Project 3 Report

Benjamin Nace

April 30, 2016

Introduction This report documents the methodology and results for a machine learning system that identifies black and white 100x100 jpeg images as being in one of five categories:

1. Smile 
2. Hat 
3. Hash 
4. Heart 
5. Dollar 

Step 1: Featurization The first step to training the machine learning system was to be able to convert the image into a vector. I used the Pillow imaging library for Python to do this. Using pillow, the image file is loaded. Then, it is converted to a grayscale image using a built-in Pillow function, just to ensure that every pixel is indeed grayscale before converting to pure black/white. It's then converted to a pure black/white image using a pixel value threshold of 175 for white pixels on a scale of 0-255. All pixels with a value less than 175 were converted to black. Finally, the pixel data was pulled out of the black/white to form a vector with one entry of either 0 (black) or 255 (white) for each pixel.

Step 2: Selecting the Approach My initial approach was to use a support vector machine, since they are generally good for machine learning. I used the scikit-learn Python package for all the machine learning implementations. I divided the data into training, testing, and validation sets with an 80%/10%/10% split within each image class. I trained the machine on the 80% training set, then set about testing it on the testing set. However, I quickly noticed an issue. The machine was very lazy, and selected the class that had the most samples in the training set to be the ONLY class it recognized. In this case, it was the dollar class. So, no matter what image I fed into it for testing, it would always say it was a dollar. I tried making the amount of samples for each class in the training set even, and twiddling with the parameters for the machine, but nothing would shake it from this behavior. So, I decided that I needed a different approach.

My next approach was to use a decision tree. I trained it using the same training set I used for the support vector machine. The result was better than with the support vector machine. It seemed to be able to correctly identify 3 out of the five classes. It would typically identify both smiles and hats as smiles, hashes and hearts both as hearts, and dollars as dollars. To determine what kind of accuracy could be achieved by this method, I used scikit-learn's built-in cross-validation functionality. I gave it all of my featurized data, and it randomly split it into 10 sets to perform cross-validation. It returned the accuracy percentages achieved by each of the 10 combinations of training and validation splits. It achieved an average accuracy in the 60% range. When I changed the function that measured the quality of a split in the tree from Gini impurity to information gain, the average accuracy improved to around 70%.

The average accuracy of the cross validation gave me an idea. I split each class of sample data into 10 sets of roughly equal size. These sets from each class were then combined into 10 roughly equal size sets containing data from each class. So, the final sets each contained about 10% of the smile samples, about 10% of the hat samples, etc. Using these 10 data sets, I trained up 10 decision trees, which are then used for prediction.

Step 3: Prediction Predicting which class a sample is in involves a voting system between the 10 decision trees. Each one is given the sample and produces the probability of that sample being in each class. The 10 probabilities per class are summed to produce the final score for each class. The class with the highest score is the one the program outputs. If multiple classes have the same score, then the one with the highest probability given by a single machine is used. And if classes are still tied, then the lowest class number is selected. Testing this approach over all of the sample data provided produced an accuracy of about 72%.