CptS 570 Assignment 1

Joe Patten

1 Analytical Part

This part will be graded as a PASS or FAIL.

1. Answer the following questions with a yes or no along with proper justification.

   (a) Is the decision boundary of voted perceptron linear?

   The decision boundary of the voted (binary) perceptron is **not** linear. To prove that the decision boundary is nonlinear, we only need to find two points that are classified the same, and that there exists a linear combination of these two points that is of a different classification, then this will be nonlinear.

   Consider, the following weight vectors with their associated c's: $(c_1, \mathbf{w}_1) = (1, (0, 1))$, $(c_2, \mathbf{w}_2) = (1, (1, -1))$, and $(c_3, \mathbf{w}_3) = (1, (-1, -1))$. Now consider the following points to be classified: $(-2, 1)$, $(2, 1)$, and $(0, 1)$. Note that $(0, 1)$ is a linear combination of $(-2, 1)$ and $(2, 1)$. $(-2, 1)$ and $(2, 1)$ have the same label of $+1$, but $(0, 1)$ has the label of -1. Thus, the voted preceptron is not linear.

   (b) Is the decision boundary of averaged perceptron linear?

   The decision boundary of the averaged perceptron is linear. Recall that the decision boundary is defined by $\langle \sum_{i=1}^{m} c_i w_i, \mathbf{x} \rangle$, which is equal to $\sum_{i=1}^{m} c_i w_i x_i$ thus is obviously linear.

2. In the class, we saw the Passive-Aggressive (PA) update that tries to achieve a margin equal to *one* after each update. Derive the PA weight update for achieving margin M.

   Consider a point needing to be classified. Instead of trying to achieve a margin of M instead of 1, then here is how the weight should be updated:

   If $y_t \langle \mathbf{w}_t \mathbf{x}_t \rangle \geq M$:

   No update

   Else:

   $\mathbf{w}_{t+1} = \tau y_t \mathbf{w}_t$

   where $\tau = \frac{M - y_t \langle \mathbf{w}_t \mathbf{x}_t \rangle}{||x_t||}$

   We can summarize this update as: $\mathbf{w}_{t+1} = \tau y_t \mathbf{w}_t$ where $\tau = \max \left\{ 0, \frac{M - y_t \langle \mathbf{w}_t \mathbf{x}_t \rangle}{||x_t||} \right\}$

3. Consider the following setting. You are provided with n training examples: $(x_1, y_1, h_1), \ldots, (x_n, y_n, h_n)$, where x_i is the input example, y_i is the class label ($+1$ or -1), and $h_i > 0$ is the importance weight of the example. The teacher gave you some additional information by specifying the importance of each training example.

   (a) How will you modify the perceptron algorithm to be able to leverage this extra information? Please justify your answer.

   A way we could leverage the extra information (of importance weights) is to make a more "aggressive" update (i.e. give more weight to more important data) for the more important training

data. We could use these weights (the higher weights for the more important data examples) to modify the loss function, and thus modify the update.

(b) How can you solve this learning problem using the standard perceptron algorithm? Please justify your answer. I'm looking for a reduction based solution.

Let $h_i \in \mathbb{R}$ be an importance weight for data $(\mathbf{x}_i, y_i)$, and let $h_{max} = \max_i h_i$. Redefine each importance weight to be $h_i = \frac{h_i}{h_{max}}$. The update becomes $w_{t+1} = w_t + \tau y_i x_i$ where $\tau = h_i$.

Thus the algorithm becomes:

```
// preprocessing step
h_max = max(h_i)
for each training example (x_i, y_i, h_i) ∈ D do
    h_i = h_i / h_max
end for

// perceptron algorithm
w = 0
mistakes = 0
for each training iteration itr ∈ {1, 2, ..., T} do
    for each training example (x_i, y_i, h_i) ∈ D do
        ŷ_i = sign(⟨w, x_i⟩)
        if y_i⟨w, x_i⟩ < 0 then
            τ = h_i
            w = w + τ y_i x_i
            mistakes++
        end if
    end for
end for
return final weight vector w
```

As stated before, some training examples are more important than others. I have altered the update to reflect his. Notice that τ represents the importance weight, and thus will update the decision boundary more aggresively when a training example is more important.

4. Consider the following setting. You are provided with n training examples: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where x_i is the input example, and y_i is the class label ($+1$ or -1). However, the training data is highly imbalanced (say 90% of the examples are negative and 10% of the examples are positive) and we care more about the accuracy of positive examples.

(a) How will you modify the perceptron algorithm to solve this learning problem? Please justify your answer.

Recall that for the original perceptron, we update a decision boundary $\mathbf{w}_t$, but are not concerned with margins. We could introduce uneven margins, i.e. a margin for each class. The margin used for positive cases (i.e. when $y_i = +1$) would be greater than the margin used for negative cases.

(b) How can you solve this learning problem using the standard perceptron algorithm? Please justify your answer. I'm looking for a reduction based solution.

Recall that in the perceptron algorithm, we update the weight vector w_t when $y_i(w_t x_i) < 0$. We can use margins $M_+ \in \mathbb{R}_+$ and $M_- \in \mathbb{R}_+$ instead of 0 (i.e. when $y_i = +1$, then check if $y_i(w_t x_i) < M_+$). I would assume that we would want $M_+ > M_-$, and would want $M_+/M_- = 9$.

// preprocessing step
$M_+ = 1$
$M_- = \frac{M_+}{9}$
for each training example $y_i \in D$ **do**
  if $y_i == 1$ **then**
    $M_i = M_+$
  else
    $M_i = M_-$
  end if
end for

// passive aggressive algorithm
$w = 0$
$mistakes = 0$
for each training iteration $itr \in \{1, 2, \ldots, T\}$ **do**
  for each training example $(x_i, y_i) \in D$ **do**
    if $y_i \langle w, x_i \rangle < M_i$ **then**
      $\tau = 1$
      $w = w + \tau y_i x_i$
      $mistakes{+}{+}$
    end if
  end for
end for
return final weight vector w

Instead of using the condition $y_i \langle w, x_i \rangle < 0$, we now use the condition $y_i \langle w, x_i \rangle < M_i$. This will introduce uneven margins as desired (thanks to our preprocessing step). Now $M_i \in \{M_-, M_+\}$.

2 Programming and Empirical Analysis Part

5.1. **Binary Classification (40 points) Learn a binary classifier to classify even labels (0, 2, 4, 6, 8) and odd labels (1, 3, 5, 7, 9).**

(a) Compute the online learning curve for both Perceptron and PA algorithm by plotting the number of training iterations (1 to 50) on the x-axis and the number of mistakes on the y-axis. Compare the two curves and list your observations.

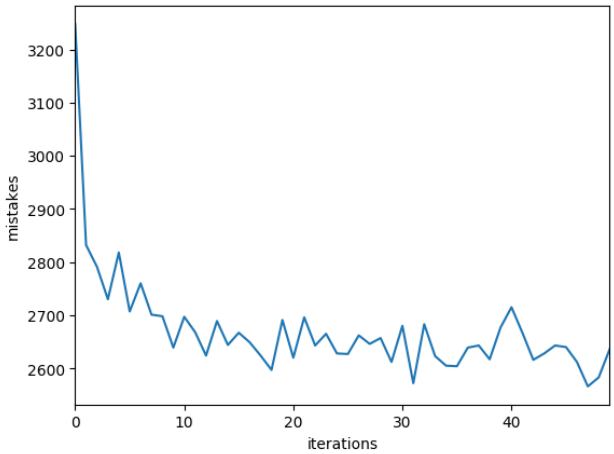

Figure 1: The plot shows the relationship between the number of mistakes encountered using the perceptron algorithm in an iteration (y axis), and the iteration number (x axis).

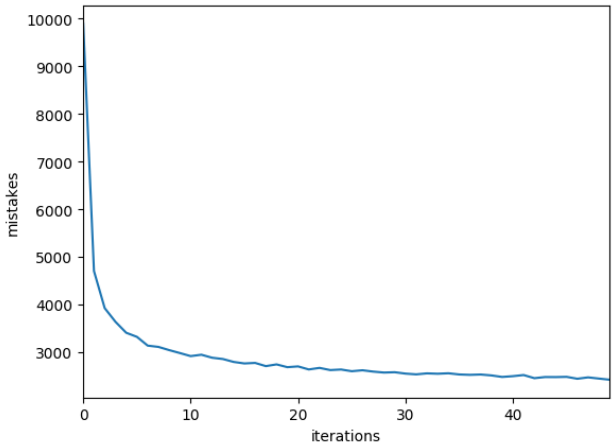

Figure 2: The plot shows the relationship between the number of mistakes encountered using the passive aggressive algorithm in an iteration (y axis), and the iteration number (x axis).

The perceptron learning curve (shown in figure 1) looks more "jagged" than the passive aggressive learning curve (shown in figure 2). This is to be expected as the passive aggressive algorithm changes the τ (using hinge loss) depending on how close to the correct margin (if it is on the wrong side of the margin) it is whereas the perceptron learning curve has a constant $\tau = 1$. Thus the passive aggressive algorithm updates the decision boundary according to how far away a training example is, whereas the perceptron algorithm updates the decision boundary if there is a mistake with the same weight each time. Therefore, we would expect the perceptron algorithm to overcorrect when a training example is barely a mistake (and close to the margin), and undercorrect when it is far away from the margin.

(b) Compute the accuracy of both Perceptron and PA algorithm on the training data and testing data for 20 training iterations. So you will have two accuracy curves for Perceptron and another two accuracy curves for PA algorithm. Compare the four curves and list your observations.

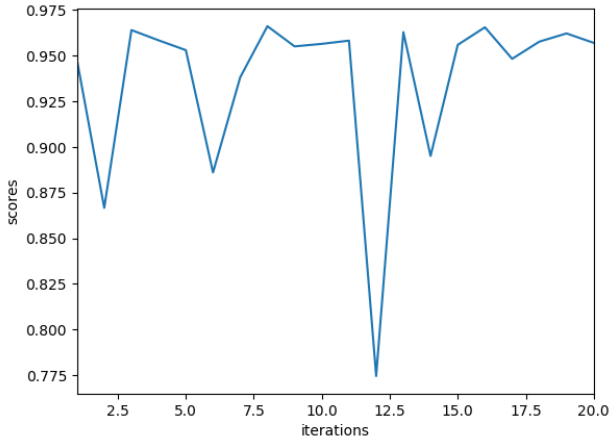

Figure 3: This plot shows the relationship between the accuracy of the perceptron algorithm based on the training examples (y axis), and the iteration number (x axis).

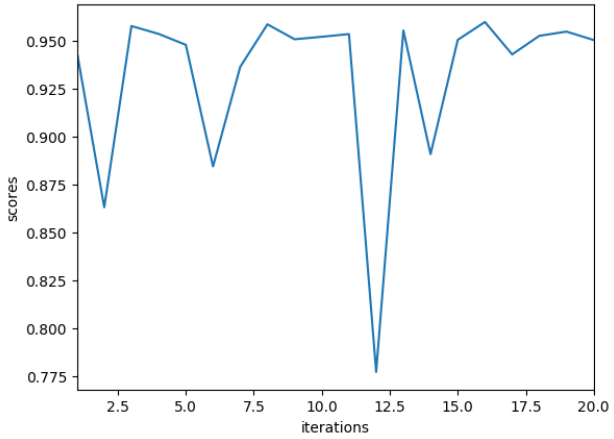

Figure 4: This plot shows the relationship between the accuracy of the perceptron algorithm based on the testing examples (y axis), and the iteration number (x axis).

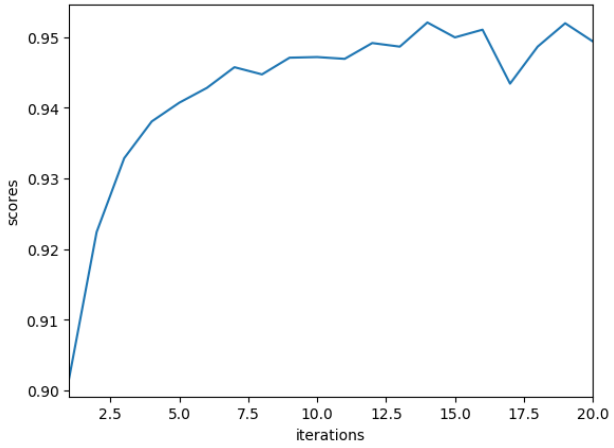

Figure 5: This plot shows the relationship between the accuracy of the passive aggressive algorithm based on the training examples (y axis), and the iteration number (x axis).

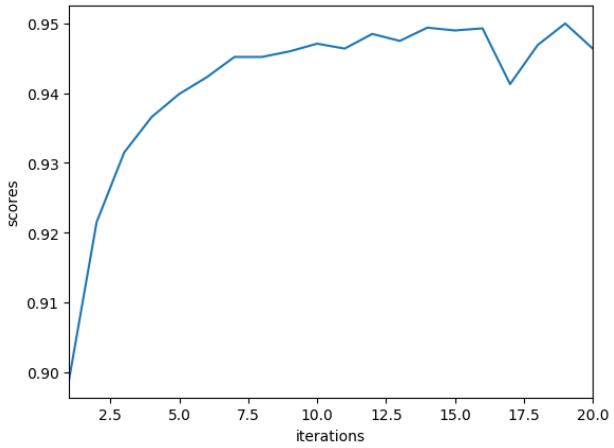

Figure 6: This plot shows the relationship between the accuracy of the passive aggressive algorithm based on the testing examples (y axis), and the iteration number (x axis).

The perceptron algorithm updates the decision boundary when it sees a mistake by the same weight each time. Thus, as we see in both the training (figure 3) and testing plots (figure 4), the accuracy after each iteration is a bit erratic.

The passive aggressive algorithm uses the loss to update the decision boundary, thus, we see that the accuracy, generally speaking, gets better little by little after each iteration. So these curves (figures 5 and 6) are less erratic.

The training and testing plots (for both the perceptron and passive algorithms) are quite similar. This might be due to the fact that the training and testing examples are quite similar. Also, since there are so many training examples (and the examples are sorted randomly in my algorithms), then one iteration over the examples will give us fairly good accuracy. I honestly would have expected the accuracy scores for training plots to increase over iterations, and the accuracy scores for testing plots to increase, and then decrease after a certain number iterations, as would be the case with overfitting.

(c) Repeat experiment (b) with averaged perceptron. Compare the test accuracies of plain perceptron and averaged perceptron. What did you observe?

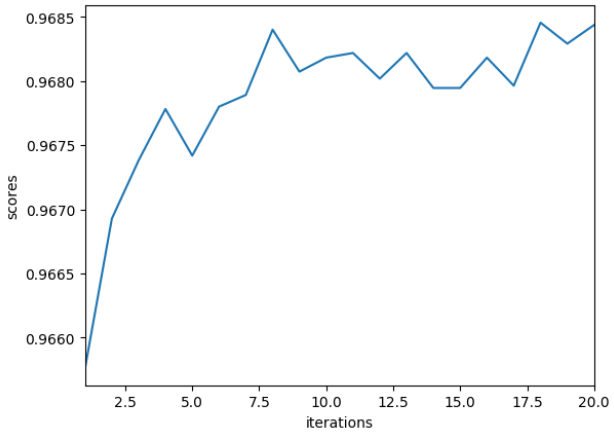

Figure 7: This plot shows the relationship between the accuracy scores of the averaged perceptron algorithm based on the training examples (y axis), and the iteration number (x axis).

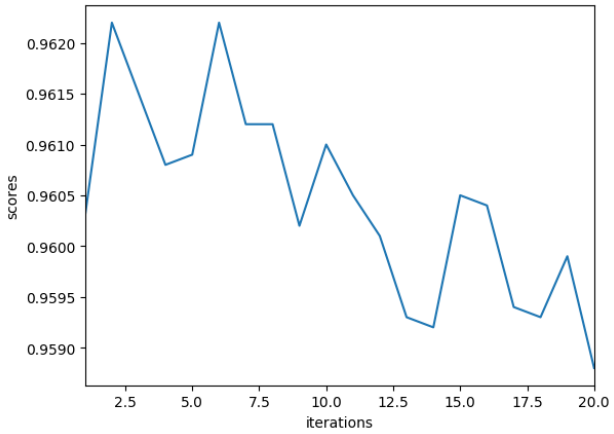

Figure 8: This plot shows the relationship between the accuracy scores of the averaged perceptron algorithm based on the testing examples (y axis), and the iteration number (x axis).

The accuracy scores for the averaged perceptron algorithm on the training examples (as shown in figure 7) and testing examples (as shown in figure 8) look a bit different at first, but notice that the accuracy for both graphs lie in such a small range. Comparing this with the perceptron accuracies, we see that the scores are less erratic. This is due to the case that we are dampening each change to the decision boundary since we are taking the average of each boundary decision. Also, the training plot shows an increase in accuracy score as iterations increase, but in our testing set, accuracy scores decrease after a certain amount of iterations. This might be due to the model overfitting.

(d) Compute the general learning curve (vary the number of training examples starting from 5000 in the increments of 5000) for 20 training iterations. Plot the number of training examples on x-axis and the testing accuracy on the y-axis. List your observations from this curve.

As shown in figure 9, there does not seem to any relationship between accuracy score and observation size. Thus, when there are two classes, observation size might not be that important (the difference between having 5,000 observations and say 30,000 observations in our training set would not affect our accuracy that much. Having 5,000 observations seems to be sufficient).

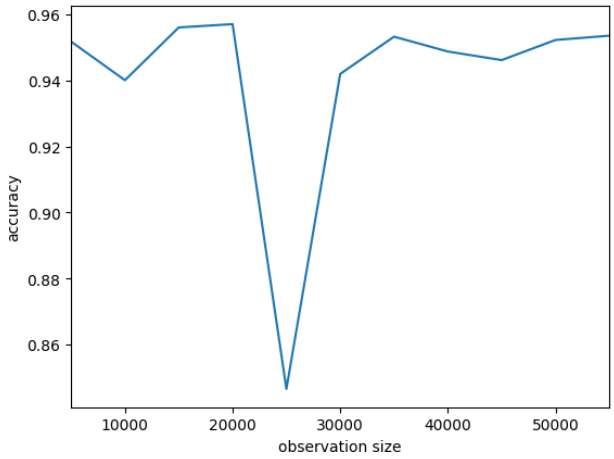

Figure 9: This plot shows the relationship between the accuracy scores (y axis) of the perceptron algorithm after 20 iterations and the observation size (x axis).

5.2. **Multi-Class Classification (40 points) Learn a multi-class classifier to map images to the corresponding fashion label.**

(a) Compute the online learning curve for both Perceptron and PA algorithm by plotting the number of training iterations (1 to 50) on the x-axis and the number of mistakes on the y-axis. Compare the two curves and list your observations.

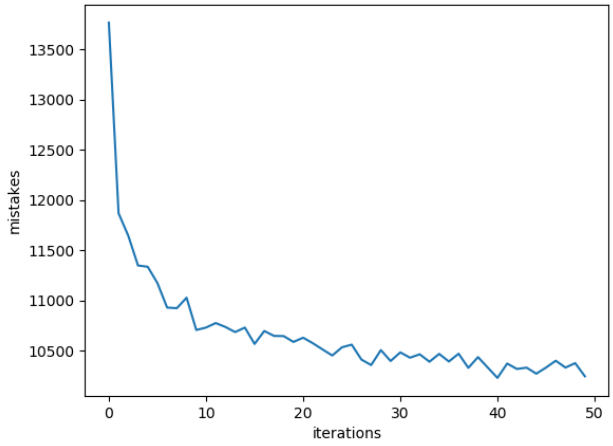

Figure 10: The plot shows the relationship between the number of mistakes encountered using the perceptron algorithm in an iteration (y axis), and the iteration number (x axis).

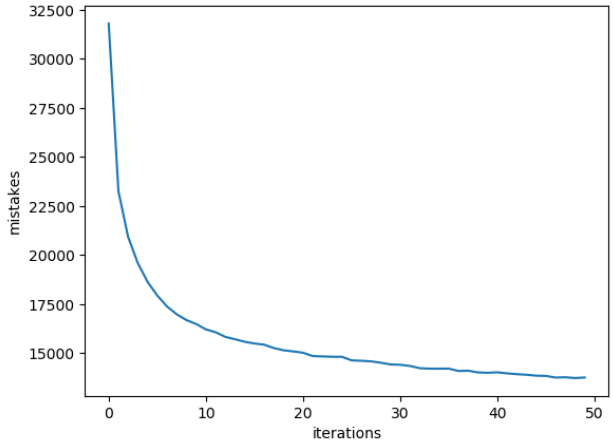

Figure 11: The plot shows the relationship between the number of mistakes encountered using the passive aggressive algorithm in an iteration (y axis), and the iteration number (x axis).

Again, the perceptron learning curve (shown in figure 10) looks more "jagged" than the passive aggressive learning curve (shown in figure 11), although it appears less jagged than its binary counterpart. Again, this is to be expected as the passive aggressive algorithm changes the τ (using hinge loss) depending on how close to the correct margin (if it is on the wrong side of the margin) it is whereas the perceptron learning curve has a constant $\tau = 1$. Thus, we would expect the perceptron algorithm to overcorrect when a training example is barely a mistake (and close to the margin), and undercorrect when it is far away from the margin.

Comparing these plots with their binary counterparts, you will notice that we are making more mistakes. This is to be expected as there are more classes.

(b) Compute the accuracy of both Perceptron and PA algorithm on the training data and testing data for 20 training iterations. So you will have two accuracy curves for Perceptron and another two accuracy curves for PA algorithm. Compare the four curves and list your observations.

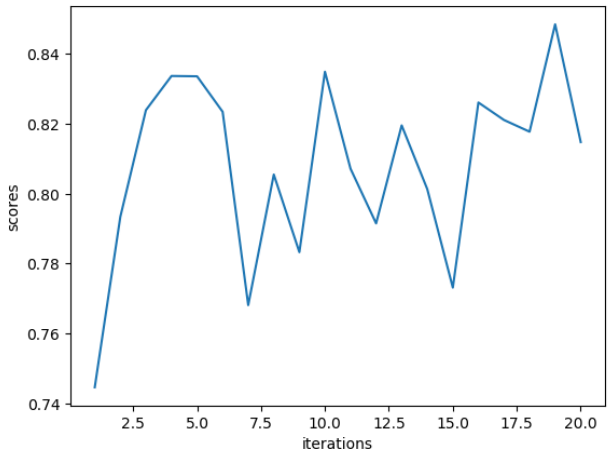

Figure 12: This plot shows the relationship between the accuracy of the perceptron algorithm based on the training examples (y axis), and the iteration number (x axis).

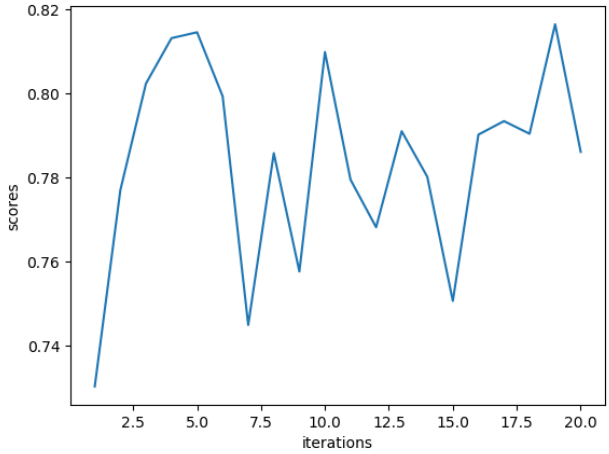

Figure 13: This plot shows the relationship between the accuracy of the perceptron algorithm based on the testing examples (y axis), and the iteration number (x axis).

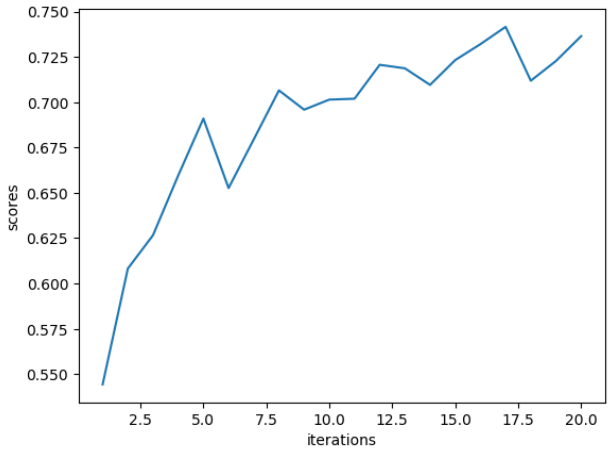

Figure 14: This plot shows the relationship between the accuracy of the passive aggressive algorithm based on the training examples (y axis), and the iteration number (x axis).

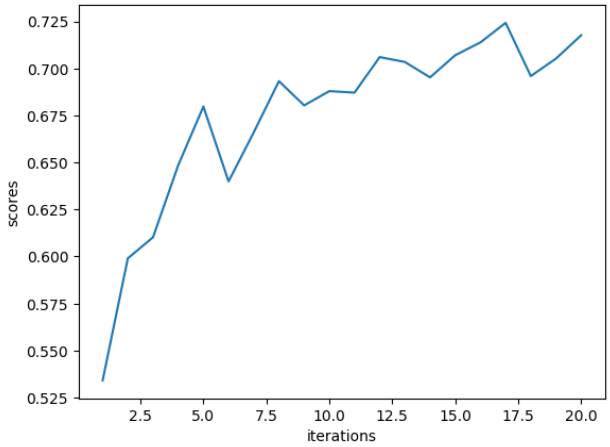

Figure 15: This plot shows the relationship between the accuracy of the passive aggressive algorithm based on the testing examples (y axis), and the iteration number (x axis).

As stated in part a, the perceptron algorithm updates the decision boundary when it sees a mistake by the same weight each time. Thus, as we see in both the training (figure 12) and testing (figure 13) plots, we see that the accuracy after each iteration is a bit erratic (similar to what we saw in the binary case). As opposed to the binary case, there seems to be a positive relationship between accuracy and iterations (as it may take more iterations to correctly classify when there are more than 2 classes).

The training (figure 14) and test (figure 15) plots for the passive aggressive algorithm show bigger improvements (when compared with their binary counterparts) in accuracy scores each iteration and are less erratic when compared to the perceptron plots.

Like we saw in binary classification, the training and testing plots (for both the perceptron and passive algorithms) are quite similar. We would expect the accuracy scores for the training plots to increase in number of iterations, and accuracy scores for the test plots to decrease after a number of iterations as would be the case when overfitting is in play.

(c) Repeat experiment (b) with averaged perceptron. Compare the test accuracies of plain perceptron and averaged perceptron. What did you observe?

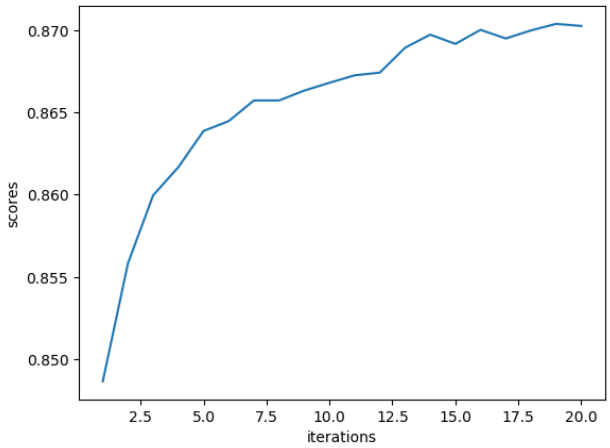

Figure 16: This plot shows the relationship between the accuracy scores of the averaged perceptron algorithm based on the training examples (y axis), and the iteration number (x axis).

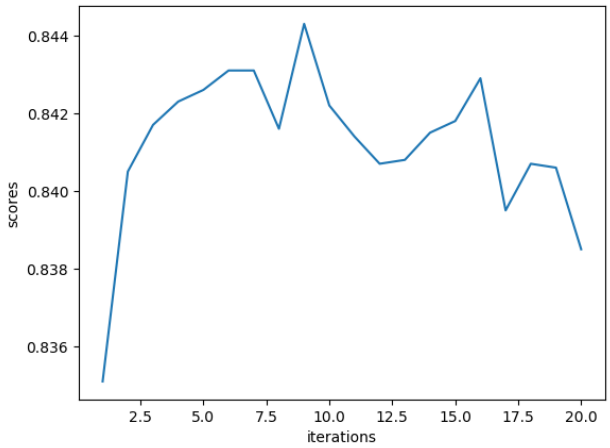

Figure 17: This plot shows the relationship between the accuracy scores of the averaged perceptron algorithm based on the testing examples (y axis), and the iteration number (x axis).

As we saw in the binary classification, the accuracy scores for the averaged perceptron algorithm on the training (figure 16) and testing (figure 17) examples look a bit different at first, but both graphs lie in such a small range. Comparing this with the perceptron accuracies, we see that the change in scores are less erratic. This is due to the case that we are dampening each change to the decision boundary since we are taking the average of each boundary. Also, the training plot shows an increase in accuracy score as iterations increase, but in our testing set, accuracy scores decrease after a certain amount of iterations. This might be due to some overfitting.

(d) Compute the general learning curve (vary the number of training examples starting from 5000 in the increments of 5000) for 20 training iterations. Plot the number of training examples on x-axis and the testing accuracy on the y-axis. List your observations from this curve.

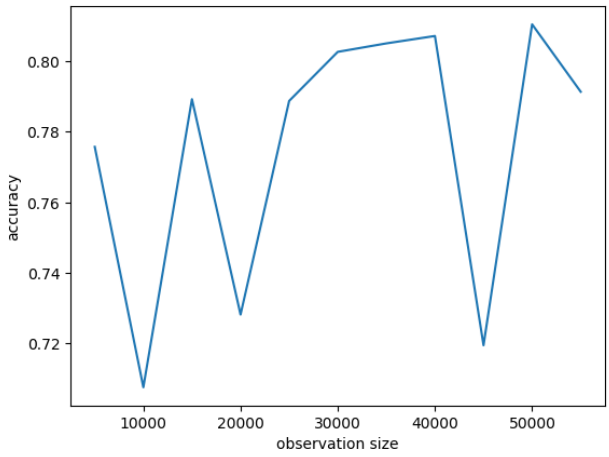

Figure 18: This plot shows the relationship between the accuracy scores (y axis) of the perceptron algorithm after 20 iterations and the observation size (x axis).

As shown in figure 18, there might be a (small) positive relationship between accuracy and observation size (although like the binary case, the change is a bit erratic). This is different than what we saw from the binary classification, as now we have more classes (10 to be exact). Thus we might require more data to more accuratly train our algorithm.