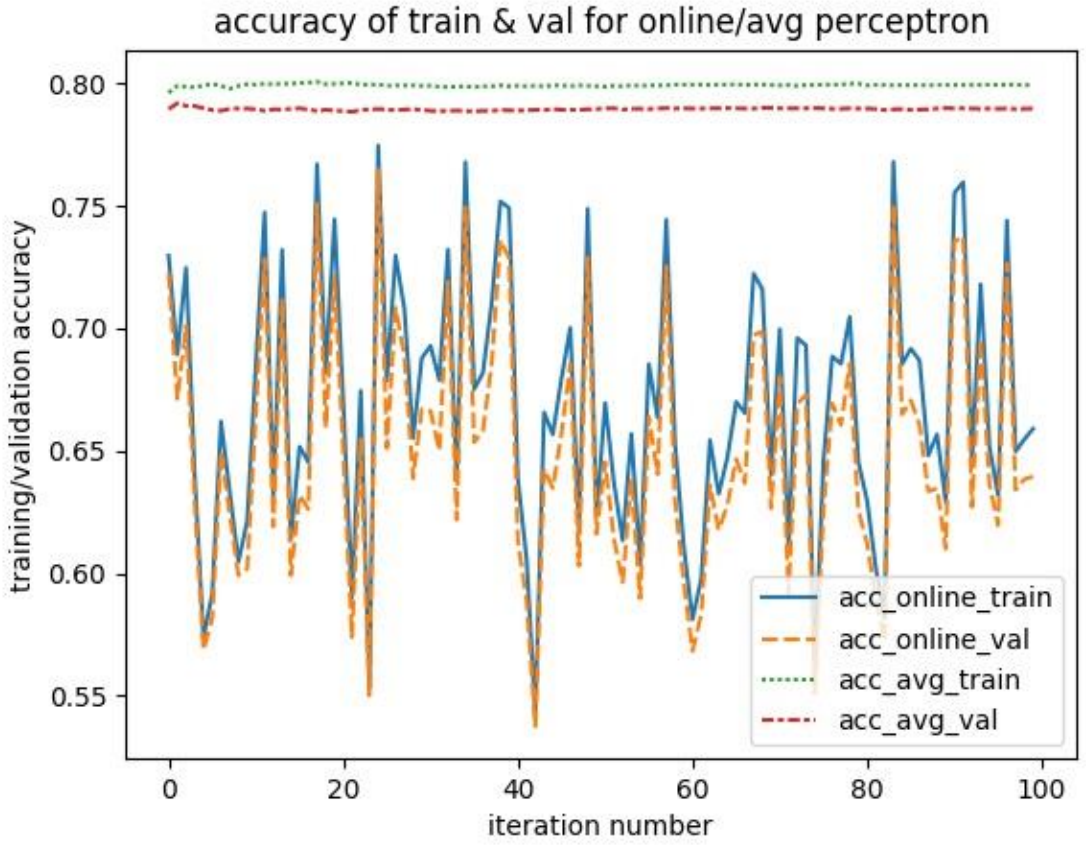
IA3 report
Name: Yijie Ren

General Introduction:
This report illustrates the results of online Perceptron, average Perceptron and Perceptron with Polynomial. It is surprising for me that all the online perceptron models, no matter it is with or without kernel, are all providing the bouncing accuracy plots, for both training and validation set. Also, I learned that using batch technique can get convergence more quickly.

Part 1:
(a)



The training/validation accuracy for online perceptron is lower than the training/validation accuracy for average perceptron. Also, the accuracy of average perceptron is bouncing back and forth, while the accuracy of average perceptron is stable despite some fluctuations at the very beginning iterations.

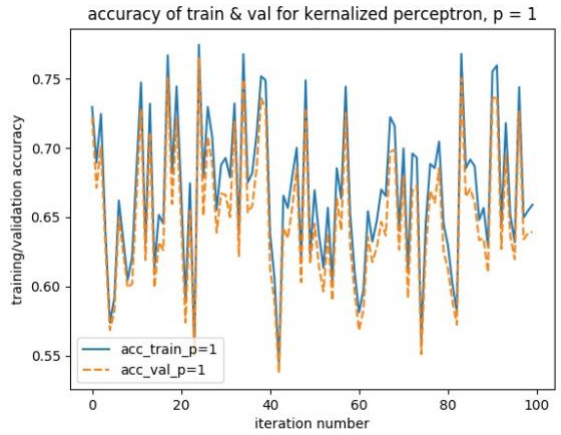
The explanation for this phenomenon could be the average perceptron always aggregates the information contained in both w and w_bar during each iteration, while the online perceptron depends more on a single training data to update w.

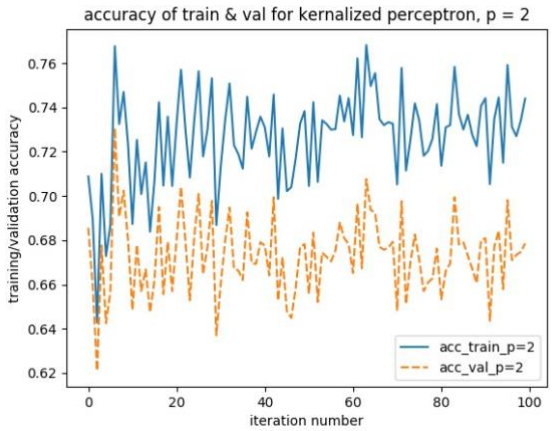
(b)
For online perceptron, according to validation accuracy, the best stopping point could be iteration # 23, and the accuracy is 0.765 at that point. For average perceptron, according to

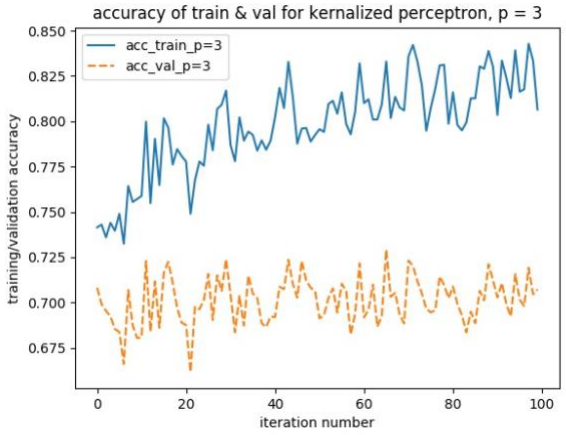
validation accuracy, the best stopping point could be iteration # 20, and the accuracy is 0.79 at that point.

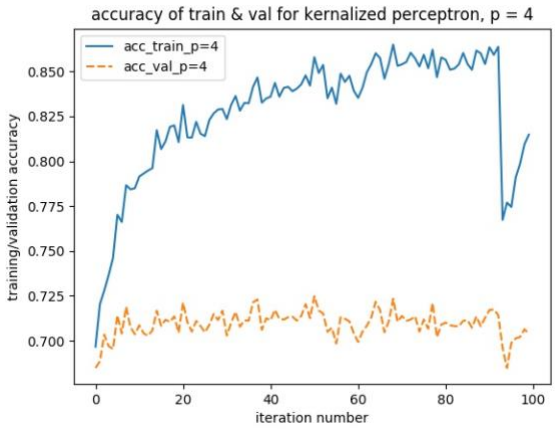
Online perceptron is more sensitive to stopping point. The practical implication could be that during training process, it is better to average the parameter, instead on solely depending on the single input data.

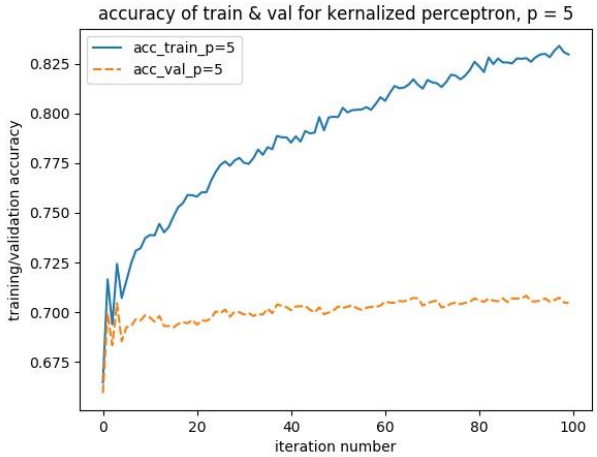
Part 2a:
(a)







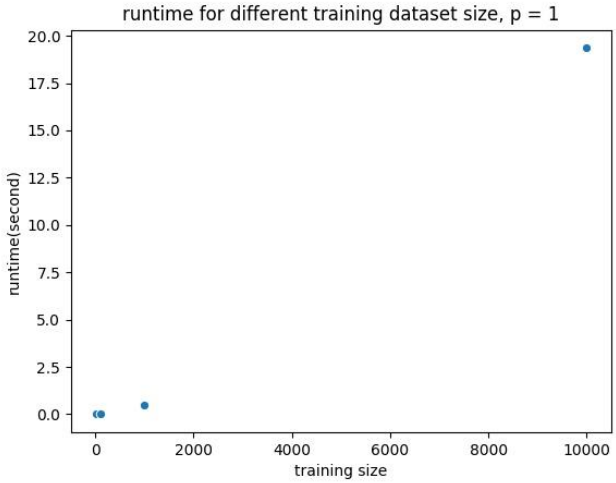




As p increases, the training accuracy is also increasing with the number of training iteration, while the validation accuracy almost remains the same.
Yes, there is the risk of overtraining (i.e., when training for too many iterations leads to overfitting) for some higher p values. The overfitting happens because the higher p value is, the more complex decision boundary is. For example, p=1 leads to a linear separation, while p=2 leads to a quadratic decision boundary, and so on.

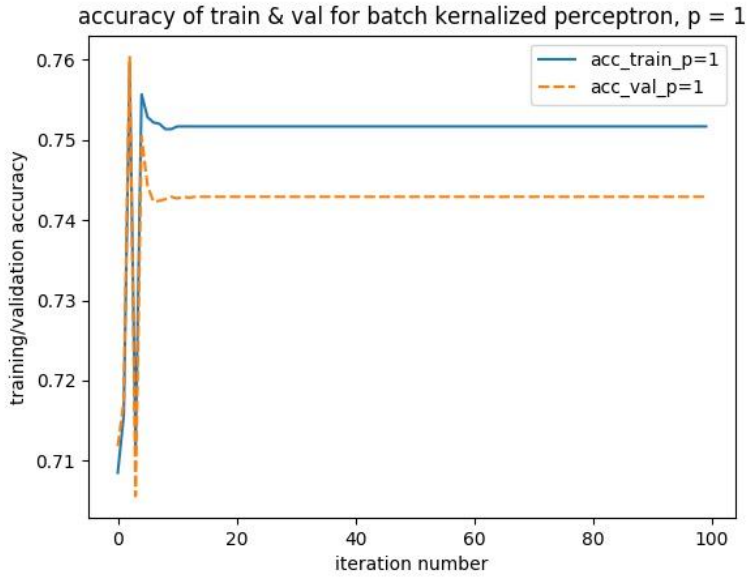
(b)
p = 1, best training accuracy is 77%, best validation accuracy is 76%
p = 2, best training accuracy is 78%, best validation accuracy is 73%
p = 3, best training accuracy is 84%, best validation accuracy is 72%
p = 4, best training accuracy is 87%, best validation accuracy is 72%
p = 5, best training accuracy is 84%, best validation accuracy is 71%
p =1 produces the best validation accuracy.
When the value of p increases, the training accuracy will always increase at first, but drop down after the model is trained "perfectly" with the known training set (i.e., the value of p reaches the "best" value). At this time, the model is pretty overfitted, so the training accuracy becomes highest.
When the value of p increases, the validation accuracy is decreasing, because the model is getting more and more overfitted. Thus, in order to get higher validation accuracy, always use the rather "simple" model, which can gain more generalization.

(c)



The asymptotic runtime in terms of the number of training examples n is $O(n^2)$ (kernelize training data $O(n^2)$ + training process (iter num 100 * for loop N * update u value N) --> $O(n^2)$). According to the figure above, the empirical runtime matches up with the asymptotic analysis.

Part 2b:
(a)



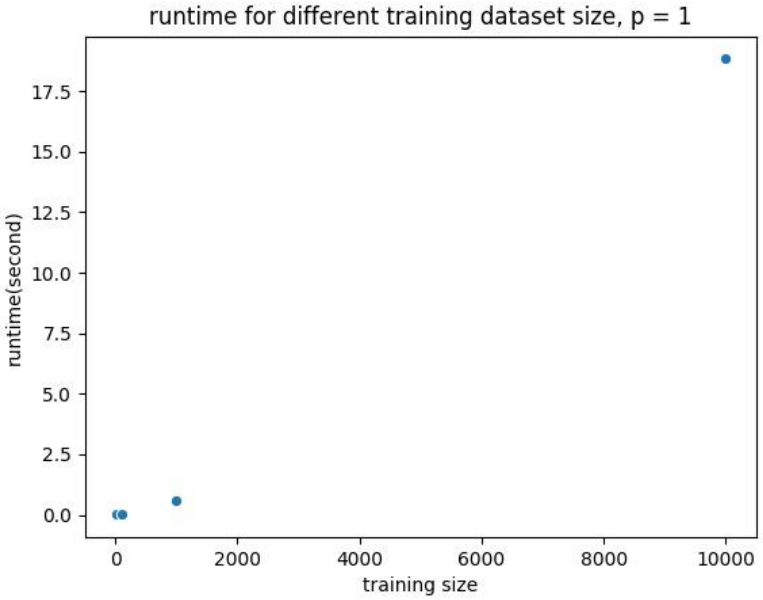
The difference from Part 2a is that the above curve converges more quickly. The explanation for this could be instead of using the last alpha from last iteration to calculate the accuracy, batch perceptron uses the accumulation of alpha to update for each iteration, which leads to more stable accuracy.

Assume the learning rate is not too large to diverge, then using higher learning rate will lead to faster convergence of the model.

(b)

Psudocode:
Input: {(x_i,y_i), i = 1, …, N }(training data), maxiter (maximum iterations), κ (kernel function)
Output: $\alpha_1,...,\alpha_N$

```
Initialize α_i ← 0, for i = 1,...,N;
for i = 1,...,N, j = 1,...,N do
    K(i,j) = κ(xi,xj);
end
while iter < maxiter do
   α_temp = α;
   for each training example x_i do
      u ← Σ_j α_j K(i,j) y_j;
      if u y_i ≤ 0 then
         α_temp_i ← α_temp_i + 1;
      end
   end
   α ← α + α_temp;
end
```



The runtime is still $O(n^2)$, as the same calculation in Part 2a(c). According to the figure above, the empirical runtime matches up with the asymptotic analysis.

There is no observation of a significant difference in runtime compared to the online algorithm. Since adding the accumulation operation for alpha costs constant time complexity, the overall time complexity remains the same.