CS440: Project : Image Classification

Sinan Sahin

May 6, 2018

Introduction

I used the files from http://inst.eecs.berkeley.edu/ cs188/sp11/projects/classification/ as a starting point. These file had the general structure needed for the project only missing the classifier parts. The files included many things that I removed because it was unnecessary for this project.

I also added couple of new argument options so I can display the information required by the project. All of the options can be seen by the command python dataClassifier.py -h

Naive Bayes Classifier

After reading the Naive Bayes explanation on the http://inst.eecs.berkeley.edu/ cs188/sp11/projects/classif I was able to add the naive bayes classifier to the project.

How the classifier works is there are three counters(Data structure usually used to map the feature[pixel], to a number which is usually a counter)

First counter is called priorDist which is the prior distribution over labels (digits, or face/not-face), P(Y).
We can estimate P(Y) directly from the training data through: P(Y) = c(y)/n
Where c(y) is the number of training instances with label y and n is the total number of training instances.

Second is actually a dictionary of counters 0 and 1 called condProb. 0 represents the black features counter and 1 represents the white features counter. This is the conditional probabilities of our features given each label

$$y : P(F_i|Y = y)$$

We do this for each possible feature value ($f_i \in 0, 1$).
$P(F_i = f_i|Y = y) = c(f_i, y)/\Sigma(f_i \in 0, 1)(f_i', y)$
where $c(f_i, y)$ is the number of times pixel F_i took value f_i in the training examples of label y.

Third is a counter named count which simply counts the number of times we see a specific counter. Its Pixel value doesn't matter because this will allow us to properly calculate conditional probability, since we need to know the total number to calculate it.

So the general algorithm is as follows:

1. Loop through each of the training data and increment the count counter for each feature we see.
2. Also increment priorDist for each label we see.
3. Also increment the right feature in condProb[1] if its a white pixel and condProb[0] if black
4. When we looped through the entire training data normalize the priorDist
5. Now we can smooth our conditional probabilities by adding a value such as 2 to every possible feature so we have no 0 values
6. After this we normalize our conditional probabilities and the classifier is finished.

While testing we need to calculate the jpint probabilities and for this I took the advice from the berkley page and used the log addition instead because multiplying many probabilities together often results in underflow, we will instead compute log probabilities which have the same argmax.

Naive Bayes Classifier Results:

- Faces

%	Training Size	Training Time	Accuracy	Std in Accuracy
10 percent	45	0.7211	71.1667	5.9722
20 percent	90	1.3429	78.8333	4.7881
30 percent	135	1.9997	85.6667	2.0728
40 percent	180	2.4483	88.0	2.8284
50 percent	225	3.1521	87.1667	0.8389
60 percent	270	3.8190	87.3333	2.1082
70 percent	315	4.4276	89.3333	0.7698
80 percent	360	4.9777	88.6667	1.8856
90 percent	405	5.5380	89.50	0.8389
100 percent	451	6.3770	90.0	0.0

- Digits

%	Training Size	Training Time	Accuracy	Std in Accuracy
10 percent	500	0.9667	73.70	1.1633
20 percent	1000	1.7308	74.3250	1.0595
30 percent	1500	2.6958	75.4250	0.8057
40 percent	2000	3.6799	76.3750	0.5737
50 percent	2500	4.6820	75.9750	0.8461
60 percent	3000	5.6673	76.3750	0.4717
70 percent	3500	6.4947	75.90	0.7394
80 percent	4000	7.4887	76.450	0.2082
90 percent	4500	8.5117	76.5750	0.250
100 percent	5000	9.4655	76.60	0.0

Perceptron

I also used the help of the berkley page for the Perceptron classifiers algorithm.
The perceptrons procedure is simpler and uses the weight system. Its procedure is as follows.

1. You hold a global counter of weights for each possible label.
2. There is one big loop for the number of iterations you would like to do.
3. With in this loop you loop through the training data points.
4. Inside this loop you keep a score for each possible label which is basically a counter of pixels values and you update the weights if the current score is higher than the previous score(weight) of the label.

Because the perceptron involves many nested loops it take a while to compute.

Perceptron Classifier Results:

- Faces

%	Training Size	Training Time	Accuracy	Std in Accuracy
10 percent	45	2.1936	81.3333	4.0369
20 percent	90	4.0166	87.0	1.6777
30 percent	135	5.8025	85.8333	1.374
40 percent	180	7.6135	84.8333	3.0
50 percent	225	9.4106	87.3333	0.0
60 percent	270	10.8560	87.3333	0.0
70 percent	315	13.1078	87.3333	0.0
80 percent	360	15.0333	87.3333	0.0
90 percent	405	16.4522	87.3333	0.0
100 percent	451	19.2845	87.3333	0.0

- Digits

%	Training Size	Training Time	Accuracy	Std in Accuracy
10 percent	500	11.5112	73.350	1.0504
20 percent	1000	22.7003	78.6250	1.9704
30 percent	1500	33.6464	78.350	2.0936
40 percent	2000	44.4779	81.050	0.9327
50 percent	2500	55.0219	78.850	1.5264
60 percent	3000	63.3562	81.0	0.9832
70 percent	3500	71.3487	81.4750	0.9069
80 percent	4000	81.4028	81.40	0.7257
90 percent	4500	90.9719	81.0	0.4082
100 percent	5000	99.9783	81.0750	0.7136

Conclusion:

It looks like there are advantages and disadvantages of both methods. When we look at a small sample space like the faces example Naive Bayes is both more accurate and faster.
But when we look at bigger sample space like the digits example Naive Bayes's accuracy is lower than the Perceptron but Perceptron took extremely long.
Even though Perceptron takes longer its standard deviation is usually closer to 0 and provides high accuracy in the 80-87% even with smaller subset of the training data.

** The actual outputs of each run can be found in text files with in the project. The output includes individual results as well as the averages.**