Submitted by: Oluwasegun Somefun, somefuno@oregonstate.edu

AI534 — IA3 Homework Report Due Nov 14th 11:59pm, 2021

1 Introduction

This report is on Perceptron and Kernels, with respect to the Implementation Assignment 3.

2 Part 1: Average Perceptron

a.) Comparing the training/validation accuracy curves of the average perceptron with those of the online perceptron, what do you observe? What are your explanation for the observation?

Generally, it can be observed that both accuracies are oscillatory as the iterations increase. For the averaged

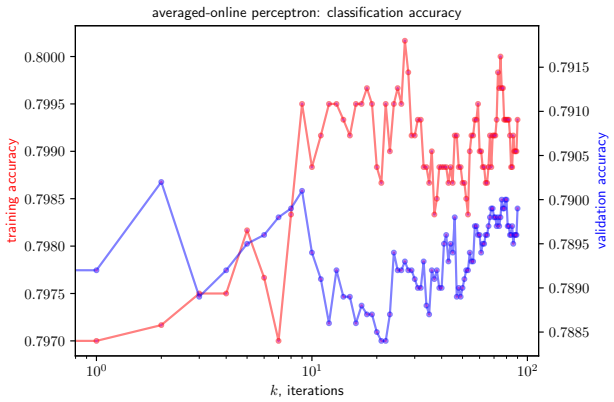

Figure 1: Averaged-Online Perceptron: Training/validation class accuracy

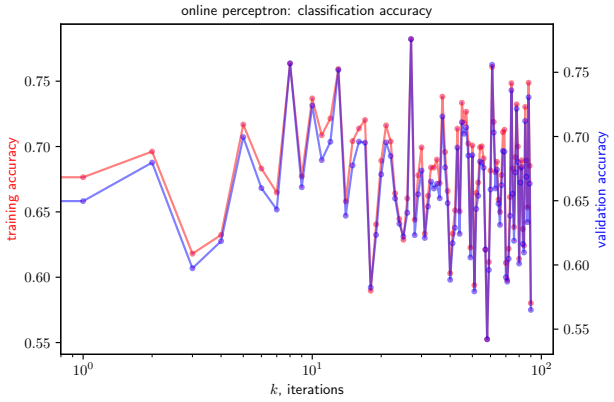

Figure 2: Online Perceptron: Training/validation class accuracy

online perceptron, although the training accuracy generally increases, the validation accuracy generally oscillates around the same point. On the other hand, for the vanilla online perceptron, both the training accuracy and the validation accuracy generally oscillates around the same point. Overall, the averaged-online perceptron gives the best taining/validation accuracy over all iterations.

This makes sense, since the averaged perceptron uses a running weighted average of the weights, which are kept for testing, hence the weights in this form are more stable for testing predictions in online learning.

b.) Which algorithm is more sensitive to the stopping point, (i.e., choosing different stopping point can lead to strong performance fluctuations) online or average perceptron? What are some practical implications of such sensitivity?

The iteration number determining the potential early stopping point k_e with respect to the class accuracies (training A_t and validation A_v) for both algorithms are tabulated in Table 1.

With respect to the iteration index k, it is clearly seen from Figures 1 and 2, that the vanilla online perceptron algorithm is more sensitive to the the chocie of a stopping point, since the accuracies are not stable, this can lead to undesired performance. One practical implication of this is that the averaged perceptron should be used instead of the vanilla form, since it generalizes better to the data. However, the performance is still oscillatory and so, we still need to do early stopping to prevent overtraining the perceptron. Otherwise, this would lead to overfitting, in the sense that the algorithm starts learning the noise in the data.

Table 1: Potential early-stopping point with respect to classification accuracy curves

Online Perceptron	A_t	A_v	k_e
Averaged	0.7993	**0.7899**	90
Vanilla	0.7008	**0.6855**	50

3 Part 2a: Online Perceptron with Polynomial Kernel

a.) As p changes, do you see different trends for how training and validation accuracies vary with the number of training iteration? Do you see the risk of overtraining (i.e., when training for too many iterations leads to overfitting) for some p value? Please explain why this happens or does not happen for different p values?

Yes. As illustrated in Figures 3 to 6, as polynomial order $p \in [1, 2, 3, 4, 5]$ changes, the training and validation accuracies display different trends as the number of training iterations increases. Particularly, the risk of overtraining almost linearly reduces as p increases. With respect to the given fixed number of max-iterations, for $p = 5$, the risk of overfitting is almost non-existing, as the class accuracies are celarly more stable and still increasing, compared to the other lower orders. The problem in using polynomial kernels for mapping non-linearly separable data to higher dimensions of linear separability is that selecting the polynomial order p becomes a model selection problem or an hyperparameter that must be tuned to ensure a balance. This could be attributed to mean that $p = 5$ gives the best bias-variance trade-off among the 5 values, and in this case it generalizes better to the validation data than the others.

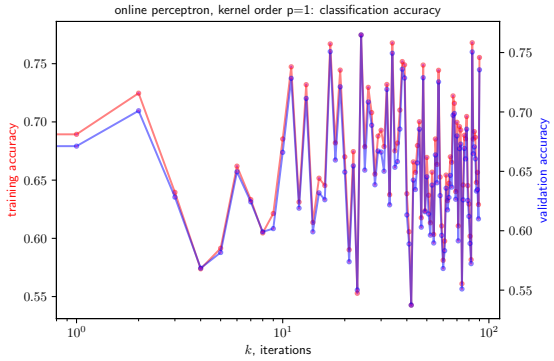

Figure 3: Kernel Perceptron $p = 1$: Training/validation class accuracy

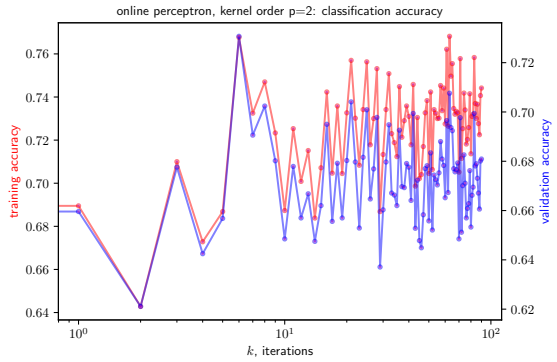

Figure 4: Kernel Perceptron $p = 2$: Training/validation class accuracy

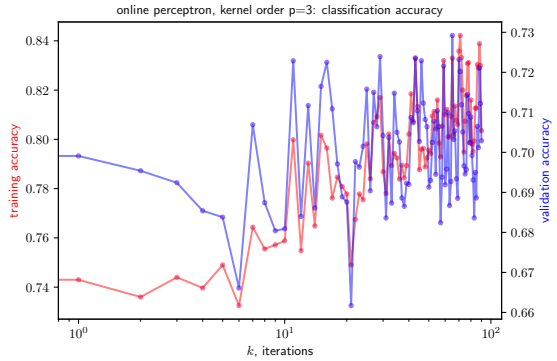

Figure 5: Kernel Perceptron $p = 3$: Training/validation class accuracy

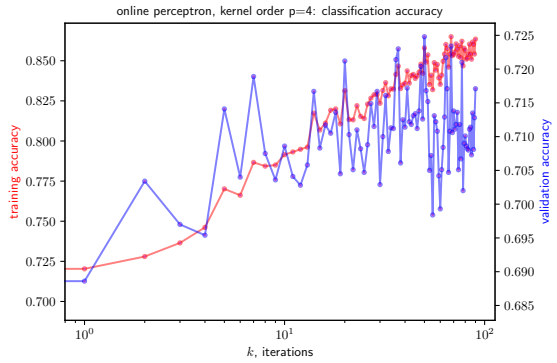

Figure 6: Kernel Perceptron $p = 4$: Training/validation class accuracy

b.) Which value of p produces the best validation accuracy? How do you think p is affecting the train and validation accuracy?

Reported in Table 2 are the best training and validation accuracy achieved for each p value (over all iterations). The best validation accuracy in terms of stability over all iterations is given by $p = 5$. The increase in p increases the capacity of the polynomial kernel for nonlinear spearability, hence an increase in p, should give increasingly better train and validation accuracy till some $p = p_e$ of diminishing returns, where the variance in the model increases (that is training accuracy increases, but validation accuracy does not)

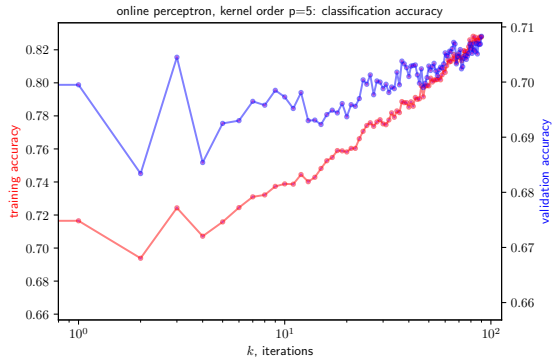

Figure 7: Kernel Perceptron $p = 5$: Training/validation class accuracy

Table 2: Polynomial Kernel: best classification accuracy values for each order p

p	A_t	A_v	k_e
1	0.7553	**0.7355**	90
2	0.7442	**0.6809**	90
3	0.8358	**0.7232**	70
4	0.8580	**0.7248**	50
5	0.8278	**0.7083**	90

c.) What is the asymptotic runtime of your algorithm (in big **O** notation) in terms of the number of training examples N? Does your empirical runtime match up with the asymptotic analysis?

The empirical run-time and asymptotic run-time of the implemented kernelized online perceptron algorithm is

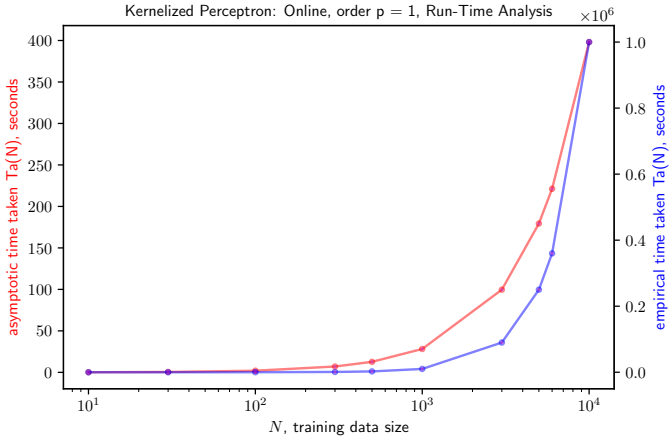

Figure 8: Algorithm Run-time: Kenelized Online Perceptron

shown in Figure 9. It is clearly seen that the asymptotic run-time (in red) agrees with the empirical run-time (in blue). The run-time data in Table 3, the regressive fit approximates to a polynomial model: $T_e(N) = 0.01\, N^{1.15}$ with a goodness of fit of 0.9744.

The asymptotic run-time (in the worst-case) of the Kernelized Online Perceptron algorithm is quadratic, that is: $T_a(N) = \mathbf{O}(N^2)$. This can be obtained by analysing the algorithm, which gives: $T_a = kdN^2 + kN$, where k is the fixed maximum iteration, d is the fixed number of input features dimension. Clearly, we observe that the computation of the kernel matrix which asymptotically is of quadratic polynomial order, contributes most to the

Table 3: Online Algorithm Run-Time Data for generating the empirical runtime model

N	$T_e(N)$ in seconds
10	352E-3
30	596E-3
50	1.65
100	1.67
300	4.21
500	12.5
1000	29.2
3000	91
5000	83
6000	99

algorithm's run-time in the worst-case.

4 Part 2b: Batch Perceptron with Polynomial Kernel

a.) Comparing the curves for batch kernel perceptron with the ones acquired with the same p value in part 2a, do you observe any differences? What are your explanations for them?

Perceptron uses a fixed learning rate of 1 for the subgradient descent. What would be the impact if we use a different learning rate?

Yes, there is a significant difference, it seems the online kenrlized algorithm is more stable and higher for $p = 5$ compared to its batch form which osciallates greatly. The reason for this is not clear, this may be due to the way the α in the kernelized batch form is being updated.

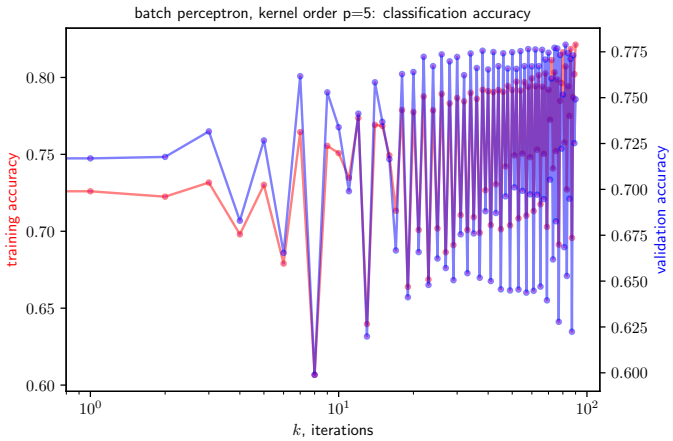

Figure 9: Batch Kernel Perceptron $p = 5$: Training/validation class accuracy

The Perceptron uses a fixed learning rate of 1 for the subgradient descent, which corresponds to a hinge-loss minimization with a margin-tradeoff penalty $C = 1$. This may indicate, in other words, that changing the learning-rate to another value, whether higher or smaller leads to a tradeoff between minimizing the maximum width of the linear decision boundary and minimizing the error.

b.) Provide the psudocode for your batch kernel perceptron algorithm. What is the asymptotic runtime of the batch kernel perceptron as a function of the number of training examples n? How does your empirical runtime match up with the asymptotic analysis? Do you observe a significant difference in runtime compared to the online algorithm? Provide an explanation for your observation.

The empirical run-time and asymptotic run-time of the implemented kernelized batch perceptron algorithm is

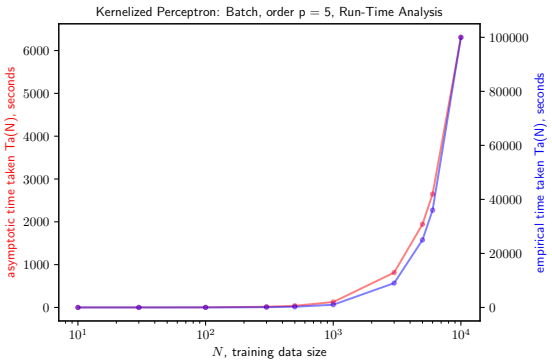

Figure 10: Algorithm Run-time: Kenelized Batch Perceptron

shown in Figure 10. It is clearly seen that the asymptotic run-time (in red) agrees with the empirical run-time (in blue). The run-time data in Table 4, the regressive fit approximates to a polynomial model: $T_e(N) = 0.001\, N^{1.7}$ with a goodness of fit of 0.9738.

Table 4: Batch Algorithm Run-Time Data for generating the empirical runtime model

N	$T_e(N)$ in seconds
10	415-3
30	537E-3
50	548E-3
100	1.2
300	2.58
500	3.93
1000	8.25
3000	33
5000	81
6000	77

The asymptotic run-time (in the worst-case) of the Kernelized Batch Perceptron algorithm is quadratic, that is: $T_a(N) = \mathbf{O}(N^2)$. This can be obtained by analysing the algorithm, which gives: $T_a = kdN^2 + k$, where k is the fixed maximum iteration, d is the fixed number of input features dimension. Clearly, we observe that the computation of the kernel matrix which asymptotically is of quadratic polynomial order, contributes most to the algorithm's run-time in the worst-case.

Gnerally, although the empirical runtime, shows that the batch algorithm is slightly faster as the training data-size increases, there was no significant visible difference in the run-time compared to the kernelized online algorithm. This is due to the kernalization matrice being computed and accessed in both algorithms accounting for quadratic polynomial time in the worst-case.

The pseudocode for the Batch Kernel Perceptron is outlined in Table 5

Table 5: Batch Kernel Perceptron Algorithm

Input	$\{(\mathbf{x}_i, y_i)\}_1^N$ (training data), k (maximum iteration), kernel function κ
Output	$\alpha_1, \ldots, \alpha_N$
1	Initialize $\{\alpha\}_i^N \leftarrow 0$ for $i = 1, ..., N$
2	for $i = 1, \ldots, N, j = 1, \ldots, N$ do
3	$K(i, j) = \kappa(\mathbf{x}_i, \mathbf{x}_j)$
4	end
5	for $t = 1$ to k do
6	for each training example $\mathbf{x}_i$ do
7	$u_i \leftarrow \sum_{j=1}^N \alpha_j y_j K(i, j)$
8	end
9	for each training example $\mathbf{x}_i$ do
10	if $u_i y_i \leq 0$ then
11	$\alpha_i \leftarrow \alpha_i + 1$
12	end
13	end
14	end