AI 534 IA3 Report

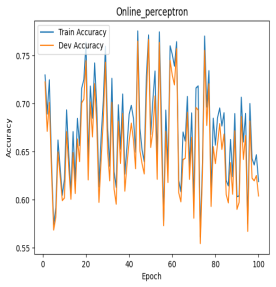
Rishab Balasubramanian

1.a. We can see that the curves in Fig.1 for online perceptron is relatively more oscillatory, compared to that of the average perceptron. We can attribute this to the fact that the online perceptron adds $w \leftarrow y_i x_i$ directly after each misclassified training example. In comparison, the average perceptron only adds a fraction of this $w \leftarrow w + z$, where $z = 0$ if the training example is correctly classified, and $z = y_i x_i/(s+1)$ if not. Thus we see that the change in weights of the average perceptron is much smaller than that of the online perceptron, and as we keep iterating through the set of training examples the denominator of the update function keeps increasing (i.e, the change in w decreases) in the case of average perceptron. This explains the smoother curves from average perceptron, compared to the oscillatory ones from online perceptron.

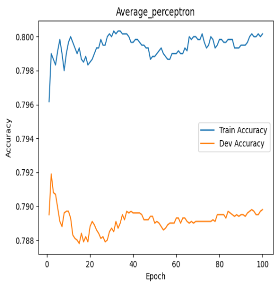
b.
Online Perceptron : Best Validation Accuracy - 76.6%, Iteration number - 48
Average Perceptron : Best Validation Accuracy - 79.19%, Iteration number - 1

As we can see, the online perceptron is more sensitive to the stopping point. This is because of how the weights are constantly changing in the case of online perceptron, which causes large variations in prediction accuracies in comparison to the average percpetron, where the change in weights become negligible as the number of iterations increases. As a result, I feel average perceptron should generally always give better results than online perceptron. One drawback of the average perceptron is that it gives more weight to data which appears earlier, than those that appear later in the dataset. As a result shuffling the data leads to better results as shown in Fig.2. We see that the best accuracies obtained after randomizing data order are:

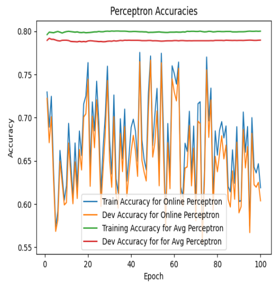
Online Perceptron : Best Validation Accuracy - 78.1%, Iteration number - 45
Average Perceptron : Best Validation Accuracy - 79.45%, Iteration number - 1



(a) Online Perceptron

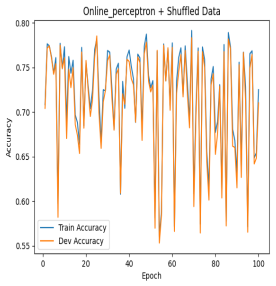


(b) Average Perceptron

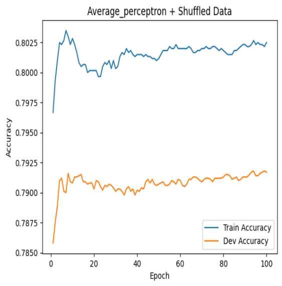


(c) Online vs Average Perceptron

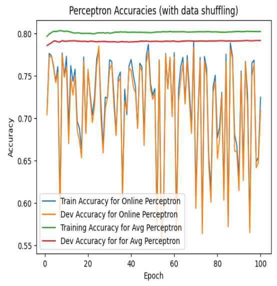
Figure 1: Training and Validation Accuracy for Perceptron Algorithm



(a) Online Perceptron



(b) Average Perceptron

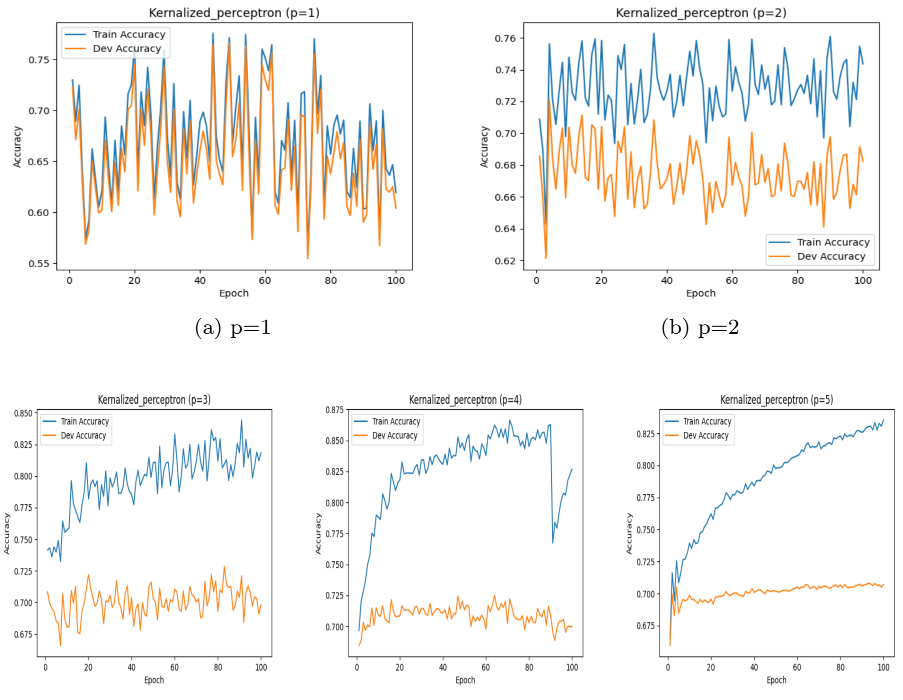


(c) Online vs Average Perceptron

Figure 2: Training and Validation Accuracy for Perceptron Algorithm with Data Shuffling

2a.a As p increases, the number of features increases as well due to the increase in dimensionality. As a result we see that for higher values of p the model overfits (i.e the training accuracy is lower than the validation accuracy, and training accuracy keeps increasing, while validation accuracy is almost constant), which is highlighted in Fig.3. We see that for $p = 1(2)$, the training and validation accuracies are similar, and follow a similar oscillatory(upward) trend. For the other values of p, we see that the training accuracy is significantly larger than the validation accuracy and keeps increasing, implying the model overfits to the training data. We can attribute this to the extra features introduced by increasing dimensions, which results in overfitting (similar to how the model overfits with a higher order polynomials for linear regression).

b Table.1 shows the best training and vaidation accuacies for all values of p. $p = 1$ produces the best validation accuracy. As the value of p increases, we see that the model overfits, and as a result the training accuracy increases, while validation accuracy slowly decreases/remains constant (highlighted clearly for $p = 5$).



(a) p=1

(b) p=2

(c) p=3

(d) p=4

(e) p=5

Figure 3: Kernelized Perceptron for different values of p

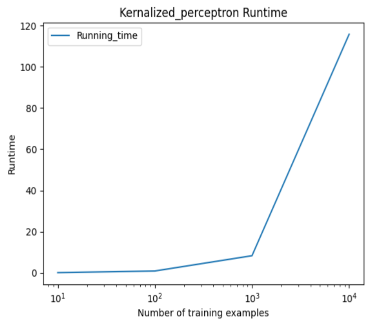
Table 1: Best Accuracies for different values of p for Kernelized Perceptron

$p = 1$		
	Accuracy	Epoch
Training	77.54%	43
Validation	76.6%	48
$p = 2$		
	Accuracy	Epoch
Training	76.28%	35
Validation	72.09%	3
$p = 3$		
	Accuracy	Epoch
Training	84.41%	90
Validation	72.86%	82
$p = 4$		
	Accuracy	Epoch
Training	86.63%	70
Validation	72.49%	63
$p = 5$		
	Accuracy	Epoch
Training	83.53%	99
Validation	70.80%	92

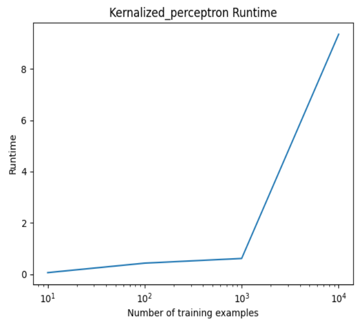
c. Fig.4a. shows the runtime of the Online Kernelized Perceptron Algorithm. The asymptotic time complexity of the algorithm is:

- Let the maximum number of iterations be m, the dimension of the data d, and number of examples n.
- Generating the kernel the matrix takes $O(n^2 d)$ time
- We train for m iterations
- Each time, we iterate through the data set, and predict the output for each example, which takes $O(nd)$ time.
- Therefore training in total takes $O(mnd)$ time, and including the time for kernelizing, we get $O(n^2 d + mnd)$.
- As $n >> m$ and $n >> d$, we can approximate this to $O(n^3)$ time.

We see that n^3 acts as an upperbound for the runtime of the algorithm for larger values of n.



(a) Online



(b) Batch

Figure 4: Runtime for Kernelized Perceptron

2b.a. 1. Fig.5 shows the plot for batch kernel perceptron. The batch kernel perceptron seems to oscillate more than the online kernel perceptron. This is because we update the α simultaneously after going throught the entire training data. Consider a scenario where examples in the training data, x_1 and x_2 are incorrectly labelled with the weights (w_t) at iteration t. In the online case, changing the weights after detecting misclassification of x_1 could possibly ensure that the next example x_2 is correctly classified. However in the batch update, as the weights are updated simultaneously, both examples x_1, and x_2 contribute to weight updates, thus causing larger changes in the α resulting in oscillations.

2. Having a learning rate would simply rescale the weights, but would not affect the decision boundary itself. If $w^T x > 0$, then for any positive value c, $cw^T x > 0$ is true. However, the algorithm would be affected if the learning rate is negative.

b. Algorithm.1 shows the algorithm for batch kernelized perceptron and Fig. 4b shows the corresponding running time. The batch perceptron, would again have $O(n^2 d)$ time for generating the kernel matrices (n = number of training examples, d = dimension of the data). For each training iteration, the algorithm would have to multiply each row of the kernel matrix with α, which takes $O(n^2)$ time. Thus the total time is $O(n^2 d + n^2 m) = O(n^3)$. We see that $O(n^3)$ provides a satisfactory upper bound on the running time of the algorithm. We however notice a significant speed up of our code, although no major changes have been made. This can be attributed to the use of python libraries (numpy), which use a efficeint static programming (C/C++) language to perform these computations, which results in the significant speed up.

Algorithm 1 Batch Kernalized Perceptron Algorithm

```
**Input** $\{(\mathbf{x}_i, y_i)_{i=1}^N\}$(for both training and dev data) , $maxiter$, $\kappa$(kernel function)
**Output** $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \cdots, \alpha_N]^T$
1: $\alpha_i \leftarrow 0$ for $i = 1, \cdots, N$
2: **for** $i = 1, \cdots N,\ j = 1, \cdots, N$ **do**
3:     $K(i, j) = \kappa(x_i, x_j) \Rightarrow K_{train} = (train * train^T)^p, K_{dev} = (dev * train^T)^p$ where $train$ is the training data, $dev$ is the dev data
4:     (*This can be done by multiplying the training data and its transpose giving a (6000*6000) matrix for the training kernel, and multiplying the validation data and training data to give (10000*6000) matrix for the validation kernel*)
5: **end for**
6: **while** $iter < maxiter$ **do**
7:     $\boldsymbol{\alpha} = \boldsymbol{\alpha} + K_{train}\boldsymbol{\alpha}\mathbf{y_{train}}$
8: **end while**
```

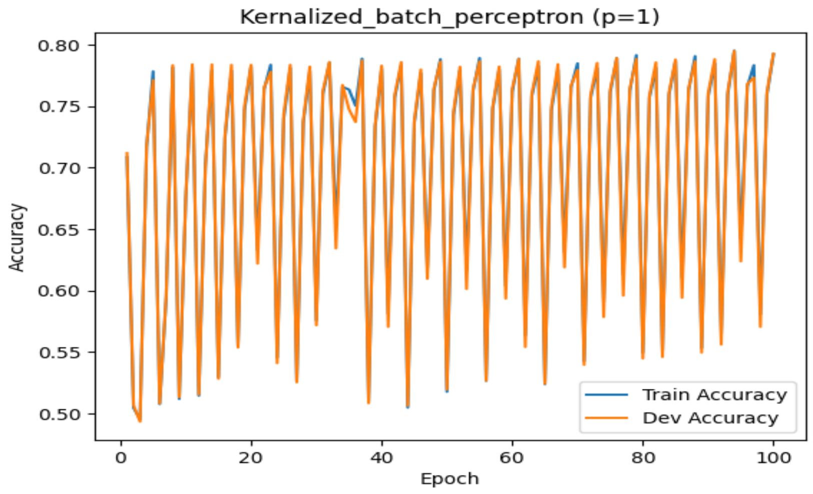


Figure 5: Batch Kernelized Perceptron