CS 520 - Introduction to Artificial Intelligence
Project : Image Classification

Chinmayee Gadgil, Lakshmi Kumar

May 02, 2017

1 Introduction

In this project, we implement three algorithms for image classification. We use two image data sets: a set of scanned handwritten digit images and a set of face images in which edges are already detected. Digits are from the MNIST data set and face images are from the easy-faces and background categories of the Caltech 101 dataset. Since the data sets are already preprocessed the classifiers perform well with simple features. [1] The three classifiers that we implement are:

- Perceptron
- Naive Bayes
- k Nearest Neighbors

2 Dataset and Image Representation

The datasets we use are as follows:

- The digit dataset consists of 5000 training samples and 1000 testing samples. Each sample image is of $28x28$ pixels.
- The face dataset consists of 451 training samples 150 testing samples. Each sample image is of $70x60$ pixels.

These datasets consist of either of the three characters - white space, '+' or '#'. We assign 0 to a white space, 225 to '+' and 255 to '#'. 0 (white space) and 255 ('#') values imply that it is a black background with white colored digits and faces on it. In case of digit dataset we also use 225 ('+') which imply that the borders between the background and the digits are gray.

3 Feature Extraction

The simple features that we extract are as follows:

- Total number of non-black pixels in the entire image - we extract the properties of the image as a whole.
- Row-wise non-black pixels - we extract the row-profile of an image. It gives the total count of the non-black pixels in every column taken row-wise.
- Column-wise non-black pixels - we extract the column profile of an image. It gives the total count of the non-black pixels in every row taken column-wise.
- Non-black pixels in small grids of the image - we divide the digit dataset in 16 grids of $7x7$ each and the face dataset in 42 grids of $10x10$ each. We extract the count of the non-black pixels from these grids.
- Hu Moments - we extract the translation, rotation and scale invariant features from the entire image. We use these features only in the case of digit database.

4 Algorithms

4.1 Perceptron

In this algorithm, we use a single neuron to classify the digits and faces. The algorithm consists of two phases: training and testing.

In the training phase, we initialize the weights randomly. We multiply the extracted features with the weights. We consider the product with the maximum value as the predicted output and compare it with the expected output to update the weights. We run this algorithm for 1000 epochs and then save the weights in the form of a '.npy' file.

For digits (multi-class classification), if the predicted and expected value do not match, we decrease the weight of the predicted value by the amount of the feature vector and increase the weight of the expected value by the amount of the feature vector.

For face/non-face (binary classification), the predicted value is divided into two classes -1 and 1. If the predicted value is greater than 0, we assign a value of 1 to it, else, we assign a value of -1 to it. If the predicted value does not match with the expected, we increase the weight by an amount equal to the product of the feature vector and predicted value.

4.2 Naive Bayes

We implement the Naive Bayes algorithm, which is a probabilistic classifier based on Bayes theorem. The assumption in the algorithm is that the features are independent of each other.

The formula for Naive Bayes as shown in Figure 1.

$$\begin{aligned}
P(y|f_1,\ldots,f_m) &= \frac{P(f_1,\ldots,f_m|y)P(y)}{P(f_1,\ldots,f_m)} \\
&= \frac{P(y)\prod_{i=1}^{m}P(f_i|y)}{P(f_1,\ldots,f_m)} \\
\arg\max_y P(y|f_1,\ldots,f_m) &= \arg\max_y \frac{P(y)\prod_{i=1}^{m}P(f_i|y)}{P(f_1,\ldots,f_m)} \\
&= \arg\max_y P(y)\prod_{i=1}^{m}P(f_i|y)
\end{aligned}$$

$$\begin{aligned}
\arg\max_y \log P(y|f_1,\ldots,f_m) &= \arg\max\nolimits_y \log P(y,f_1,\ldots,f_m) \\
&= \arg\max_y \left\{ \log P(y) + \sum_{i=1}^{m} \log P(f_i|y) \right\}
\end{aligned}$$

Figure 1: Bayes Theorem - Formula [1]

We first calculate the probability distribution $P(Y)$ over the individual classes in the training data set (for digit dataset $y = 0,1,2,3,4,5,6,7,8,9$ and for faces $y = 0,1$). We then calculate $P(f_i|y)$ i.e. the probabilities of the individual features in the feature vector for every test image. We calculate this probability by counting the instances where the particular feature matches with the corresponding value in the test feature vector for certain y and then we divide this count by the total number of the occurrences of y. We calculate this probability for every feature and then calculate their logarithmic values. We then sum it over all the individual features and add to the logarithmic value of $P(y)$. The maximum value of the above sum gives us the prediction.

4.3 k Nearest Neighbors

We implement the k Nearest Neighbours (kNN) classification algorithm, which classifies an object based on a majority vote of its' nearest neighbors.

In the training phase, we save the feature matrix as a '.npy' file which we retrieve in the test phase. In the test phase, we compute the euclidean distances between the features of the test sample and the saved feature matrix. We sort the distance array in increasing order to get the corresponding indices. We consider the first k-values to be the nearest neighbors. We consider the value which occurs most frequently in the nearest neighbors to be the predicted value. If the value does not match the expected value, we increase the error count by one.

5 Results

We randomly vary the data points from 10% to 100% during training and obtain the following results for the three algorithms.

5.1 Perceptron

Digits:

Data-points	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Accuracy	72.7%	69.0%	68.3%	75.7%	70.2%	68.3%	64.6%	78.1%	74.1%	75.2%
Time(sec)	3.36	6.33	9.25	12	14.82	18.56	21.14	23.91	27.98	30.12

Mean = 71.62%, Standard deviation = 3.98%

Faces:

Data-points	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Accuracy	85.33%	89.33%	80.66%	84.66%	82%	82%	84%	82%	82.66%	80.66%
Time(sec)	0.53	0.76	1.04	1.24	1.51	1.81	2.02	2.33	2.48	2.81

Mean = 83.33%, Standard deviation = 2.5%

5.2 Naive Bayes

Digits:

Data-points	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Accuracy	65%	72%	72.7%	74.5%	74.9%	75.7%	76.6%	76.6%	76.8%	76.9%
Time(min)	1.51	2.88	4.44	5.69	7.04	8.46	10.05	13.62	17.45	20.14

Mean = 74.16%, Standard deviation = 3.46%

Faces:

Data-points	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Accuracy	83.33%	81.33%	79.33%	77.33%	81.33%	83.33%	85.33%	86%	88.66%	88.67%
Time(sec)	1.82	2.91	4.13	5.23	6.05	6.91	7.90	8.99	9.77	10.83

Mean = 83.46%, Standard deviation = 3.58%

5.3 k Nearest Neighbors

Digits: k = 3

Data-points	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Accuracy	66.2%	73.3%	76.2%	77.8%	79.0%	79.1%	79.5%	79.8%	80.7%	81.3%
Time(sec)	1.8	3.07	4.0	5.05	5.91	7.23	8.25	9.02	10	11.44

Mean = 77.28%, Standard deviation = 4.30%

Faces: k = 1

Data-points	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Accuracy	67.33	68.66%	69.33%	74.66%	70%	73.33%	74%	76.66%	79.33%	80%
Time(sec)	0.56	0.72	0.89	1.12	1.26	1.45	1.63	1.83	2.1	2.19

Mean = 73.33%, Standard deviation = 4.22%

6 Conclusion

In conclusion, we see that as the training data set is increased, the time taken to train the system also increases. The prediction accuracy however does not show any such trend. It fluctuates over a mean value. This is because, in case of perceptron, the trained models tend to change every time since we initialize the weights randomly. Hence we get different outputs for repeated testing of the algorithm.

In case of kNN, the values of the features and the euclidean distances remain constant and we get same results for repeated testing of the algorithm.

In case of naives bayes classifier, the values of the probabilities remain constant and we get same results for repeated testing of the algorithm.

We observe that Naive Bayes gives the best prediction accuracy for faces and k Nearest Neighbors gives the best prediction for digits.

References

[1] UC Berkeley CS188 - Intro to AI - Project 5 : Classification
`http://inst.eecs.berkeley.edu/~cs188/sp11/projects/classification/classification.html`