(CS534) Machine Learning

Implementation Assignment 3: Perceptron

A-young Kang (ID:933-610-350)

General Introduction

In this assignment, online perceptron and average perceptron are implemented for Part 1. The training and validation accuracies of those two perceptron algorithms are compared. For Part 2, kernel trick is applied to perceptron algorithm and the online and batch versions of the algorithm are implemented. Polynomial kernel with degree *p*, $\kappa(x_1, x_2) = \left(x_1^T x_2\right)^p$, is used and we observe how the *p* value affects the training and validation accuracy. The training and validation accuracies and the empirical runtime of the two algorithms are also compared. The upper limit on the number of training iterations is 100.

Part 1. Average Perceptron

(a) Apply your implemented algorithm to learn from the training data with maxiter = 100. Plot the train and validation accuracy of w (online perceptron) and $\overline{w}$ (average perceptron) at the end of each training iteration.

Figure 1 shows training and validation accuracies of online perceptron and those of average perceptron.

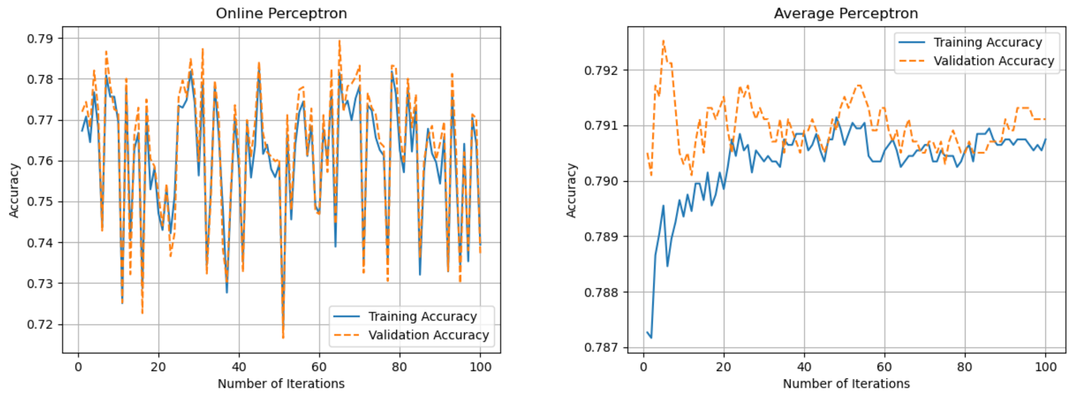


Figure 1 Training and Validation Accuracies (Left: Online Perceptron, Right: Average Perceptron)

(b) What are your observations when comparing the training accuracy and validation accuracy curves of the average perceptron with those of the online perceptron? What is your explanation for the observations?

Training and validation accuracy curves of the average perceptron are smoother and less fluctuate than the online perceptron's. The online learning updates the weights after each mistake and then makes predictions, so it is sensitive to individual examples. On the other hand, the average perceptron makes its predictions based on the averaged weights of intermittent ones, which makes the accuracy curve smoother. The average perceptron also achieves higher validation accuracy (0.79253) than the online perceptron does (0.78929)

(c) Focusing on average perceptron, use the validation accuracy to decide the best number of iterations to stop.

For this validation set of data, the best number of iterations to stop is 5 where the algorithm achieves the highest validation accuracy of 0.79253.

Part 2. Perceptron with Polynomial Kernel

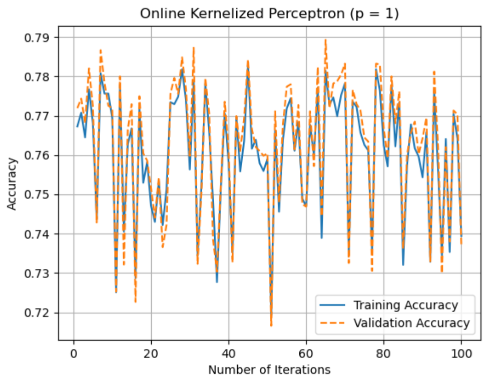
Part 2a. Online Kernelized Perceptron

(a) Apply the kernelized perceptron with different p values in [1, 2, 3, 4, 5] with maxiter = 100. Note that p = 1 will return to the vanilla online perceptron, so you should expect similar behavior compared to part 1.

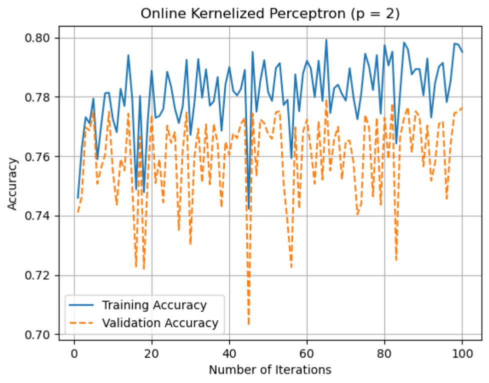
The accuracies of the kernelized perceptron with p = 1 are the same as those of the online perceptron as shown in Figure 2(a).

(b) For each p value, at the end of each training iteration use the current model (aka the current set of α's) to make prediction for both the training and validation set. Record and plot the train and validation accuracy as a function of training iterations.

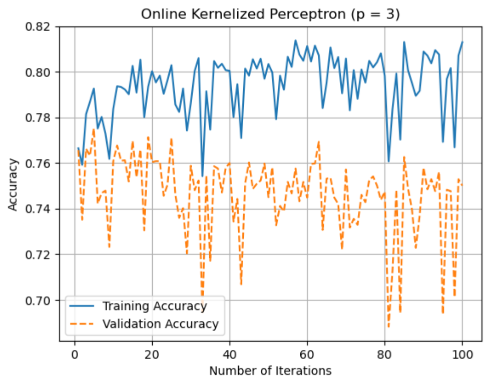
Figure 2 shows training and validation accuracies of the online kernelized perceptron.



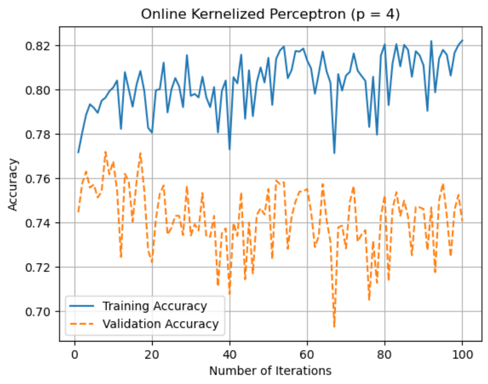
(a) p = 1



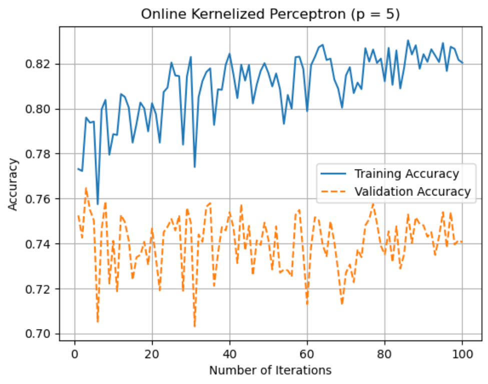
(b) p = 2



(c) p = 3



(d) p = 4



(e) p = 5

Figure 2 Training and Validation Accuracies of Online Kernelized Perceptron with different p values

(c) Record the best validation accuracy achieved for each p value over all iterations. Plot the recorded best validation accuracy versus p. How do you think p is affecting the train and validation accuracy and what is your explanation for the observation?

The best training and validation accuracies achieved are recorded in Table 1. As you can see in Figure 3, the best training accuracy increases as the p value increases while the best validation accuracy decreases. Using a higher degree polynomial kernel (larger p value) is equivalent to working in a larger feature space, so it can lead to overfitting. In other words, the algorithm tries to fit the training data more and therefore results in lower accuracy on the validation set. Therefore, the best value for p is 1 for this data set.

p	**1**	**2**	**3**	**4**	**5**
Best Training Accuracy	0.78279	0.79920	0.81373	0.82229	0.83035
Best Validation Accuracy	0.78929	0.77919	0.77515	0.77192	0.76465

Table 1 Best Training and Validation Accuracy

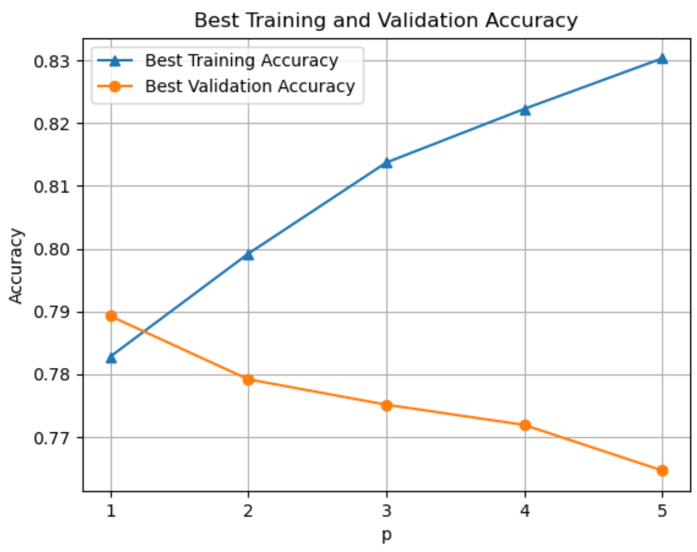


Figure 3 Best Training and Validation Accuracy

(d) What is the asymptotic runtime of your algorithm in terms of the number of training examples n? Try to make your implementation as efficient as possible. For the p value chosen above, plot the empirical runtime of your algorithm as a function of the iterations.

The asymptotic runtime of the algorithm is O(n^3) since we have two for loops nested and each loop, we compute the polynomial kernel of two vectors with length n, of which the time complexity is O(n). Then, I have removed for loops and used matrix operations to speed up and got the empirical runtime shown in Figure 4. When p = 1, it takes 2.02216 seconds to compute the kernel matrix. Then, time for the *while* statement is measured—it takes around 0.32 seconds/iteration and around 32 seconds for 100 iterations.

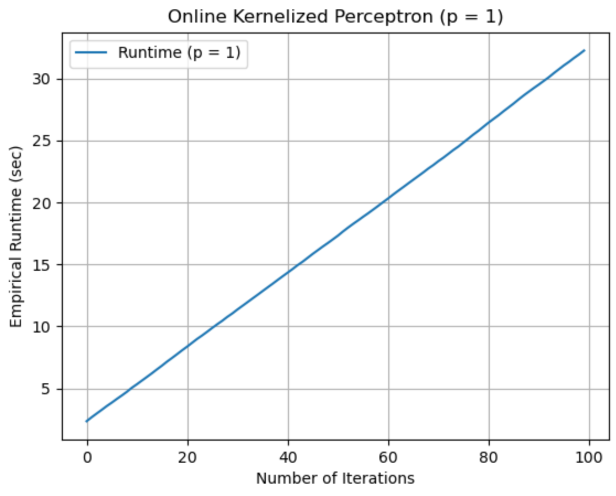


Figure 4 Empirical Runtimes of Online Kernelized Perceptron with p = 1

Part 2b. Batch Kernelized Perceptron

```
Algorithm 1: Batch Kernelized Perceptron
   Input: $\{(\mathbf{x}_i, y_i)_{i=1}^N\}$(training data), $maxiter$(maximum iterations),
          $\kappa$(kernel function), $\lambda$(learning rate)
   Output: $\alpha_1, ..., \alpha_N$
1  Initialize $\alpha_i \leftarrow 0$ for $i$=1,...,$N$
2  for $i = 1, ..., N, j = 1, ..., N$ do
3      $K(i, j) = \kappa(\mathrm{x_i}, \mathrm{x_j})$;
4  while $iter < maxiter$ do
5      $\Delta_i \leftarrow 0$ for $i$=1,...,$N$
6      for each training example $\mathrm{x_i}$ do
7          $u \leftarrow \sum_j \alpha_j K(i, j) y_j$
8          if $uy_i \leq 0$ then $\Delta_i \leftarrow \Delta_i + 1$
9      $\Delta \leftarrow \Delta/N$
10     $\alpha \leftarrow \alpha + \lambda\Delta$
```

(a) Apply your batch kernel perceptron with the best p value selected in part 2a. You should tune the number of iterations to maximize the validation accuracy. Report the iteration number with the best validation accuracy you achieve. Explain why changing the learning rate does not impact the learning trajectory and the learned decision boundary?

The pseudocode for batch kernelized perceptron is described in Algorithm 1. The α's are updated after observing the entire training set. For this validation set of data, the number of iterations to maximize the validation accuracy is 89 where the algorithm achieves the highest validation accuracy of 0.79414 when using p = 1, maxiters = 100. The reason why changing the learning rate does not impact anything is because it just uniformly scales the α's and thus the classifier will be the same.

(b) Record and plot the training accuracy and validation accuracy as a function of the iterations. Comparing these curves with the ones acquired with the same p value in part 2a, what do you observe? What are your explanations for the observation?

Figure 5 shows the training and validation accuracies of the batch kernelized perceptron with p = 1. Comparing the curves with those of the online learning with the same p value, we can see massive drops in accuracy. Almost every update, the accuracy goes up and down. Since we are doing update after seeing all the examples, all the misclassified examples contribute to the update at once, which moves the classifier in a direction that corrects the error relatively fast. However, our data is not linearly separable, there still be misclassified examples. Next update relies on all the misclassified examples and therefore, just one update could ruin it and result in a huge decrease in accuracy.

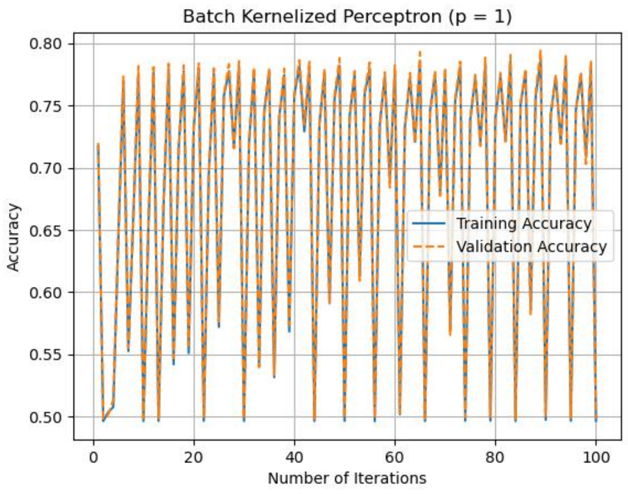


Figure 5 Training and Validation Accuracies of Batch Kernelized Perceptron with p = 1

(c) What is the asymptotic runtime of your algorithm in terms of the number of training examples n? Try to make your implementation as efficient as possible. Plot the empirical runtime of your algorithm as a function of the iterations. How does it compare to the runtime of the online algorithm?

The asymptotic runtime of the algorithm is $O(n^3)$ since we have two for loops nested and each loop, we compute the polynomial kernel of two vectors with length n, of which the time complexity is O(n). Then, I have eliminated all for loops and implemented it as matrix operation, and got the empirical runtime shown in Figure 6. It takes 2.03077 seconds to compute the kernel matrix. Then, time for the *while* statement is measured—it takes around 0.035 seconds/iteration and around 3.47 seconds for 100 iterations. Comparing to the runtime of the online version of the perceptron algorithm (32 seconds for 100 iterations), it is about 9.2 times faster. This would be because we can use matrix operation to process all examples at once when doing batch update.

* Since it takes almost the same amount of time to compute the kernel matrix in both the online and batch algorithms, I only compared the runtime of the *while* statement of the two algorithms.

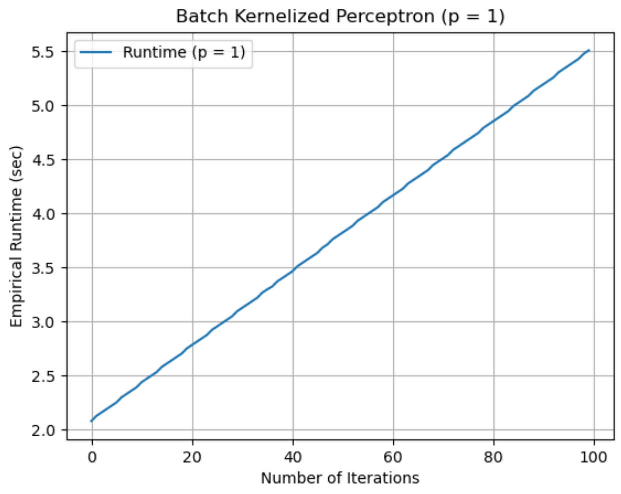


Figure 6 Empirical Runtime of Batch Kernelized Perceptron with p = 1