IA3

Yongsung Cho

Computer Science, Oregon State University

AI 534: Machine Learning

Prof. Fern

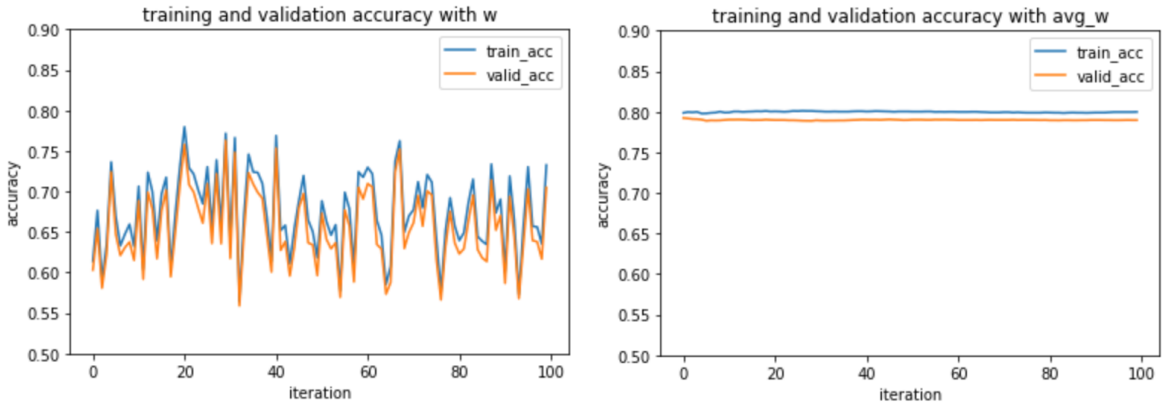
Due Nov 14, 2021

Preprocessing

Dataset is same as IA2 train and validation dataset. As I mentioned on IA2 report, most features were one-hot encoding, so, no other preprocessing was required. However, normalization was needed because Age, Annual_Premium, and Vintage are numeric numbers. I did normalization and training in the same way that we did in IA2.

Part 1 Average Perceptron

(a) Plot the train and validation accuracy of w (online perceptron) and $\overline{w}$(average perceptron) as a function of the number of training iterations.



The left graph was trained using an online perceptron, and it is training and validation accuracy accordingly. The right chart was trained using an average perceptron and the training and validation accuracy for it.

Question: 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?

Online perceptron has more changes in accuracy than average perceptron. In other words, the online perceptron graph is more fluctuating than the average perceptron graph. Furthermore, the average perceptron remains almost constant, with high accuracy of about 80% of the accuracy. This is because the iteration increases the s value, so the updated amount of change is not significant. On the other hand, the online perceptron updates the w value for each training sample. The accuracy changes significantly because the amount of change for w is large every time the iteration increases.

(b) For both average and online perceptron, use the validation accuracy to decide the best stopping point and report the iteration number and resulting validation accuracy.

online perceptron / Iteration: 30 with accuracy 0.7629 (76%)

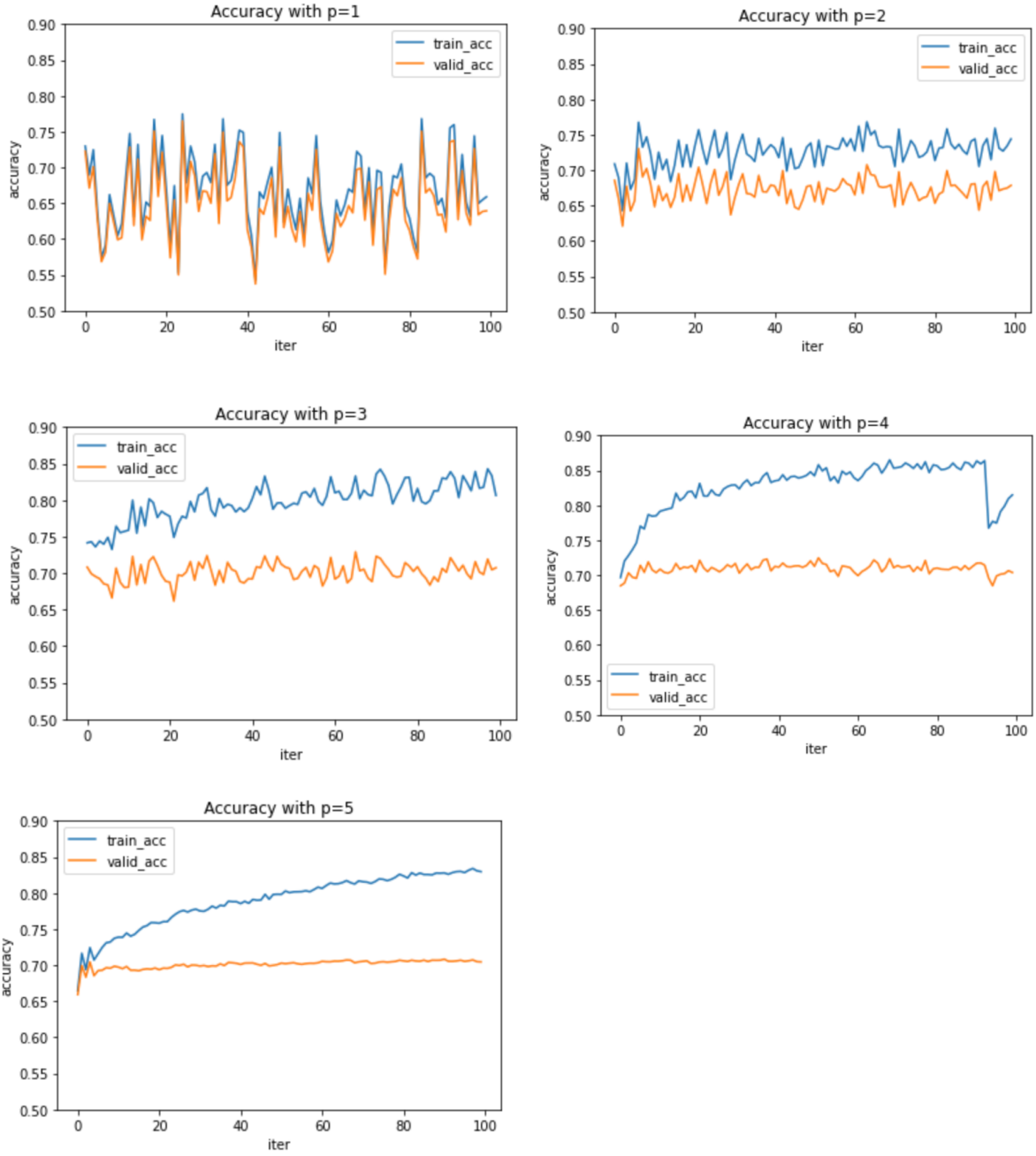
average perceptron / Iteration: 1 with accuracy 0.7923 (79%)

Question: 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?

Online perceptron is more sensitive to the stopping point. Because accuracy fluctuates more than average perceptron, and if I change just several iterations, the accuracy will increase or decrease dramatically in online perceptron. This situation implies that the updated values are too big, and it doesn't update weight values efficiently. In addition, because it is an online perceptron, I updated the weight vector per each example. Per each iteration, I updated weight many times, which is the same as the size of the training dataset. In our case, the weight vector is updated 6000 times per iteration.

Part 2 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. For each p value, at the end of each training iteration (the while loop) 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.



As the P-value increases, the fluctuation of the graph tends to decrease. In addition, the difference between training accuracy and validation accuracy increased as the number of items increased in most charts.

Question: 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?

As described above, the fluctuation of graphs tended to decrease as the P-value increased; The fluctuation decreased at p=5 compared to P=1. Moreover, as the number of items increases, the difference between training accuracy and validation accuracy increases. In other words, p=1 graphs showed similar graph shape and value with training accuracy and validation accuracy. But at p=5, there was an increase in the difference between the two accuracies, showing a difference of about 15%. The training accuracy and validation accuracy were rapidly lowered when the iteration was about 90 in the P=4 accuracy graph. This is thought to be overfitting because the local minimum exists at p=4, and then training to the Optimal point again. This phenomenon did not occur because the local minimum does not exist in other p values.

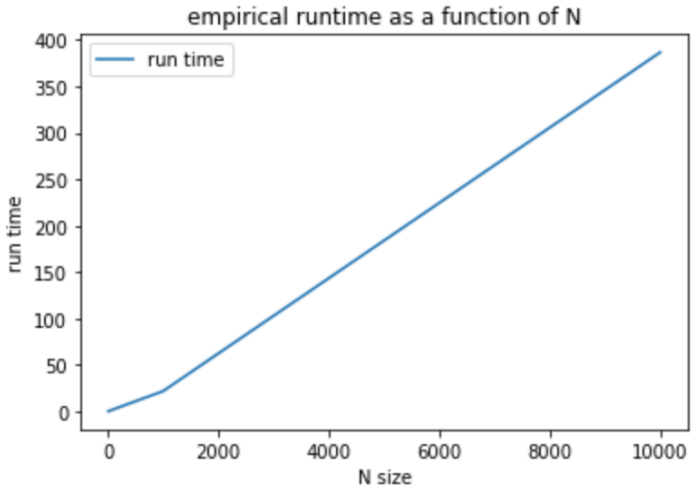
(b) Report the best training and validation accuracy achieved for each p value (over all iterations).

P value	P=1	P=2	P=3	P=4	P=5
Training accuracy	0.774667	0.768167	0.842833	0.865	0.834
Validation accuracy	0.7648	0.7307	0.7292	0.7248	0.7083

Question: Which value of p produces the best validation accuracy? How do you think p is affecting the train and validation accuracy?

At P=1, the highest validation accuracy was 76%. As the P-value increases, training accuracy tends to increase except for p=5. However, on the contrary, as the p-value of the validation Accuracy increases, the Accuracy tends to decrease. This result shows that the p-value is increased, and overfitting is performed on the training dataset. Because it was overfitting on the training dataset, the increase in p-value affected the validation accuracy.

(c) For p = 1, plot the empirical runtime of your algorithm (with a fixed number of iterations) as a function of N, the training data size. You can do this by measuring the runtime of your algorithm on training data of different sizes ranging from 10, 100, 1000 to 10000 (you can use the subset of the validation data for the last size).

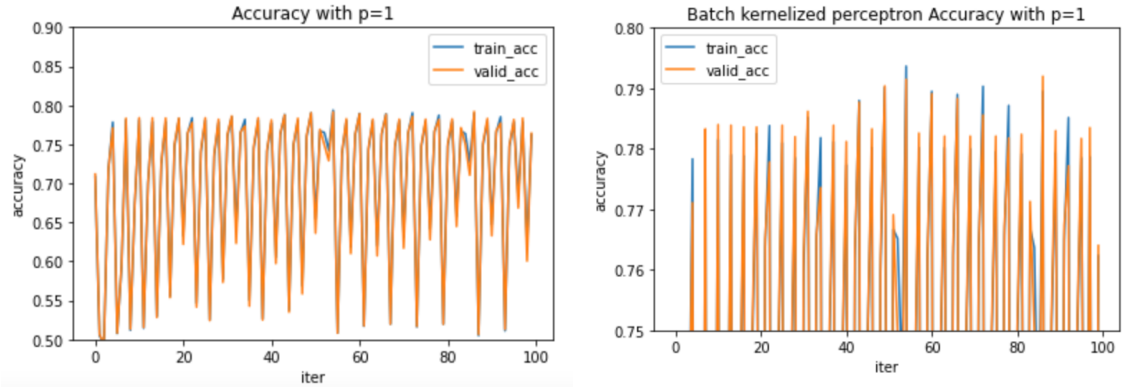


Question: what is the asymptotic runtime of your algorithm (in big O notation) in terms of the number of training examples n? Does your empirical runtime match up with the asymptotic analysis?

With a fixed number of iteration, run time increases linearly according to the size of n. Because the number of iteration is fixed, I don't need to consider iteration for loop. As a result, run time increases linearly as the size of n increases, which can be seen to follow the asymptotic analysis of O(n). The Empirical runtime graph shows that the N size and run-time are increasing linearly, so it can be said that O(n).

Part 2b Batch kernelized perceptron

(a) Apply your batch kernel perceptron with the p value that achieved the best validation accuracy in part 2. Record and plot the training accuracy and validation accuracy in a single figure as a function of the iterations.



In part2(a), when the p-value is 1, I achieved the best validation accuracy of around 76%. So, I used p=1 for recording and plot training accuracy and validation accuracy with batch kernelized perceptron. The above graph plots training accuracy and validation accuracy on one graph. However, the difference between the two values is not significant, so it shows a similar graph shape. The right graph is a graph that magnifies the portion between 0.75 and 0.8. Looking at the chart, you can see that the two graphs are subtly different, but the difference is not significant.

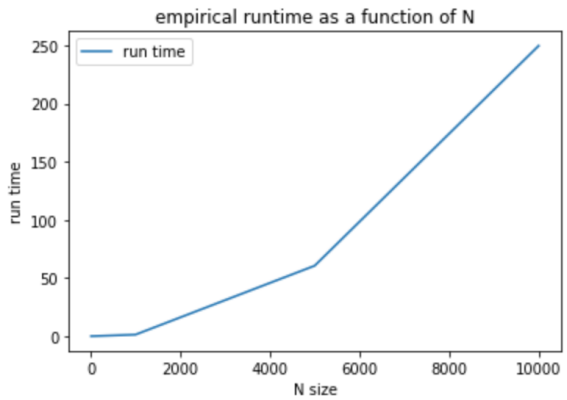
Question1: Comparing the curves for batch kernel perceptron with the ones acquired with the same p value in part 2a, do you observe any differences? What are your explanations for them?

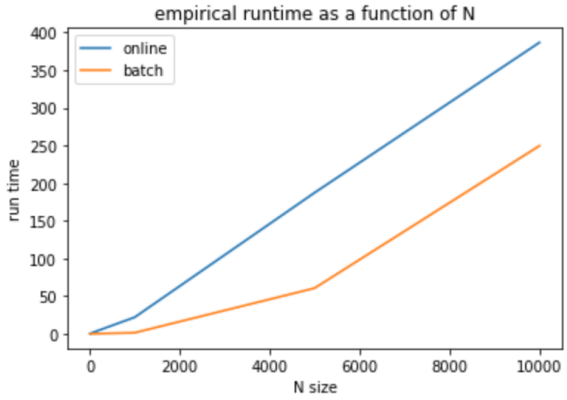
Comparing the curves for batch Kernel perceptron with the ones obtained with the same p-value in part 2a, I found that the batch Kernel perceptron graph fluctuates more. The accuracy of the Online perceptron fluctuated significantly too, but the batch algorithm showed a tendency to fluctuate more frequently and significantly. This difference is due to the difference in algorithms. The online perceptron updates alpha for each example and predicts the following example with the updated alpha value. Hence, the error probability for the following example is relatively small. However, when the batch algorithm is applied, all examples are predicted as one alpha value. The amount of alpha change for them is updated to alpha at once. So, changes to alpha are greatly affected by each iteration. In other words, each iteration fluctuates more heavily because the updated value is too large.

Question2: Perceptron uses a fixed learning rate of 1 for the subgradient descent. What would be the impact if we use a different learning rate?

The Batch Kernelized perceptron graph fluctuation is significant because the learning rate is very large at 1. If we use learning rates such as 0.1 and 0.001, the size of the alpha updated at a time is reduced, and the graph is expected to fluctuate less than the current graph. Moreover, the current learning rate is too large, so it does not show convergence, but the chart will be converged if the learning rate is lowered.

(b) Follow the same procedure as in part 2a, plot the empirical runtime of your algorithm (with fixed number of iterations) as a function of the training data size.





Question: 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.

Pseudocode for batch kernel perceptron

Input: $\left\{(x_i, y_i)\right\}_{i=1}^{N}(training\ data), maxiter\ (maximum\ iterations), \kappa\ (kernel\ function)$
Output: $\alpha_1, \ldots, \alpha_N$
$Initialize\ a_i \leftarrow 0\ for\ i = 1, \ldots, N$;
$Initialize\ temp\_a_i \leftarrow 0\ for\ i = 1, \ldots, N$;
for $i = 1, \ldots, N, \quad j = 1, \ldots, N\ do$
$K(i,j) = \kappa(x_i, x_j)$; // $Compute\ the\ Gram\ Matrix$
end
while $iter < maxiter\ do$
$temp\_a_i \leftarrow 0\ for\ i = 1, \ldots, N$;
$u \leftarrow \sum_j \alpha_j K(i,j) y_j$; // $make\ prediction\ for\ xi$
if $uy \leq 0$ ***then***
$temp\_a \leftarrow temp\_a + 1$; // $Mistake\ on\ example\ i, increment\ the\ counter$
$\alpha_i \leftarrow \alpha_i + temp\_a_i$;
end

Unlike the Online Kernelized perceptron, the batch Kernelized perceptron removed the for loop repeating each example, so it was expected that asymptotic analysis would become O(1). However, it is estimated that run time increases as N size increases because the NumPy matrix multiplication is required. The right graph shows that the batch kernelized perceptron has a less run time than the online kernelized perceptron. Therefore, I couldn't obtain the result that the asymptotic analysis is made into O(1) by removing the for loop repeating each example. Still, it can be seen that it affects run time reduction.