Image Classification

Brendan Waters

May 17, 2015

This project had very little guidance and thus my approach was simply to search the internet for useful tools to complete the task of image classification. After some research on the tools available, I decided to use scikit because it seemed the most straightforward and it was already installed on my virtual machine via anaconda. I have provided reference links to the webpages I found useful when doing this project. Each of these links contains a basic tutorial on how to use scikit's various tools such as image loading, rgb to grayscale to black and white image conversion as well as a support vector machine which actually classifies these images.

After collecting all of these tools it took a bit of trial and error to get them to work together to solve the problem. For example, it was unclear that when I loaded the images they were defined as a 2D array with 100 features (each containing 100 ones or zeros) but what I really needed was a 1D array with 10,000 features (each pixel being a feature). So the array had all the values I needed, they just weren't formatted correctly. Luckily it turned out that numpy has a reshape method which easily converts the arrays into the format I needed.

Once I had everything set up, I decided to use the linearSVC because by its description on scikit learn's website, it seemed the most appropriate for the given task of image classification with multiple (5) classes. These are the results of the SVM:

brendan@brendan-VirtualBox: /AI/proj3$ python ProcessImage.py Validation/01/85.jpg

Classification report for classifier LinearSVC(C=1.0, $\text{class}_w eight = None, dual = True, fit_i ntercept = True, intercept_s caling = 1, loss =' squared_h inge', max_i ter = 1000, multi_c lass =' ovr', penalty =' l2', random_s tate = None, tol = 0.0001, verbose = 0)$:

class	precision	recall	f1-score	support
1	0.85	1.00	0.92	40
2	0.89	0.93	0.91	27
3	0.83	0.91	0.87	44
4	0.96	0.70	0.81	37
5	0.98	0.93	0.95	43
avg / total	0.90	0.90	0.89	191

Confusion matrix:

```
[[40 0 0 0 0]
[ 0 25 0 1 1]
[ 3 1 40 0 0]
[ 4 0 7 26 0]
[ 0 2 1 0 40]]
```

The SVM is only about 90% accurate when training on half of the given data and testing on the other half, having a score of 0.90 for both precision and recall. This is relatively low compared with other classmates I talked to who used different kinds of SVM's and achieved accuracies of around 97%. So the accuracy of my program could be improved but I thought it would be more interesting to leave it as is to show the difference in results when using different types of machines.