Implementation assignment 2:

Behnam Saeedi, Kamilla Aslami, Mostafa Estaji
CS534: Machine Learning
Due Oct 27th 11:59pm
Fall 2018

CONTENTS

1 INTRODUCTION

This assignment we implemented the perceptron learning algorithm with training and validation. Furthermore, we will produce the kernel functions for the perceptron and produce the necessary graphs for the write up portion.

1.1 How to Run

In order to run the code please refer to the file ”Readme.md”. For your convenience the instructions are also included here.

- Use command ”make” or ”python ia_2.py” in order to run the code
- Use ”make plot” or ”python ia_2.py -p” to store the plots into file
- Use ”make save” or ”python ia_2.py -s” to save the kernel tables for faster future runs

2 PART 1

In this section we followed the pseudo code and the instructions listed in the documentation. We recorded our results based on the specification and produced the following graphs and tables.

2.1 A

The accuracy computations for both training and validation sets are shown in the following figure (Figure 1).

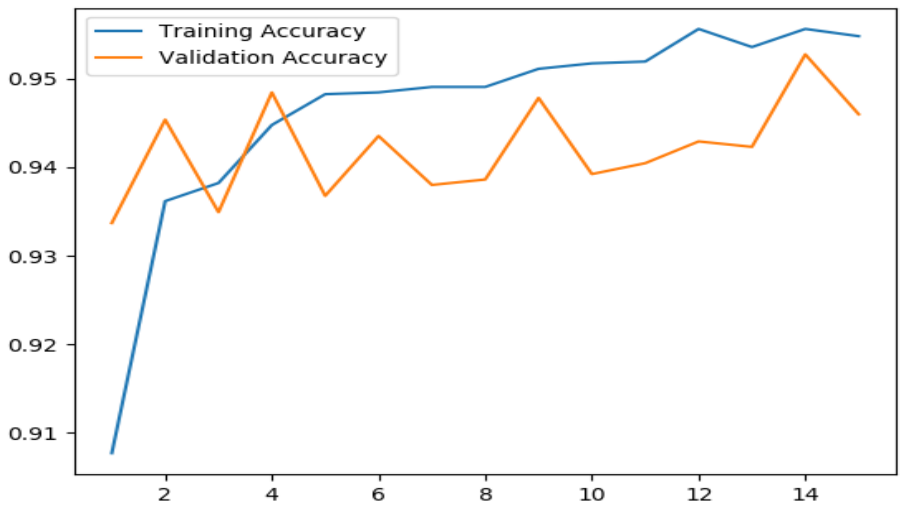

Figure 1. Training and Validation comparison for Part 1

The following tables represents the accuracy behaviour of the training set and validation set, respectively.

Iteration	Accuracy
1	0.907733224223
2	0.936170212766
3	0.93821603928
4	0.944762684124
5	0.948240589198
6	0.948445171849
7	0.949058919804
8	0.949058919804
9	0.951104746318
10	0.951718494272
11	0.951923076923
12	0.955605564648
13	0.953559738134
14	0.955605564648
15	0.954787234043

Table 1
Training set for data set Part 1

Iteration	Accuracy
1	0.933701657459
2	0.945365254758
3	0.934929404543
4	0.948434622468
5	0.936771025169
6	0.943523634131
7	0.937998772253
8	0.938612645795
9	0.947820748926
10	0.939226519337
11	0.940454266421
12	0.942909760589
13	0.942295887047
14	0.952731737262
15	0.9459791283

Table 2
Validation set for data set Part 1

2.2 B

In this example, it did not reach 100%. Usually, this means that data is not linearly separable, but in our case, it happened because we limited the number of iterations to 15. It can also mean that the input classes are difficult to separate (are close to the decision boundary), and it will take many iterations for the algorithm to converge. We can potentially increase the number of iterations to achieve zero error, but it is very likely to cause

overfitting, and therefore decrease accuracy on the validation set. To check convergence (100% accuracy on the training set) we decided to keep iterating till such a case is met. We noticed than in epoch=1600 the training set's accuracy reaches 100%. This is shown in Figure 2.

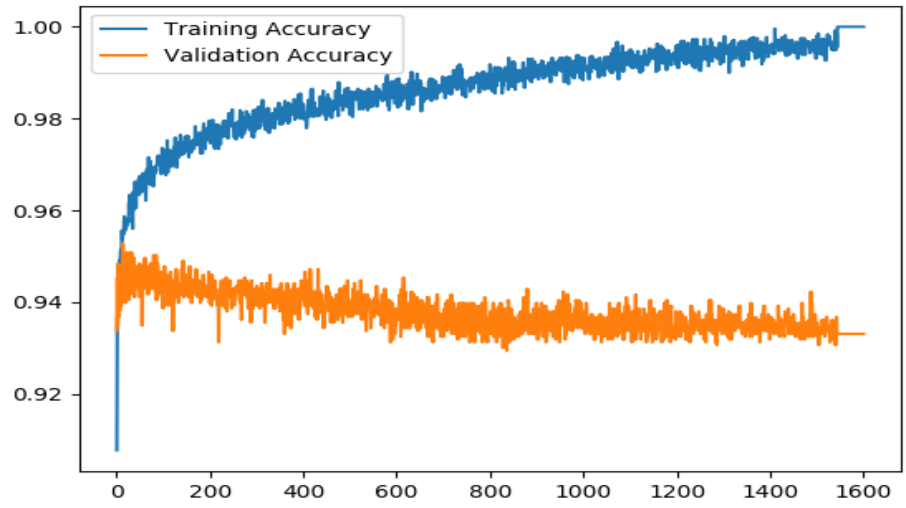

Figure 2. Training and Validation comparison for 1600 iterations

2.3 C

According to Table 2, the best validation accuracy was achieved on 14-th iteration, so we used this *iters* value to make predictions for the samples in the test set. The predictions were computed and stored in the designated file "olabel.csv". A quick bash search shows that all data stored in this file are either 1 or -1.

3 PART 2

For this section we followed the instructions for part two and produced the following results.

3.1 A

For this section the algorithm used was the average perceptron algorithm. The results were stored in file "alable.csv" and the following tables were produced:

These tables showcase the accuracy as a function of iteration. The table 3 is the training data and the table 4 is validation data.

Iteration	Accuracy
1	0.907733224223
2	0.936170212766
3	0.93821603928
4	0.944762684124
5	0.948240589198
6	0.948445171849
7	0.949058919804
8	0.949058919804
9	0.951104746318
10	0.951718494272
11	0.951923076923
12	0.955605564648
13	0.953559738134
14	0.955605564648
15	0.954787234043

Table 3
Training set for data set Part 2

Iteration	Accuracy
1	0.944751381215
2	0.949662369552
3	0.951503990178
4	0.950890116636
5	0.950276243094
6	0.950276243094
7	0.951503990178
8	0.95211786372
9	0.95211786372
10	0.952731737262
11	0.952731737262
12	0.952731737262
13	0.952731737262
14	0.952731737262
15	0.953959484346

Table 4
Validation set for data set Part 2

3.2 B

The following figure (Figure 3) is the plotted graph for the two data sets for Training (in blue) and Validation (in orange) through out the course of 15 iterations.

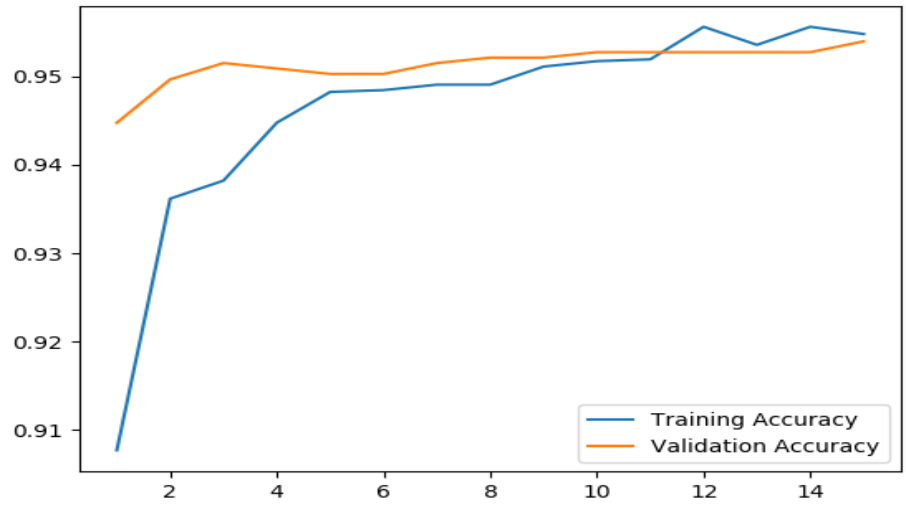

Figure 3. Training and Validation comparison for Part 2

3.3 C

The comparison of Figure 1 and Figure 3 reveals that accuracy curves are considerably smoother in average perceptron method. This can be explained by the nature of these two algorithms. The weights in online perceptron get updated after checking each training example. This introduces noise in the overall accuracy behavior since not all other training examples are updated. In other words, update of one example might have a negative impact on the updates of subsequent examples. However, in average perceptron the whole batch of training examples are updated at the end of each iteration. In other words, sum of all weight updated is recorded as the average updated weight at the end of each iteration. It goes without saying that this effect is more clearly seen on the validation set since the validation accuracy is independent of training or learning algorithm. The above observation can again be seen in the comparison of the two plots. Therefore, as expected and observed, the accuracy curve is much smoother. Also, it appears that the validation accuracy for the average version is higher than online perceptron (about 1%).

4 PART 3

In this section we attempt to create the kernel function for the perceptron. This section posed many challenges which we will discuss in the following subsections.

4.1 A, B and C

For this section we computed the Kernel function. However, this posed several challenges. For example, the Gram matrix generation and computation cost in each iteration is so large that it would have been very tedious

for us to recompute this in each debug steps. So instead, we stored these Gram matrices for each p (order of polynomial function), the first time a computation was to happen and afterwards, we reused the kernel matrices which were produced. In our program, we have a flag ”-s” or ”–save” that has the task of storing the kernel file. This was specially difficult because the size of the kernel matrices was larger than team member’s Z drive limitation. Finally after producing all the kernels for values $[1, 2, 3, 7, 15]$, we produced the following figures (figures 4,5,6,7 and 8):

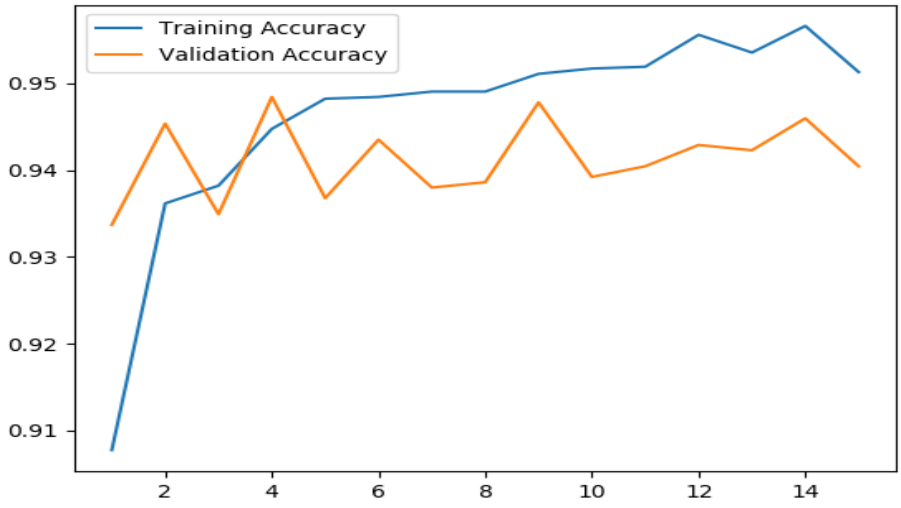

Figure 4. Training and Validation comparison for Part 3: p = 1

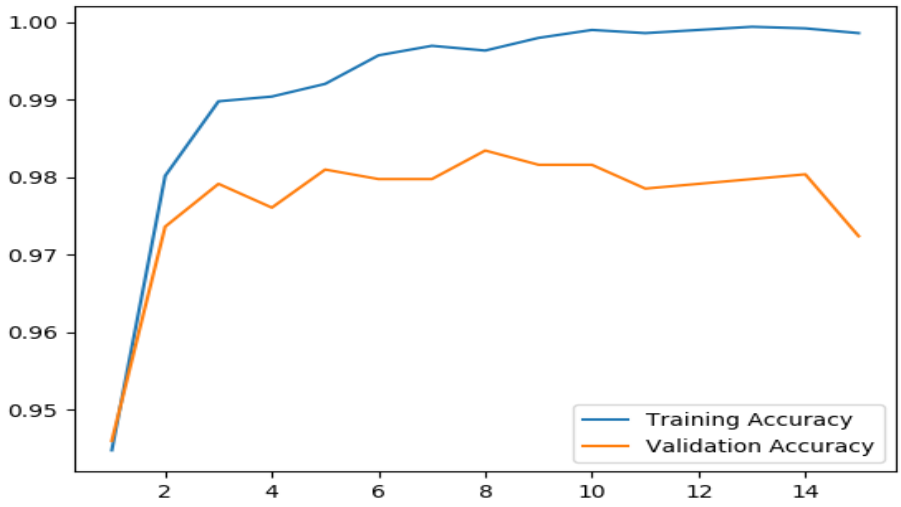

Figure 5. Training and Validation comparison for Part 3: p = 2

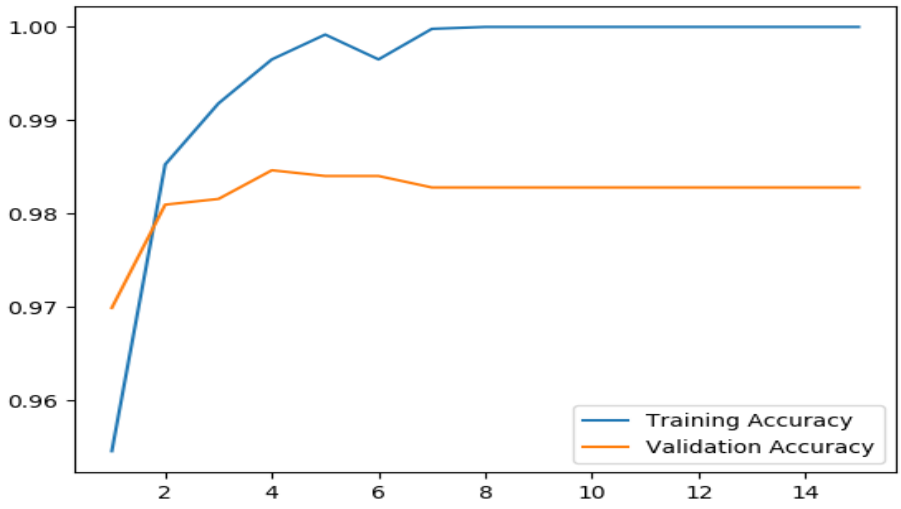

Figure 6. Training and Validation comparison for Part 3: p = 3

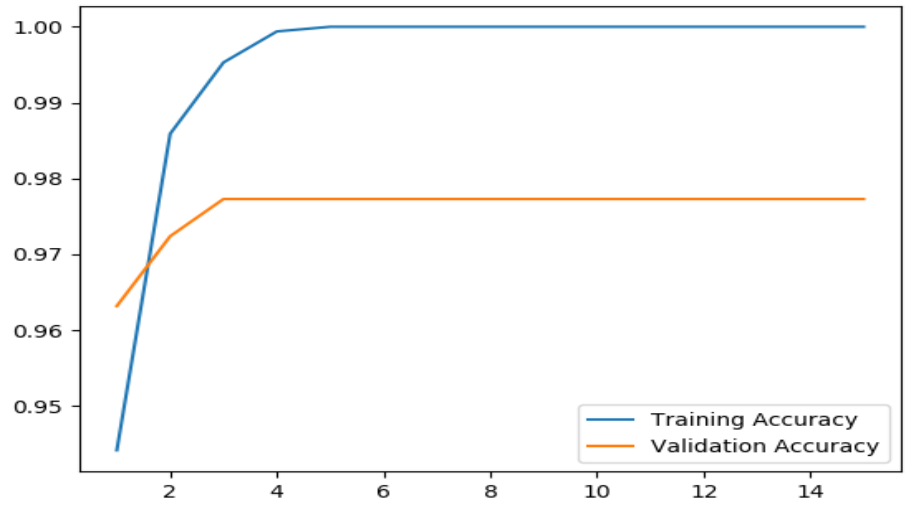

Figure 7. Training and Validation comparison for Part 3: p = 7

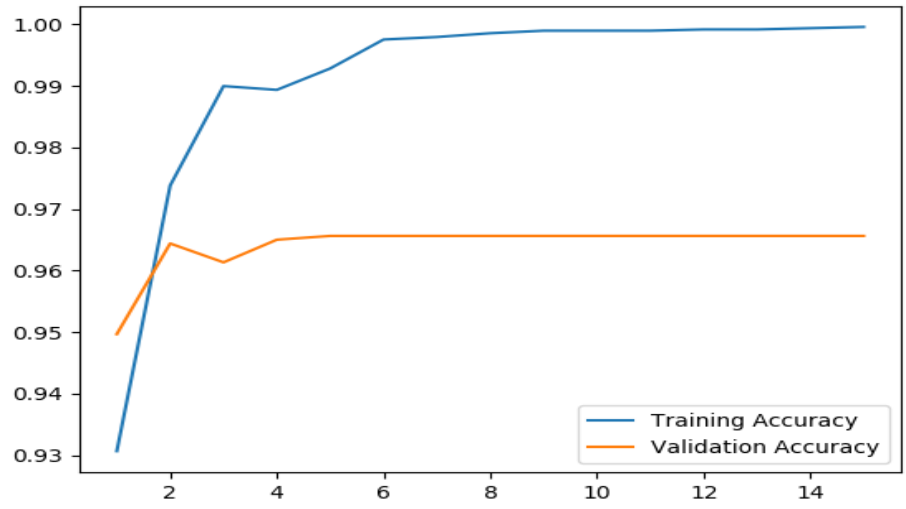

Figure 8. Training and Validation comparison for Part 3: p = 15

4.2 D

Figure 9 illustrates the recorded best validation accuracies versus degrees. The results are rather interesting. In polynomial kernel, the degree parameter controls the flexibility of the decision boundary: higher degree kernels yield a more flexible decision boundary. We can see this effect in our results - the larger p gets the higher accuracy for the training set. For $p = 7$ and $p = 15$ training accuracy even reaches 100%. However, the validation results shows that high training accuracy does not necessarily give us good predictions. For instance, $p = 3$ (Figure 6) shows better validation accuracy comparing to $p = 7$ (Figure 7) or $p = 15$ (Figure 8).

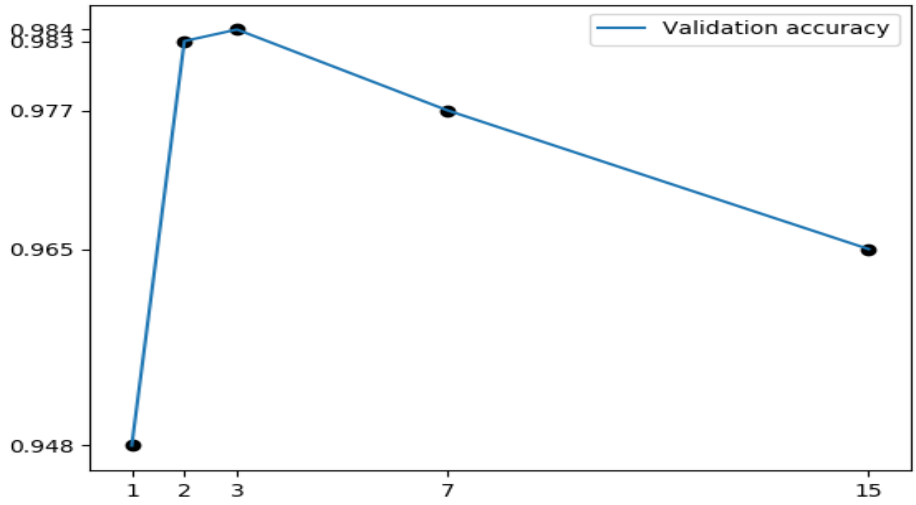

Figure 9. Best validation accuracies versus degrees

4.3 E

We achieved the best results with $p = 3$ and 6 iterations, so these values were used to compute α and predict the test data-set.