David Tang
008282339
CMPE 255
Program 2 Report
Rank: 33
F1-Score: .4385

To start off, I archived **traffic-small** and copied it over to my computer. I wanted to do some simple sanity tests on the small dataset before I move on to the real big one.

For feature extraction for the images, I tried two techniques:

- Normalized Color Histogram
    - This was done using OpenCV's **imread** and **calcHist** functions
    - I had to reshape the output matrix from each **calcHist** call in order to call **numpy's vstack** function against my python list of histograms. The **vstack** function converts a python list to a numpy array, which will later be used for faster matrix operations.
    - I used the same parameters shown in the example from https://docs.opencv.org/3.3.1/d1/db7/tutorial_py_histogram_begins.html for **calcHist**. The constant parameters ensured that each vector was the same size.
- Histogram of Oriented Gradients (HOG)
    - This was done using scimage's **hog** function to create the histogram
    - I first **resized** each image before I calculated the HOG so that the vector dimensions would be the same length. The vectors must be the same length in order to **vstack** them.
    - When I glanced over each image in a file browser, I noticed some were tall and some were wide but approximately the same proportions if tall images were rotated 90 degrees. Since I did not want to disproportionately stretch/shrink an image too much, I decide to **resize** the image to a square.
    - I've experimented with the anti-aliasing feature. Toggle this feature did not make much of a different for me.

I went with HOG because it seems to provide better accuracy in classification.

For feature selection, I tried using several techniques provided by the **sklearn** library:

- PCA
- LDA
- SVD
- Random Projection

Unfortunately for me, none of the feature selection techniques I tried above did well for me. They ranged from bad to extremely bad. I think the issue I ran into isn't that the classifier is bad, but probably because the vector representation I created from the images were not accurate or I missed something else. The one I stick with is SVD since I got the best worst results with it.

Initially, I wrote my own KNN classifier since I was already accustomed to it from homework 1. I also used cosine similarity as my metric in KNN. Since I wanted to try other classifiers and didn't want to implement it myself, I refactored my code to use **sklearn**'s classifiers.

For classification, I have tried using the following classifiers:

- KNN
- ADA Boost
- Naive Bayes
- Decision Trees

I noticed that the KNN classifier did the best by far. The ADA Boost classifier did the worst and classified most objects as **14**. The other two just seemed worse in general compared to KNN so I went with the KNN classifier.

I tried various **k** neighbors for KNN in the small dataset. At first, I tried a suggestion I found online where it said a good starting point is **k=sqrt(n)** where **n** is the number of samples you have. When I tried this with the small 4000ish traffic dataset and **k=63**, my classifications got a lot worse. I decided to smart my **k** small again. Each time, I would eyeball and compare the known labels against my results. I eventually chose **k=5** because it seemed to give relatively reasonable results. Additionally, when I set **weights** to either **uniform** or **distance** I didn't really notice much of a difference.

The various combinations of feature extraction/selection and classifiers didn't make any huge difference. I feel like my results are almost random. Most of the other people in the class got F-scores of 70%+ while I am lingering around 40%. I'm starting to believe that there is something more fundamental that I am doing incorrectly.

Most important of all, I took advantage of our HPC that has 28 CPUs. This cut down a lot of my testing time against the full test set. I inspected the CPU specs by running **lscpu**. I tried to leverage all that power by passing in an extra parameters to by KNN classifier like so:

```
KNeighborsClassifier(n_neighbors=5, weights='distance', n_jobs=28)
```

When it came time to classify the objects, I noticed that my CPU utilization jumped to 2500%+. I inspected this in **top**. This probably saved me an hour or two on each run that I did when I planned to submit my results.