

## Introduction

In this project, we developed variations of the perceptron algorithms. We aim to classify handwritten digits of numbers 3 or 5 using perceptron algorithms and compare the performance of the variations of it. We implemented three perceptron algorithms; Online Perceptron, Average Perceptron, Polynomial Kernel Perceptron.

In Part 1, we discuss Online Perceptron algorithm. We trained a linear classifier as follow;

$$\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x}) \quad , \text{ which gives output of -1 or 1}$$

Then, we record accuracy of prediction on the given validation samples for each iteration to see the effect of the number of iterations to accuracy. In conclusion, the train accuracy does not reach to 100%, and we discuss the reason in Part 1. In addition, we apply the trained linear classifier to test data sets to predict the target value for given data sets. In Part 2, we provide the discussion of Average Perceptron. In this part, we want to deal with the issue we discovered in Part 1 by utilizing Average Perceptron by keeping running average weight. We compute our output as follow;

$$\hat{y} = \text{sign}(\hat{\mathbf{w}}^T \mathbf{x}) \quad , \text{ where } \hat{\mathbf{w}} \text{ is a running average}$$

We present the changes of the train and validation accuracies with the increase of the number of the iterations. Additionally, we compare the validation accuracy of Average Perceptron to the Online Perceptron. In Part 3, we introduce polynomial Kernel Perceptron which is appropriate for complex model. In order to experiment the algorithm, we provide the feature vectors in a higher dimensions using Kernel function as follow;

$$k_p(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^T \mathbf{y})^p$$

We define a Gram matrix  $\mathbf{K}$  with size  $N \times N$  where  $N$  is the number of training samples. We compute  $\alpha$  to predict validation sets. Then, we compare the accuracies of the algorithm with different  $p$  value. We discuss how  $p$  value affects to the train and validation performance. Finally, we use our best  $\alpha$  to predict the test data.

## Part 1. Online Perceptron

a.

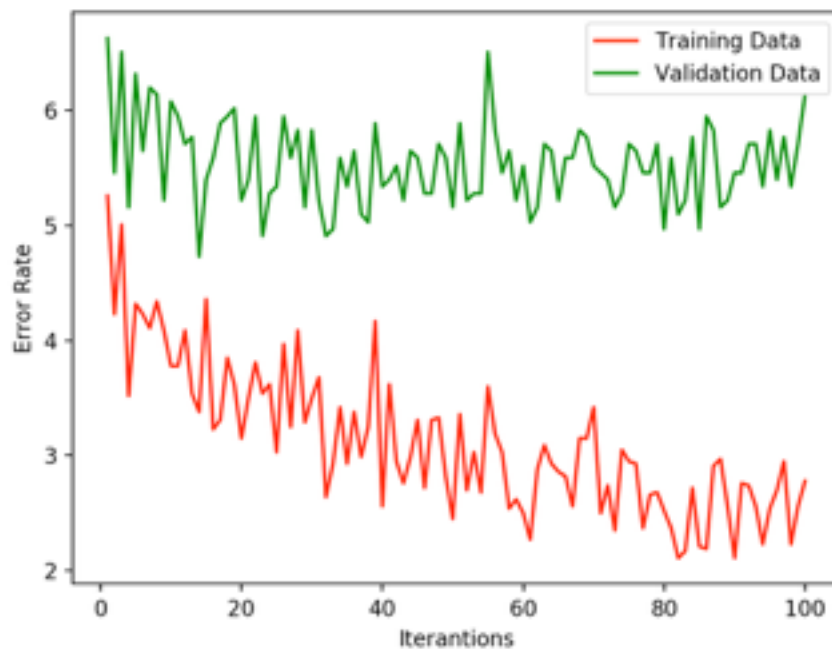


Figure 1.

epoch: 1	Training error rate: 5.26 %	Validation error rates: 6.63 %
epoch: 2	Training error rate: 4.23 %	Validation error rates: 5.46 %
epoch: 3	Training error rate: 5.81 %	Validation error rates: 6.51 %
epoch: 4	Training error rate: 3.52 %	Validation error rates: 5.16 %
epoch: 5	Training error rate: 4.32 %	Validation error rates: 6.32 %
epoch: 6	Training error rate: 4.23 %	Validation error rates: 5.65 %
epoch: 7	Training error rate: 4.11 %	Validation error rates: 6.2 %
epoch: 8	Training error rate: 4.34 %	Validation error rates: 6.14 %
epoch: 9	Training error rate: 4.89 %	Validation error rates: 5.22 %
epoch: 10	Training error rate: 3.78 %	Validation error rates: 6.08 %
epoch: 11	Training error rate: 3.78 %	Validation error rates: 5.95 %
epoch: 12	Training error rate: 4.89 %	Validation error rates: 5.71 %
epoch: 13	Training error rate: 3.54 %	Validation error rates: 5.77 %
epoch: 14	Training error rate: 3.38 %	Validation error rates: 4.73 %
epoch: 15	Training error rate: 4.36 %	Validation error rates: 5.4 %

Figure 2.

Figure 1. shows the error rate versus iteration number of the train and validation. Moreover, we present a part of the exact error rate in Figure 2.

**b. Does the train accuracy reach to 100%?**

No.

Why? Most data are not linear separable. We have only learned a linear function in training. It is very difficult for a linear model to linear separate such non-linear separable data. Furthermore, it becomes a XOR problem which means that the learning process cannot stop without a limited iterations number due to the inseparability. (since it does not converge ??)

**c. Use the validation accuracy to decide the test number for iteration.**

The best result of validation data set in error rate appears in the 14th iterations at 4.73% error rate in validation data set. Figure 3. gives closer look to it, and Figure 4. shows a part of results when applying the model we trained to the test set.

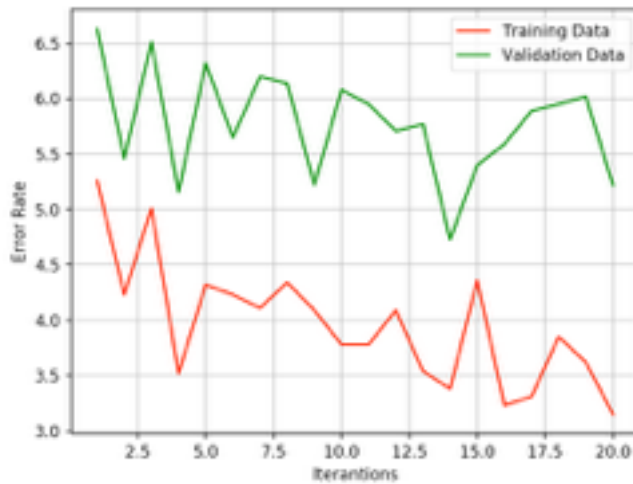


Figure 3.

oplabel.csv

	A	B	C	D
1	-1			
2	-1			
3	-1			
4	1			
5	-1			
6	-1			
7	1			
8	-1			
9	1			
10	-1			
11	-1			
12	1			
13	-1			
14	-1			
15	1			
16	1			
17	-1			
18	-1			
19	1			
20	1			
21	1			
22	1			
23	1			
24	1			
25	-1			

Figure 4.

Moreover, we discovered that there is a overfitting issue in this perceptron algorithm. In Figure 1., it is very clearly shown.

## Part 2. Average Perceptron

### a. Implementation

[avg\\_perceptron.py](#) file attached

### b. Plot the train and validation accuracies versus the iteration numbers where items = 1, ... , 15

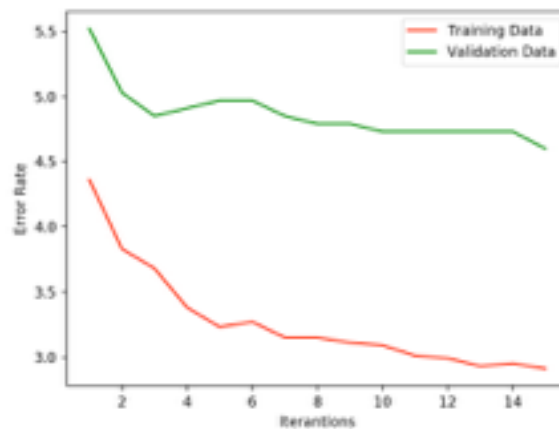


Figure 5.

c.

Figure 6. shows the first 100 iterations error rate of both perceptron and average perceptron on both training set and validation data set.

In Figure 6., yellow line is showing the result of online perceptron on validation, while black line is showing the result of online perceptron on training data. Green line and the red line represents the error rate of average perceptron on validation data and training data respectively.

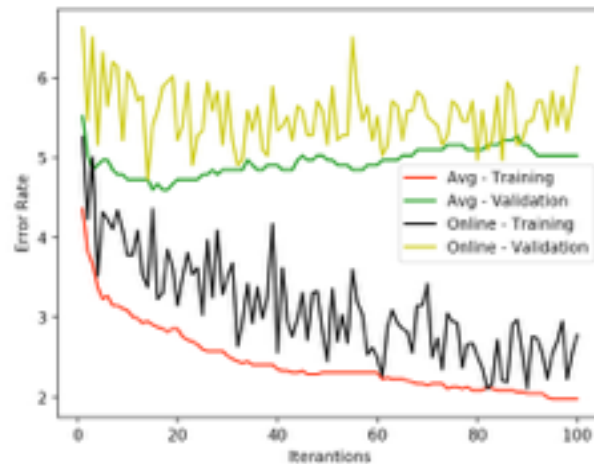


Figure 6.

From Figure 6., it is hard to see that the result of average perceptron is much more smooth than online perceptron. The naive online perceptron has a really huge fluctuation of the error rate in validation accuracy, while the fluctuation of naive perceptron is very rough. Beside, just like online perceptron, there is also a slight tendency of overfitting problem. When the training error rate is going down, the error rate of validation data set is going up slightly.

Thus, we believe that the average perceptron could improve the performance of perceptron algorithm on validation data set for trying to maximize its margin. The idea of online perceptron is updating when meeting a mistake, while the idea of average perceptron is trying to decrease the number of errors from remembering the more-repeated-mistakes data (There should be several rows of data, the perceptron might make more # of mistakes on some particular mistakes. ) Essentially, it is reducing the effect of ordering to the online perceptron. Because average perceptron gave more weight on mistakes data, thus it can reduce the ordering influence to the result. While online perceptron has to stop itself depends on the ordering of the data in a separable data. After decreasing the influence of ordering, the results of average perceptron cannot only be more smooth than online perceptron but also has a lower error rate due to the personality of ordering relative independence which could also help to maximize its margin. In

addition, Figure 6. clearly shows that Online perceptron is oscillating compared to Average Perceptron.

### Part 3. Average Perceptron

#### a. Polynomial Kernel function Kp

Pseudo Code

```
kernel(x1,x2,p){
    s = 1 + dot(x1,x2)
    return pow(s,p)
}
```

#### b. Define a Gram matrix K

Pseudo Code

```
K[][] = N by N integer array
for i = 0 to N
    for j = 0 to N :
        K[i][j] = kernel(data[i],data[j],p)
```

c.

3) Record the train and validation accuracy for each iteration

Figure 7. shows the error rate of training data set with different polynomial degree p with increase of iterations

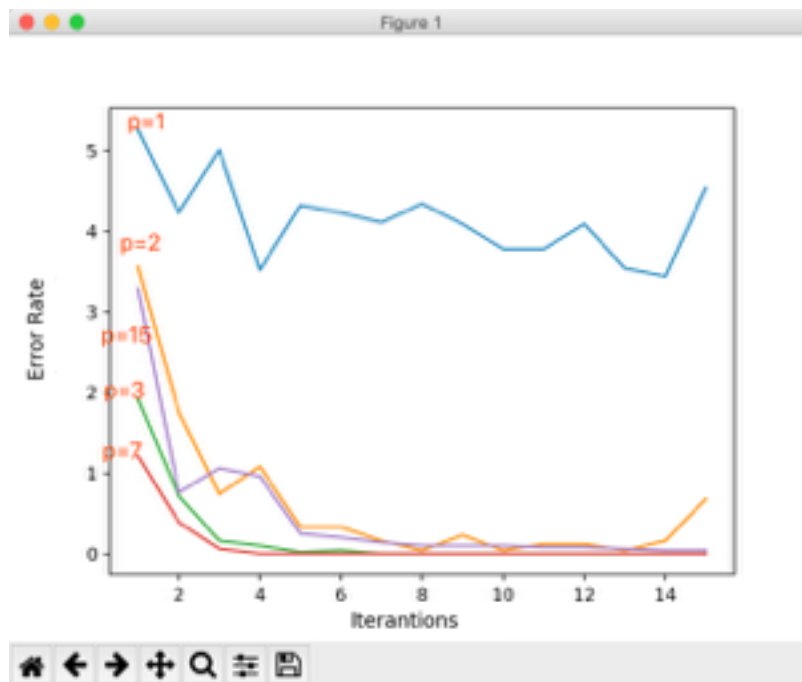


Figure 7. Kernel Perceptron on Training Data

Figure 8. shows that the result of kernel function running on validation data set.

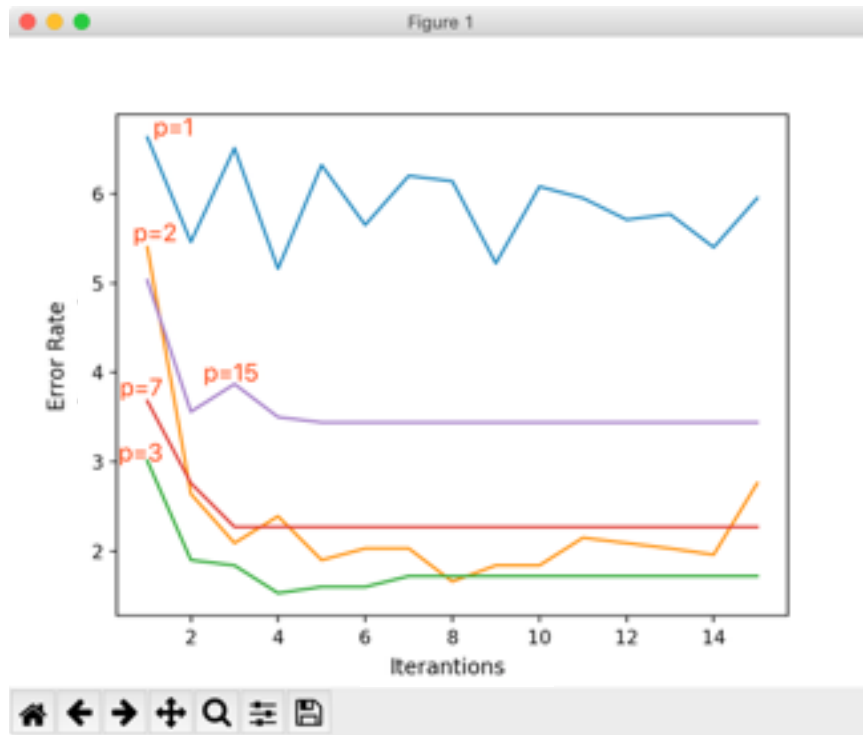


Figure 8. Kernel Perceptron on Validation Data

4) Record the best validation accuracy achieved for each  $p$  over all iterations  
The best validation accuracy we have is 1.53% error rate at the 4th epoch with  $p=3$ .

d. Plot the recorded best validation accuracies versus degree

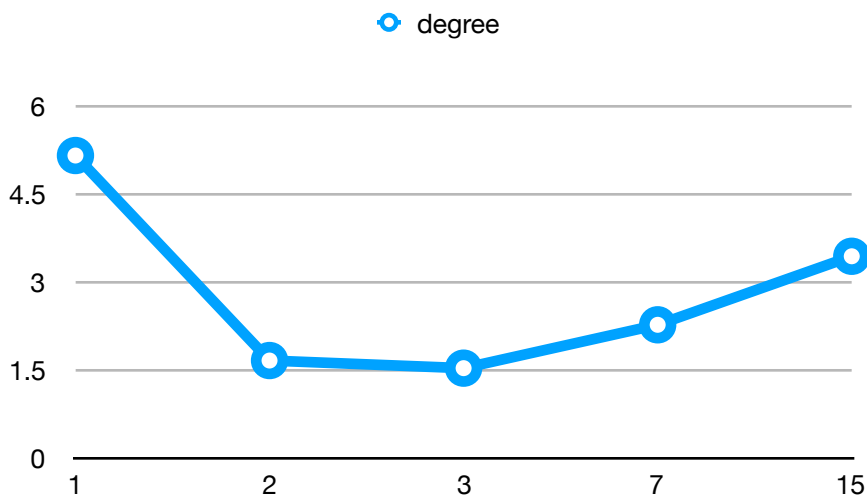


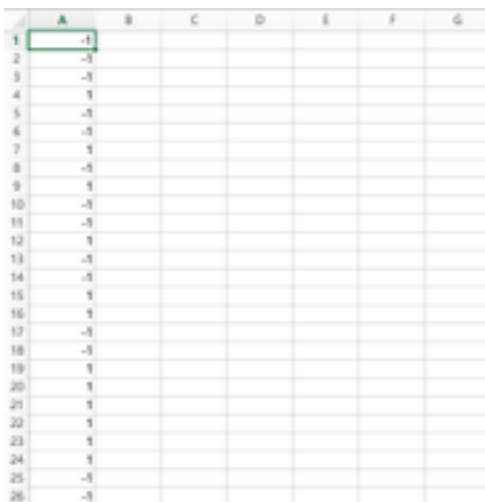
Figure 9.

Generally, when  $p$  is bigger, it is easier to get an overfitting problem. However, when  $p$  is not big enough, the linear model ( $p=1$ ) cannot get a good result. Therefore, in Figure 9., we can find out that we can get minimized error rate when  $p=3$ , which means neither a small  $p$  nor a very huge  $p$  cannot get the best answer.

This result is fitting on our intuition. In a linear model with a small  $p$ , the data set is relative linear non-separable. Thus, the error rate result of this model also shows a rough fluctuation. While, with a very large  $p$ , the training set can always get a really good result (when  $p \geq 3$ , error rate reach to 0%). However, when the error rate in training set is 0%, the update of model will stop, then the error rate in validation data set will keep a relative higher number. Beside, from our experiment, with a higher  $p$ , the model is easier and sooner to get an overfitting problem and error rate easier reaches to 0. In the meanwhile, the update would stop and the best error rate will keep in a higher position. Thus, the shape of figure of best error rate versus degree number is more like a bowl shape.

e.

kplabel.csv



	A	B	C	D	E	F	G
1	-1						
2	-1						
3	-1						
4	1						
5	-1						
6	-1						
7	1						
8	-1						
9	1						
10	-1						
11	-1						
12	1						
13	-1						
14	-1						
15	1						
16	1						
17	-1						
18	-1						
19	1						
20	1						
21	1						
22	1						
23	1						
24	1						
25	-1						
26	-1						

Figure 10. shows a part of results when applying the model we trained to the test set.

Figure 10.

## Discussion

1. Most of results follow our expectation. The main problem of learning algorithm is either the dimension is not enough for linear separable or the dimension is too high and cause an overfitting problem. Thus, the feature engineering work is important.
2. In practice, we think that we should better to use average perceptron first for a relative better result ,compared to online perceptron, and a relative faster algorithm ,compared to kernel. Then

we could use kernel to approximate what the dimensionality the model should be. Last, instead of using kernel, we may add feature combination manually and apply average perceptron again with the experience from the kernel.

3. A surprise to us is the error rate result of this experiment is pretty low. Thus, we checked the data and where the data is from. Then we found out that all of data has been preprocessed compare to the data on Kaggle.com. We think that that is why this our result in this assignment is pretty good. (In fact, in CS519, we also implemented perceptron. However, the error rate for that is around 18% for online perceptron and 14% for average perceptron. )