Supervised Learning - Coursework 1

Yinji Zhu 18062537 yinji.zhu.18@ucl.ac.uk
Yao Zhao 18071889 yao.zhao.18@ucl.ac.uk

December 13, 2018

1 Kernel perceptron

1. Introduction

The aim of this exercise is to train a classifier to recognize digits which have 16x16 pixels. The dataset is obtained from **zipcombo** , which is a moderately large dataset having the size of 9298 rows and 257 columns. According to the **zipcombo** dataset, digits which need to be recognized are from 0 to 9.

1. Perceptron

The aim of the perceptron is to find a hyper-plane S: $w * X = 0$, which can separate S into two parts. The first part is larger than zero, the other part is less than or equal to 0. In this case, all sample can be separated into two parts. To classify all simple points correctly, values of w needs to be updated when mistakes are faced.

2. Kernel

In some case, samples are hard to be separated by simply linear functions. However, kernel allows one to map the data to a higher dimensional space so that the class of functions is larger than simply linear functions. In this exercise, two types of kernel are considered, which have the formats of polynomial and Gaussian.
The polynomial kernel can be written as:

$$K_d(p, q) = (p, d)^d$$

And the Gaussian kernel can be written as:

$$K(p, q) = e^{-c||p-q||^2}$$

Where d is a positive integer which can control the dimension of the polynomial and c is the width of the kernel.

2. Algorithm

Programs are implemented in Matlab. The basic algorithm of this exercise is online learning, which updates the value of α row by row. α is a matrix which has the terms of 0 and -1. The value of α updates epoch by epoch. When training the dataset epoch-wise, mistakes of classification will decrease and then converge in most of situations.

The process of ten classes classifier kernel perceptron can be seen below:

	Ten Classes Kernel Perceptron (training)
Input	$\{$ (x$_1, y_1), ..., (x_m, y_m)\} \in (R^n, \{-1, +1\})^m$
Initialization	$\alpha(1,2,..,m)_t(0,1,..,9)_j = 0$
Prediction	Upon receiving the t_{th} instance x_t, the predict value of y_t is: and for j $\subseteq \{0,1..9\}$j is the classes number $Ypredicted_{t,j} = sign(\sum_{i=1}^{t} \alpha_{t,j} K(x_t, x_i))$
Update	if $y_t = j, Y_{t,j} = 1$ else $Y_{t,j} = -1$ if $Y_{i,j} * Ypredicted_{i,j} < 0$, $\alpha_{i,j} = sign(Y_{i,j})$ where $sign(<= 0) = -1, sign(> 0) = 1$
Error Calculation	When $Ypredicted_{t=m,j=9}$, $yvalue_t = j.$ And if $yvalue_t \neq y_t$ mistake = mistake + 1 else mistake = mistake

In general, the training process has five steps: input, initialization, prediction, update and error calculation. Inputs are declared firstly, and then the value of alpha is initialized to 0. The predicted value of y is calculated row by row. If the predicted value of y is not the same with the expected value, α value in that term needs to update to the sign of Y in that term. After updating all the training dataset, mistakes between the predicted and expected values will be summed.

The testing process is similar with the training process. The difference between train and test is that the test algorithm does not need to initialize and update α. In this case, it just needs to use the α which has been calculated in the process of training to calculate the predict value y.

As what has been mentioned above, the data file is obtained from **zipcombo.dat**, which has 9298 records. Separating **zipcombo** into 80% train and 20% test, so the records of train set and test set are 7438 and 1860 respectively. Moreover, the first column of **zipcombo** represents the digits, and the others columns of **zipcombo** represent intensities of grey values, which are 16x16 matrices.

Several improvements were done to increase the speed of the algorithm: 1. The calculation of kernel is outside **for** loop of epoch, since the iteration will largely increase the computational expensive. In this case, a matrix substitutes iteration. 2. The statement sign() obtains three values: -1, 0 and 1. To display -1 when the value in sign() is 0, the statement **if** is necessary. However, it will also largely increase computational cost. Consequently, a very small bias is added in the sign() statement and then it will not have the situation that sign() equals to 0. 3.

Mistakes are calculated together by using matrices and summing the different number between the expected and predicted y.

3. Results

1. Knowing that the polynomial kernel can be written as: $K_d(p,q) = (p,d)^d$, perform 20 runs for d = 1,...,7. Each run randomly splits **zipcombo** into 80% train and 20%test. Flashing dataset in train and test by using the statement **randperm**, the train result of 20 runs for d = 1,...,7 is:

d	1	2	3	4	5	6	7
mean	7.90%	0.91%	0.27%	0.11%	0.07%	0.06%	0.05%
std	0.0013	0.0010	0.00053	0.00035	0.00029	0.00026	0.00028

The test result of 20 runs for d = 1,...,7 is:

d	1	2	3	4	5	6	7
mean	9.59%	3.92%	3.07%	2.78%	2.77%	2.75%	2.83%
std	0.0201	0.0068	0.0046	0.0033	0.0036	0.0044	0.0036

2. Comparing with the question 1, this question uses 5-fold cross validation to select the "best" parameter d^* and then use the best d to retrain on full 80% training set. Consequently, the dataset is divided into 5 equal parts, and each one has 1448 records. The outputs of 20 d^* and 20 test errors are shown in two matrix. Using statements **mean** and **std** to calculate a mean test error±std and a mean d^* with std. Performing 20 runs, the result is: mean test error = 2.70% ± 0.0032 and d^* = 4.65±1.1367.

3. Each parameter d^* produces confusions. Based on question 2, the aim of this question is to output a 10x10 confusion matrix, where each cell contains a confusion error and its standard deviation. Performing 20 runs, the final mean confusion error matrix is:

	0	1	2	3	4	5	6	7	8	9
0	0	0.05	0.65	0.25	0.15	1.35	1.20	0	1.15	0.25
1	0.35	0	0.25	0.25	1.1	0.1	0.35	0.4	0.6	0.1
2	0.6	0.2	0	0.8	1.05	0.55	0.4	0.8	0.4	0.1
3	0.65	0.2	1.35	0	0.05	1.65	0	0.3	1.8	0.3
4	0.3	0.85	1.1	0.2	0	0.5	0.9	1.25	0.8	1.95
5	0.5	0	0.2	2.25	0.3	0	0.25	0.	1.85	0.1
6	0.4	0.5	0.3	0.05	0.6	1.05	0	0	0.25	0.05
7	0.15	0.1	0.85	0.5	0.25	0.15	0	0	0.65	2.3
8	0.2	0.1	0.65	1.6	0.25	0.75	0.55	0.45	0	0.55
9	0.15	0.2	0	0.05	1.45	0.35	0.1	1.1	0.65	0

The final standard deviation matrix of confusion error is:

CONTINUED

	0	1	2	3	4	5	6	7	8	9
0	0	0.22	0.88	0.44	0.37	1.22	1.15	0	1.30	0.44
1	0.74	0	0.44	0.44	1.07	0.45	0.59	0.9	0.88	0.45
2	0.88	0.41	0	0.77	0.89	0.76	0.60	0.95	0.68	0.31
3	0.67	0.52	1.18	0	0.22	1.14	0	0.47	1.40	0.66
4	0.47	0.99	1.12	0.41	0	0.69	0.64	1.12	1.11	1.67
5	0.76	0	0.41	2.17	0.57	0	0.55	0.41	1.27	0.31
6	0.60	0.69	0.66	0.22	0.82	1.00	0	0	0.44	0.22
7	0.37	0.31	0.93	0.76	0.44	0.49	0	0	0.67	1.38
8	0.41	0.31	0.75	1.39	0.64	0.79	0.83	0.60	0	0.69
9	0.37	0.41	0	0.22	1.19	0.59	0.31	1.21	0.59	0

4. The training set in this part uses the whole dataset which has 9298 records. Keep training alpha and calculating the number of mistakes until the number of mistakes is less than or equal to 5. Running this program, the five hardest digits to predict can be seen below:

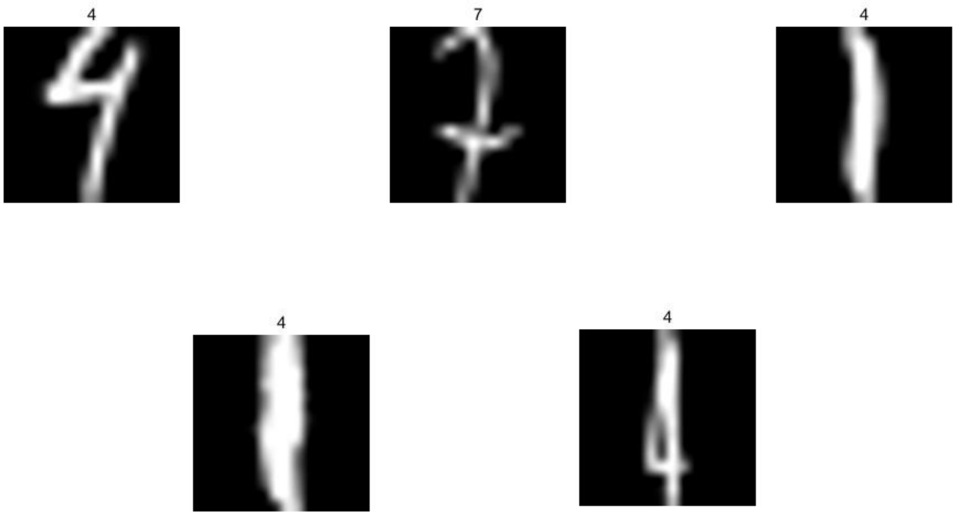

Figure 1: five hardest digits to predict

5. Change the kernel from the polynomial kernel $K_d(p,q) = (p,d)^d$ to the Gaussian kernel $K(p,q) = e^{-c||p-q||^2}$, where c is the width of the kernel. Since the normal function of the Gaussian can be written as: $K(p,q) = exp(-\frac{||p-q||^2}{2\sigma^2})$, c equals to $\frac{1}{2\sigma^2}$.

   Performing 20 runs, the train result for σ = 1,...,9 is:

σ	1	2	3	4	5	6	7	8	9
mean	23.88%	0.084%	0.031%	0.030%	0.036%	0.053%	0.089%	0.18%	0.33%
std	0.0071	0.00033	0.00021	0.00020	0.00016	0.00022	0.00024	0.00045	0.00061

   The test result of 20 runs for σ = 1,...,9 is:

σ	1	2	3	4	5	6	7	8	9
mean	28.01%	5.02%	4.05%	2.93%	2.58%	2.64%	2.84%	3.02%	3.15%
std	0.0145	0.0049	0.0032	0.0028	0.0021	0.0026	0.0033	0.0044	0.0041

Performing 5-fold cross validation, the result of 20 runs for σ = 1,...,9 is: mean test error = 2.57% $\pm$ 0.0029 and σ^* = 5.55$\pm$0.5104.

6. The method in this part uses single to single. Performing 20 runs, the train result for d = 1,...7 is:

d	1	2	3	4	5	6	7
mean	2.11%	0.28%	0.033%	0.018%	0.014%	0.018%	0.067%
std	0.0036	0.0021	0.00052	0.00059	0.00033	0.00043	0.00021

The test result of 20 runs for d = 1,...,7 is:

d	1	2	3	4	5	6	7
mean	10.36%	6.89%	5.82%	5.65%	5.84%	5.76%	5.78%
std	0.0240	0.0092	0.0067	0.0082	0.0062	0.0063	0.0043

Performing 5-fold cross validation, the result of 20 runs for d = 1,...,7 is: mean test error = 5.52% $\pm$ 0.0069 and d^* = 4.45$\pm$1.099.

7. The first algorithm comparing to the kernel perceptron is Support Vector Machine. It uses the library in Matlab, which includes the statement **fitcecoc**. Performing 20 runs, the train result for σ = 5,...,10 is:

σ	5	6	7	8	9	10
mean	0.075%	0.089%	0.11%	0.14%	0.23%	0.36%
std	0.000075	0.00013	0.00021	0.00020	0.00019	0.00041

The test result of 20 runs for σ = 5,...,10 is:

σ	5	6	7	8	9	10
mean	12.24%	4.95%	3.48%	3.10%	3.06%	2.90%
std	0.0081	0.0041	0.0035	0.0046	0.0029	0.0041

Performing 5-fold cross validation, the result of 20 runs for d = 5,...,10 is: mean test error = 5.52% $\pm$ 0.0069 and σ^* = 4.45$\pm$1.099.

The second algorithm comparing to the kernel perceptron is Random Forests. It uses the library in Matlab, which includes the statement **fitensemble**. Performing 20 runs, the train result for bag = 200 keeps 0. The test result of 20 runs for bag =100, 150 and 200 is:

bag	100	150	200
mean	3.41%	3.19%	3.23%
std	0.0056	0.0038	0.004

CONTINUED

Discussion

In question 1, the train can converge to the error which approximates to 0, it means that the classifier can fit the train value well. However, when the number of epoch increases, the error in the test decreases in the beginning and then gradually increases. The reason of it is the overfit of classifier to the train dataset. In this case, when the number of epoch increases, the error in the training decreases and converges to an interval, since it can have an excellent fit to the training set. It means that it is too accurate to the training to fit other sets. Consequently, it needs to find a suitable epoch. Comparing epoch from 1 to 10, epoch seems to have the best effect when it is 5.

Comparing the mean errors of d from 1 to 7, when d equals to 5, it has the minimum value of error. It is also because of the overfit of the classifier to the training set. Comparing the polynomials of each dimension, when the value of d increases, the fit will gradually fit and then overfit. Consequently, the value of d needs to be derived.

Since the best d always oscillates in each run so the way of 5-fold cross-validation is applied, which can calculate the best parameter d^* and then retrain on the full 80% training set, which will decrease the error comparing with a fixed parameter d. The mean test error in question 2 is 2.70%, which is smaller than each of the mean test error without 5-fold cross-validation.

Comparing the five digits shown in Figure 1, the first digit is similar to the digit 9, the second digit has a bar in the middle and the fifth one is similar to the digit 1. Furthermore, the third and forth digits misclassify the real value from digit 1 to digit 4. Consequently, the digits which are hard to predict are convincing. The digits in dataset still have some confusing digits. In this case, the program may display different digits in each runing time.

In question 5, as what can be seen in the train result table, the mean error is small enough when the value of σ is larger than 1. When $\sigma = 5$, the mean test error has the minimum value, which is 2.58%. Knowing that $c = \frac{1}{2\sigma^2}$, so in the situation of c = 0.02, the mean test error has the minimum value.

Comparing the minimum mean results of Gaussian to the polynomial Kernel, 2.57% is smaller than 2.70%. Consequently, it seems that Gaussian Kernel has a better fit to the test set

There are two methods chosen for generalizing 2-class classifiers to k-class classifiers, the first one is creating 10 classifiers and comparing all weights together. Which has been written in the algorithm part. The other one is comparing weights single to single, which means that classifier needs to compare 9 times to obtain the best predicted value of y. In this case, a 9x9 matrix is created to comparing weights. The matrix can be written as:

1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	0
18	19	20	21	22	23	24	0	0
25	26	27	28	29	30	0	0	0
31	32	33	34	35	0	0	0	0
36	37	38	39	0	0	0	0	0
40	41	42	0	0	0	0	0	0
43	44	0	0	0	0	0	0	0
45	0	0	0	0	0	0	0	0

For example, predicting the value of y between 0 and 1, if sign(predicted) = +1, 0 will compare with 2 then. If sign(predicted) = -1, 1 will compare with 2 then. If the predicted value if not as the expected value, the classifier updates. After the value comparing with 9, the program will handle the next row of code. In the test part, values of classifiers will not be updated, the predict values of each y still need to be compared one to one. After all calculation have done, use a matrix to calculate mistakes.

The epoch of this algorithm chooses 2, since the algorithm has a quick converges in training set. As what can be seen in the table, when d = 4, the program can obtain the smallest mean test error. When using 5-fold cross-validation, the mean test error is less than the value without 5-fold cross-validation. Consequently, the final test errors are reasonable.

As for the term of time complexity, the Gaussian kernel seems to have the smallest time complexity comparing with the other two algorithms since the Gaussian needs less parameters. One to one class method with polynomial kernel has more iterations comparing with one to all classes. Consequently, it has larger time complexity than one to all classes method with polynomial kernel.

In question 7, the algorithm Support Vector Machine is similar with the algorithm in one to one methods, so it still has a large time complexity. When σ = 10, the program can obtain the smallest mean test error, which is 2.90%. As for the algorithm Random Forests, when bag = 150, the program can obtain the smallest mean test error.

2 PART II

2.1

a. Here are sample complexity plots for 4 classification algorithms. In this section, due to the extremely high time complexity, it is necessary to trade-off accuracy and computation time. The details of estimating sample complexity will be shown in next section.

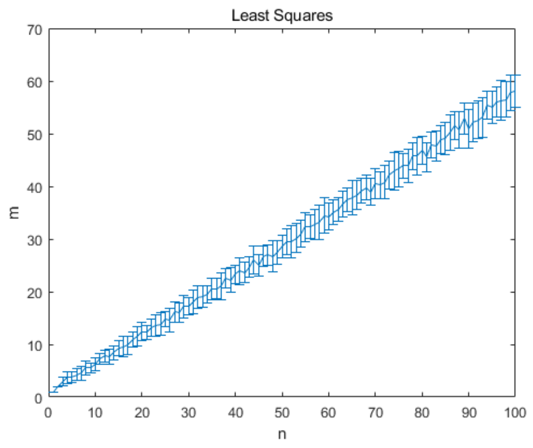

Figure 2: Estimated number of samples (m) to obtain 10% generalisation error versus dimension (n) for least squares.

CONTINUED

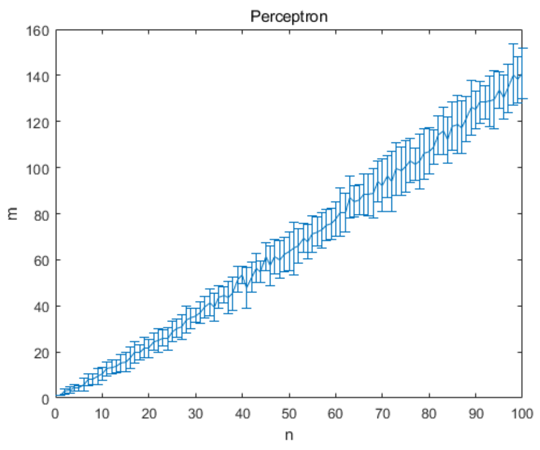

Figure 3: Estimated number of samples (m) to obtain 10% generalisation error versus dimension (n) for perceptron.

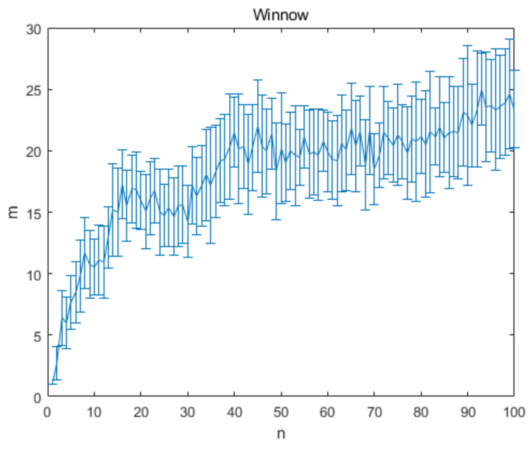

Figure 4: Estimated number of samples (m) to obtain 10% generalisation error versus dimension (n) for winnow.

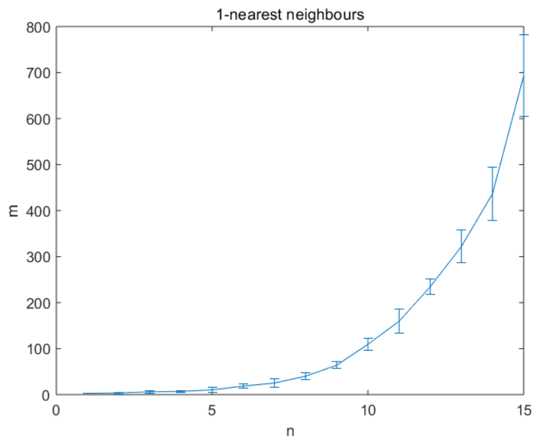

Figure 5: Estimated number of samples (m) to obtain 10% generalisation error versus dimension (n) for 1-nearest neighbours.

b. Due to the expensive computational cost of computing sample complexity, it is necessary to trade-off accuracy and computation time.
i) For each dimension **n** , there is a possibility matrix which contains 2^n possibilities sampled from $\{-1, 1\}^n$. In the ideal situation, the pattern should be taken randomly from the possibility matrix based on the example quantity **m**. For instance, if n=3 and m=2, the possibility matrix will have 2^n=8 rows; since the pattern is consist by two rows from possibility matrix, there are $(2^n)^m = 2^{mn}$ different patterns. However, that will have 2^{mn} times computation which will take a long time. Therefore, to achieve best simulation, it is a good choice to sample from the possibility matrix randomly and then take the average. The generalization error for exact **m** and **n** will be the expectation of cumulative errors in the test data set. Besides, repeating the whole procedure 20 times will minimize the error. For example, for each n, there will be 20 minimum values of m once the program has finished. Hence, the average value of them will get more close to the ideal value.
The trade-off method will be displayed in the following pseudo-code.

CONTINUED

Algorithm 1 Estimation method using trade-off

```
for $test\_test = 1 \to 20$ do %trade-off
    for $n = 1 \to max\_n$ do
        $m = 1$
        while 1 do
            % generate a m×n matrix sample randomly from $\{-1, 1\}^n$
            $[x\_train, y\_train] = data\_generation(m, n)$ %trade-off

            $training\ algorithm$ (1NN, least squares, winnow, perceptron)

            %generate test data set which as a much bigger m and same n as training data set
            $[x\_test, y\_test] = data\_generation(m\_test, n)$ %trade-off

            $error = 0$
            if $prediction! = y\_test$ then
                $error = error + 1$
            end if
            $generalization\_error = \frac{error}{m\_test} error$
            if $generalization\_error < 0.1$ then
                $break$
            else
                $m = m + 1$
            end if
        end while
        $M\_set(n, test\_time) = m$
    end for
end for
```

c. For least squares and perception classification algorithms, the sample complexity grows linearly as a function of dimension so that $m = \theta(n)$.
For 1 nearest neighbour algorithm, the relationship between m and n seems to be a exponential function.
Besides, for winnow algorithms, the plot of sample complexity can be indicated as a logarithmic function.
Therefore, for small n, 1 nearest neighbour algorithm have better performance. For example, sample quantity m increase slowly for $n < 10$ in Figure 4; for other algorithms, m will be larger. Similarly, winnow algorithm is better than others due while $n > 30$.
d. (VAPNIK AND CHERVONENKI therorem)
Suppose that m was drawn uniformly at random from $\{1, ..., m\}$ and perceptron algorithm has been trained on examples$(x_1, y_1), ...(x_{m+1}, y_{m+1})$ until finding the convergence.The mistakes occur on a total of examples with indices $i_1, ..., i_k$. Assume R = $max_{1 \leq j \leq k} ||x||$, and let

$$\gamma = max_{||u||=1}\, min_{1 \leq j \leq k}\, y_{i_j}(u * x_{i_j})$$

Assume $\gamma > 0$ with probability one. Hence if we run perceptron algorithm to convergence on training data$(x_1, y_1), ...(x_m, y_m)$, the probability that the final perceptron does not predict y_{m+1} on the test instance x_{m+1} should be

$$\frac{1}{m+1} E[min\{k, (\frac{R}{\gamma})^2\}]$$

In this problem, the generalization error should be 0.1 and then k is 0.1m, s=m+1. Hence, the upper bound is

$$\frac{1}{s} E[min\{0.1m, (\frac{R}{\gamma})^2\}]$$

Because we randomly draw from $\{1, ..., m\}$, then the upper bound is

$$p_{\hat{m,n}} = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{s} E[min\{0.1m, (\frac{R}{\gamma})^2\}]$$

END OF COURSEWORK