AI Project- Image Classification

Abhishek Prajapati, Rushit Sanghrajka

December 16, 2016

1 Introduction

In this project, we used different classifiers to recognize digits and faces. We used Percepton Classifier, Naive Bayes Classifier, And k Nearest Neighbors Classifiers on the given image data set.

2 Algorithm

- First of all we read all the train and test data, and extract all of its features and store it in the memory for faster and easier access. Then we pass the train data to the algorithm to train the data using one of three algorithm (i.e Perceptron.train(train data)).

- **Percepton Algorithm**
  In perceptron we keep the weight vector w^y of each image y. Given a list of features of an image we compute the image y whose vector weight is most similar to the input vector, and we choose the class with the highest score to predicated the label for that data instance. We do that by using this formula $score(f, y) = \sum_i f_i * w_i{}^y$. Then we train the data using training data. In training data, we iterate each instance at a time and calculate the label with the highest score $y^{'=argmaxscore(f,y")}$ then we compare $y^{'}$ to the true label y. If $y' = y$, we've gotten the instance correct, and we do nothing. Otherwise, we guessed y' but we should have guessed y. That means that w^y should have scored f higher, and $w^{y'}$ should have scored f lower, in order to prevent this error in the future. We update these two weight vectors accordingly:

  $$w^y + = f$$
  $$w^{y'} - = f$$

  Then we classify the result using test data to find the accuracy of an algorithm.

- **Naive Bayse Algorithm**
  In naive Bayes, we find the most probable label given the feature using bayes theorem.

$P(y|f_1, ..., f_m = P(y) \prod_{i=1}^{m} P(f_i|y)/P(f_1, ...f_m)$

$argmax P(y|f_1, ..., f_m) = argmax P(y) \prod_{i=1}^{m} P(f_i|y)$

then our Naive Bayes estimates the images true value from the training data using $P(y) = c(y)/n$

We also the smoothing factor to count every possible observation value. If the k = 0, the smoothing probabilities are un-smoothed, and as the k grows larger the probabilities are smoothed more and more. we use this formula for smoothing $P(F_i = f_i|Y = y) = c(f_i) + k / \sum_{f_i^l \in 0,1} (c(f_i^l, y) + k)$

then we use the classify the train result using test data comparing and calculating joint probability of the feature and its value.

- **K Nearest Neighbors Algorithm**
  In KNN we classify image by vote of the majority of K neighbors. K neighbors are measured using the distance function. $\sum_{i=1}^{k} |X_i - Y_i|$. And if K = 1 then the case is simply assigned to the class of its nearest neighbor. We choose the best value of K by inspecting the data. And the training phase only consists of storing feature vectors and class labels. Then we use test lebels to classify our results.

3 Result

When we use only 10% of the training data the algorithms learns less and its accuracy is less then what we get if we use 100% of the data. I have given few results from our algorithm to see the what different results we get if we use 10% of the training data and 100% of the training data. When we use 100% of the train data, the algorithm learns well and identifies the images more accurately. However, the training time does increases when we use more data. However for the KNN algorithm on digits we need to keep the k low for some reason to get 70% accuracy, and on faces it gives 50% on most cases because the features for the faces are too thin and detailed to give a good result in KNN.

10% training data:

==========

Running Perceptron Classifier on Digits
*RESULT OF PERCEPTRON CLASSIFIER ON DIGITS**
Error rate: 0.265
Accuracy: 0.735
Number of Errors: 265 out of 1000
Total training time: 245ms

==========

Running Perceptron Classifier on Faces
*RESULT OF PERCEPTRON CLASSIFIER ON Images**
Error rate: 0.4066666666666667
Accuracy: 0.5933333333333334
Number of Errors: 61 out of 150
Total training time: 13ms

==========

Running Naive Bayes Classifier on Digit
*RESULT OF Naive Bayes CLASSIFIER ON digits**
Error rate: 0.514
Accuracy: 0.486
Number of Errors: 514 out of 1000
Total training time: 39ms
==========
Running Naive Bayes Classifier on faces
*RESULT OF Naive Bayes CLASSIFIER ON images**
Error rate: 0.486666666666667
Accuracy: 0.513333333333333
Number of Errors: 73 out of 150
Total training time: 4ms

100% training data:
==========
Running Perceptron Classifier on Digits
*RESULT OF PERCEPTRON CLASSIFIER ON DIGITS**
Error rate: 0.186
Accuracy: 0.814
Number of Errors: 186 out of 1000
Total training time: 1596ms
==========
Running Perceptron Classifier on Faces
*RESULT OF PERCEPTRON CLASSIFIER ON Images**
Error rate: 0.133333333333333
Accuracy: 0.8666666666666667
Number of Errors: 20 out of 150
Total training time: 126ms

==========

Running Naive Bayes Classifier on digits
*RESULT OF Naive Bayes CLASSIFIER ON DIGITS**
Error rate: 0.312
Accuracy: 0.688
Number of Errors: 312 out of 1000
Total training time: 259ms
==========

Running Naive Bayes Classifier on faces
*RESULT OF Naive Bayes CLASSIFIER ON images**
Error rate: 0.1133333333333333
Accuracy: 0.8866666666666667
Number of Errors: 17 out of 150
Total training time: 41ms

4 Lesson Learned

The algorithm learns well and becomes more accurate as we train it more and more.