RUTGERS, THE STATE UNIVERSITY OF NEW JERSEY

Artificial Intelligence Project Report

Image Classification

Tongpeng Zhang(tz136)
Yuwei Huang(yh480)
May 4,2016

1. Description

For our project, we Implement three classification algorithms for detecting faces and classifying digits: a naive Bayes classifier, a perceptron classifier and a Convolutional Neural Network classifier. We test our classifiers on two image data sets: a set of scanned handwritten digit images and a set of face images in which edges have already been detected. Even with simple features, our classifiers will be able to do quite well on these tasks when given enough training data.

2. Implemented Algorithm

2.1 Naïve Bayes Classifier

A naive Bayes classifier models a joint distribution over a label Y and a set of observed random variables, or features, $\{F_1, F_2, \ldots F_n\}$, using the assumption that the full joint distribution can be factored. To classify a datum, we can find the most probable label given the feature values for each pixel, using Bayes theorem:

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

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

Now we calculate the P(y) and P(f | y).
We can estimate P (Y) directly from the training data:

$$\hat{P}(y) = \frac{c(y)}{n}$$

where $c(f_i,y)$ is the number of times pixel F_i took value f_i in the training examples of label y.
In our code,first we add all pixels(28*28/60*70) of training data in every label, and calculate the count of pixel = 1 and pixel = 0 separately, then we use the number of pixel = 1 in one label divided by the number of data in this label, same as when pixel = 0.Then, we have $\hat{P}(f_i|y)$.

Now we need to make sure that no parameter ever receives an estimate of zero,and use Laplace smoothing, which adds k counts to every possible observation value:

$$P(f_i|y) = \frac{c(f_i, y) + k}{\sum_{f_i} (c(f_i, y) + k)}$$

In this project, we use 10 values of k to smooth the $P(f_i|y)$ and calculate each of them. Because multiplying many probabilities together often results in underflow, we will instead compute log probabilities which have the same argmax:

$$\begin{aligned} argmax_y log(P(y|f_1, ..., f_m) &= argmax_y logP(y, f_1, ..., f_m) \\ &= argmax_y \left\{ log(P(y)) + \sum_{i=1}^{m} logP(f_i|y) \right\} \end{aligned}$$

Thus, we can get the maximum probability of the label given all the features.

2.2 Perception

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers: functions that can decide whether an input (represented by a vector of numbers) belongs to one class or another. A perceptron uses a weight vector w^y of each class y. Given a feature list f, the perceptron compute the class y whose weight vector is most similar to the input vector f. Formally, given a feature vector f, we score each class with:

$$\text{score}(f, y) = \sum_i f_i w_i^y$$

In our code, first,we use features and reallabel to store every training data and their real labels respectively. And then, we use loop to calculate each data by multiplying its feature and weight. Also we use a list called results to store them. Then we choose the class with highest score as the predicted label for that data instance.

$$y' = \arg\max_{y''} score(f, y'')$$

In our code, we set the predicted label as label to store the highest label in the results list using argMax(). 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:

$$w^y += f$$

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

In our code, first, we compare the reallabel and the label, if they are not equal, we add features to the weight of reallabel and subtract the features from the weight of the label. After lots of times iteration, the weigh of features will not change again, and the algorithm is done.

2.3 An algorithm of own choice --- Convolutional Neural Network

2.3.1 Single Hidden Layer Neural Network

The framework of a single hidden layer Neural Network model is shown in figure 2. The inputs are all the pixels of the datam, and the outputs are the classification result.

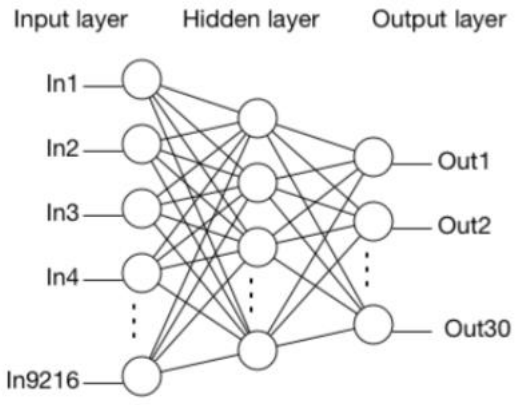

Figure 1-Single Hidden Layer Neural Network

2.3.2 CNN Model

Here we used a more complicated method: Convolutional Neural Network, which has been electively used in many vision tasks such as digit classification, face recognition and scene parsing. In CNN, the first layer may check the edge orientation and transport information to the second layer units for checking edge combinations like circle, square or corner. As we going up the network, the features are getting more and more complex, abstract, and obviously fewer. Here is an example of the framework of CNN model. In our model, we used two convolutional-layers followed by two max-pooling layers and then used dropout-layer and dense-layer.

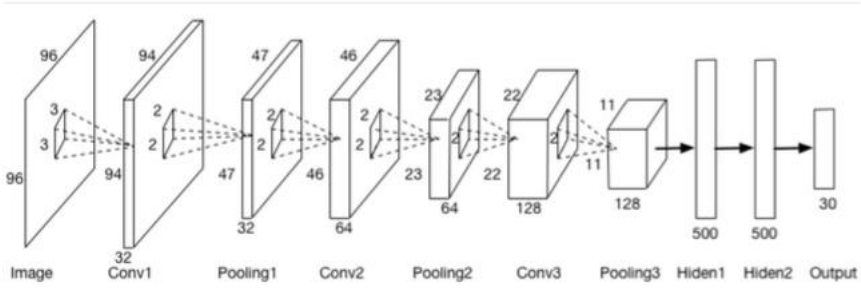

Figure 2-Convolutional Neural Network

Max-pooling Layers:

To reduce the complexity of computation, we need a max-pooling layer after every convolutional layer in CNN. The function of max pooling is simply down- scaling the image by splitting it into several non-overlapping rectangle parts and outputting only the maximum value in each part. The size of the max-pooling rectangle could either be 2x2, 3x3 or other supposed values based on real situation, and we find it is better to use max-pooling rectangle of 2x2.

Dropout:

Another important layer is dropout, we randomly ignoring half of the hidden layer nodes and still keeping the input and output nodes untouched. Every time we train the weights, a random half of the hidden layer neurons are used for training. In this way, several (suppose 5 runs) different networks are generated for same recognition aim, and we store each trained weights and corresponding nodes. When test images are inputted, the dropout layer will use the former trained neurons and weights to predict the

input into 5 results, and the final result is the average of these all subsets.

3. Result and Analysis

For three classifiers, we use 10% 20%, ...,100% of the data points that are reserved for training. First we set k = 2 in Naïve Bayes, iteration = 3 in Perceptron and iteration = 3 in Convolutional Neural Network.

3.1 Digits

The prediction accuracy of different algorithms is as follow:

Number of data	Naïve Bayes accuracy	Perceptron accuracy	CNN accuracy
10%	74%	70%	80%
20%	76%	74%	80%
30%	74%	70%	80%
40%	74%	68%	80%
50%	74%	70%	83%
60%	78%	72%	84%
70%	76%	82%	85%
80%	78%	72%	87%
90%	78%	78%	90%
100%	78%	74%	91%

The prediction errors as a function of the number of data points used for training are as follows

Number of data	Naïve Bayes Prediction Error	Perceptron Prediction Error	CNN Prediction Error
10%	26%	30%	20%
20%	24%	26%	20%
30%	26%	30%	20%
40%	26%	32%	20%
50%	26%	30%	17%
60%	22%	28%	16%
70%	24%	18%	15%
80%	22%	28%	13%
90%	22%	22%	10%
100%	22%	26%	9%
Average Error	24%	27%	16%
Standard Deviation	1.78%	4.02%	4%

The time needed for training as a function of the number of data points used for training.(In perceptron, iteration =3)

Number of data	Naïve Bayes Run time	Perceptron Run time	CNN Run time
10%	1.53s	16.7s	17.3s
20%	2.22s	34.9s	49.9s
30%	2.85s	50.9s	69.6s

40%	3.71s	61.5s	96.2s
50%	4.32s	106.03s	170.8s
60%	4.94s	121.4s	201.8s
70%	5.81s	148.6s	252.2s
80%	6.59s	171.2s	289.3s
90%	6.93s	197.22s	333.6s
100%	8.19s	213.65s	363.4s

3.2 Face

The prediction results are as follows

Number of data	Naïve Bayes accuracy	Perceptron accuracy	CNN accuracy
10%	56%	76%	70%
20%	78%	86%	72%
30%	82%	88%	83%
40%	88%	86%	89%
50%	84%	88%	91%
60%	88%	92%	92%
70%	88%	92%	93%
80%	92%	94%	94%
90%	92%	94%	95%
100%	92%	96%	96%

The prediction errors as a function of the number of data points used for training are as follows

Number of data	Naïve Bayes Prediction Error	Perceptron Prediction Error	CNN Prediction Error
10%	44%	24%	30%
20%	22%	14%	28%
30%	18%	12%	17%
40%	12%	14%	11%
50%	16%	12%	9%
60%	12%	8%	8%
70%	12%	8%	7%
80%	8%	6%	6%
90%	8%	6%	5%
100%	8%	4%	4%
Average Error	16%	10.8%	12.5%
Standard Deviation	10.31%	5.52%	8.95%

The time needed for training as a function of the number of data points used for training.

Number of data	Naïve Bayes Run time	Perceptron Run time	CNN Run time
10%	1.69s	2.19s	4.73s
20%	1.76s	3.88s	17.8s
30%	1.89s	5.59	31.3s

40%	2.12s	7.91s	54.7s
50%	2.5s	9.28s	64.9s
60%	2.97s	11.56s	80.5s
70%	3.47s	13.28s	92.4s
80%	3.69s	15.19s	98.7s
90%	4.07s	17.25s	113.7s
100%	4.50s	18.65s	121.5s

3.3 Comparing different k values in naïve Bayes classifier

In this section, we compared different smoothing parameter k in the naïve Bayes classifier and try to find the best k value. We use the digit data in this section.

Value of k	Validation Accuracy	Test Accuracy
1	80%	84%
2	78%	82%
3	78%	80%
4	76%	76%
5	76%	74%
6	76%	74%
7	74%	68%
8	70%	68%

We can find larger the k is, worse the performance, so k=1 is the best value.

3.4 Comparing different number of iterations in perceptron

In this section, we compared the prediction accuracy of perceptron classifier with different number of iterations and try to find the best iteration. We use the digit data to test.

Number of iterations	Validation Accuracy	Test accuracy
1	64%	62%
2	74%	64%
3	74%	70%
4	86%	68%
5	86%	78%
6	84%	74%
7	86%	72%
8	84%	66%

We can find 5 iterations will have the best prediction performance.

3.5 Analysis

From the data we recorded, we can get some conclusions.

Firstly, Comparing the performance of the three algorithms, we can find generally the CNN algorithm has the best prediction performance, averagely, it has the highest prediction accuracy both in digits and faces. But the CNN also need longest time to train, so its time performance is the worst. Between the naïve Bayes and the perceptron, perceptron algorithm has better prediction performance but takes more time to train, this may because it need to iterate several times. Naïve Bayes has the relatively lower prediction accuracy, but much faster than other algorithms.

Secondly, all the three algorithms have higher prediction accuracy in faces than digits. This is easy to explain, the face classification only

has two kinds: yes or no while the digits have 10 kinds-from 0 to 9. So to recognize digits is more difficult than faces.

Thirdly, for the relationship between the prediction accuracy and number of training data used. We can find the in the digits recognition, more training data may not get higher accuracy, but the accuracy of 100% still higher than the 10% accuracy. In the face recognition, more training data we used, the higher accuracy we get. We think the reason of this result may be we have 5000 number training data but only 450 face data. The number of digits is too large and has reached the limitation of the algorithm, so when we continuously use more data, the performance cannot be better, even become worse. But the face data is less, so we use more training data, the performance can become better.

Forth, for the smooth parameter k in naïve Bayes and the number of iterations in the perceptron, we can find the bigger the k is, the lower the prediction accuracy. That may because the larger k value will cause over fitting and is harmful to the performance. For the iterations in the perceptron, we can find too small or too large number of iterations will both make negative effects on the performance. In order to solve these problems, in the naïve Bayes algorithm, we use the validation data set to adjust the hyper parameter k and choose the best model.

4. Conclusion

In this project, we implement three algorithms to detecting faces and digits. The naïve Bayes algorithm is based on probability, it calculates the probability of different labels in the condition of given specific features and find the label with highest probability. The perceptron algorithm uses the weights vectors of different labels to evaluate each label and modify these weights every time it gets wrong results. The CNN algorithm uses the neuron network. All the three algorithms have their own features. Generally, all these three algorithms have quite good performance in this project, if we continuously to improve these algorithms, we believe they can handle more complicated tasks.