HW 2

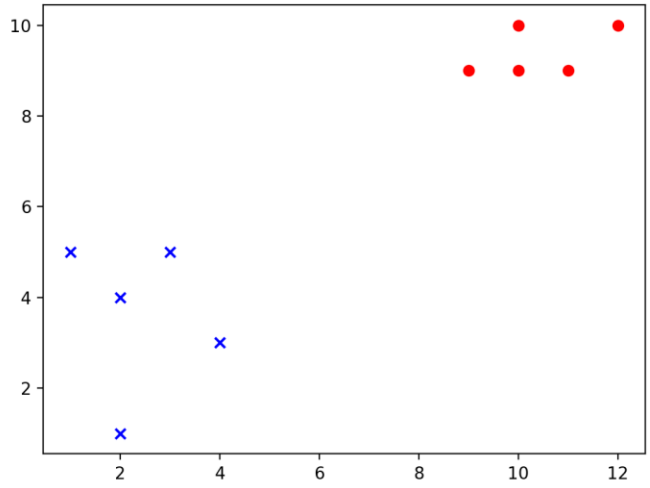
Elias Crum

1.

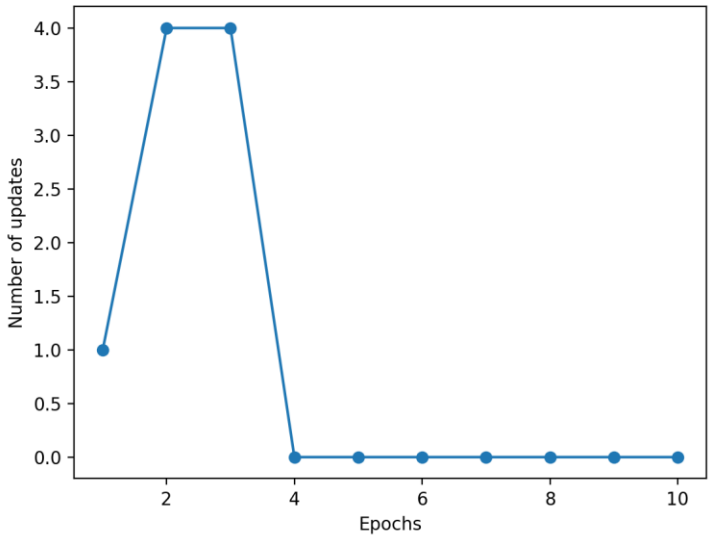
I laid out my perceptron algorithm in a similar way to the textbook in that I made a Perceptron class with the parameters eta (learning rate) and niter (the number of iterations). There are two methods that allow for weight updates and weight testing. The fit() method iterates through the rows of the dataset and calls the predict() method. The predict() method calculates the dot product of the weight array (which is initialized to be 0s for each feature) and returns the label (1 or -1) depending on the relation of the dot product to the threshold (which is initialized as 0 and is the first item in the weight array). My predict() method is different to the book's algorithm in that it is a single method that does the dot product calculation and label prediction within one method rather than two. The label returned from the predict() method is then subtracted from the actual label within the input dataset and multiplied by eta. This is the same as the books algorithm as it is the central step of the perceptron model. The result is 0 if the labels match but is != 0 if the labels do not match. The result is then multiplied by the row values to yield the delta w value in. the same way as the book. The weight array and the threshold value are then updated in a similar way to the book. This process is iterated for the designated number of iterations in the same way as the books algorithm. The main difference between my algorithm and the books algoritm is how I implemented the row iterations, the predict() method, and the way I compared the prediction to the actual label. Conceptually the books algorithm and my algoritm are almost the same (likely because I read the chapter before I built my algorithm).

2.

Linearly separable dataset graphical display:



Linearly separable dataset error tracking after each iteration:

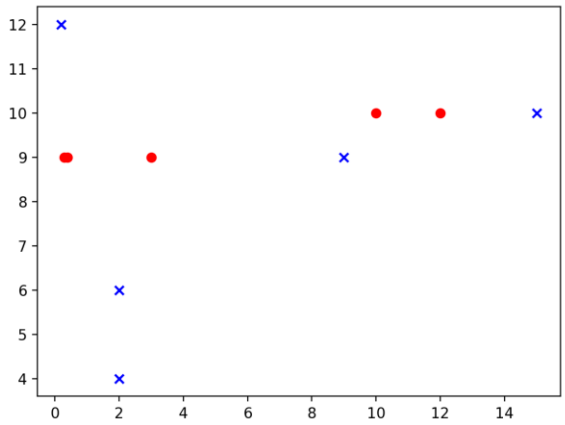


As seen in the image above, the perceptron algorithm found a decision boundary after 3 iterations because there was 0 error during that iteration of training. Thus, the data was linearly separable and the weights converged to values that allowed for correct classification of the training data.

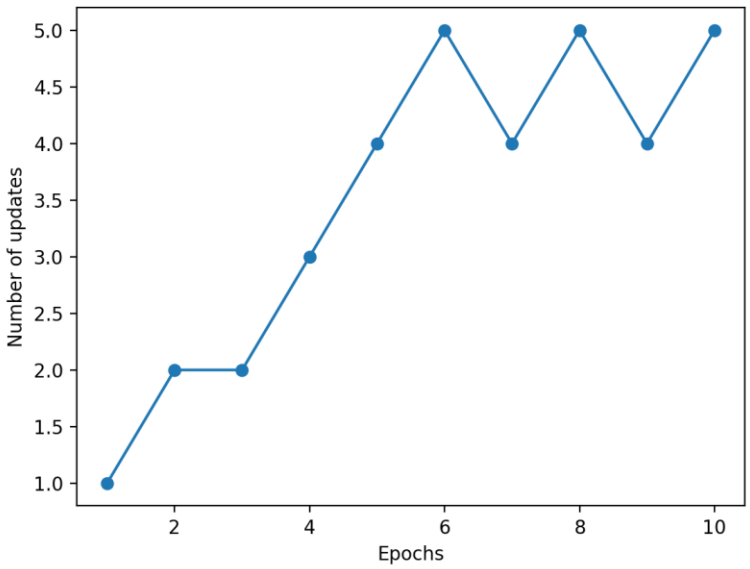
Linearly separable dataset error (on final iteration): 0.0 – 100% accuracy.

The error was 0% after the final iteration showing that the algorithm was able to apply weights that allowed for complete separation of the data based on the feature values. Essentially, because the dataset was linearly separable, the algorithm was able to accurately classify everything via weight changes after 3 iterations.

Non-linearly separable dataset graphical display:



Non-linearly separable dataset error tracking after each iteration:

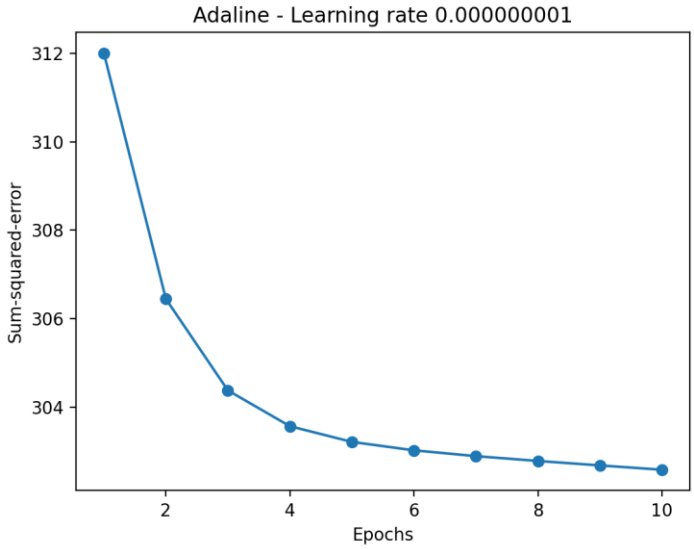


Non-linearly separable dataset error (on final iteration): 0.5 accuracy.

The algorithm did not converge. As is seen in the image, the error increases with each iteration then oscillates at high error values showing there is no convergence.

The error was 50% after the final iteration showing that the algorithm was not able to apply weights that allowed for complete separation of the data based on the feature values. Essentially, because the dataset was not linearly separable, the algorithm was not able to accurately classify everything via weight changes after 10 iterations. Because the algorithm could not find a way to change the weights to achieve a lower error rate, the weight changes instead kept increasing the error of the algorithm each iteration.

3.



Adaline performance on test data: 0.663 accuracy (using batch version)

What were the most predictive features of your Titanic model? Provide evidence.
Most predictive:

4.

The most predictive features of the Titanic dataset within the Adaline algorithm cannot be determined easily. Logically, it would make sense to look at the final weight values and assume the smaller (closer to 0) the weight value, the less predictive that feature is, as it will factor less into the final activation value for a given row. But, when the weight changes are made, the error is multiplied by the values of a specific row (as in the formula for the weight adjustment, J). Thus, the feature weights cannot be compared to each other as though they are representative of the importance of a given feature but rather are a representation of the how the Adaline algorithm is descending the gradient, a practically useless value for logical interpretation. Therefore, we cannot determine the most predictive features based on the information provided by the Adaline algorithm, instead I attempted to see if there was a change in model accuracy with the dropping of features. With all features present, accuracy was 66%. Without the features sex, PassengerID and Name, the accuracy was reduced slightly to 65%. When dropping the features Sex, PassengerID, Name, Age, Pclass, SibSp, Fare, Cabin, Embarked, and Parch the accuracy remained at 65% suggesting that these features are not important in predictions. By contrast, the accuracy decreased to 35% when Ticket was dropped along with Sex, PassengerID, Name, Age, Pclass, SibSp, and Fare, suggesting Ticket is one of the most predictive features. Interestingly, when only Ticket was dropped there was no drop in accuracy suggesting that there is a complex relationship between features that contributes to the prediction for each passenger. Following this rudimentary analysis of feature importance, I found that the importance of the features cannot be well established without more sophisticated

methods of analysis.

5.

Baseline Model: Did semi-poorly poorly: 40% accurate predictions
Perceptron Model: Did very poorly: 33% accuracy (likely due to non-linearly separable data)
Adaline Model: Did better: 66.3% accuracy

The Adaline model performed the best, and substantially better than the baseline which is no surprise because of its use of gradient descent (especially with the 10 features present). The perceptron algorithm performed poorer than the baseline likely because the titanic dataset is not linearly separable.