CSE 250B: Homework 1

Xiyun Liu A53099348

October 13, 2016

0.1 Description of the idea for prototype selection

Considering that the training set may have training error, we should first remove the *Outliers*. Then, we could only put the important points into the subset of the training set.

1. Remove the *Outliers* which is not classified in the correct label using the training set excluded this data.

2. Initialize a new set S with a random element from the trainingset TR.

3. Randomly select an element from the training set TR and classify this image using the current training subset S by 1-NN. If it is mislabeled, put this training point into S and remove it from TR.

4. Repeat step 3 until the size of S equals to M. If no more data can be added into the subset S, then we randomly select points from TR to fill S.

0.2 Concise and unambiguous pseudocode

Algorithm 1 Prototype selection

Input: labeled training set *trainingSet*, number M
Output: subset *trainingSubset* of size M;

```
1: procedure CreateSebset(trainingSet,M)
2:     for count ∈ 0...trainingSet.size() do
3:         data = trainingSet.pop(0)
4:         NNIndex = findNearestNeighbor(data, trainingSet)]
5:         if data.label == trainingSet[NNIndex].label then
6:             trainingSet.append(data)
7:     n ← trainingSet.size()
8:     trainingSubset ← [trainingSet[randomnumber]]
9:     while trainingSubset.size() < M do
10:        i ← random.randint(0, n)
11:        data ← trainingSet[i]
12:        index = findNearestNeighbor(data, trainingSubset)
13:        if data.label != trainingSet[index].label then
14:            trainingSubset.append(data)
15:            trainingSet.remove(data)
       return trainingSubset
16: procedure findNearestNeighbor(data,training)
17:    nearestIndex ← −1
18:    nearestDistance ← maximum integer
19:    for j ∈ 0.....training.size() do
20:        distance ← l2distance from data to training[j]
21:        if distance < nearestDistance then
22:            update nearestDistance
23:            nearestIndex ← j
       return nearestIndex
```

0.3 Experimental results

Compare the performance between uniform-random selection and my method.
The correctness rate:

M	Uniform-random Selection	Prototype Selection
1000	88.55%	90.31%
5000	93.48%	94.64%
10000	94.72%	95.39%

The error rate:

M	Uniform-random Selection	Prototype Selection
1000	11.45%	9.69%
5000	6.52%	5.36%
10000	5.28%	4.61%

Calculate the error bars and confidence interval for correctness percentage at M = 1000,5000, 10000 using Random Selection 1-NN.

M	1000	5000	10000
Number of Samples	100	50	10
Mean	88.5526%	93.4818%	94.7295%
Standard Deviation	0.3917%	0.1938%	0.1247%
Standard Error	0.039166%	0.0274017%	0.0394395%
Error Bars	88.1609% to 88.9443%	93.2880% to 93.6756%	94.6048% to 94.8542%
Confidence Interval	87.7848% to 89.3203%	93.1019% to 93.8616%	94.4850% to 94.9739%

The standard error is calculated by dividing the standard deviation by the square root of number of measurements that make up the mean (often represented by N).

$$Standard\ Error = \sigma / \sqrt[2]{N}$$

The error bars are basically calculated by mean and standard deviation. Using mean μ and standard deviation σ, we can state that, based on the experiments, the correctness percentage at M = 1000 is 88.5526% $\pm$ 0.3917%.

$$Error\ Bars = \mu \pm \sigma$$

For the confidence interval at the 95% confidence level:

$$Lower\ limit = \mu - Z_{.95} * \sigma$$
$$Upper\ limit = \mu + Z_{.95} * \sigma$$

where Z is the 0.95 critical value of the standard normal distribution which can be found in the table of the standard normal distribution. $Z_{.95} = 1.96$

0.4 Critical evaluation

My method is a clear improvement over random selection, which can be demonstrated by the above tables.

However, there are also some problem with the new method. First,it can take a long time to run, particularly on very large data sets. Based my experiment, it takes more than 1h to remove the *Outliers*, and takes more than 2 hours to generate a subset with size of 10000 from the raw training set.

For further improvement, we could think of some method to reduce the running time, for example, using PCA to reduce the dimensions of the input images from 784 to 100, and then using CNN to create a reasonable subset from the new training set.

For the next step, I would like to try using PCA to reduce the dimensions of the input image to increase the running speed and see if the accuracy drops after using PCA.

1 Appendix

M = 1000: [0.8866, 0.8818, 0.8805, 0.8925, 0.886, 0.8889, 0.8856, 0.8828, 0.8803, 0.8863, 0.8828, 0.8882, 0.881, 0.8926, 0.8843, 0.8837, 0.8862, 0.8889, 0.882, 0.8829, 0.8846, 0.8982, 0.8879, 0.8837, 0.8856, 0.8885, 0.8856, 0.889, 0.8898, 0.8823, 0.8886, 0.8884, 0.8795, 0.8779, 0.8834, 0.885, 0.8803, 0.8871, 0.8889, 0.879, 0.8857, 0.8852, 0.8766, 0.8866, 0.8858, 0.884, 0.8831, 0.8824, 0.8869, 0.8855, 0.8874, 0.8845, 0.8866, 0.89, 0.8865, 0.8862, 0.8859, 0.8849, 0.8887, 0.8794, 0.882, 0.8861, 0.8878, 0.896, 0.8822, 0.8888, 0.8862, 0.8895, 0.8829, 0.8859, 0.8901, 0.8825, 0.8851, 0.8905, 0.8876, 0.8856, 0.8896, 0.889, 0.8865, 0.8921, 0.8811, 0.8811, 0.8794, 0.8829, 0.8869, 0.8849, 0.8915, 0.8912, 0.8829, 0.8792, 0.8869, 0.8842, 0.8882, 0.8827, 0.878, 0.8891, 0.8781, 0.885, 0.8907, 0.8835]

M = 5000: [0.935, 0.9359, 0.9332, 0.9337, 0.9353, 0.9338, 0.9378, 0.9328, 0.9401, 0.9354, 0.937, 0.9358, 0.9351, 0.9357, 0.9324, 0.937, 0.9313, 0.9357, 0.934, 0.9341, 0.9326, 0.9321, 0.938, 0.9351, 0.9355, 0.9336, 0.9358, 0.9334, 0.9355, 0.9339, 0.9369, 0.9301, 0.9328, 0.9358, 0.9367, 0.9361, 0.9339, 0.9338, 0.935, 0.933, 0.9378, 0.9351, 0.9361, 0.9326, 0.9313, 0.9341, 0.9346, 0.9375, 0.9348, 0.9363]

M = 10000: [0.9494, 0.948, 0.9484, 0.946, 0.9461, 0.946, 0.947, 0.9465, 0.948, 0.9447, 0.9488, 0.9476, 0.9455, 0.949, 0.947, 0.9487, 0.9463, 0.9481, 0.9473, 0.9475]