COMP0078

Supervised Learning

Coursework 2

Student Number: 22108699

December 14th, 2022

Contents

1 Part I: Kernel Perceptron

1.1 Discussion

1.1.1 Method to choose a suitable number of epoch.

While training, we split a validation train/test set from the train dataset and use it to determine the number of epoch. We don't stop our training until the validation error not decrease in two consecutive cycles. Hence, the number of epoch is not fixed and by our experiments, the average of number of the epoch is around 4.

1.1.2 Methods to build a k-class kernel perceptron.

Two methods are introduced to generalizing a k-class perceptron. One-versus-rest is applied in problem 1-5 and One-versus-one in problem 6.

One-versus-rest: splits a multi-class classification into one binary classification problem per class. In our case, we have 10 binary classifiers. For the input data we will return 10 prediction results called confidences corresponding to the 10 class. We final choose the class with the biggest result.

One-versus-one: splits a multi-class classification into one binary classification problem per each pair of classes. In our case, we have 45 binary classifiers. Each classifiers will vote and we choose the class has the largest votes. While there are several class has the same votes, we choose the result by uniform distribution.

1.1.3 Compare the Polynomial kernel with the Gaussian kernel.

See corresponding section.

1.1.4 Implementation of the kernel perceptron

If there is a mistake, according to the update rule, we will update the parameter $w_{t+1} = w_t + y_t x_t$ and rewrite it to $w_t = \sum_{j\in \text{ mistakes}} \alpha_j x_j$, $\alpha_j = y(x_j)$. By introducing the kernel method, the predicted label of the t_{th} instance x_t is

$$\begin{aligned}\hat{y}_t =& w_t \cdot \Phi\left(x_t\right) \\ =& \left(\sum_{j\in \text{ mistakes}} \alpha_j \Phi\left(x_j\right)\right) \cdot \Phi\left(x_t\right) \\ =& \sum_{j\in \text{ mistakes}} \alpha_j K\left(x_j, x_t\right)\end{aligned}$$

In practice, we initialize the $\alpha \in \Re^{(k,m)}$ as a zero matrix and $\alpha_{(i,j)}$ will be updated if the i_{th} class is wrong for instance x_j.

	Algorithm for one-vs-rest:
Input:	$x_1, y_1, \ldots, (x_m, y_m) \in (R^n, -1, +1)^m$
Initialization:	$\alpha_0 = 0$ for all classifier
Prediction:	Upon receiving the t_{th} instance x_t, predict $\hat{y}_t = \text{sign}\left(\sum_{i=0}^{t-1} \alpha_i K(x_i, x_t)\right)$ for all classifier
Update:	if $\hat{y}_t \neq y_t$ then $\alpha_t = \alpha_t + y_t$

1.2 Experiments

1.2.1 Basic results

We apply the k-class Perceptron model on zipcombo.dat dataset. We use 80% of the dataset to train our model and test it on the rest. While training, we split 10% from training dataset as the validating dataset to determine the number of epoch. The parameter of model is updated only during the training process.

d-values	Training error rate	Test error rate
1	0.86 ± 0.137	0.1106 ± 0.0147
2	0.0133 ± 0.0085	0.044 ± 0.0086
3	0.0116 ± 0.023	0.0341 ± 0.0046
4	0.0029 ± 0.028	0.275 ± 0.04
5	0.0035 ± 0.0034	0.0302 ± 0.0057
6	0.0028 ± 0.003	0.03 ± 0.0038
7	0.0025 ± 0.003	0.0289 ± 0.0033

Table 1: Error rate of k-class Perceptron with polynomial kernel with different d.

1.2.2 Cross-validation

We run 20 times to find the best d^*, for each run, we apply 5-fold cross-validation to select d^* among this run and train and test our model again with d^*. Test error is recorded.

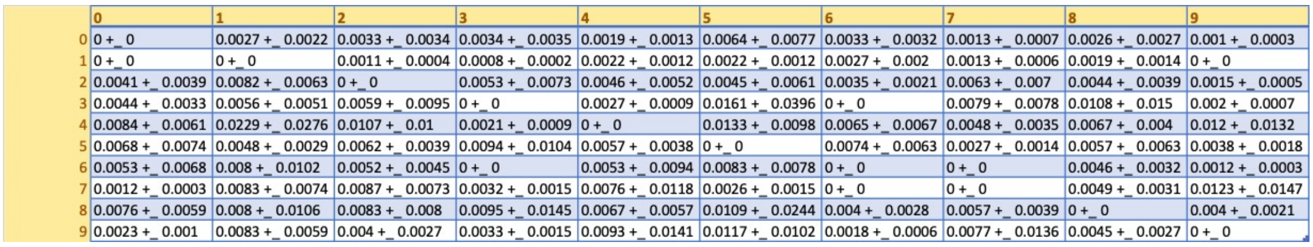
d^*	Mean test error rate
6.0500 ± 1.0235	0.0438 ± 0.0043

1.2.3 Confusion matrix

we repeat the above experiment and record the confusion matrix of test data for each run and average them for elements in the same position (i, j). for every position (i, j) in the confusion matrix, K_i is the true label and K_j is the wrong predicted label while we have the label set K. The mean of confusion matrix and its standard deviation is attached in Fig 1.

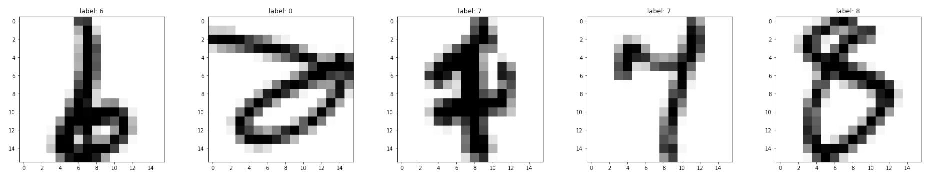
1.2.4 Results analysis

There are the five hardest digits with their labels from the above experiments shown in FIg 2. As we can see, these digital images are blurred and even humans have a hard time recognizing these numbers.



	0	1	2	3	4	5	6	7	8	9
0	0 +_ 0	0.0027 +_ 0.0022	0.0033 +_ 0.0034	0.0034 +_ 0.0035	0.0019 +_ 0.0013	0.0064 +_ 0.0077	0.0033 +_ 0.0032	0.0013 +_ 0.0007	0.0026 +_ 0.0027	0.001 +_ 0.0003
1	0 +_ 0	0 +_ 0	0.0011 +_ 0.0004	0.0008 +_ 0.0002	0.0022 +_ 0.0012	0.0022 +_ 0.0012	0.0027 +_ 0.002	0.0013 +_ 0.0006	0.0019 +_ 0.0014	0 +_ 0
2	0.0041 +_ 0.0039	0.0082 +_ 0.0063	0 +_ 0	0.0053 +_ 0.0073	0.0046 +_ 0.0052	0.0045 +_ 0.0061	0.0035 +_ 0.0021	0.0063 +_ 0.007	0.0044 +_ 0.0039	0.0015 +_ 0.0005
3	0.0044 +_ 0.0033	0.0056 +_ 0.0051	0.0059 +_ 0.0095	0 +_ 0	0.0027 +_ 0.0009	0.0161 +_ 0.0396	0 +_ 0	0.0079 +_ 0.0078	0.0108 +_ 0.015	0.002 +_ 0.0007
4	0.0084 +_ 0.0061	0.0229 +_ 0.0276	0.0107 +_ 0.01	0.0021 +_ 0.0009	0 +_ 0	0.0133 +_ 0.0098	0.0065 +_ 0.0067	0.0048 +_ 0.0035	0.0067 +_ 0.004	0.012 +_ 0.0132
5	0.0068 +_ 0.0074	0.0048 +_ 0.0029	0.0062 +_ 0.0039	0.0094 +_ 0.0104	0.0057 +_ 0.0038	0 +_ 0	0.0074 +_ 0.0063	0.0027 +_ 0.0014	0.0057 +_ 0.0063	0.0038 +_ 0.0018
6	0.0053 +_ 0.0068	0.008 +_ 0.0102	0.0052 +_ 0.0045	0 +_ 0	0.0053 +_ 0.0094	0.0083 +_ 0.0078	0 +_ 0	0 +_ 0	0.0046 +_ 0.0032	0.0012 +_ 0.0003
7	0.0012 +_ 0.0003	0.0083 +_ 0.0074	0.0087 +_ 0.0073	0.0032 +_ 0.0015	0.0076 +_ 0.0118	0.0026 +_ 0.0015	0 +_ 0	0 +_ 0	0.0049 +_ 0.0031	0.0123 +_ 0.0147
8	0.0076 +_ 0.0059	0.008 +_ 0.0106	0.0083 +_ 0.008	0.0095 +_ 0.0145	0.0067 +_ 0.0057	0.0109 +_ 0.0244	0.004 +_ 0.0028	0.0057 +_ 0.0039	0 +_ 0	0.004 +_ 0.0021
9	0.0023 +_ 0.001	0.0083 +_ 0.0059	0.004 +_ 0.0027	0.0033 +_ 0.0015	0.0093 +_ 0.0141	0.0117 +_ 0.0102	0.0018 +_ 0.0006	0.0077 +_ 0.0136	0.0045 +_ 0.0027	0 +_ 0

Figure 1: Confusion matrix with polynomial kernel



(a) (b) (c) (d) (e)

Figure 2: Five hardest digits.

1.2.5 Gaussian kernel

We repeat the problem 1 and 2 with the Gaussian kernel method in order to enlarge our feature dimensions. The parameter c is chosen from set $S = \{0.01, 0.1, 1, 10, 100\}$. By repeating the first problem, we have a training error rate and testing error rate for each parameter c. As we can see, when $c = 0.01$ we can get a local minimum of testing error rate.

c-values	Training error rate	Test error rate
0.01	0.0041 ± 0.0040	0.0297 ± 0.0048
0.1	0.0105 ± 0.0225	0.0528 ± 0.0054
1	0.0297 ± 0.0329	0.0695 ± 0.0041
10	0.2241 ± 0.0267	0.1656 ± 0.0136
100	0.7231 ± 0.0030	0.7195 ± 0.0100

Table 2: Error rate of k-class Perceptron with gaussian kernel with different c.

Then for problem 2, we redesign our set $S = \{0.001, 0.02575, 0.0505, 0.07525, 0.1\}$ considering the efficiency of computing. By 5-fold cross validation, we can get a optimal c^* and corresponding testing error. (Due to a mistake, we did not save the results, but you can run function **experiment__5__2** in file **perceptron.py** to get the results). Roughly speaking, $d^* \approx 0.01$ with lower testing error rate compared with polynomial kernel model.

1.2.6 Alternate method

One-versus-one: splits a multi-class classification into one binary classification problem per each pair of classes. In our case, we have 45 binary classifiers. Each classifiers will vote and we choose the class has the largest votes. While there are several class has the same votes, we choose the result uniformly.

2 Part II: Laplacian Interpolation

2.1 Results

	1	2	4	8	16
50	0.0301± 0.0022	0.0312± 0.0	0.0299± 0.0047	0.0304± 0.0059	0.0287± 0.0098
100	0.0197± 0.0015	0.0191± 0.0022	0.0201± 0.0019	0.0209± 0.0019	0.0205± 0.0044
200	0.0074± 0.0005	0.0074± 0.0006	0.0074± 0.0008	0.0074± 0.0009	0.0077± 0.001
400	0.0063± 0.0	0.0063± 0.0	0.0062± 0.0003	0.0063± 0.0003	0.0063± 0.0005

Table 3: Errors and standard deviations for semi-supervised learning via LI

	1	2	4	8	16
50	0.1515± 0.0924	0.0719± 0.0422	0.0777± 0.0589	0.0643± 0.0329	0.05± 0.022
100	0.0952± 0.1918	0.0434± 0.0215	0.0409± 0.0047	0.0375± 0.0069	0.0369± 0.0111
200	0.0313± 0.0341	0.0176± 0.0059	0.0184± 0.0033	0.0156± 0.0033	0.0166± 0.0042
400	0.0305± 0.0753	0.012± 0.0025	0.0125± 0.0024	0.0119± 0.0018	0.0114± 0.002

Table 4: Errors and standard deviations for semi-supervised learning via LKI

2.2 Observations

Something interesting:

1. Getting more knowledge could help us classify, as two table show, more known data, less error rate for a fixed dataset. But knowing too much may hurt us. In the last column of two tables, the error rates decrease as the Known proportions decreasing. $Error\ rates(Known\ proportions)$ may be a convex function.
2. LKI is more sensitive to the proportions of known data than LI. The standard deviations in the second table is higher than the number in the first table in the most cases.

2.3 Compare

Overall, the semi-supervised learning via LI is better than via LKI in the current datasets (more stable, significantly less error rates). One reason may be that the data of grey digits is highly non-linear.

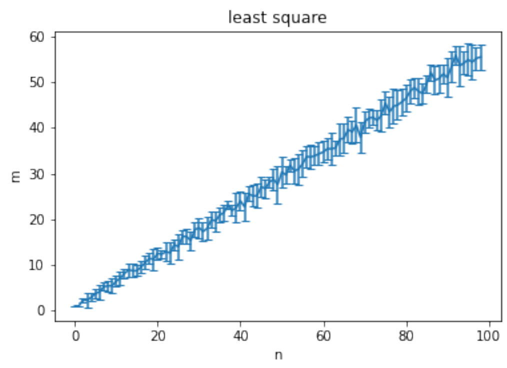
2.4 Improvement

Normalization of data could be a method to improve the performance of LKI technology.

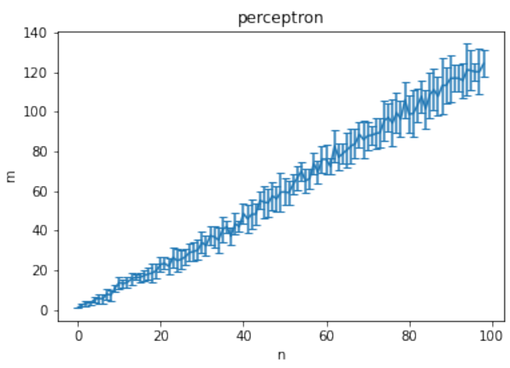
3 Part III: Sparse Learning

3.1 A

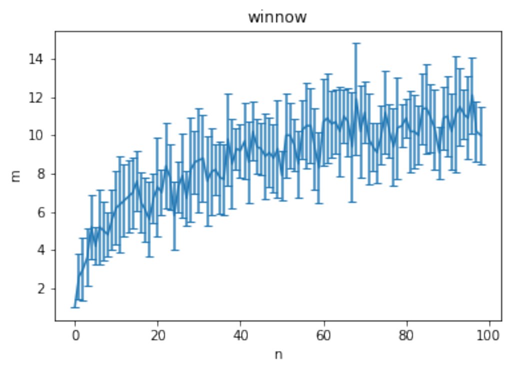
We first implement four binary classification algorithms and measure the sample complexity of four algorithms. While the generalization error is smaller than 0.1, the m is recorded for each n and repeat it for getting mean and standard deviation. The images of m-versus-n are shown below



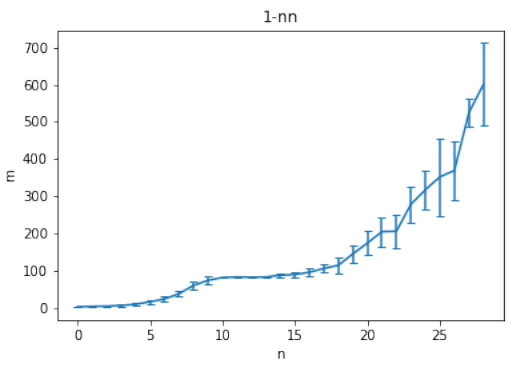
(a)



(b)



(c)



(d)

Figure 3: The number of samples (m) to achieve 10% generalization error versus to dimension n for four algorithm.

3.2 B

Given n, we increasing our m from 1 to 10000. for a pair of (m, n), we create our training dataset $(X_{train}, y_{train}), X_{train} \in \Re^{m,n}$ as $m = 10000$ training data can cover almost all sampling space and we set the size of testing data to $(10 * n, n)$. For every pair (m, n), we repeat it for ten times and record the smallest m that obtain 10% generalization error versus to n in each iteration.

Though m up to 10000, it is impossible to sample all data in n-dimension space. It is noticed that when n is small, the corresponding m is smooth with a low standard deviation, and as n increases, a large m is needed to get a small generalization error and because we create our training data randomly, the sampling space in each iteration is various and m has a larger standard deviation. The biases could occurs when one data generated several times in our testing dataset.

3.3 C

Easy to see the there exists a linear relation between m and n in perceptron and least square model. For the linear relation $m = \alpha n$, $\alpha_{perceptron}$ is larger than $\alpha_{least\ square}$. Hence, the sample complexity of perceptron and least square algorithm are $\Theta(n)$.

For winnow algorithm, the shape of curve $m(n)$ is similar to log function, that is, $m = a\log(n), a > 0$. Considering the standard deviation, a is bounded by $a \in [b, c], b, c \in \Re, b > 0$. Hence, we conclude the sample complexity of winnow algorithm is $\Theta(\log(n))$.

For 1-NN method, it looks like there is a exponential relation between m and n. The curve can be fitted by a exponential function $m = exp(an), a \in \Re$ and hence we conclude the sample complexity of 1-NN method is $\Theta(\exp(n))$.

Based our observations, the winnow algorithm is the most efficient method to this sparse learning problem if n is big enough and 1-NN is real time-consuming as its sample complexity is $\Theta(\exp(n))$. Least square and perceptron have a closed performance and solve it in a linear time.

3.4 D

Let $\hat{p}_{m,n}$ be the upper bound on the probability that the perceptron will make a mistake on the sth sample after training on samples. According to the Nobikoff Perceptron Bound, the mistakes M is bounder above by $M \leq \left(\frac{R}{\gamma}\right)^2$ with $R := \max_t \|\mathbf{x}_t\|$ when there exists a vector $\|\mathbf{v}\| = 1$ and $(\mathbf{v} \cdot \mathbf{x}_i)\, y_t \geq \gamma$ for a constant value γ. In this problem, we have $R = \sqrt{1 + 1 + \ldots + 1} = \sqrt{n}$. The minimum separation between two points is 1 and thus we have $\gamma = 1$. Hence, $M \leq \left(\frac{R}{\gamma}\right)^2 \leq n$. By the Batch Bounds given by online bounds theorem, we have $\text{Prob}\left(\mathcal{A}_s\left(\mathbf{x}'\right) \neq y'\right) \leq \frac{B}{m}$. Because B is the mistake bound for algorithm $\mathcal{A}$ for any such set. Combined above two theorem, we have $\text{Prob}\left(\mathcal{A}_s\left(\mathbf{x}_s\right) \neq Y_s\right) \leq \frac{B}{m} = \frac{n}{m}$, that is ,the upper bound is $\hat{p}_{m,n} = \frac{n}{m}$.

3.5 E

By part C, we have seen that the sample complexity of 1-NN is $\Theta(\exp(n))$. Based on the idea of 1-NN, to get a small generalization error, we are supposed to sample the data points as many as we can in every subspace of $\Re^n$, denoted it as α, then we get a total α^n data points in space $\Re^n$, that is, the relation between generalization error and dimension n is exponentially. From part C, we can assume $m = \exp(an), a \in \Re$ is a good lower bound of the sample complexity and hence we have $m = \Omega(f(n)) = \Omega(\exp(an))$.