Chenyun Yu

Prof. Fern

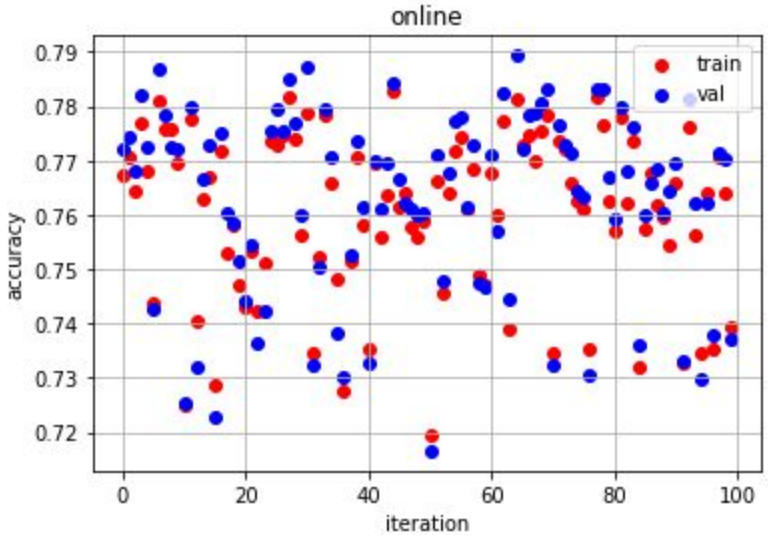
CS-534 MACHINE LEARNING

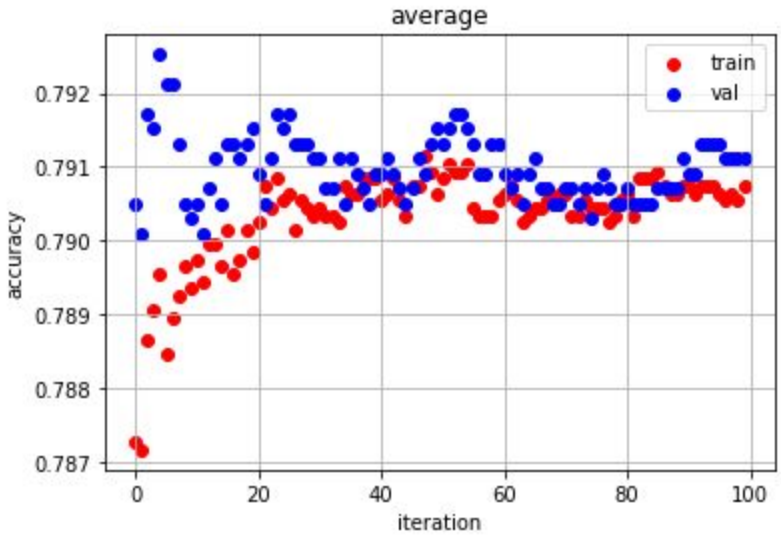
Implementation 3 Reporter

Part1 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 $\bar{w}$ (average perceptron) at the end of each training iteration.

Answer. The train and validation accuracy show as following:





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?

Answer. In my opinion, although the online perceptron method can achieve a not bad accuracy rate, its accuracy does not seem to be better as the iteration increases. The accuracy rates scatter between 0.7 to 0.8 randomly(or, at least looks like randomly), which means the prediction driven by the last weight is also random.

Average perceptron will be a better predictor because the convergence trend is very clear, I think the main reason causing the difference between online and average is "deviant" and "maverick" points will draw the wights to a different direction many times, which makes the convergence becomes very hard. However, the average perceptron can minimize the influence of those points to some degree.

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

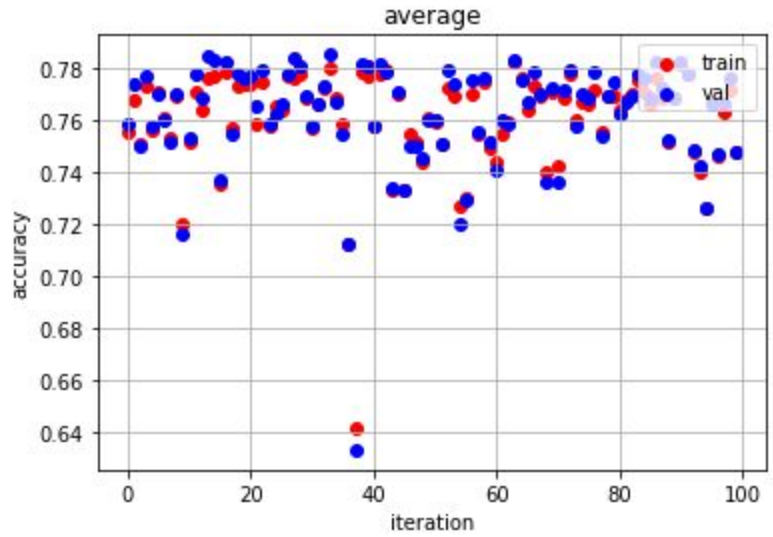
Answer. Based on my observation, I think that 50 times iteration is enough.

Part 2a. Perceptron with Polynomial Kernel

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.

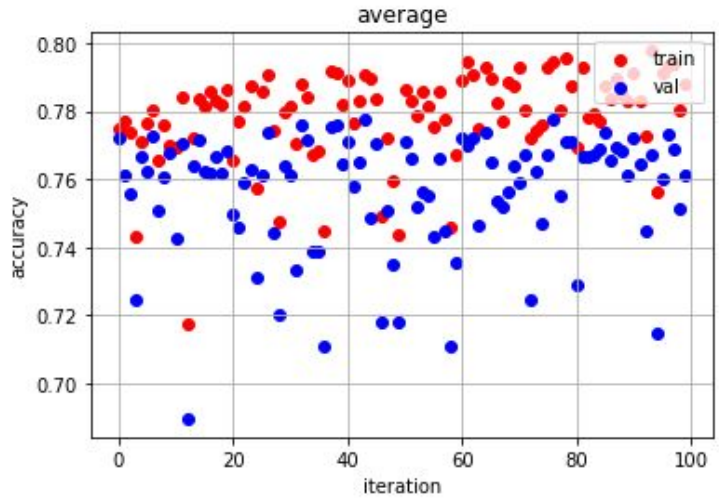
Answer.

P = 1: max training accuracy = 0.782, max validation accuracy = 0.785

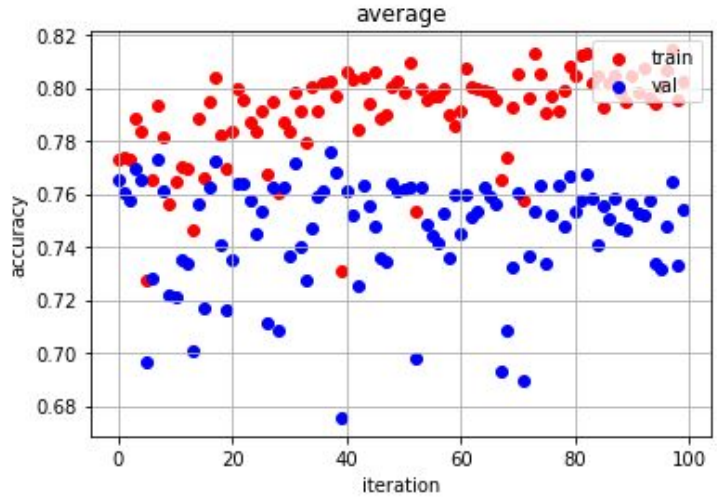


Except for a certain singularity located 36th iteration, the p1 kernelized perceptron has a similar un-convergence trend, and the accuracy is also scattered between 0.72 to 0.8.

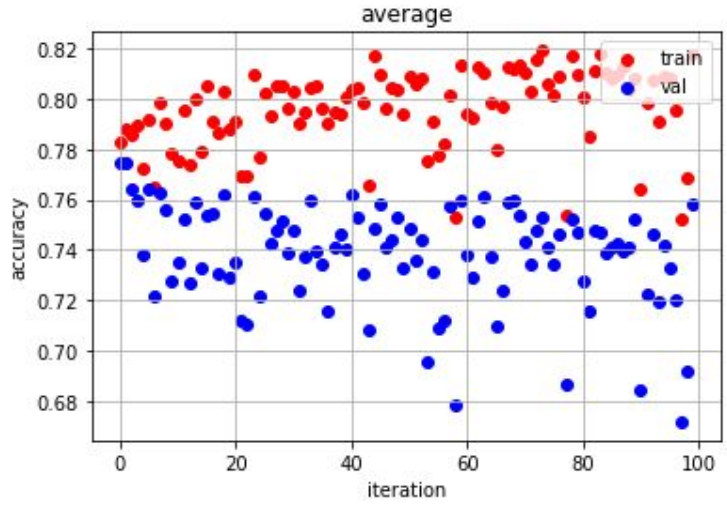
P =2: max training accuracy = 0.798, max validation accuracy = 0.777



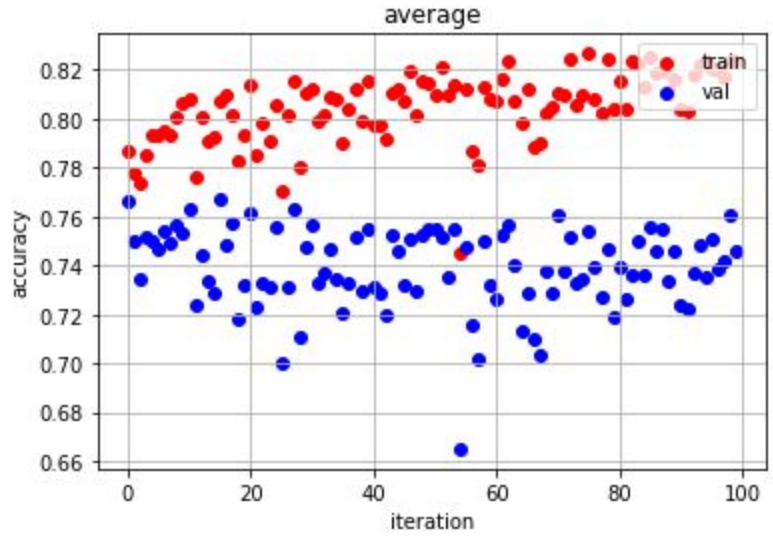
P=3: max training accuracy = 0.814, max validation accuracy = 0.776



P = 4, max training accuracy = 0.819, max validation accuracy = 0.774

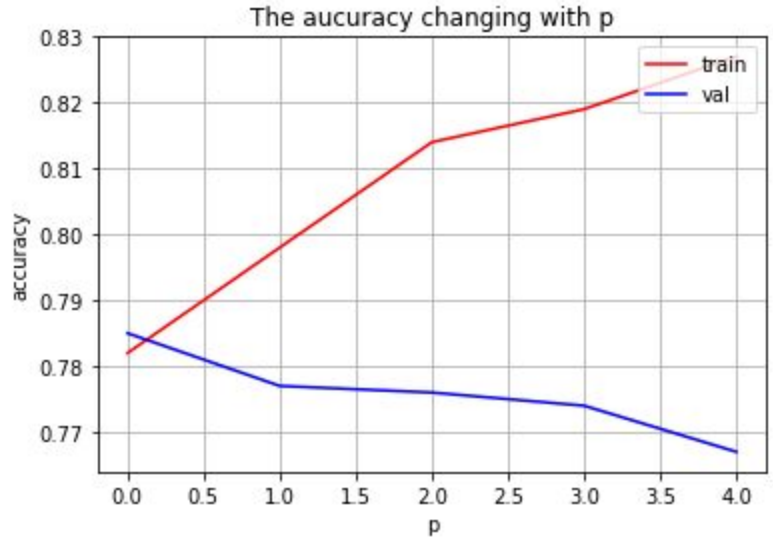


P =5 , max training accuracy = 0.827, max validation accuracy = 0.767



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 a prediction for both the training and validation set. Record and plot the train and validation accuracy as a function of training iterations.
Answer. Shown above

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?
Answer. I think it is a kind of overfitting, for this set of data p = 1 is enough for linearly separate the data, using higher order polynomial function will lead to overfitting. The evidence of overfitting happening is the training accuracy still becomes better, but the validation accuracy actually worsens.



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.

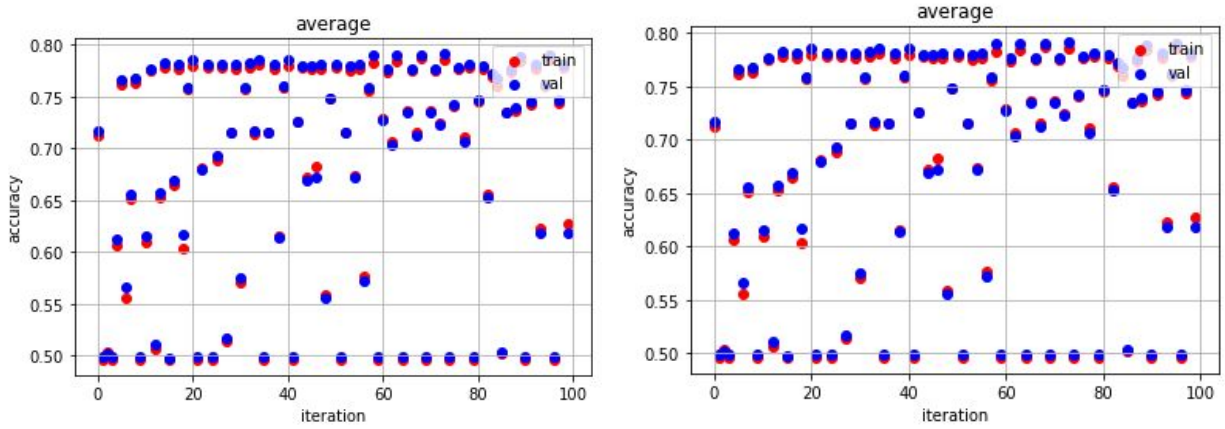
Answer. I think runtime is not the first thing we should concern, the overfitting phenomenon happens when P larger than 1. The only value of P that can be chosen is 1.

Part 2b. Batch Kernelized Perceptron

a) Apply your batch kernel perceptron with the best p value selected in part 2a. You should tune your learning rate as well as the number of iterations to maximize the validation accuracy. Report the configuration of the learning rate and iteration number with the best validation accuracy you achieve.

Answer. I don't think the learning rate does any effort for perceptron training because when we use the learning rate, we always want to do the numerical computations to approach some value. However in the kernelized perceptron, the update step is just about **the sign, not its value**, and the learning rate is always a positive num, it will not change the sign.

P = 1, learning rate = 0.1 (left) P = 1, learning rate = 0.1 (right)



Another possibility is my implementations are wrong.

```
for _ in range(maxIter):
    u = np.dot(gram_matrix, self.alpha * self.train_y)
    # (?) I don't think learning rate does any effort,
    # because other computations are only about sign, not value
    # but the learning rate is always a positive num,
    # it will not change the sign.
    u *= learning_rate
    for i in range(N):
        if u[i] * self.train_y[i] <= 0:
            self.alpha[i] += 1
```

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?

Answer. As shown above, the difference between the training accuracy and validation accuracy becomes smaller. I don't know how to exactly express my idea, I think this batch updating method makes each iteration likes a "fresh start".

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?