
7 131.0 0.857143
8 130.0 0.857143
9 27.0 0.857143
10 185.0 0.857143
11 108.0 0.750000
12 102.0 0.750000
13 184.0 0.750000
14 128.0 0.750000
15 123.0 0.750000
16 40.0 0.750000
17 142.0 0.750000
18 152.0 0.750000
19 155.0 0.750000
20 71.0 0.714286
21 91.0 0.714286
22 88.0 0.714286
23 161.0 0.714286
24 143.0 0.714286
25 183.0 0.714286
26 197.0 0.714286
27 25.0 0.714286
28 173.0 0.714286
29 139.0 0.625000
...
114 98.0 0.250000
115 77.0 0.250000
116 61.0 0.250000
117 14.0 0.142857
118 54.0 0.142857
119 139.0 0.142857
120 191.0 0.142857
121 106.0 0.142857
122 31.0 0.142857
123 90.0 0.142857
124 89.0 0.142857
125 83.0 0.142857
126 51.0 0.142857
127 93.0 0.142857
128 73.0 0.125000
129 174.0 0.125000
130 42.0 0.125000
131 35.0 0.125000
132 134.0 0.125000
133 53.0 0.125000
134 192.0 0.000000
135 55.0 0.000000
136 70.0 0.000000
137 76.0 0.000000
138 74.0 0.000000
139 100.0 0.000000
140 47.0 0.000000
141 48.0 0.000000
```

## MilestoneHW8:



The screenshot shows the AWS Management Console on the left and a terminal window on the right. The console displays the EC2 instance details for instance ID i-0943c9874609d602, which is a running m4.xlarge instance in the us-east-1a availability zone. The terminal window shows the output of a command, likely a classification task, displaying the accuracy of a bagging classifier for rbf and polynomial (order 2) kernels.

**EC2 Instance Details:**

Instance ID	Instance Type	Availability Zone	Instance State
i-0943c9874609d602	m4.xlarge	us-east-1a	running

**Terminal Output:**

```
Kernel SVM with Cross-validation:
rbf Accuracy: 57.962963%
poly2 Accuracy: 67.592593%

Bagging classifier for rbf and polynomial (order 2) kernels:
rbf bagging Accuracy: 52.407407%
poly2 bagging Accuracy: 65.370370%
ubuntu@ip-172-31-41-158:~$
ubuntu@ip-172-31-41-158:~$
ubuntu@ip-172-31-41-158:~$
```

## Exploration of logistic classification:

The screenshot displays the AWS Management Console on the left and a terminal window on the right. The console shows an EC2 instance named 'i-0943c98746094602' of type 'm4.xlarge' in the 'us-east-1a' availability zone, currently in a 'running' state. The terminal window shows the output of a Python script named 'Competition\_logistic.py'.

```
ubuntu@ip-172-31-41-158:~$ python3 Competition_logistic.py
My own logistic one-vs-rest:
Accuracy: 58.85556%
Misclassification Error: 41.94444%

Sklearn LogisticRegression:
Accuracy: 58.85556%
Misclassification Error: 41.94444%

one-vs-rest
adding image transformations 1 at a time
base : 55.83333%
color : 56.89259%
compress : 45.83333%
crop : 54.49674%
crop_to_corner : 42.96296%
homography : 53.98148%
mirror : 56.15740%
rotate30 : 38.88889%
scale : 58.41666%

multinomial
adding image transformations 1 at a time
base : 71.28370%
color : 66.48148%
compress : 58.33333%
crop : 63.47222%
crop_to_corner : 65.46296%
homography : 89.81481%
mirror : 91.52778%
rotate30 : 49.72222%
scale : 68.28703%
ubuntu@ip-172-31-41-158:~$
```

## Exploration of linear support vector machines (SVM):

The screenshot displays the AWS Management Console on the left and a terminal window on the right. The console shows the same EC2 instance as in the previous image. The terminal window shows the output of a Python script named 'Competition\_linearSVC.py'.

```
ubuntu@ip-172-31-41-158:~$ python3 Competition_linearSVC.py
My own linear svm:
Accuracy: 65.37837%
Misclassification Error: 34.62963%

Sklearn LinearSVC:
Accuracy: 65.37837%
Misclassification Error: 34.62963%

one-vs-one
adding image transformations 1 at a time
base : 70.89259%
color : 66.99674%
compress : 57.63889%
crop : 64.21296%
crop_to_corner : 65.13889%
homography : 89.69185%
mirror : 92.45378%
rotate30 : 48.89259%
scale : 68.65740%

one-vs-rest
adding image transformations 1 at a time
base : 56.28370%
color : 52.12963%
compress : 47.54629%
crop : 66.94444%
crop_to_corner : 52.45378%
homography : 67.77778%
mirror : 69.67592%
rotate30 : 39.86111%
scale : 53.28703%

multinomial
adding image transformations 1 at a time
base : 69.72222%
color : 65.21148%
compress : 56.57407%
crop : 62.26810%
crop_to_corner : 64.58333%
homography : 88.65740%
mirror : 91.52778%
rotate30 : 46.59674%
scale : 66.75925%
ubuntu@ip-172-31-41-158:~$
```



## Exploration of kernel support vector machines (SVM):

The screenshot displays the AWS Management Console interface for an EC2 instance and a terminal window showing the output of a Python script.

**AWS Management Console:**

- Instance Details:** The instance `i-0943cf98746094602` is a `m4.xlarge` instance in the `us-east-1a` availability zone, currently in a `running` state.
- Status Checks:** The instance is running with the following details:
  - Instance ID: `i-0943cf98746094602`
  - Instance state: `running`
  - Instance type: `m4.xlarge`
  - Elastic IPs: None
  - Availability zone: `us-east-1a`
  - Security groups: `launch-wizard-13`
  - Public DNS (IPv4): `ec2-34-204-77-13.compute-1.amazonaws.com`
  - IPv4 Public IP: `34.204.77.13`
  - Private DNS: `ip-172-31-41-158.ec2.internal`
  - Private IP: `172.31.41.158`

**Terminal Output:**

```
ubuntu@ip-172-31-41-158:~$ python3 Competition_SVC.py
rbf
adding image transformations 1 at a time
base : 58.348148%
color : 51.898148%
compress : 44.814815%
crop : 72.453784%
crop_to_corner : 51.018519%
homography : 84.212963%
mirror : 86.898148%
rotate30 : 37.487487%
scale : 51.018519%
polynomial(order 2)
adding image transformations 1 at a time
base : 66.944444%
color : 61.283784%
compress : 51.527778%
crop : 83.425926%
crop_to_corner : 66.185185%
homography : 88.878388%
mirror : 92.777778%
rotate30 : 45.148148%
scale : 62.179248%
ubuntu@ip-172-31-41-158:~$
```

## Appendix C

### Amazon Web Services (AWS) S3 Storage (Pickle Files – Saved Models)

Pickle Files – Features:

US East (N. Virginia)

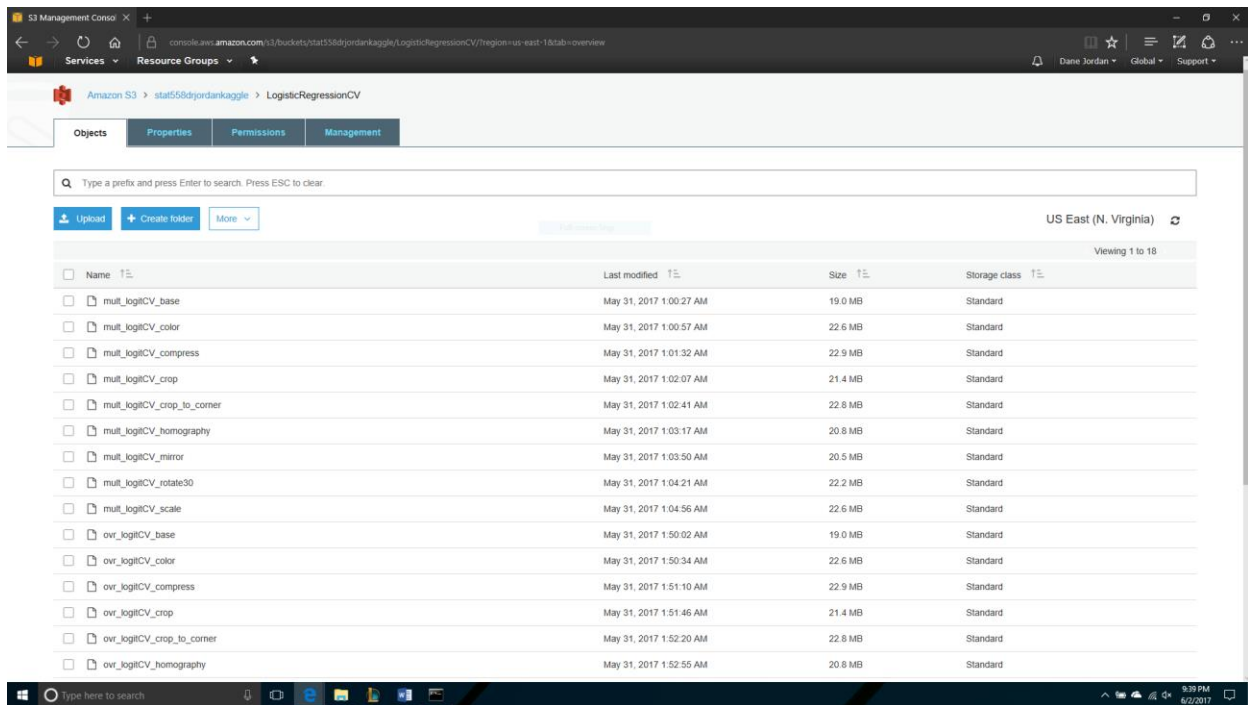
Name	Last modified	Size	Storage class
HW7 Milestone	--	--	--
LinearSVC	--	--	--
LogisticRegressionCV	--	--	--
SVC	--	--	--
color_features	May 16, 2017 5:15:27 AM	67.5 MB	Standard
compress_features	May 16, 2017 4:59:17 AM	67.5 MB	Standard
crop_features	May 16, 2017 4:55:14 AM	67.5 MB	Standard
crop_to_corner_features	May 16, 2017 5:23:02 AM	67.5 MB	Standard
homography_features	May 16, 2017 5:04:37 AM	67.5 MB	Standard
minor_features	May 16, 2017 4:51:55 AM	67.5 MB	Standard
rotate30_features	May 16, 2017 6:17:47 AM	67.5 MB	Standard
scale_features	May 16, 2017 5:10:47 AM	67.5 MB	Standard
test_features	May 16, 2017 2:21:50 AM	67.5 MB	Standard
train_features	May 16, 2017 2:22:42 AM	67.5 MB	Standard
train_labels	May 16, 2017 2:23:18 AM	54.1 KB	Standard

Pickle Files – MilestoneHW7 Models

US East (N. Virginia)

Name	Last modified	Size	Storage class
mut_gridCV	May 30, 2017 9:07:56 PM	630.7 KB	Standard
mut_kernelgridCV	May 30, 2017 9:07:58 PM	16.9 MB	Standard
mut_linSVC	May 30, 2017 9:08:26 PM	628.5 KB	Standard
ovo_gridCV	May 30, 2017 9:08:27 PM	51.1 MB	Standard
ovo_kernelgridCV	May 30, 2017 9:09:46 PM	16.9 MB	Standard
ovo_linSVC	May 30, 2017 9:10:13 PM	51.1 MB	Standard
ovr_gridCV	May 30, 2017 9:11:32 PM	630.6 KB	Standard
ovr_kernelgridCV	May 30, 2017 9:11:33 PM	16.9 MB	Standard
ovr_linSVC	May 30, 2017 9:12:00 PM	628.5 KB	Standard

## Pickle Files – Logistic Classification Models



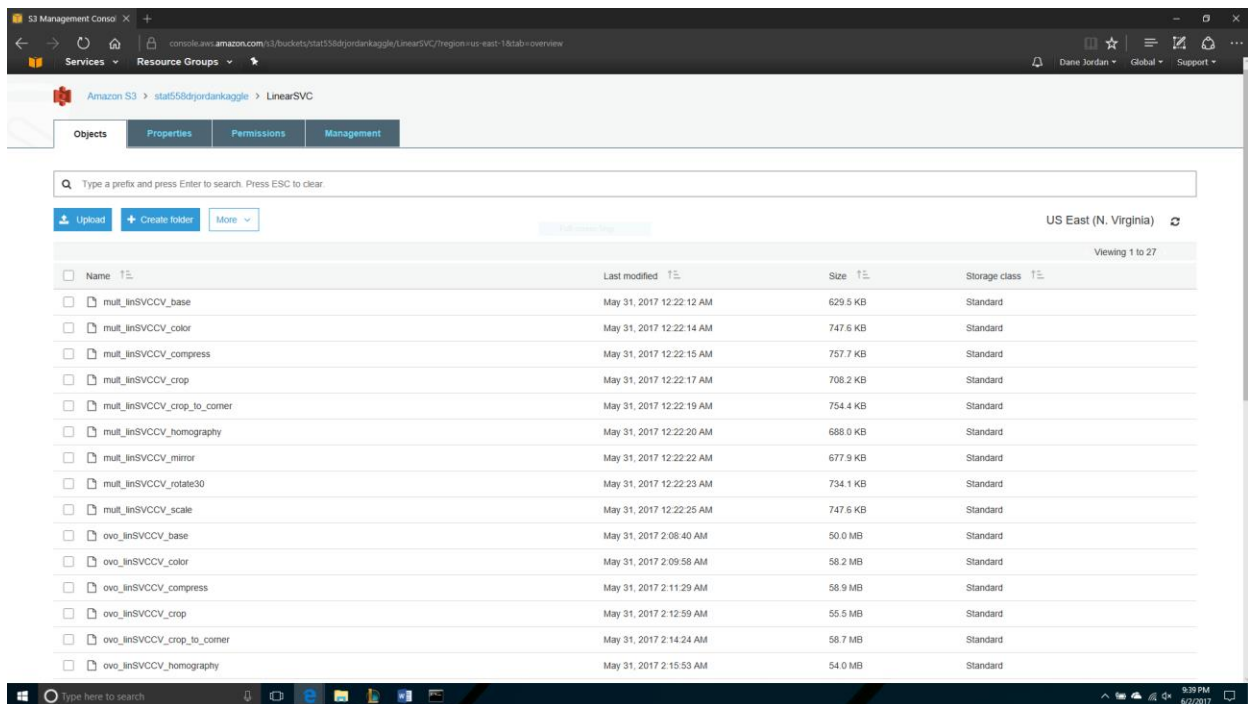
Amazon S3 > stat558jordankaggle > LogisticRegressionCV

US East (N. Virginia)

Viewing 1 to 18

Name	Last modified	Size	Storage class
mult_logitCV_base	May 31, 2017 1:00:27 AM	19.0 MB	Standard
mult_logitCV_color	May 31, 2017 1:00:57 AM	22.6 MB	Standard
mult_logitCV_compress	May 31, 2017 1:01:32 AM	22.9 MB	Standard
mult_logitCV_crop	May 31, 2017 1:02:07 AM	21.4 MB	Standard
mult_logitCV_crop_to_corner	May 31, 2017 1:02:41 AM	22.8 MB	Standard
mult_logitCV_homography	May 31, 2017 1:03:17 AM	20.8 MB	Standard
mult_logitCV_mirror	May 31, 2017 1:03:50 AM	20.5 MB	Standard
mult_logitCV_rotate30	May 31, 2017 1:04:21 AM	22.2 MB	Standard
mult_logitCV_scale	May 31, 2017 1:04:56 AM	22.6 MB	Standard
ovr_logitCV_base	May 31, 2017 1:50:02 AM	19.0 MB	Standard
ovr_logitCV_color	May 31, 2017 1:50:34 AM	22.6 MB	Standard
ovr_logitCV_compress	May 31, 2017 1:51:10 AM	22.9 MB	Standard
ovr_logitCV_crop	May 31, 2017 1:51:46 AM	21.4 MB	Standard
ovr_logitCV_crop_to_corner	May 31, 2017 1:52:20 AM	22.8 MB	Standard
ovr_logitCV_homography	May 31, 2017 1:52:55 AM	20.8 MB	Standard

## Pickle Files – Linear SVMs



Amazon S3 > stat558jordankaggle > LinearSVC

US East (N. Virginia)

Viewing 1 to 27

Name	Last modified	Size	Storage class
mult_linSVCV_base	May 31, 2017 12:22:12 AM	629.5 KB	Standard
mult_linSVCV_color	May 31, 2017 12:22:14 AM	747.6 KB	Standard
mult_linSVCV_compress	May 31, 2017 12:22:15 AM	757.7 KB	Standard
mult_linSVCV_crop	May 31, 2017 12:22:17 AM	708.2 KB	Standard
mult_linSVCV_crop_to_corner	May 31, 2017 12:22:19 AM	754.4 KB	Standard
mult_linSVCV_homography	May 31, 2017 12:22:20 AM	688.0 KB	Standard
mult_linSVCV_mirror	May 31, 2017 12:22:22 AM	677.9 KB	Standard
mult_linSVCV_rotate30	May 31, 2017 12:22:23 AM	734.1 KB	Standard
mult_linSVCV_scale	May 31, 2017 12:22:25 AM	747.6 KB	Standard
ovo_linSVCV_base	May 31, 2017 2:08:40 AM	50.0 MB	Standard
ovo_linSVCV_color	May 31, 2017 2:09:58 AM	58.2 MB	Standard
ovo_linSVCV_compress	May 31, 2017 2:11:29 AM	58.9 MB	Standard
ovo_linSVCV_crop	May 31, 2017 2:12:59 AM	55.5 MB	Standard
ovo_linSVCV_crop_to_corner	May 31, 2017 2:14:24 AM	58.7 MB	Standard
ovo_linSVCV_homography	May 31, 2017 2:15:53 AM	54.0 MB	Standard

## Pickle Files – Kernel SVMs

The screenshot displays the Amazon S3 Management Console interface. The breadcrumb navigation shows the path: Amazon S3 > stat558djordankaggle > SVC. The 'Objects' tab is selected, showing a list of files in the 'US East (N. Virginia)' region. The files are listed in a table with columns for Name, Last modified, Size, and Storage class. The files are all pickle files (.pkl) and are stored in the 'Standard' storage class. The table shows 20 files, with the first 10 visible in the screenshot.

Name	Last modified	Size	Storage class
poly2_SVCCV_base	May 31, 2017 12:26:06 AM	16.9 MB	Standard
poly2_SVCCV_color	May 31, 2017 12:26:34 AM	38.5 MB	Standard
poly2_SVCCV_compress	May 31, 2017 12:27:40 AM	39.3 MB	Standard
poly2_SVCCV_crop	May 31, 2017 12:28:41 AM	35.8 MB	Standard
poly2_SVCCV_crop_to_corner	May 31, 2017 12:29:37 AM	38.8 MB	Standard
poly2_SVCCV_homography	May 31, 2017 12:30:37 AM	34.8 MB	Standard
poly2_SVCCV_homography_mirror	May 31, 2017 12:31:32 AM	51.0 MB	Standard
poly2_SVCCV_mirror	May 31, 2017 12:32:50 AM	34.2 MB	Standard
poly2_SVCCV_rotate30	May 31, 2017 12:33:43 AM	38.6 MB	Standard
poly2_SVCCV_scale	May 31, 2017 12:34:43 AM	38.5 MB	Standard
rbf_SVCCV_base	May 31, 2017 12:35:43 AM	17.4 MB	Standard
rbf_SVCCV_color	May 31, 2017 12:36:11 AM	39.6 MB	Standard
rbf_SVCCV_compress	May 31, 2017 12:37:12 AM	40.1 MB	Standard
rbf_SVCCV_crop	May 31, 2017 12:38:14 AM	37.6 MB	Standard
rbf_SVCCV_crop_to_corner	May 31, 2017 12:39:13 AM	39.9 MB	Standard