Project 3 Report

Esteban Guillen

Intro

For Project 3 we were asked to classify songs into 10 different genres (blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, and rock) using the audio of each song. We were given the code to extract FFT and MFCC features. Our task was to create a third feature and see if we could improve classification performance.

Method for third feature extraction

After exploring a number audio libraries and reading a number of papers I decided to build my feature from the FFT and MFCC features. I came across a method for finding the most important features by using a forest of trees (source: Python Machine Learning by Sebastian Raschka). I combined FFT and MFCC features and fed them into the forest (ExtraTreesClassifier) and then extracted out the 50 strongest features to use for my feature. I called this feature Top 50 FFT and MFCC. I noticed early in testing that the MFCC features were producing stronger classification results (compared to FFT) and not surprisingly all the MFCC features (features 1000-1012) were included in the top 50 features. It was a nice surprise to see a few FFT features rank high on the list of top 50. This gave me hope that this feature set could produce better results than just MFCC alone. Below are are listing of the top 50 features and a graph showing their relative importance value (how strong of a feature it is). The top 50 features seemed to be the optimal value. I tried higher and lower values but the top 50 gave me the best results

```
Project3 — bash — 96×54
bash                    top
Estebans-MacBook-Pro:Project3 esteban$ python classify.py &
[1] 52079
Estebans-MacBook-Pro:Project3 esteban$
Feature ranking:
1. feature 1001 (0.009288)
2. feature 1000 (0.008343)
3. feature 1007 (0.004502)
4. feature 1002 (0.003987)
5. feature 1003 (0.003768)
6. feature 1005 (0.003298)
7. feature 0 (0.002771)
8. feature 1006 (0.002577)
9. feature 1009 (0.002232)
10. feature 991 (0.001759)
11. feature 1011 (0.001718)
12. feature 1004 (0.001689)
13. feature 105 (0.001647)
14. feature 1012 (0.001622)
15. feature 99 (0.001597)
16. feature 96 (0.001596)
17. feature 948 (0.001584)
18. feature 73 (0.001581)
19. feature 69 (0.001579)
20. feature 3 (0.001557)
21. feature 903 (0.001555)
22. feature 959 (0.001525)
23. feature 92 (0.001525)
24. feature 77 (0.001509)
25. feature 61 (0.001509)
26. feature 961 (0.001506)
27. feature 1010 (0.001503)
28. feature 968 (0.001491)
29. feature 962 (0.001490)
30. feature 93 (0.001478)
31. feature 114 (0.001461)
32. feature 997 (0.001458)
33. feature 981 (0.001456)
34. feature 89 (0.001455)
35. feature 79 (0.001452)
36. feature 109 (0.001431)
37. feature 998 (0.001427)
38. feature 31 (0.001414)
39. feature 62 (0.001411)
40. feature 29 (0.001403)
41. feature 984 (0.001399)
42. feature 95 (0.001395)
43. feature 75 (0.001394)
44. feature 990 (0.001391)
45. feature 101 (0.001390)
46. feature 1008 (0.001384)
47. feature 56 (0.001384)
48. feature 963 (0.001383)
49. feature 115 (0.001379)
50. feature 174 (0.001370)
```

Figure 1- Listing of Top 50 Features

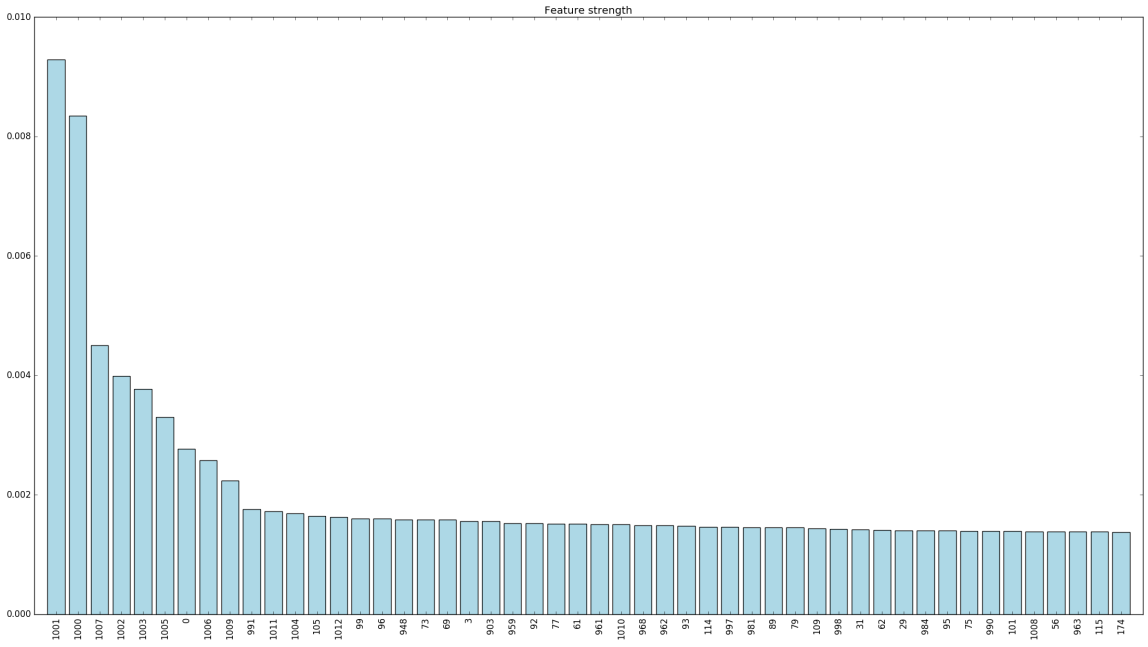

Figure 2- Graph of Top 50 FFT and MFCC Features

Reasons for choosing feature extraction method

I went with the top 50 feature approach after having little success using features found in other audio libraries. None of the ones that I tried (things like tempo, pitch, energy, etc.) gave me good results. In lecture the instructor alluded to combining and or reducing the FFT and MFCC features to build a third feature set. So I looked into PCA but didn't have much luck (possibly using it wrong). Finally, I came across the feature extraction method using a forest of trees and was happy with the results I got.

Random Forests

I chose to investigate random forests as one of my classifiers because I was interested in evaluating and ensemble approach. Building weak learners and combining them to produce a strong learner sounded like a good approach and I was building my third feature set from forest of trees, so using a random forest classifier seemed like a logical approach.

SVM

In just about every machine learning paper I read there seems to be some mention of SVM and most of the time they perform as good or better than the other approaches. I also had some experience with SVM in the past for Sentiment Analysis and had good success. Also after covering SVM in class recently and gaining some insight into the inner workings of them I though they would be a good choice for my second classifier. With SVM having the ability to build non-linear decision boundaries I though they would perform well.

Results

I divided the training data into a training set (80%) and test set (20%) and performed 10-fold cross validation on the training set. It is interesting to note that the two classifiers performed very similar with Random Forests having a slight edge for each feature set.

Table 1- Summary of Average CV Accuracy and 95% Confidence Interval for the 10 CV Accuracy Result Values

	FFT	MFCC	Top 50 FFT & MFCC
Random Forest	39.41%, +/- 2.54% [36.87 , 41.95]	54.49%, +/- 2.83% [51.66 , 57.32]	62.75%, +/- 2.75% [60.00 , 65.50]
SVM	39.04%, +/- 2.95% [36.10 , 41.99]	53.59%, +/- 3.75% [49.84 , 57.34]	59.82%, +/- 2.03% [57.79 , 61.85]

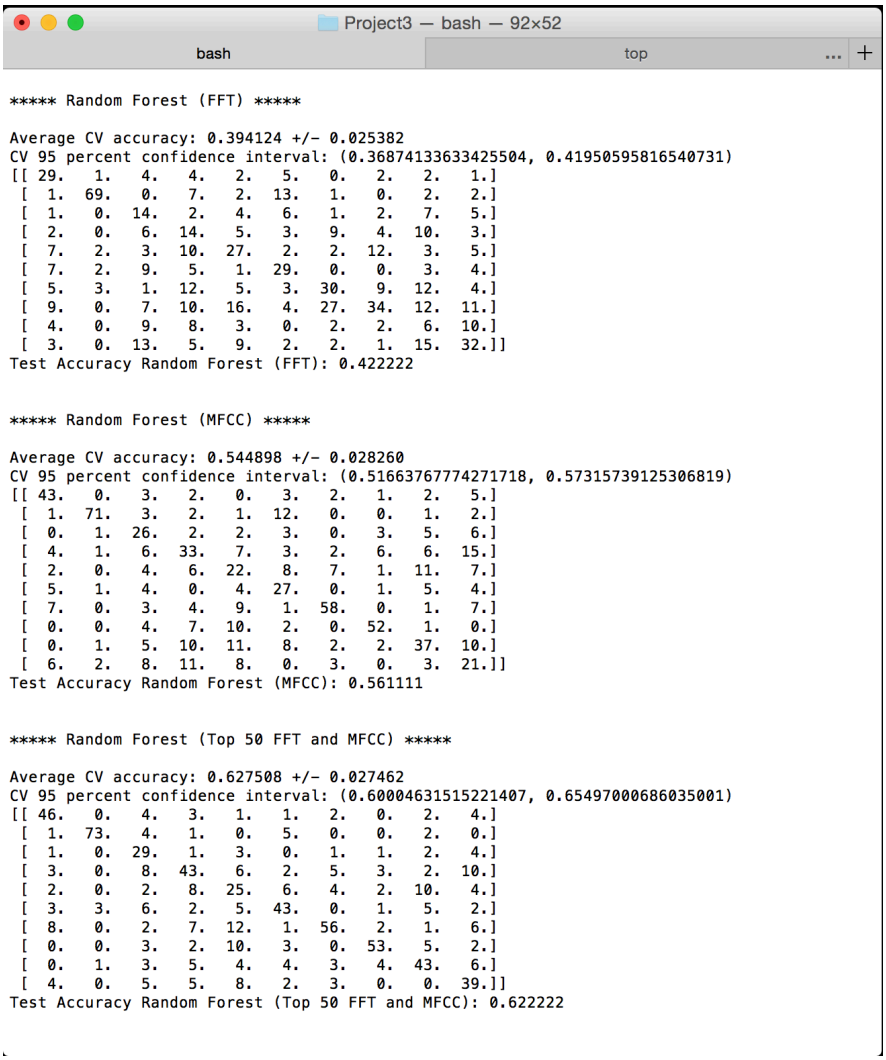

Figure 3- Console Output showing CV Accuracy, CV Confidence Interval, and Test Set Accuracy for Random Forest

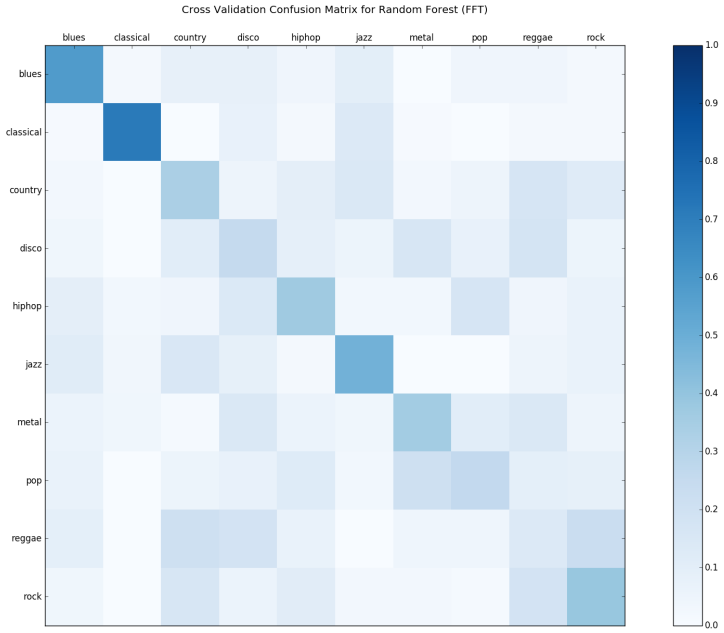

Figure 4- Confusion Matrix from CV runs for Random Forest using FFT

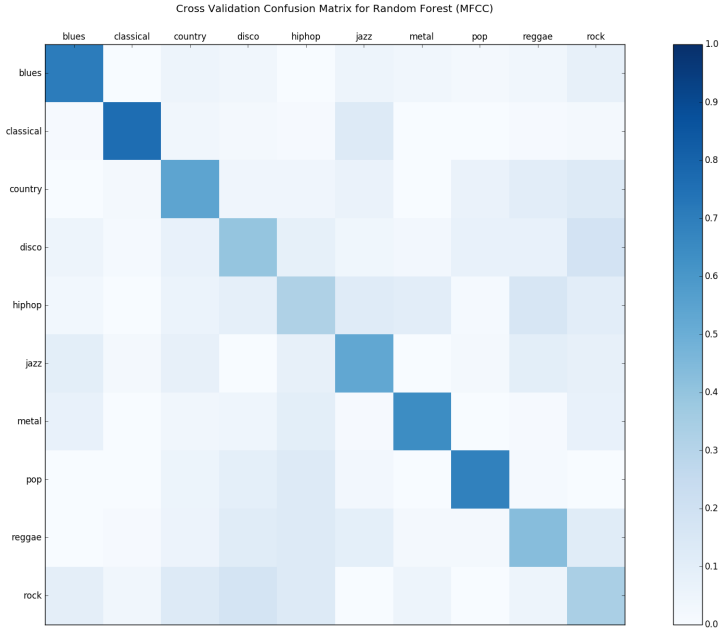

Figure 5- Confusion Matrix from CV runs for Random Forest using MFCC

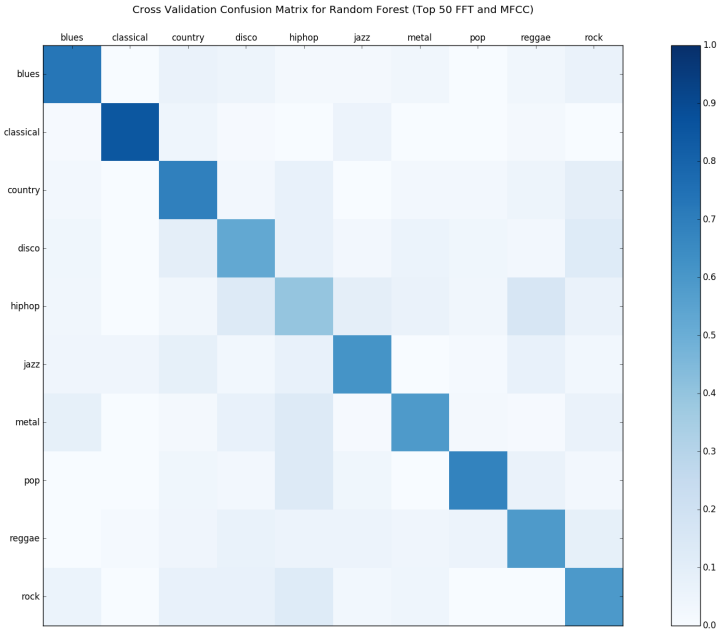

Figure 6- Confusion Matrix from CV runs for Random Forest using Top 50 FFT and MFCC

```
Project3 — bash — 92×52
bash                                    top

***** SVM (FFT) *****

Average CV accuracy: 0.390425 +/- 0.029458
CV 95 percent confidence interval: (0.36096637701511441, 0.41988307993380153)
[[ 36.   8.   5.   6.   7.   4.   6.   3.   5.   2.]
 [  9.  57.   0.   2.   3.   9.   1.   0.   2.   2.]
 [  3.   2.  27.  11.   5.   3.   1.   2.  10.   8.]
 [  1.   1.   8.  15.   7.   5.   9.   5.   6.   7.]
 [  6.   0.   4.   5.  18.   2.   5.  10.   2.   7.]
 [  1.   3.   4.   0.   0.  32.   0.   1.   1.   0.]
 [  4.   3.   2.   9.   1.   6.  31.   9.   9.   5.]
 [  6.   3.   7.  12.  20.   3.  11.  32.  11.  18.]
 [  2.   0.   5.   7.   5.   2.   5.   4.  11.   6.]
 [  0.   0.   4.  10.   8.   1.   5.   0.  15.  22.]]
Test Accuracy SVM (FFT): 0.438889


***** SVM (MFCC) *****

Average CV accuracy: 0.535904 +/- 0.037542
CV 95 percent confidence interval: (0.49836221349833143, 0.57344663326592982)
[[ 52.   2.   3.   4.   3.   8.   8.   1.   2.  10.]
 [  2.  70.   0.   2.   0.   7.   0.   0.   1.   0.]
 [  1.   0.  33.   8.   7.   6.   1.   3.   5.   7.]
 [  3.   0.   6.  33.  10.   5.   2.   6.  11.  13.]
 [  1.   0.   3.   6.  25.   4.  10.   7.  13.   8.]
 [  4.   3.   6.   5.   4.  32.   1.   1.   8.   5.]
 [  2.   0.   3.   1.   9.   0.  47.   0.   3.   3.]
 [  0.   0.   4.   5.   4.   1.   1.  46.   2.   1.]
 [  0.   2.   1.   7.   5.   3.   0.   1.  23.   7.]
 [  3.   0.   7.   6.   7.   1.   4.   1.   4.  23.]]
Test Accuracy SVM (MFCC): 0.577778


***** SVM (Top 50 FFT and MFCC) *****

Average CV accuracy: 0.598219 +/- 0.020284
CV 95 percent confidence interval: (0.57793472812303326, 0.61850265323117404)
[[ 52.   1.   1.   4.   5.   4.   5.   1.   5.   5.]
 [  2.  72.   0.   1.   0.   6.   0.   0.   1.   0.]
 [  2.   0.  42.  10.   0.   5.   2.   2.   4.   6.]
 [  2.   0.   8.  36.   6.   5.   3.   3.   8.  10.]
 [  1.   0.   1.   4.  27.   5.   7.   7.  10.   5.]
 [  1.   2.   4.   2.   4.  39.   1.   0.   2.   2.]
 [  5.   0.   0.   5.  12.   0.  49.   0.   3.   3.]
 [  1.   1.   1.   3.   7.   3.   1.  45.   3.   8.]
 [  1.   1.   3.   5.   8.   0.   2.   1.  32.   3.]
 [  1.   0.   6.   7.   5.   0.   4.   7.   4.  35.]]
Test Accuracy SVM (Top 50 FFT and MFCC): 0.583333
```

Figure 7- Console Output showing CV Accuracy, CV Confidence Interval, and Test Set Accuracy for SVM

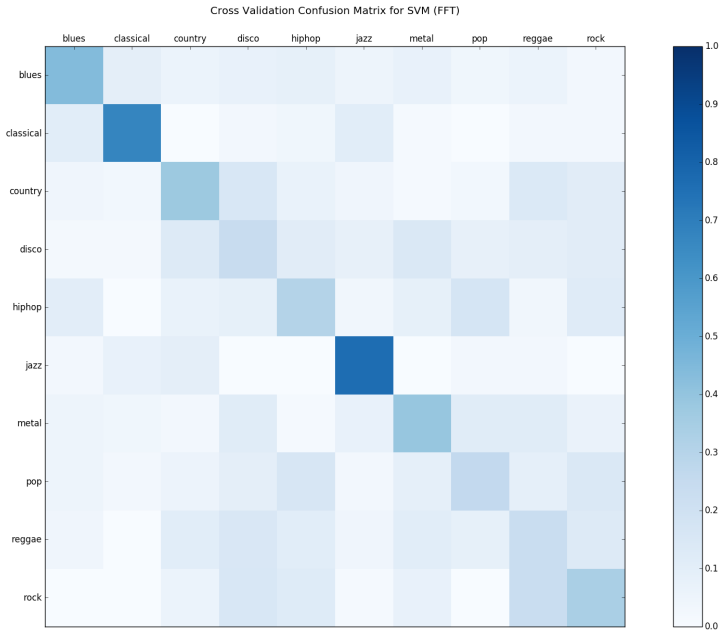

Figure 8- Confusion Matrix from CV runs for SVM using FFT

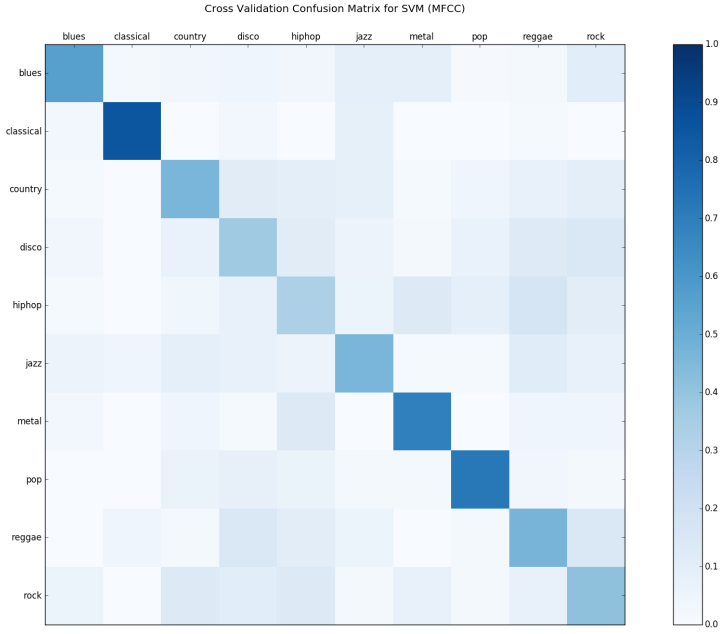

Figure 9- Confusion Matrix from CV runs for SVM using MFCC

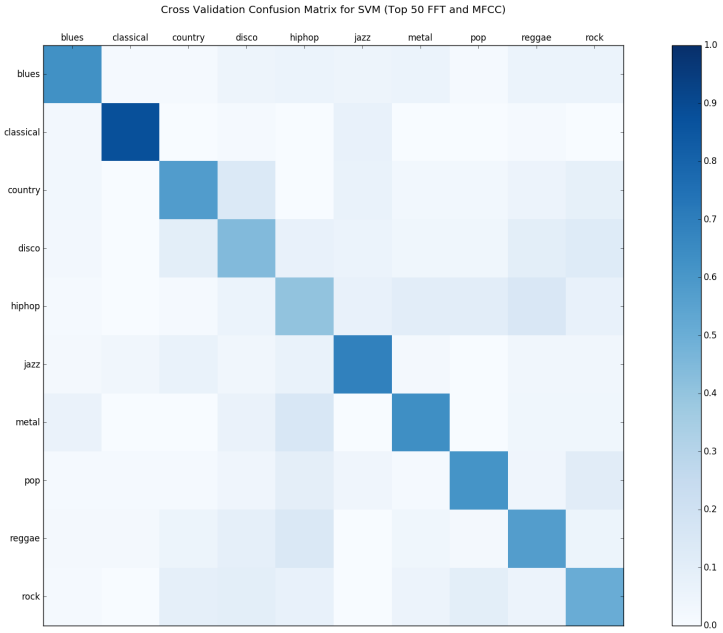

Figure 10- Confusion Matrix from CV runs for SVM using Top 50 FFT and MFCC

Explanation of Results

Overall I thought the results were very good and consistent with the results of other students in the class and researchers using similar datasets. I was very happy with my top 50 approach for feature selection. It was great to see significant improvement by selecting the strongest features. I was also pleased to see that my accuracy on the test sets were basically as good or better than the average CV accuracy. This leads me to believe there is very little bias in my training methods. The test set was not involved in the training of the classifiers (when doing CV) and it was not involved in generating the top 50 FFT and MFCC features (only the test set was used). I expect to see similar accuracy results on the validation test set (all training data was used for fitting the classifiers when making predictions on the validation set). I will be interested to see the validation results, if the accuracy is bad for the top 50 FFT and MFCC features it could be due to some bias in the training set. For example, a few sample could have produced a high ranking on the top 50 list and you might not have similar results come up in the validation data. Possibly creating a top 100 might help to avoid over fitting the classifier.

Future Improvements

I mentioned this in the previous section but I would consider expanding my top 50 list to possibly a top 100 list of FFT and MFCC features. I would also revisit some of the features that I couldn't get good performance out of. I was testing them in isolation and just gave up after seeing poor classification results. If I combine some of those features with the FFT and MFCC

features and then use my feature extraction method some of them could potentially crack the top 50 or top 100 list and help improve classification accuracy.

References

I made heavy use of the scikit-learn website for code examples. I also used ideas presented in "Python Machine Learning" and "Building Machine Learning Systems with Python"