CS 5350/6350: Machine Learning Fall 2023

Homework 3

Handed out: 24 Oct, 2023
Due date: 11:59pm, 7 Nov, 2023

- You are welcome to talk to other members of the class about the homework. I am more concerned that you understand the underlying concepts. However, you should write down your own solution. Please keep the class collaboration policy in mind.
- Feel free to discuss the homework with the instructor or the TAs.
- Your written solutions should be brief and clear. You do not need to include original problem descriptions in your solutions. You need to show your work, not just the final answer, but you do *not* need to write it in gory detail. Your assignment should be **no more than 15 pages**. Every extra page will cost a point.
- Handwritten solutions will not be accepted.
- *Your code should run on the CADE machines.* **You should include a shell script, `run.sh`, that will execute your code in the CADE environment. Your code should produce similar output to what you include in your report.**

  You are responsible for ensuring that the grader can execute the code using only the included script. If you are using an esoteric programming language, you should make sure that its runtime is available on CADE.
- Please do not hand in binary files! We will *not* grade binary submissions.
- The homework is due by **midnight of the due date**. Please submit the homework on Canvas.

1 Paper Problems [36 points + 15 bonus]

1. [8 points] Suppose we have a linear classifier for 2 dimensional features. The classification boundary, i.e., the hyperplane is $2x_1 + 3x_2 - 4 = 0$ (x_1 and x_2 are the two input features).

x_1	x_2	label
1	1	1
1	-1	-1
0	0	-1
-1	3	1

Table 1: Dataset 1

(a) [4 points] Now we have a dataset in Table 1. Does the hyperplane have a margin for the dataset? If yes, what is the margin? Please use the formula we discussed in the class to compute. If no, why? (Hint: when can a hyperplane have a margin?)

x_1	x_2	label
1	1	1
1	-1	-1
0	0	-1
-1	3	1
-1	-1	1

Table 2: Dataset 2

(b) [4 points] We have a second dataset in Table 2. Does the hyperplane have a margin for the dataset? If yes, what is the margin? If no, why?

Solution. A hyperplane can have a margin if it perfectly separates the two classes in the dataset. The margin is defined as the distance from the hyperplane to the nearest point from either class. We should be using the formula for general hyperplane h: $w^T x + b = 0$, to check whether the hyperplane $2x_1 + 3x_2 - 4 = 0$ has a margin for the datasets in both tables. We can calculate the distance of each point in the datasets to the hyperplane $2x_1 + 3x_2 - 4 = 0$, $dist(x_0, h) = |w^T x_0 + b|/||w||$, where weight vector $w = [2, 3]$, and bias term $b = 4$. So, we calculate the distance from a point (x_1, x_2) to the hyperplane as $distance = \frac{|2x_1+3x_2-4|}{\sqrt{2^2+3^2}} = \frac{|2x_1+3x_2-4|}{\sqrt{13}}$.
(a) To check whether the hyperplane $2x_1 + 3x_2 - 4 = 0$ has a margin for the dataset in Table 1, we need to compute the distance from each point to the hyperplane, and then take the minimum of these distances. However, we first need to check the sign of the decision function for each point. If the sign matches the label of the point, then the point is on the correct side of the hyperplane. If any point is on the wrong side, then the hyperplane does not have a margin for the dataset.
For the point $(1, 1)$ with label 1:
$2 * 1 + 3 * 1 - 4 = 1 > 0$
For the point $(1, -1)$ with label -1:
$2 * 1 + 3 * (-1) - 4 = -3 < 0$
For the point $(0, 0)$ with label -1:
$2 * 0 + 3 * 0 - 4 = -4 < 0$
For the point $(-1, 3)$ with label 1:
$2 * (-1) + 3 * 3 - 4 = 3 > 0$
Since the sign of the decision function matches the label for each point, the hyperplane separates the two classes, and we can proceed to compute the distances:
For the point $(1, 1)$ with label 1:
$distance = \frac{|2*1+3*1-4|}{\sqrt{13}} = \frac{1}{\sqrt{13}} \approx 0.28$
For the point $(1, -1)$ with label -1:
$distance = \frac{|2*1+3*(-1)-4|}{\sqrt{13}} = \frac{3}{\sqrt{13}} \approx 0.83$
For the point $(0, 0)$ with label -1:

$distance = \frac{|2*0+3*0-4|}{\sqrt{13}} = \frac{4}{\sqrt{13}} \approx 1.11$
For the point $(-1, 3)$ with label 1:
$distance = \frac{|2*(-1)+3*3-4|}{\sqrt{13}} = \frac{3}{\sqrt{13}} \approx 0.83$
The margin is the minimum of these distances, which is 0.28. Therefore, the hyperplane has a margin of 0.28 for the dataset in Table 1.
(b) To check whether the hyperplane $2x_1 + 3x_2 - 4 = 0$ has a margin for the dataset in Table 2, we need to compute the distance from each point to the hyperplane, and then take the minimum of these distances. However, we first need to check the sign of the decision function for each point. If the sign matches the label of the point, then the point is on the correct side of the hyperplane. If any point is on the wrong side, then the hyperplane does not have a margin for the dataset.
For the point $(1, 1)$ with label 1:
$2 * 1 + 3 * 1 - 4 = 1 > 0$
For the point $(1, -1)$ with label -1:
$2 * 1 + 3 * (-1) - 4 = -3 < 0$
For the point $(0, 0)$ with label -1:
$2 * 0 + 3 * 0 - 4 = -4 < 0$
For the point $(-1, 3)$ with label 1:
$2 * (-1) + 3 * 3 - 4 = 3 > 0$
For the point $(-1, -1)$ with label 1:
$2 * (-1) + 3 * (-1) - 4 = 3 < 0$
Since the decision function for the point $(-1, -1)$ with label 1 is negative, this point is on the wrong side of the hyperplane. Therefore, the hyperplane does not have a margin for the dataset in Table 2.

2. [8 points] Now, let us look at margins for datasets. Please review what we have discussed in the lecture and slides. A margin for a dataset is not a margin of a hyperplane!

x_1	x_2	label
-1	0	-1
0	-1	-1
1	0	1
0	1	1

Table 3: Dataset 3

(a) [4 points] Given the dataset in Table 3, can you calculate its margin? If you cannot, please explain why.

(b) [4 points] Given the dataset in Table 4, can you calculate its margin? If you cannot, please explain why.

Solution.A margin of a dataset is defined as the maximum margin achievable by any hyperplane. To calculate the margin of a dataset, we need to find the weight vector w

x_1	x_2	label
-1	0	-1
0	-1	1
1	0	-1
0	1	1

Table 4: Dataset 4

that maximizes the margin. The margin for a given weight vector w and a dataset can be calculated as the minimum distance from any data point to the decision boundary defined by w.
However, finding the optimal weight vector w that maximizes the margin is a complex optimization problem and is better solve using an algorithm, for example SVM algorihtm. Nevertheless, for simple datasets, such as in Tables 3 and 4, we might be able to find the optimal weight vector and the margin by inspection or by using simple geometric arguments.
(a) Looking at the dataset in Table 3, we can see that the data points are linearly separable. By inspection, we can see that the decision boundary $x_1 = 0$ separates the data points with label -1 on the left from the data points with label 1 on the right. The margin for this decision boundary is the minimum distance from any data point to the decision boundary. The margin can be calculated as $margin = min_i \frac{|w^T x_i + b|}{||w||}$, where $w = [1, 0]$ and $b = 0$, so the margin is 1.
(b) Looking at the dataset in Table 4, we can see that the data points are not linearly separable. Therefore, we cannot find a decision boundary that separates the data points with different labels, and the margin is undefined.

3. [**Bonus**] [5 points] Let us review the Mistake Bound Theorem for Perceptron discussed in our lecture. If we change the second assumption to be as follows: Suppose there exists a vector $\mathbf{u} \in \mathbb{R}^n$, and a positive γ, we have for each $(\mathbf{x}_i, y_i)$ in the training data, $y_i(\mathbf{u}^\top \mathbf{x}_i) \geq \gamma$. What is the upper bound for the number of mistakes made by the Perceptron algorithm? Note that $\mathbf{u}$ is unnecessary to be a unit vector.

   Solution. The standard Mistake Bound Theorem assumes that there exists a unit vector u such that for all (x_i, y_i), $y_i(u^T x_i) \geq \gamma$, where $\gamma > 0$. However, the modification suggests that u is not necessarily a unit vector. To find the mistake bound in this case, we can normalize u by its length to make it a unit vector and then apply the unmodified theorem.
   Given $u \in \mathbb{R}^n$ (not necessarily a unit vector) and $\gamma > 0$ such that $y_i(u^T x_i) \geq \gamma$, we can define a new vector $v = \frac{u}{||u||}$, which is a unit vector. Now for each (x_i, y_i), we have:

   $$y_i(v^T x_i) = y_i \left(\frac{u}{||u||}^T x_i \right) = \frac{y_i(u^T x_i)}{||u||} \geq \frac{\gamma}{||u||}$$

   The modified mistake bound can the be computed as follows:

   $$R^2 \left(\frac{||u||}{\gamma} \right)^2 = \left(\frac{R||u||}{\gamma} \right)^2$$

   Where R is the maximum norm of any x_i in the dataset, i.e. $R = max_i ||x_i||$. Thus, the

new mistake bound is proportional to the square of the product of the norm of u and the maximum norm of any input vector x_i, divided by the square of γ. THerefore, the upper bound on the number of mistakes the Perceptron algorithm would make under the modified assumption that u is need not be a unit vector is $\left(\frac{R||u||}{\gamma}\right)^2$.

4. [10 points] We want to use Perceptron to learn a disjunction as follows,

$$f(x_1, x_2, \ldots, x_n) = \neg x_1 \vee \neg \ldots \neg x_k \vee x_{k+1} \vee \ldots \vee x_{2k} \quad \text{(note that } 2k < n\text{)}.$$

The training set are all 2^n Boolean input vectors in the instance space. Please derive an upper bound of the number of mistakes made by Perceptron in learning this disjunction.

Solution: To derive an upper bound on the number of mistakes the Perceptron will make on the given disjunction, we can apply the Mistake Bound Theorem. We need to determine he appropriate values for R and γ for the given function $f(x_1, x_2, \ldots, x_n) = \neg x_1 \vee \neg \ldots \neg x_k \vee x_{k+1} \vee \ldots \vee x_{2k}$ where $2k \leq n$.

First, we can find the norm $R = ||x||$, which is the maximum norm of any input vector x_i. For Boolean vectors with components of 0 or 1, the norm is at most $\sqrt{n}$, since each component can contribute at most 1 to the sum of squares. In other words: $||x|| = \sqrt{1^2 + 1^2 + \cdots + 1^2} = \sqrt{n}$

Next, we can find the margin γ. Since, we are dealing with Boolean input vectors, our disjunction will output true (1) for any input where at least one of the literals is true. We can set our weight vector u such that:

- We assign a weight of $+1$ to literals $\neg x_1$ to $\neg x_k$ since their logical negation in the function should trigger a true outcome.
- We assign a weight of $+1$ to literals x_{k+1} to $\neg x_{2k}$ since they are directly included in the disjunction.
- We assign a weight of 0 to the remaining features.

Given that any example where the disjunction is true will have at least one of these features being true, the smallest positive product of $u^T x$ for correctly classified examples will be at least 1. Therefore: $\gamma = 1$.

Based on our findings, we now can say that since $R = \sqrt{n}$ and $\gamma = 1$, by applying the Mistake Bound Theorem, the number of mistakes M Perceptron will make is:

$$M \leq \left(\frac{R}{\gamma}\right)^2 = \left(\frac{\sqrt{n}}{1}\right)^2 = n$$

Therefore, the upper bound on the number of mistakes the Perceptron will make on this disjunction, given the training set of all 2^n Boolean input vectors is n.

5. [10 points] Prove that linear classifiers in a plane cannot shatter any 4 distinct points.

Solution. To show that linear classifiers in a plane cannot shatter any set of 4 distinct points, we need to demonstrate that there is no linear classifier (i.e., no line in the plane) that can separate the 4 points in all possible ways that they might be labeled. This relates to the concept of VC-dimension (Vapnik–Chervonenkis dimension).

A set of points is said to be shattered by a classifier if, for all possible assignments of

labels to those points, there exists a classifier that can perfectly separate the points into their assigned classes. To shatter 4 points in the plane with a linear classifier, we would need to be able to draw a line that separates any possible labeling of the points into 2 classes (assume positive and negative).

Proof by contradiction:

Step 1: Assume 4 points can be shattered.

Let's assume that we have 4 distinct points in a plane and that it's possible for a linear classifier to separate them in all possible ways.

Step 2: Label 3 points in a non-collinear configuration.

First, consider any 3 of the 4 points. Since no 3 of our distinct points are collinear (they don't all lie on the same line), we can always draw a line that separates one of the points from the other two, regardless of how we label them.

Step 3: Introduce the fourth point.

Now, we introduce the fourth point. There are two cases:

1) The fourth point is on the same side of the line as the single point. If we want to classify this fourth point into the opposite class from the single point, then we cannot draw a line that separates it from the single point without also misclassifying one of the other points.

2) The fourth point is on the same side as the other two points. In this case, if we want to separate the single point and the fourth point from the two others, we again face the problem that any line that correctly classifies the single point cannot correctly classify the fourth point without misclassifying one of the two.

In both cases, we find that there's no line that can be drawn to separate the points into the two different classes as required.

Step 4: Show contradiction.

By showing that there is at least one labeling of the points that cannot be separated by a single line (i.e., one configuration that cannot be shattered), we demonstrate that our initial assumption (that 4 points can be shattered by a linear classifier) is false.

Therefore, we conclude that linear classifiers in a plane cannot shatter any set of 4 distinct points. The VC-dimension of a linear classifier in a plane (a line) is 3, since it can shatter any set of 3 non-collinear points but not a set of 4.

6. [**Bonus**] [10 points] Consider our infinite hypothesis space $\mathcal{H}$ are all rectangles in a plain. Each rectangle corresponds to a classifier — all the points inside the rectangle are classified as positive, and otherwise classified as negative. What is VC($\mathcal{H}$)?

   Solution. The VC-dimension (Vapnik-Chervonenkis dimension) of a hypothesis space is the maximum number of points that can be shattered by that space. To shatter a set of points means that for every possible way of labeling these points (with two classes, positive and negative), there exists a hypothesis in the space that can classify the points exactly as they are labeled.

   For the hypothesis space $\mathcal{H}$ of all rectangles on a plane, let's consider how we can shatter points:

   One point: We can always draw a rectangle around a single point to classify it as positive and leave all other points as negative. So, $VC(\mathcal{H}) \geq 1$.

   Two points: We can draw a rectangle such that both points are inside (both pos-

itive), or one is inside and the other is outside (one positive, one negative, in either arrangement). So, $VC(\mathcal{H}) \geq 2$.

Three points: No matter how the three points are arranged, we can draw a rectangle in such a way to include any one of the three points (the others being outside), any two of the points (the other being outside), or all three. So, $VC(\mathcal{H}) \geq 3$.

Four points: If we place four points at the corners of a rectangle, we can shatter this set as well. We can include any one of the four points, any pair of adjacent or opposite points, or any three points, or all four. So, $VC(\mathcal{H}) \geq 4$.

Five points: If we arrange five points such that no four points are the corners of a rectangle formed by those points (for instance, four points forming a rectangle and one point inside this rectangle), there is no way to draw a rectangle that includes the center point while excluding all the corner points. Thus, this set of five points cannot be shattered by our hypothesis space of rectangles.

Therefore, since we can shatter four points but not five, the VC-dimension of the hypothesis space of all rectangles in a plane is $VC(\mathcal{H}) = 4$.

2 Practice [64 points]

1. [2 Points] Update your machine learning library. Please check in your implementation of ensemble learning and least-mean-square (LMS) method in HW1 to your GitHub repository. Remember last time you created the folders "Ensemble Learning" and "Linear Regression". You can commit your code into the corresponding folders now. Please also supplement README.md with concise descriptions about how to use your code to run your Adaboost, bagging, random forest, LMS with batch-gradient and stochastic gradient (how to call the command, set the parameters, etc). Please create a new folder "Perceptron" in the same level as these folders.

   Solution. Changes can be seen on GitHub: https://github.com/milenabel/CS6350-ML2023/.

2. We will implement Perceptron for a binary classification task — bank-note authentication. Please download the data "bank-note.zip" from Canvas. The features and labels are listed in the file "bank-note/data-desc.txt". The training data are stored in the file "bank-note/train.csv", consisting of 872 examples. The test data are stored in "bank-note/test.csv", and comprise of 500 examples. In both the training and testing datasets, feature values and labels are separated by commas.

   (a) [16 points] Implement the standard Perceptron. Set the maximum number of epochs T to 10. Report your learned weight vector, and the average prediction error on the test dataset.

   Solution. Learned weight vector:

   $[53 \ -61.086591 \ -42.70582 \ -40.30786 \ -3.146269]$.

   Average prediction error on the test dataset: 0.020000000000000018.

(b) [16 points] Implement the voted Perceptron. Set the maximum number of epochs T to 10. Report the list of the distinct weight vectors and their counts — the number of correctly predicted training examples. Using this set of weight vectors to predict each test example. Report the average test error.

Solution. Table of distinct vectors and their counts can be seen on Github as a csv and tex files. (The tex file failed to insert in this document due to its size.)

Link: https://github.com/milenabel/CS6350-ML2023/blob/main/Perceptron/figs/...

The average test error: 0.014000000000000012

(c) [16 points] Implement the average Perceptron. Set the maximum number of epochs T to 10. Report your learned weight vector. Comparing with the list of weight vectors from (b), what can you observe? Report the average prediction error on the test data.

Solution. Learned weight vector:

[36.38211009 −46.60072831 −29.06735968 −30.14480192 −8.94214493].

Part b results with similar vectors from the table:

131 [36. −43.811101 −33.60522 −28.54354 −10.474387], 84

133 [36. −42.690241 −25.54332 −37.17104 −11.082187], 6

135 [36. −44.920141 −27.05802 −34.04964 −9.699887], 49

137 [36. −48.670141 −31.00822 −25.49584 −9.661117], 23

141 [36. −44.993221 −29.15622 −37.91314 −5.236147], 38

Magnitude Differences: Although the weight vectors from the voted perceptron vary across the epochs, the average perceptron consolidates the weights into a single vector that captures the average influence across all updates. The average weights don't exactly match any single vector from the voted perceptron but appear to be a consolidation.

Influence of Counts: The counts associated with the weight vectors from the voted perceptron indicate how many times a particular weight vector was used to make correct predictions. While the average perceptron doesn't account for this directly in its final weight vector, the impact of repeatedly used weights in the voted perceptron might be reflected in the average weight vector due to the nature of frequent updates in similar directions.

Observation on the Test Error: To compare the effectiveness of the learned models, we look at the average prediction error on the test data. This gives us an empirical measure of how well each model generalizes to unseen data. Both voted and average perceptrons resulted in the same error on the testing data (as seen below).

Insight into Algorithm Behavior: The voted perceptron generates multiple models, each with its voting power, while the average perceptron seeks to find a single model that performs well on average. The average perceptron's weight vector represents the cumulative knowledge gained over all training examples and epochs, smoothed over time.

Conclusion: The average perceptron's learned weight vector is a single representation that seems to capture the overall trend of the weight adjustments made

during training, while the voted perceptron retains individual weight vectors with varying degrees of influence. Comparing them can reveal how stable the learning process is and whether certain features are consistently influential across different iterations and versions of the perceptron algorithm. The differences in the magnitude of the weights and the presence of certain features across multiple vectors suggest that while there is a general direction of learning, the stochastic nature of the training data and the order in which it is presented can result in variations that the average perceptron's single vector aims to smooth out.

Average prediction error on the test dataset: 0.014000000000000012.

(d) [14 points] Compare the average prediction errors for the three methods. What do you conclude?

Solution.

When comparing the average prediction errors for the standard, voted, and average perceptron, we can draw the following conclusions:

Error Rates: The voted and average perceptrons both have an average test error of 0.014, while the standard perceptron has a slightly higher average test error of 0.02. This indicates that both the voted and average perceptrons perform better than the standard perceptron on the test data.

Consistency and Stability: The voted and average perceptrons use methods that incorporate the history of the weight adjustments throughout the training process. The voted perceptron does this by maintaining a weighted vote system for each weight vector, whereas the average perceptron does this by maintaining a cumulative sum of the weight vectors. Both methods seem to offer a more consistent and stable hypothesis that generalizes better to unseen data compared to the standard perceptron, which only keeps the last weight vector found at the end of the training process.

Noise: The similar performance of the voted and average perceptrons could suggest that these methods are more robust to noise in the training data. By considering the history of the weight vectors, these algorithms might be less prone to overfitting to the noise compared to the standard perceptron.

Algorithm Efficiency: In terms of computational efficiency, the average perceptron could be preferred over the voted perceptron. The average perceptron only needs to perform calculations for a single final weight vector during testing, whereas the voted perceptron must compute predictions using potentially many weight vectors and sum their weighted votes.

In conclusion, while the standard perceptron is the simplest algorithm and easier to understand, the voted and average perceptrons offer better performance, with the added complexity being justified by a lower average prediction error. They both are more suitable for practical applications where prediction accuracy on unseen data is crucial. The choice between voted and average may then come down to considerations of implementation complexity, memory requirements, and computational efficiency at prediction time.