ASSIGNMENT 2

DIZHI MA

1.

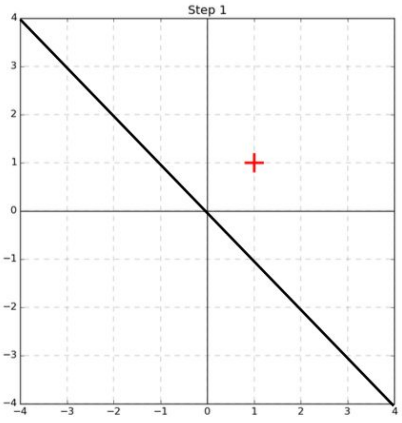
1.1

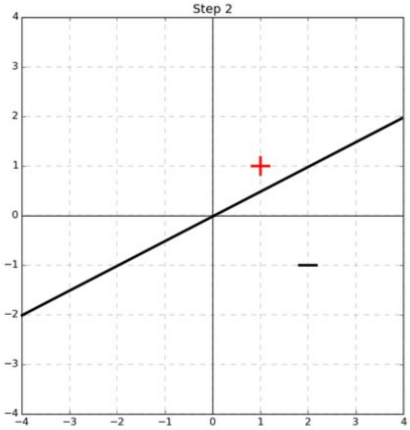
step1: $a = wx = 0\,, ay = 0 \leq 0\,, w = w + yx = [1, 1]$

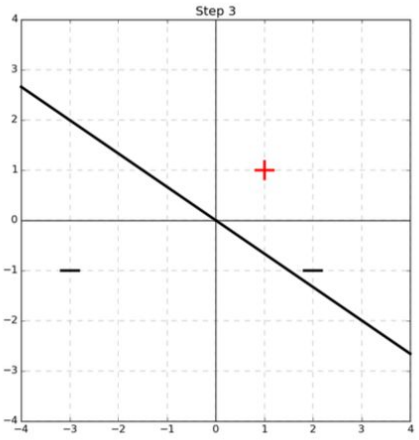
step2: $a = wx = 1\,, ay = 0 \leq 0\,, w = w + yx = [-1, 2]$

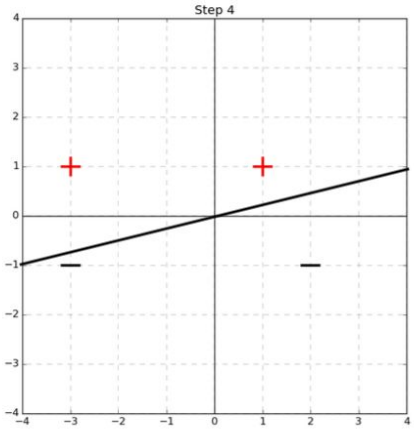
step3: $a = wx = 1\,, ay = 0 \leq 0\,, w = w + yx = [2, 3]$

step4: $a = wx = -3\,, ay = 0 \leq 0\,, w = w + yx = [-1, 4]$



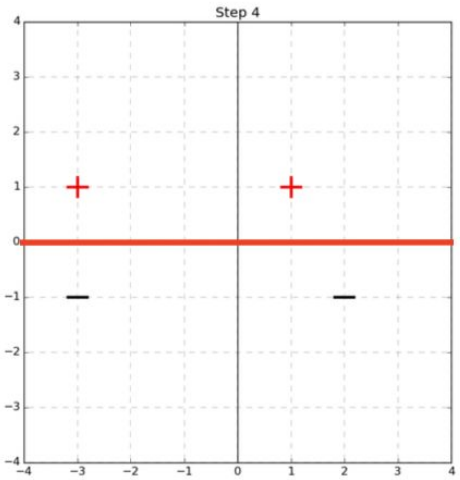






1.2

No, it doesn't. The maximum-margin decision boundary showed as follow



1.3

Assume there is a unit vector w^* that can linearly separate the data, with the margin $\gamma > 0$.

Each time we make mistake the weight will update:

$$w^{(t+1)} = w^{(t)} + y_t x_t$$

Dot product with w^* on both side:

$$w^* w^{(t+1)} = w^*(w^{(t)} + y_t x_t)$$
$$w^* w^{(t+1)} = w^* w^{(t)} + w^* y_t x_t$$
$$w^* w^{(t+1)} \geq w^* w^{(t)} + \gamma$$

Thus, after m mistake we get:

$$w^* w^{(t+1)} \geq m\gamma \quad (1)$$

The norm of $w^{(t+1)}$:

$$||w^{(t+1)}||^2 = ||w^t||^2 + 2y_t(w^t x_t) + ||x_t||^2 \leq ||w^t||^2 + ||x_t||^2 \leq ||w^t||^2 + R^2$$

Thus, after m mistake we get:

$$||w^{(t+1)}||^2 \leq mR^2 \quad (2)$$

Substitue (2) into (1)

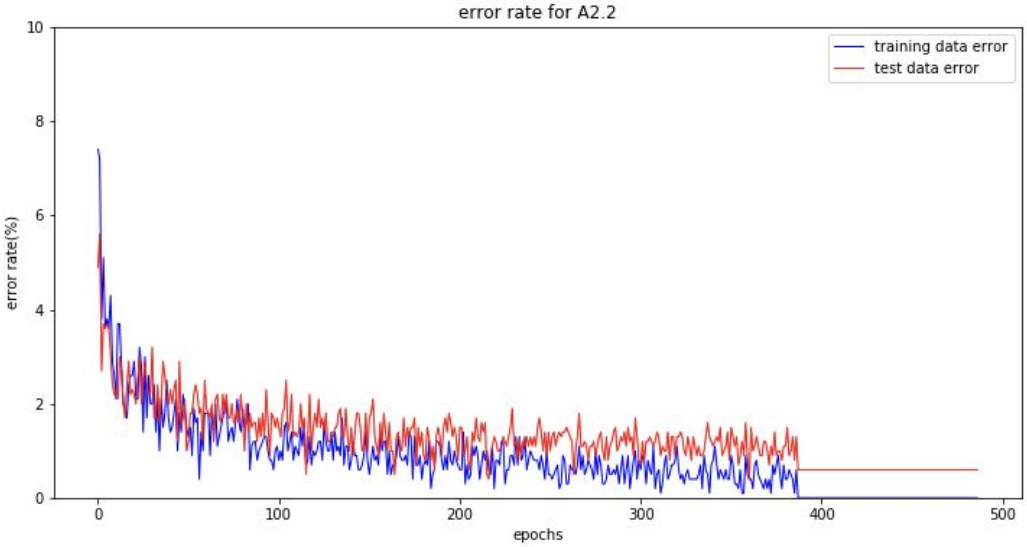
$$m \leq R^2/\gamma^2$$

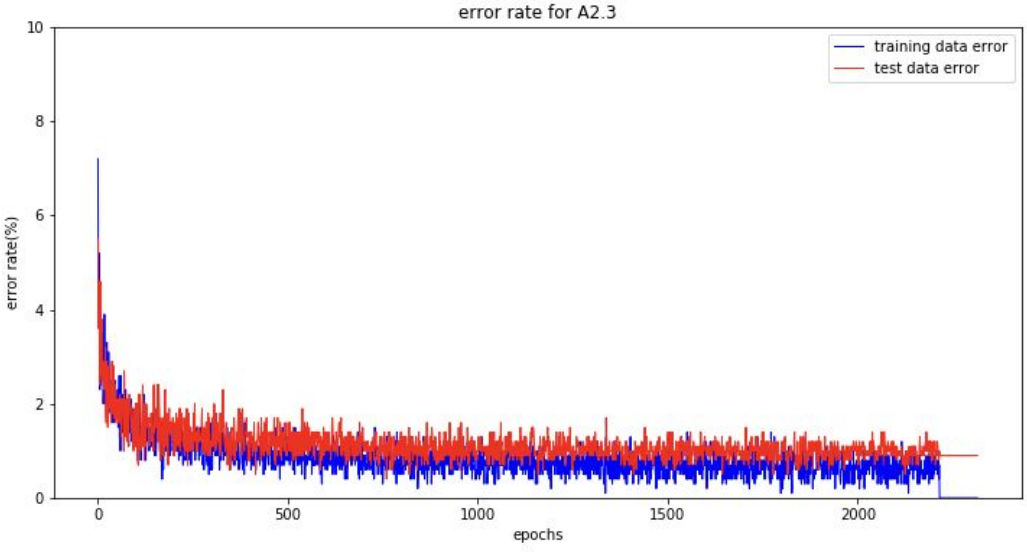
Therefore, in this question we can make at most $R^2/\gamma^2 = 25/0.25 = 100$ mistakes.

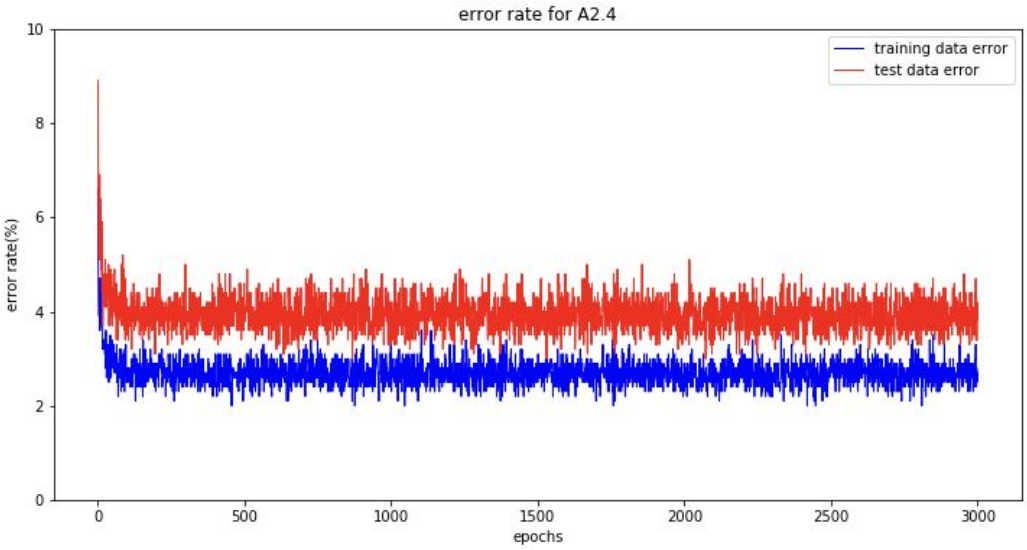
2.

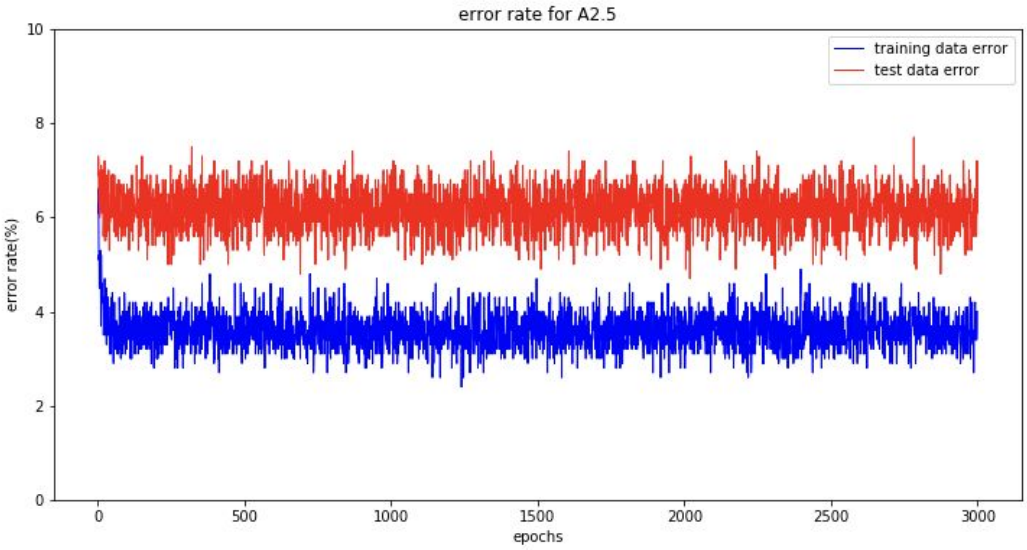
2.1

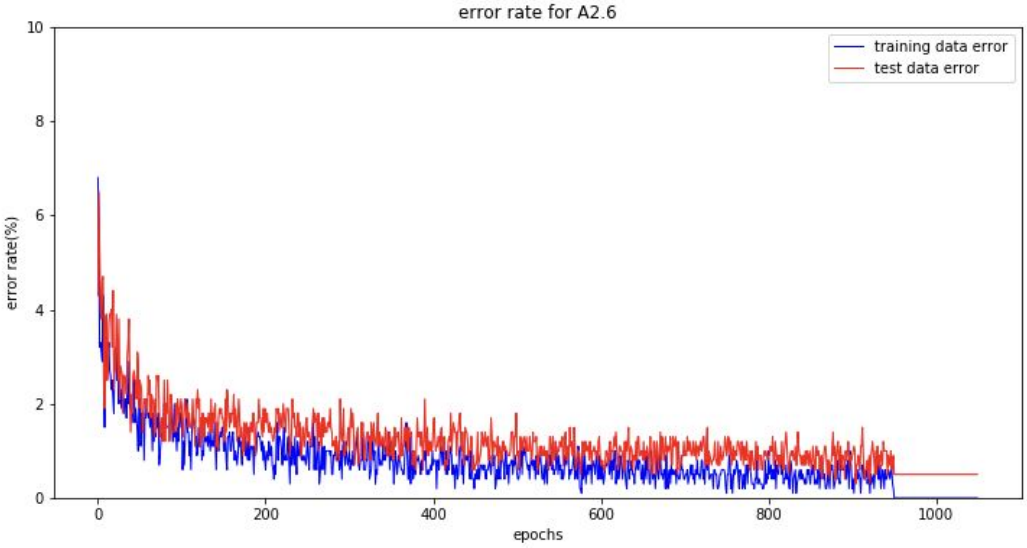
In my program the perceptron will run for 3000 epochs if the error rate is not stable(in here, stable means that the error rate not changing in 100 epochs). Perceptron performance plots showed as followed:

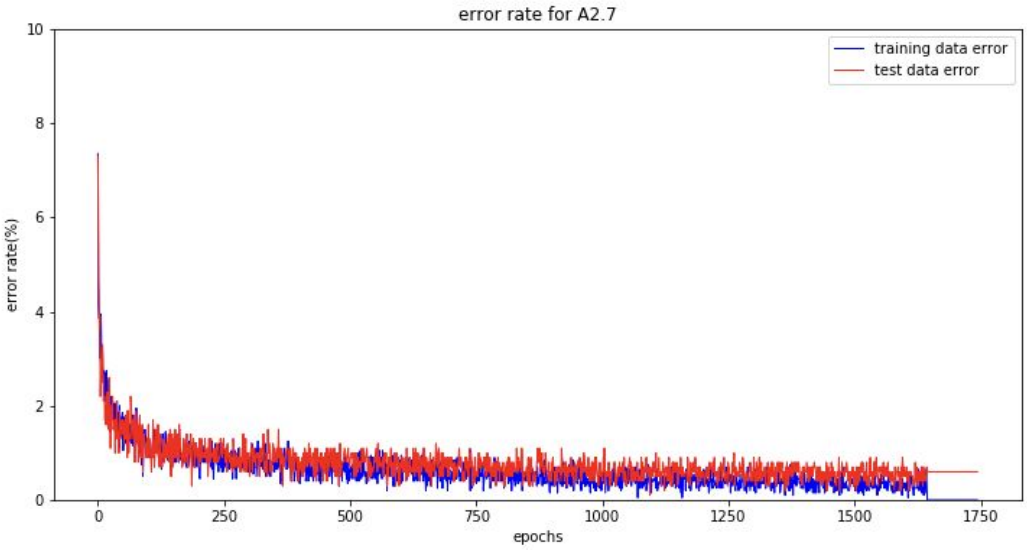


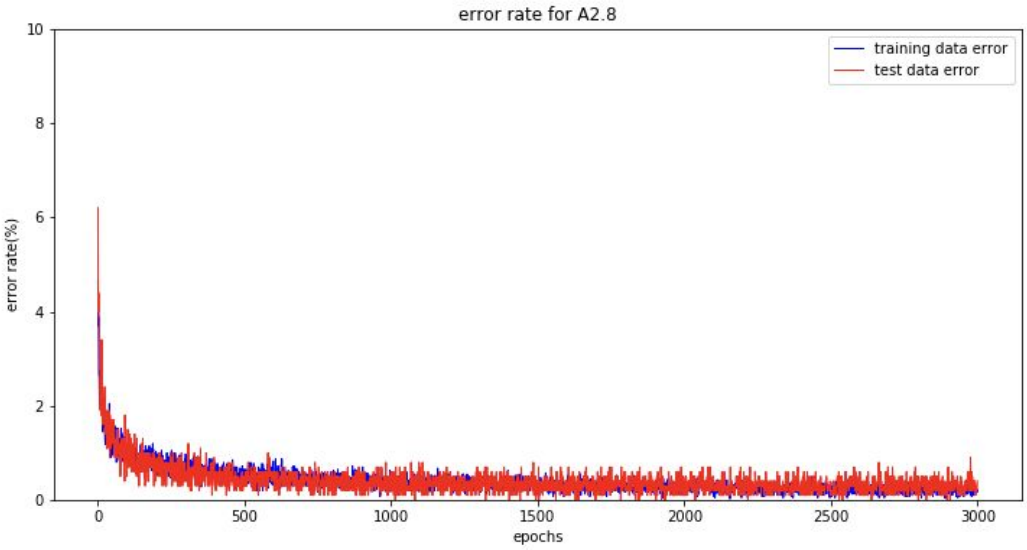


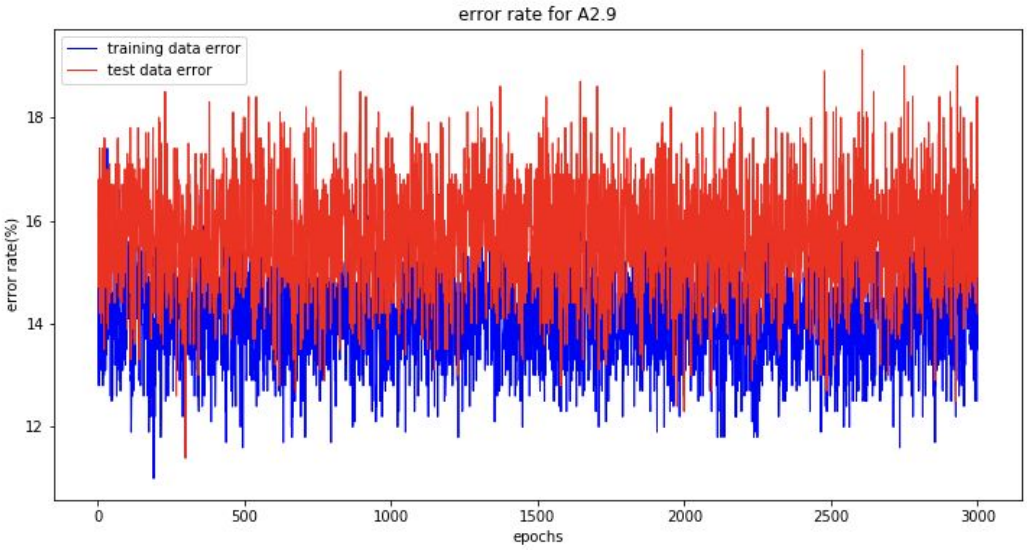

error rate for A2.4
training data error
test data error
error rate(%)
epochs
0
2
4
6
8
10
0
500
1000
1500
2000
2500
3000


error rate for A2.5
training data error
test data error
error rate(%)
epochs
0
2
4
6
8
10
0
500
1000
1500
2000
2500
3000


error rate for A2.6
training data error
test data error
error rate(%)
epochs
0
2
4
6
8
10
0
200
400
600
800
1000


error rate for A2.7
training data error
test data error
error rate(%)
epochs
0
2
4
6
8
10
0
250
500
750
1000
1250
1500
1750





2.2

Whether the dataset is linearly separable will decided by the last error rate, if error rate is 0, then the training data is separable and there exist a hyperplane that can divide the examples with different label. From the plots we can see that dataset 2,3,6,7 are separable.

2.3

Whether the data is overfit will decided by the performance of training data error and testing data error, if the training data error is going down and testing data error is going up means the overfitting problem occur. In all the plot, I didn't see this happened.

2.4

The final weight will be absolute scaling and those weights has the values that is smaller than 0.01 will be consider as noise. The reason why I don't use normalization is that after normalize

the weight, there will definitely have a 0 term. And what we want to do is to calculate all the term in the weight and figure out which ones are relatively too close to 0 that can be ignore. The normalization method will incorrectly consider certain term as noise in this way, and absolute scaling method will work better in here. I've run all dataset for several times, and took those at least showed twice as a noise feature.

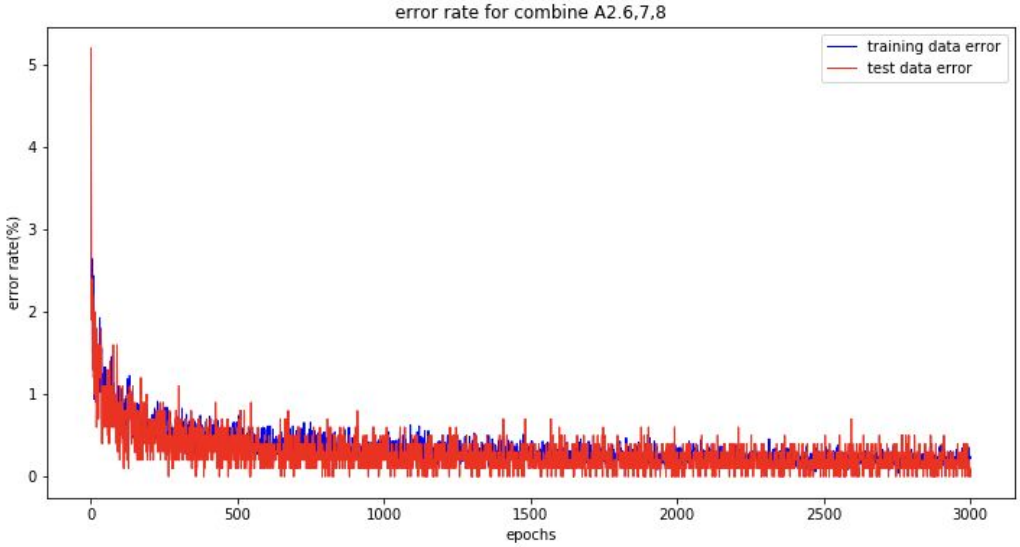
2.5

20% of the training data will be carve out as a developed data, the remaining data will be the training data for the perceptron. When training the perceptron, in each epoch, we will get a error rate of develop data and train data. The epochs will be tuned based on the error rate of develop dataset -- we will report these error rate every 100 epochs for 1000 epochs, if the training data gone through all the epoch and can't get an error rate of 0 for develop data, we will report the epoch with minimum develop data error rate. Or, if the develop dataset get an error rate of 0, we will report that epoch that got this result.

dataset	2	3	4	5
2	Yes. Margin is 0.005.	No	9th, and 17th feature should be noise.	with 217 epoch, we get error rate of develop data/ train data/ test data:0/ 0.6/ 1.5
3	Yes. Margin is 0.002	No	6th, and 19th feature should be noise.	with 199 epoch, we get error rate of develop data/ train data/ test data:0/ 1.3/ 2.6
4	No	No	2rd feature should be the noise.	with 200 epoch, we get. error rate of develop data/ train data/ test data:2.0 / 2.0 / 3.5
5	No	No	2rd, 5th, 7th feature should be the noise.	with 500 epoch, we get error rate of develop data/ train data/ test data:2.0 / 3.375 / 6.9
6	Yes. Margin is 0.0003	No	10th, 12th, 16th, 18th, 19th feature should be noise	with 485 epoch, we get error rate of develop data/ train data/ test data:0.0 / 0.75 / 1.09
7	Yes. Margin is 0.0049	No	10th, 12th, 16th, 18th, 19th feature should be noise	with 438 epoch, we get error rate of develop data/ train data/ test data:0/ 0.44/ 0.5

8	No	No	10th, 12th, 16th, 18th, 19th feature should be noise	with 398 epoch, we get error rate of develop data/ train data/ test data:0.0 / 0.65 / 0.4
9	No	No	1st, 11th, 13th, 17th feature should be noise	with 600 epoch, we get error rate of develop data/ train data/ test data:13.0 / 13.3 / 13.5

2.surprise.
When compare these three dataset's perceptron result, we can see that their noise features have some repeated term -- the 10th ,12th, 18th features. My idea is constructing a new train dataset that is combine all the data in data 6, 7, 8, then use the data and remove the noise feature for each example, and use the new dataset to train a new perceptron.



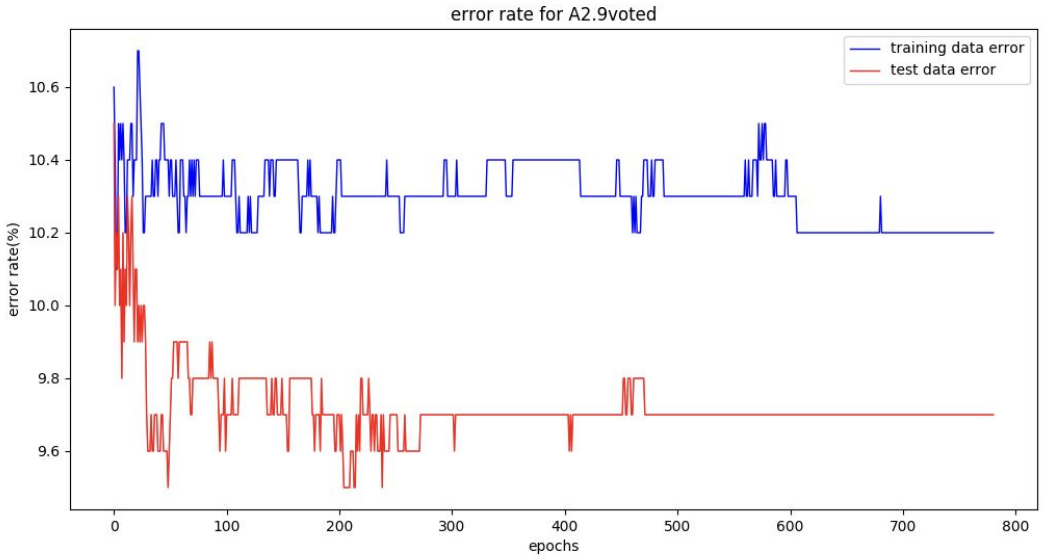
The error rate improve and reached 0% for the test data when using the new data to train the perceptron at the epoch of 3000, where when using the dataset 6, 7, 8 got the error rate 0.5%, 0.6%, and 0.4%. The answer for the question 2-5:

2	3	4	5
Yes. Margin is 0.59	No	14th feature should be noise	with 3000 epochs, we get error rate of develop data/ train data/ test data:0.2/ 0.15/ 0

3.

In this part I will focus on dataset 9, because the error rate of the data set is the highest and it can't be stable after all the epochs.

I used averaged perceptron for the dataset, averaged performance introduce a cache weight and cache bias, that is, if the weight survive longer, it will be more important for the final result. In the former question, I set the maximum epochs as 3000, and when the error rate of the train data becoming 0 the training will stop -- since there is no more error to drive the update. This time, I will introduce another hyperparameter to tune the epoch. Consider the error rate of all the epoch as a list, I want to stop training when the last n of them are the same, the n is my new parameter, I call it same error rate number. The parameter can prevent the training process wasting time on those training data that is hard to linear separate. In my experiment, the former dataset that can't be linearly divided by the original perceptron will result with error rate flipping around a certain range, in averaged perceptron, this situation is happening, too -- the error rate will stick at a certain error rate for a very long time and even if it change after, the performance won't improve a lot. When set the parameter as 100, the performance showed as followed.



As we can see, there are two main improvement. First, the learning epochs has been shorten. With the origin perceptron algorithm, it takes thousands of epochs and still can't be stabilized, now it only take less than 800 epochs to get a final result. Second, the error rate has been reduced to 10%, which is almost the best performance for the original algorithm.

4.

With $K = 19$, PCA lower dimension data with perceptron performance showed as follow. As we can see, there is no much improvement. The PCA is lower the dimension by decomposing the original data to the eigenvectors and eigenvalues, and remove the trivial terms, lower the dimension of the data. With this process, some of the noise influence has been reduced. However, perceptron can lower the noise influence itself by lower the weight of that feature, the noise feature will finally get a pretty small weight at the end of training. This may explain why there is little progress when using PCA for the data. Also, as we can see in the third plot, the values of the eigenvalues divided by sum of all eigenvalues is not varying a lot, this may meaning that there is no noise features, this can also be a reason that lowering the dimension can't get better performance. In terms of K, I chose to use 95% of the top eigenvalue, and it gave me the dimension as 19.

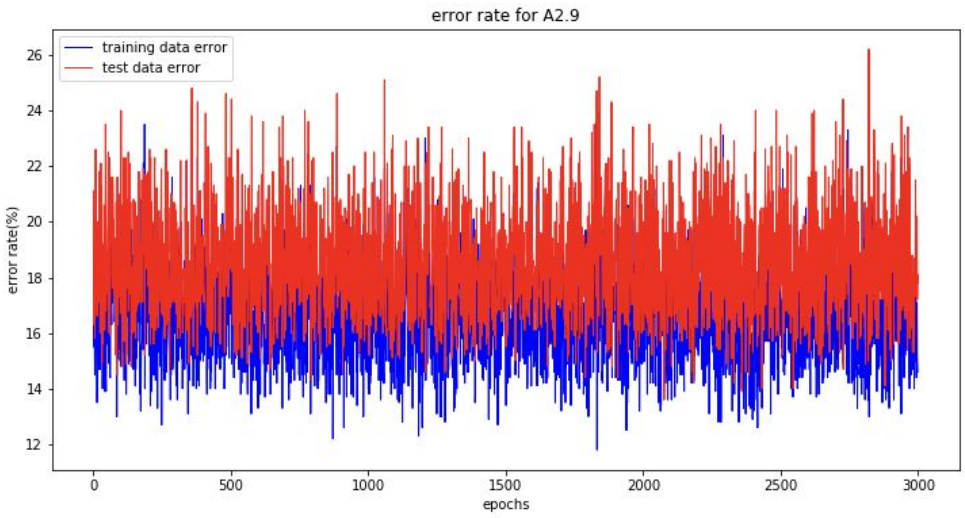


fig1. pca lower dimension data training result with perceptron

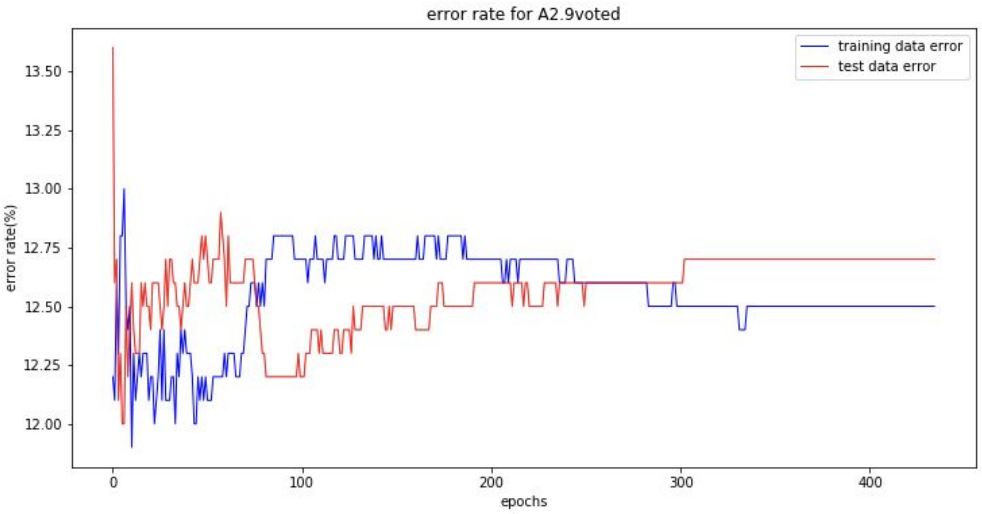


fig2. pca lower dimension data training result with averaged perceptron

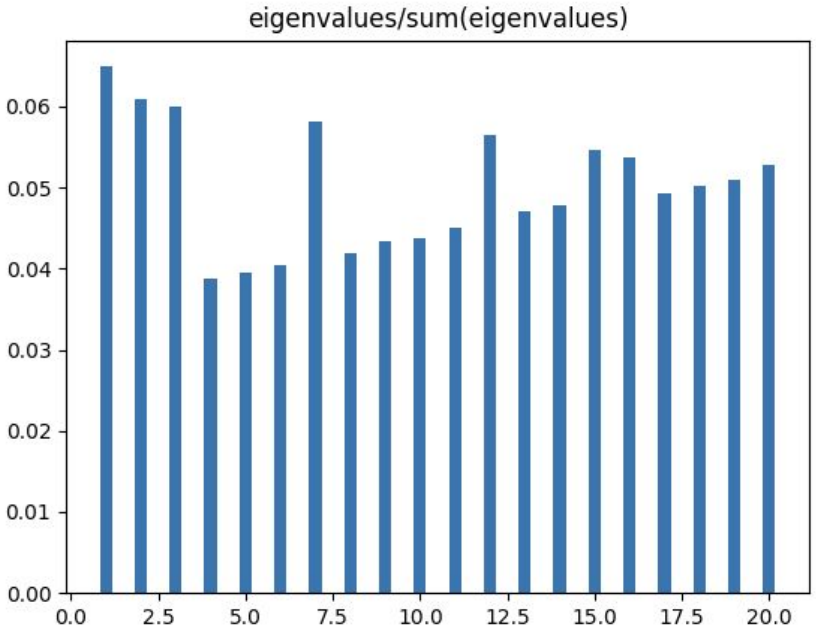


fig3. eigenvalues/sum(engenvalues)