Homework #1 CPTS-570 Machine Learning

Ricardo Rivero

September 19, 2024

Analytical Part

1a. Is the decision boundary of voted perceptron linear?

Yes, the decision boundary of voted perceptron is linear because it is a linear combination of linear functions. If we recall the decision boundary of the perceptron, it is determined by the equation:

$$\mathbf{w} \cdot \mathbf{x} + \mathbf{b} = 0 \tag{1}$$

Where $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is the input feature vector and $\mathbf{b}$ is the bias term. Therefore, if the equation is linear, the decision boundary is linear. Building on that, the voted perceptron expands on the concept of the perceptron by maintaining multiple weight vectors throughout the training process, and associating each training vector with a counter $\mathbf{c_i}$, which represents the number of consecutive correct predictions made by $\mathbf{w_i}$ before it encounters and error and it is updated. At the end of the training process, the decision boundary is a weighted sum of the hyperplanes defined by each weight vector, and is defined by the equation:

$$\sum_{i=1}^{k} c_i(\mathbf{w}_i \cdot \mathbf{x} + b_i) = 0 \tag{2}$$

1b. Is the decision boundary of averaged perceptron linear?

Yes, the decision boundary of the averaged perceptron is linear. Averaged perceptron is builds on the standard perceptron by doing a cumulative sum of each weight vector and keeping the count the number of weights that have been evaluated throughout the training process, therefore for a $\mathbf{c}$ number of vectors, the averaged weight is determined by the equation:

$$\mathbf{w}_{average} = \frac{1}{\mathbf{c}} \sum_{i=1}^{c} \mathbf{W}_i \tag{3}$$

Then, the decision boundary is defined by the linear equation:

$$\mathbf{w}_{average} \cdot \mathbf{x} + \mathbf{b} = 0 \tag{4}$$

2. Derivation of the Passive-Aggressive weight update for achieving margin M.

From the Passive-Aggressive (PA) update we saw in class, we can derive a PA weigth update that achieves a custom margin $\mathbf{M}$ we can set the algorithm to solve the dual optimization problem, such that when an update is required:

$$\max_{\tau \geq 0} \quad -\frac{1}{2} \|\mathbf{x}_t\|^2 \tau^2 + \tau(1 - \mathbf{y_t}(\mathbf{w}_t \cdot \mathbf{x}_t)) \tag{5}$$

Therefore, if we set our margin to be $\mathbf{M}$, we can modify the Lagrangian to substitute the 1 for $\mathbf{M}$, which then can be optimized and solved as:

$$\tau = \max \left\{ 0, \frac{M - y_t(w_t \cdot x_t)}{\|x_t\|^2} \right\}$$

After which, the weight w_{t+1} is determined by the linear equation:

$$w_{t+1} = w + \frac{M - y_t(w_t \cdot x_y)}{\|x\|^2} \cdot y_t x_t \tag{6}$$

Which we can simplify as:

$$w_{t+1} = w + \tau y_t x_t \tag{7}$$

3a. Training examples with importance weight: modifying the perceptron algorith to leverage the extra information.

First, let us assume that the importance weights are integer values. In that case, we can leverage the perceptron algorithm by transforming the dataset, such that we present each example i h_i times in the dataset, for example: if we have an example (x_1,y_1,h_1=3), we then represent this example as transformed copies in the dataset as (x_1,y_1), (x_1,y_1), (x_1,y_1). Then, for the implementation of the algorithm, we would just add a preprocessing step, were the dataset is transformed:

Algorithm 1 Dataset Transformation

Require: Original Dataset: $\{(\mathbf{x}_i, y_i, h_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$, $y_i \in \{-1, +1\}$, and $h_i > 0$.

Ensure: Transformed Dataset: $\{(\mathbf{x}_j, y_j)\}_{j=1}^m$, where $m = \sum_{i=1}^n h_i$.

Initialize an empty list for the transformed dataset, $D_{\text{transformed}} \leftarrow \emptyset$.
for $i = 1$ **to** n **do**
  if h_i is an integer **then**
    for $k = 1$ **to** h_i **do**
      Add $(\mathbf{x}_i, y_i)$ to $D_{\text{transformed}}$.
    end for
  end if
end for
Return $\mathcal{D}_{\text{transformed}}$

3b. Training examples with importance weight: solving the problem using the standard perceptron.

Since a reduction-based approach was used where the dataset was transformed, the standard perceptron algorithm can be implemented as discussed in class:

Algorithm 2 Online Binary-Classifier Learning Algorithm

Input: $\mathcal{D}_{\text{transformed}} = \{(\mathbf{x}_i, y_i, h_i)\}_{i=1}^n$, T = maximum number of training iterations
Output: w, the final weight vector
Initialize the weights $\mathbf{w} \leftarrow \mathbf{0}$
for each training iteration $itr \in \{1, 2, \ldots, T\}$ **do**
  for each training example $(\mathbf{x}_t, y_t) \in \mathcal{D}$ **do**
    $\hat{y}_t \leftarrow \text{sign}(w \cdot \mathbf{x}_t)$ {Predict using the current weights}
    if $\hat{y}_t \neq y_t$ **then**
      $w \leftarrow w + \tau y_t \mathbf{x}_t$ {Update the weights}
    end if
  end for
end for
return **w**

4a. Training with imbalanced dataset: Modifying the perceptron algorithm to solve the learning problem.

In this case, the positive examples would be the minority class, which we will call $y_{minority}$. Then, if the goal is to train the model to be more accurate in identifying the positive examples, we can modify the perceptron algorithm such that it penalizes the errors on the minority class, and makes more aggressive updates as a function of the proportion of the minority class. Therefore, for any error in the minority class, we would update the learning rate as defined by the equation:

$$\eta = \frac{\beta}{\rho_{y_i}} \tag{8}$$

Where η is the learning rate, β is a hyperparameter that controls the scaling of the learning rate, which in the standard perceptron is equal to 1, and ρ_{y_i} is the proportion of the class, such that:

$$\eta = \begin{cases} \frac{\beta}{0.9}, & \text{if } y_i \in y_{\text{majority}} \\ \frac{\beta}{0.1}, & \text{if } y_i \in y_{\text{minority}} \end{cases}$$

Finally, the weight update for the perceptron would be defined as:

$$w_{t+1} = w_t + \frac{\beta}{\rho_{y_i}} x_t y_t \tag{9}$$

4b. Training with imbalanced dataset: Solving the problem using the standard perceptron.

To use the standard perceptron algorithm, we can implement a reduction-based solution that consists in applying a Cluster-based undersampling of the majority class [1], and removing the redundant training examples. However, in case that the training examples are unique, we can set a cutoff for the values of distance to the center of the cluster for each member of a cluster. With this approach, the goal is to obtain a balanced dataset where the number of examples per classes are equal.

Then, we can implement it as a pre-processing step in the standard perceptron algorithm.

Algorithm 3 Cluster-Based Undersampling for Class Imbalance

Input:

- Dataset: $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, where $\mathbf{x}_i \in \mathbb{R}^d$, $y_i \in \{-1, +1\}$
- Majority Class Label: y_{maj}
- Number of Clusters: k
- Clustering Algorithm: K-Means

Output:

- Balanced Dataset: $\mathcal{D}_{\text{balanced}}$

Step 1: Separate Majority and Minority Classes
$\mathcal{D}_{\text{maj}} \leftarrow \{(\mathbf{x}_i, y_i) \in \mathcal{D} \mid y_i \in y_{\text{maj}}\}$
$\mathcal{D}_{\text{min}} \leftarrow \{(\mathbf{x}_i, y_i) \in \mathcal{D} \mid y_i \notin y_{\text{maj}}\}$
Step 2: Apply Clustering to Majority Class
$\mathcal{C} \leftarrow \text{Cluster}(\mathcal{D}_{\text{maj}}, k)$ {Apply K-Means clustering}
Step 3: Select Representative Samples from Each Cluster
$\mathcal{D}_{\text{maj}_{\text{selected}}} \leftarrow \emptyset$
for $c \in \mathcal{C}$ **do**
    $\mathbf{x}_{\text{rep}} \leftarrow \text{SelectRepresentative}(c)$ {Select centroid or closest point to centroid}
    $\mathcal{D}_{\text{maj}_{\text{selected}}} \leftarrow \mathcal{D}_{\text{maj}_{\text{selected}}} \cup \{(\mathbf{x}_{\text{rep}}, y_{\text{maj}})\}$
end for
Step 4: Combine Selected Majority Samples with Minority Samples
$\mathcal{D}_{\text{maj}_{\text{selected}}} \leftarrow \mathcal{D}_{\text{maj}_{\text{selected}}} \cup \mathcal{D}_{\text{min}}$
Return $\mathcal{D}_{\text{balanced}}$
Run Standard Perceptron on $\mathcal{D}_{\text{balanced}}$

Programming and Empirical Analysis.

5.1 Binary Classification

5.1a. Online learning curve for Perceptron and PA algorithms.

These curves show that the Perceptron makes less mistakes compared to the PA algorithm from the start. The number of mistakes at iteration 1 is around 3500 for the Perceptron, compared to approximately 4100 mistakes for the PA. At the end of the 50 iterations, the Perceptron was making 2900 mistakes, compared to the 3600 mistakes made by the PA algorithm, representing a better separation of the data achieved by the Perceptron.

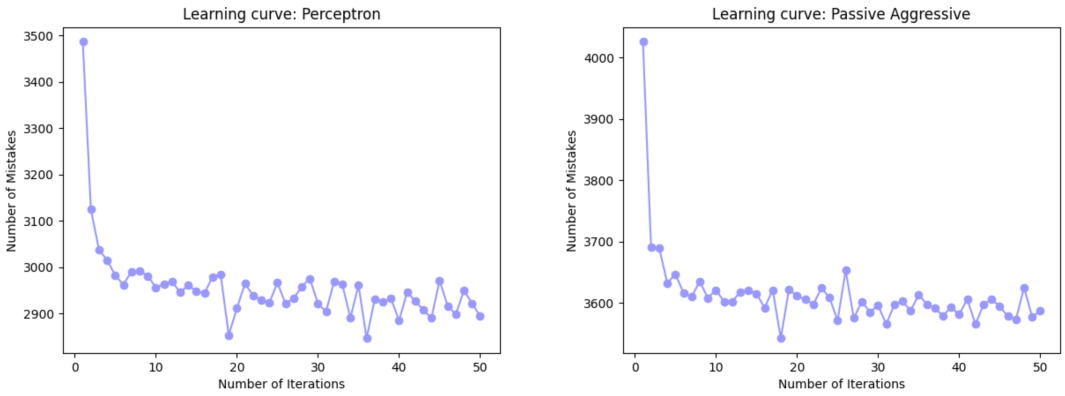

Figure 1: Comparison between the online learning curves of the Perceptron and the Passive-Aggressive algorithms.

5.1b. Accuracy of Perceptron and PA on training and testing data.

The accuracy of the Perceptron was similar to the accuracy of the Passive-Aggressive, however, the PA seems to have converged as evidenced by the small variations on the accuracy for both datasets. On the other hand, the Perceptron algorithm might showed worse generalization in the test data, which could be due to the weight update rule struggling to achieve a margin that confidently separates the data.

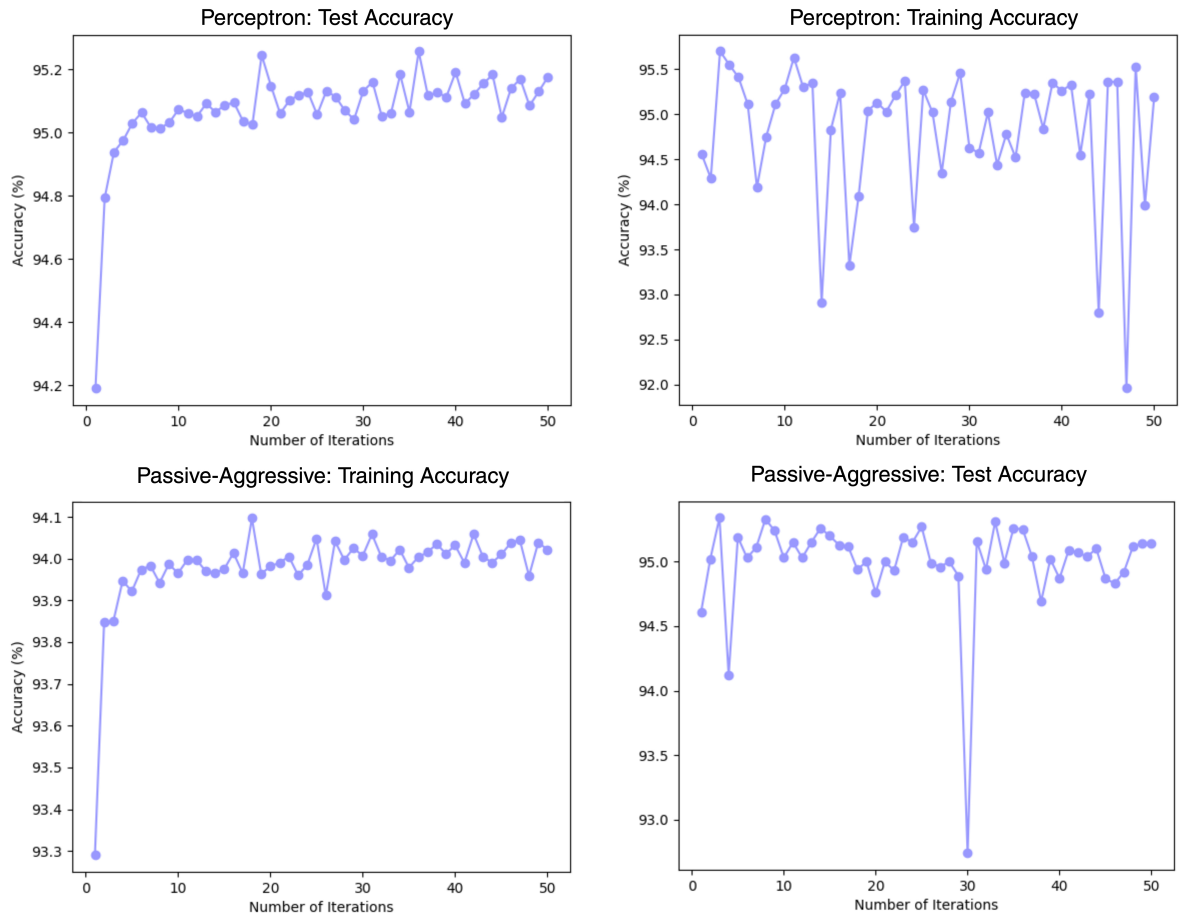

Figure 2: Comparison between the training and testing accuracies of the Perceptron and Passive-Aggresive algorithms.

5.1c. Accuracy of Averaged Perceptron and Plain Perceptron on training and testing data.

At the end of the 20 epochs, the accuracy of the plain perceptron was 93.6 and 91.67 for the training and test data respectively. For averaged Perceptron the accuracies were 95.15, and 96.04, which suggests that averaged Perceptron has better generalization than Plain Perceptron for binary classification of the fashion MNIST data.

5.1d. General learning curves of the Perceptron and the Passive-Aggressive algorithms.

Both algorithms quickly converge to a value around 92% accuracy, which can be interpreted as suggesting that adding more data is not causing the learner to increase its accuracy in the testing data, therefore, it has reached its maximum generalization potential.

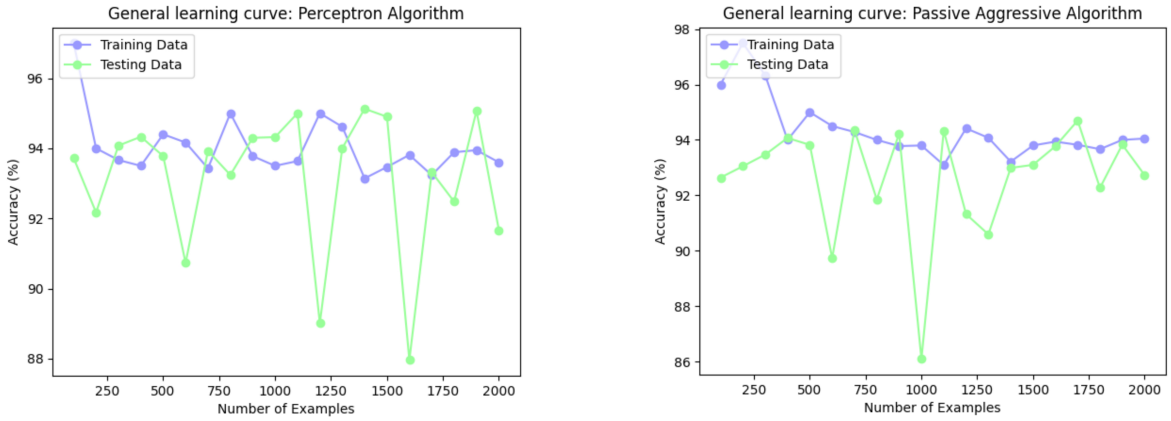

Figure 3: General learning curves for Perceptron and Passive-Aggressive algorithms.

5.2 Multiclass Classification.

5.2a Online learning curve for Multiclass Perceptron and Passive-Aggressive algorithms.

Similar to what was observed for the binary classifier, the standard perceptron makes less mistakes than the Passive-Aggressive, with a final number of mistakes of approximately 11500, compared to PA's 13200 at iteration 50 (see Fig 4). However, both algorithms make more mistakes when learning to classify the multilearn problem, which reflects on these algorithms being more robust at classifying binary data.

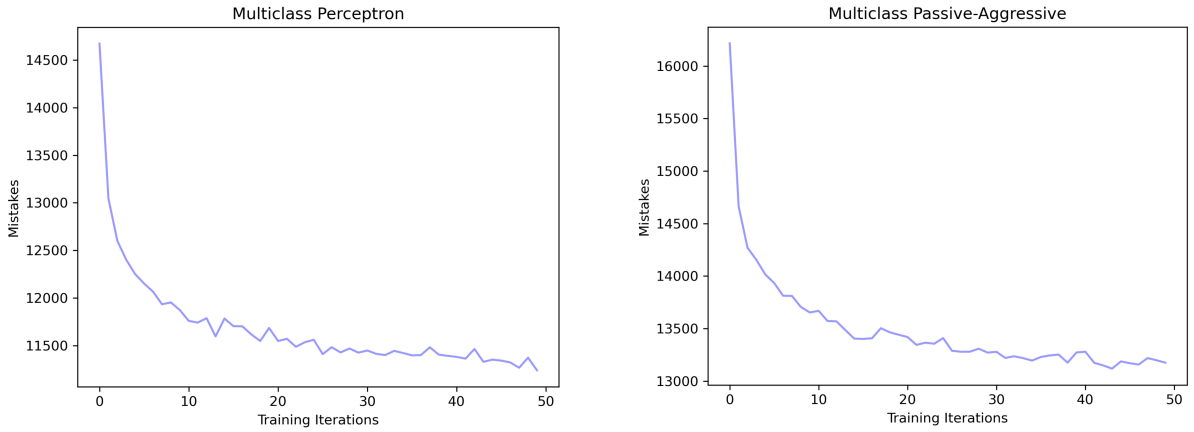

Figure 4: Online learning curve for Multiclass Perceptron and Passive-Aggressive algorithms.

5.2b Accuracy curves for Multiclass Perceptron and Passive-Aggressive algorithms on the training and testing data.

As suggested by the online learning curve, we observe from the curves that the training accuracy converges quickly for both the Perceptron and PA algorithm, however, on the testing data, the accuracy does not converge towards a single value and rather has an amplitude of approximately ± 0.1. Overall, the Perceptron has a higher average accuracy of 0.78 in the testing data, compared to 0.75 for the PA algorithm. Finally, as observed in the online learning curve, both algorithm perform worse for multiclass classification than for binary classification problems.

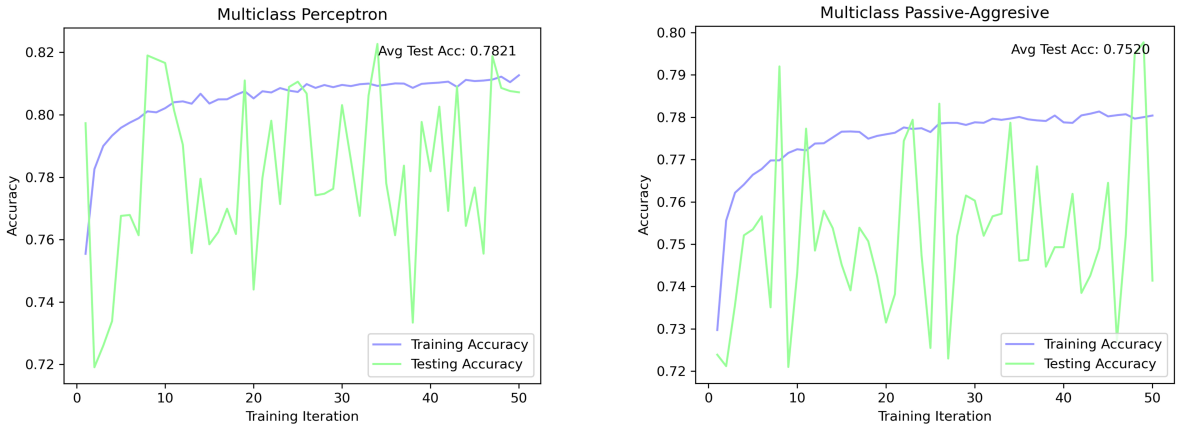

Figure 5: Accuracy curves for Multiclass Perceptron and Passive-Aggressive algorithms on the training and testing data.

5.2c Accuracy curves for the implementation of the averaged Perceptron on the training and testing data.

When implementing the averaged perceptron, the accuracy on the training dataset remained unchanged, due to the the averaged perceptron being essentially a standard perceptron during the training, after which, the averaging of the sum of weights over the number of updates is done and applied in predicting the class of the examples in the test data. In that sense, the average accuracy of the averaged perceptron in the test data was 0.84, which is around 0.06 higher than the average accuracy of the standard perceptron, suggesting that the Averaged Perceptron is able to generalize better than Standard perceptron in this specific classification problem.

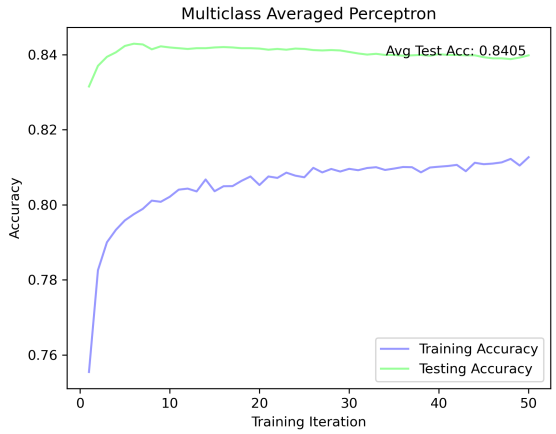

Figure 6: Accuracy curves for Multiclass averaged Perceptron on the training and testing data.

5.2d General learning curve for the Perceptron and Passive-Aggressive algorithms over varying dataset sizes.

For both training algorithms, 5,000 training examples are enough to achieve convergence in the training dataset. However, throughout the iterations, increasing the number of training examples was not correlated with an increase in the testing accuracy. As observed before, the accuracy achieved by the Perceptron was

higher than the PA's accuracy, showing that for this specific classification problem, a dataset comprised of 10,000 training examples and implementing a Standard perceptron algorithm is the better option.

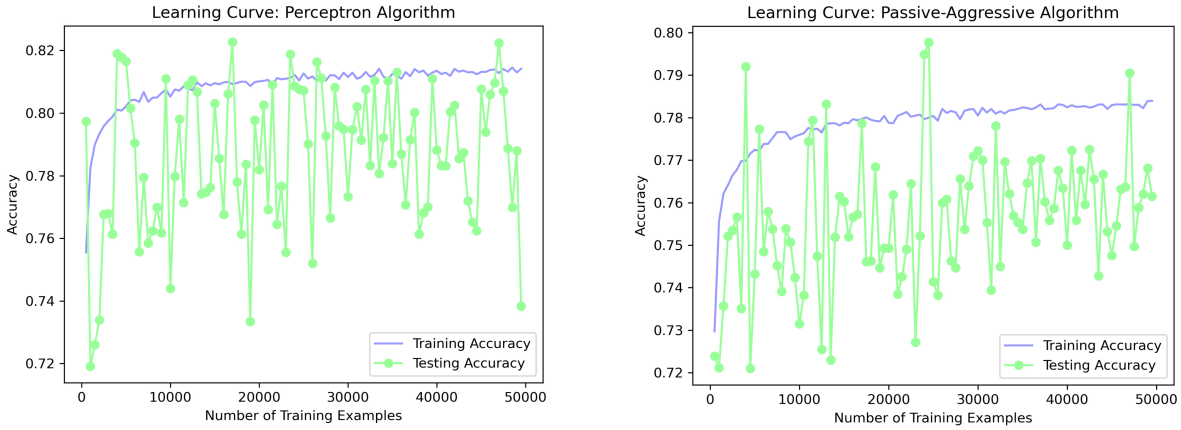

Figure 7: General learning curves for the Perceptron and Passive-Aggressive algorithms over dataset sizes ranging from 500 to 50,000.

References

[1] W.-C. Lin, C.-F. Tsai, Y.-H. Hu, and J.-S. Jhang, "Clustering-based undersampling in class-imbalanced data," *Information Sciences*, vol. 409-410, pp. 17–26, Oct. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025517307235