ML - HW3

Luke Schreiber

November 2023

1 Paper Problems

1.1

1.1.1 a

We know that a hyperplane cannot have a margin if the function is not linearly seperable. However, it is clear that the function is linearly separable. A perceptron is gauranteed to converge if the data is linearly separable. On desmos graph this is made clear with purple dots representing positive labels and green representing negative labels (*(Fig. 1)*. To find the margin of the dataset, one must find the minimum distance any point in the dataset and the hyperplane. To do this, we can use the equation described in class of

$$\gamma = min \frac{y_i(w^T x_i + b)}{||w||}$$

written in terms of the distance from a single data point to the hyperplane, this equation is

$$\frac{|b + w_1 x_1 + w_2 x_2|}{\sqrt{w_1^2 + w_2^2}}$$

When we find the distance for each data point, we find the minimum of these distances to be roughly *Margin = 0.28*. (From the point 1,1).

1.1.2 b

We first must inspect that the dataset 2 is linearly separable. It appears when I graph the data that the last data point contradicts the other signs of the surrounding datapoints. This means the dataset is not linearly separable by the defined hyperplane. Therefore, the margin DNE. A hyperplane margin cannot exist on a dataset that is not linearly seperable by the defined hyperplane.

1.2

1.2.1 a

The margin of a hyperplane is the distance between the hyperplane and the closest data points from either class. The margin for a dataset is the smallest possible margin of any hyperplane that separates the dataset. The margin of a hyperplane is a measure of how well a specific hyperplane separates the data. The margin for a dataset is a measure of how well any possible hyperplane can separate the data. To calculate the margin of the dataset, we must first find the support vectors, then compute the distance between the hyperplane and the support vectors. In python, this is done by loading the data as an np.array, then fitting an SVC (Support Vector Classification) to the data by clf.fit(data, y). Then we compute the margin using margin = clf.support vectors of 0 - clf.support vectors of 1. For this dataset the margin vector output was [-1, 1, 0] whose norm (length) is square root of two, or 1.141.

1.2.2 b

We first must inspect that the dataset 2 is linearly separable. It appears when I graph the data that the last data point contradicts the other signs of the surrounding datapoints. This means the dataset is not linearly separable by any hyperplane. There is no 2d straight line that can seperate the colors of these dots. Therefore, the margin DNE. A hyperplane margin cannot exist on a dataset that is not linearly seperable.

1.3

It stands to reason that the upperbound of the number of mistakes made by the perceptron algorithm is still $\frac{R^2}{\gamma^2}$. The original assumption states that there exists a unit vector u such that for some margin, we have $y_i(u^T x_i) >= margin$ for all i. The modified assumption only states that there exists a vector u such that for some margin, we have $y_i(u > x_i) >= margin$for all i.

The modified assumption is weaker than the original assumption because the original assumption implies the modified assumption. This is because if u is a unit vector, then u ¿ xi is equivalent to u^T xi ¿= 0. Therefore, if the original assumption holds, then the modified assumption must also hold.

Since the modified assumption is weaker than the original assumption, it follows that the upper bound.

1.4

A perceptron will have a number of mistakes less than or equal to $\frac{R^2}{\gamma^2}$. This is true when the data has a margin γ and every point inside a circle of radius R. Here, the radius of the ball that contains all of the training data is $\sqrt{2n}$ because that is the max distance between any two datapoints, and the margin of the

hyperplane is at least 1. Therefore, the upperbound for the number of mistakes of the perceptron is $\frac{\sqrt{2n}^2}{1^2}$ which simplifies to 2n.

1.5

A linear classifier that exists in a plane can shatter any 3 distinct points but not any four distinct founds. The linear classifier can only divide the plane into two half spaces. There are 16 total ways to label 4 distinct points, so there must be two labelings that cannot be achieved by any linear classifier.

Proof: Let there exist a linear classifier that can shatter any 4 distinct points on a plane. This means there must be a line that can divide the plane into two half spaces such that a labeling of the four points can be made by choosing the half-space that contains the points labeled positive and excluding the half-space with negative labels. There are 16 ways to label 4 distinct points. For each labeling for these points, we construct a line that divides the plane into two half-spaces so each labeling can occur. There are only 8 ways to divide the plane into two half spaces with a line, however. Therefore there are at least two labels that are not achieved by any possible line. Therefore, there is no linear classifier that can shatter 4 distinct points.

1.6

The VC dimension of a hypothesis space is the largest number of points that can be shattered by H. When rectangles are in a plane, the VC dimension is four because there are four points that can be shattered by rectangles. A set of points is only shattered by a hypothesis space when there is a hypothesis in that space which classifies each point as positive or negative. There are many examples of sets of four points where rectangles at similar verticies can classify points as positive and negative, and can be shattered.

2 Practice

2.1

I updated my machine learning library, added a README.md, and added a new folder titled perceptron.

2.2

2.2.1

After implimenting the standard perceptron in python with T = 10 epochs, the learned weight vector was [4102.605, 11291.479 8154.935, -5277.336]. The average prediction error for the test dataset was 0.724, which is obviously a lot of error.

2.2.2

After implimenting the voted perceptron in python with T = 10 epochs, the learned weight vectors and their counts is [-20.578034 -17.529043 -9.164973 -16.836971] 1 [-18.8312856 -14.068138 -17.351032 -11.6960836] 1 [-17.1234107 -14.812823 -15.603061 -11.2288112] 1 [-16.5776125 -14.256978 -13.582958 -9.2649804] 1 [-16.2773989 -14.294888 -15.962774 -11.4624434] 1 [-15.7728758 -7.847718 -14.97832 -9.2909102] 1 [-14.8279922 -12.990393 -15.833349 -11.8765132] 1 [-14.2749291 -14.859943 -13.995755 -13.8128836] 1 [-13.2754734 -9.654928 -11.70373 -12.1575434] 1 [-8.9926167 -5.258883 -8.381915 -9.1471432] 1. The average prediction error for the test dataset was 0.658, which is an improvement but still too much error.

2.2.3

My learned weight vector for the average perceptron is [-15811.70503946 -11401.3072864 -7556.1951926 -12690.15358546]. The test error is only 0.074. Comparing with the previous weight vectors it is obvious that this weight vector just used averages to get to its final iteration.

2.2.4

Ultimately, the average prediciton error for the averaged perceptron was the best method because it is the *most efficient, simplest, and yeilded the least errors.* The standard perceptron was not the best because it only updated one weight vector and did not keep track of others. The voted vector was inefficient as well, but better than standard. Voted perceptrons are typically more accurate than standard perceptrons. This is because they take into account the votes of other perceptrons and only update the weight vector if the majority of the perceptrons disagree with the current weight vector. Averaged perceptrons are typically the most accurate of the three algorithms. This is because they take into account the average of the outputs of all perceptrons and update the weight vector in the direction of the average output. My averaged perceptron was a lot more accurate than the others here.

The way in which weights are voted and averaged have the greatest effect on the accuracy of the perceptron according to my accuracy of prediction outcomes. Perceptrons are well suited for problems where the data is linearly seperable. With this data it is hard to tell. When I modified the T parameter the prediction accuracy sometimes got worse which could be evidence of overfitting.