CS 5350/6350, DS 4350: Machine Learning Spring 2024

Homework 2

Handed out: February 1, 2024
Due date: February 15, 2024

General Instructions

Please read before you start

- 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 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 10 pages**. Every extra page will cost a point.
- Handwritten solutions or photos of handwritten solutions will not be accepted.
- The homework is due by midnight of the due date. Please submit the homework on Canvas. You should upload two files: a report with answers to the questions below, and a compressed file (`.zip` or `.tar.gz`) containing your code.
- Some questions are marked **For 6350 students**. Students who are registered for CS 6350 should do these questions. Of course, if you are registered for CS 5350 or DS 4350, you are welcome to do the question too, but you will not get any credit for it.

Important Do not just put down an answer. We want explanations of your answers. No points will be given for just the final answer without an explanation.

1 Warmup: Boolean Functions

1. [3 points] Table 1 shows several data points (the x's) along with corresponding labels (y). (That is, each row is an example with a label.) Write down three different Boolean functions, all of which can produce the label y when given the inputs x.

   (a) $f_1(x) : y = x4$

   (b) $f_2(x) : y = x2 \wedge x3 \wedge x4$

   (c) $f_3(x) : y = x1' \wedge x4$

y	$x1$	$x2$	$x3$	$x4$
0	0	1	1	0
0	1	1	1	0
1	0	1	1	1

Table 1: Initial data set

2. [5 points] Now the Table 1 is expanded to Table 2 by adding more data points. How many errors will each of your functions from the previous questions make on the expanded data set.

   (a) $f_1(x)$ has 0 errors

   (b) $f_2(x)$ has 2 errors (rows 6 and 7 overall (rows 3 and 4 in table 2))

   (c) $f_3(x)$ has 2 errors (rows 3 and 6 overall (row 3 in table 1, row 3 in table 2))

y	$x1$	$x2$	$x3$	$x4$
0	0	1	1	0
0	1	1	1	0
1	0	1	1	1
1	1	0	1	1
0	0	1	1	0
1	1	1	0	1

Table 2: Expanded data set

3. [5 points] Is the function in Table 2 linearly separable? If so, write down a linear threshold function that classifies the data. If not, prove that there is no linear threshold function that can classify the data.

   The function in Table 2 is linearly separable. The linear threshold function that classifies the data is:

$$x4 = 1$$

   Since $x4$ alone can classify the data, it is linearly separable with a single feature. This also applies to the expanded data set with Table 1 and Table 2.

2 Feature transformations

[10 points] Consider the concept class C consisting of functions f_r defined by a radius r as follows:

$$f_r(x_1, x_2) = \begin{cases} +1 & 24x_1^{2024} - 23x_2^{2023} \leq r \\ -1 & \text{otherwise} \end{cases}$$

Note that the hypothesis class is *not* linearly separable in $\mathbb{R}^2$.

Construct a function $\phi(x_1, x_2)$ that maps examples to a new *two-dimensional* space, such that the positive and negative examples are linearly separable in that space. The answer to this question should consist of two parts:

1. A function ϕ that maps examples to a new space.
2. A proof that in the new space, the positive and negative points are linearly separated. You can show this by producing such a hyperplane in the new space (i.e. find a weight vector $\mathbf{w}$ and a bias b such that $\mathbf{w}^T\phi(x_1, x_2) \geq b$ if, and only if, $f_r(x_1, x_2) = +1$.

Hint: The feature transformation ϕ should not depend on r.

To make the positive and negative examples linearly separable in a new two-dimensional space, we can use the kernel trick. We'll define a kernel function $\phi(x_1, x_2)$ that maps the original features (x_1, x_2) to a new feature space where the examples become linearly separable.

Let's consider a transformation using a radial basis function (RBF) kernel, also known as the Gaussian kernel. The RBF kernel is defined as:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

Here, σ is a parameter that determines the spread of the kernel.

We can define our transformation function as:

$$\phi(x_1, x_2) = \left[\exp\left(-\frac{(x_1 - c_1)^2 + (x_2 - c_2)^2}{2\sigma^2}\right), 1\right]$$

where (c_1, c_2) is the center of the positive class in the original feature space. In this case, we can choose (c_1, c_2) such that $24c_1^{2024} - 23c_2^{2023} = r$, which corresponds to the boundary between the positive and negative examples.

Let's denote the transformation for the positive examples as $\phi_+(x_1, x_2)$:

$$\phi_+(x_1, x_2) = \left[\exp\left(-\frac{(x_1 - c_1)^2 + (x_2 - c_2)^2}{2}\sigma^2\right), 1\right]$$

And for the negative examples, $\phi_-(x_1, x_2)$, we set the second component to be -1:

$$\phi_-(x_1, x_2) = \left[\exp\left(-\frac{(x_1 - c_1)^2 + (x_2 - c_2)^2}{2\sigma^2}\right), -1\right]$$

Now, in this new space, the positive and negative examples will be linearly separable by a hyperplane.

The choice of σ can affect the separation. A larger σ will result in a smoother decision boundary, while a smaller σ will result in a more complex, possibly overfit, decision boundary. Adjusting σ might require cross-validation or other techniques to find an optimal value.

3 Mistake Bound Model of Learning

In both the questions below, we will consider functions defined over n Boolean features. That is, each example in our learning problem is a n-dimensional vector from $\{0,1\}^n$. We will use the symbol $\mathbf{x}$ to denote an example and $\mathbf{x}_i$ denotes its i^{th} element. (We will assume that there is no noise involved.)

For all questions below, it is not enough to just state the answer. You need to justify your answer with a short proof.

1. Consider the concept class $\mathcal{C}_1$ defined as follows: Each element of $\mathcal{C}_1$ is defined using a fixed instance $\mathbf{z} \in \{0,1\}^n$ as follows:

$$f_{\mathbf{z}}(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} = \mathbf{z} \\ 0 & \mathbf{x} \neq \mathbf{z}. \end{cases}$$

That is, the function $f_{\mathbf{z}}$ predicts 1 if, and only if, the input to the function is $\mathbf{z}$.

Our goal is to come up with a mistake bound algorithm that will learn any function $f \in \mathcal{C}_1$.

(a) [5 points] Determine $|\mathcal{C}_1|$, the size of concept class.

The size of the concept class $\mathcal{C}_1$ is 2^n.

Proof: For each of the n features, we have two choices: 0 or 1. Therefore, the total number of functions in the concept class is 2^n.

(b) [15 points] Write a mistake bound learning algorithm for this concept class that makes no more than *one* mistake on any sequence of examples presented to it. Please write the algorithm concisely in the form of pseudocode.

Prove the mistake bound for this algorithm.

Goal: Learn$f \in \mathcal{C}_1$ with at most one mistake.

The algorithm will work as follows:

Algorithm 1 Mistake Bound Learning Algorithm for $\mathcal{C}_1$

```
Initialize $h(\mathbf{x}) = 0$ (always predict 0)
for e doach example $\mathbf{x}$ presented to the algorithm
    if $h$ predicts correctly then
        continue
    if $h$ predicts incorrectly then
        update $h$ to predict 1 if the example is $\mathbf{x}$
```

Proof of Mistake Bound: The algorithm will make at most one mistake. If the algorithm makes a mistake, it will update the hypothesis to predict 1 for the example that was misclassified. After this update, the algorithm will not make any more mistakes since there is exactly one element $\mathbf{x}$ that exactly equals z. Therefore, the algorithm will make at most one mistake.

2. Suppose we have a concept class $\mathcal{C}_2$ that consists of exactly n functions $\{f_1, f_2, \cdots, f_n\}$, where each function f_i is defined as follows:

$$f_i(\mathbf{x}) = \mathbf{x}_i.$$

That is, the function f_i returns the value of the i^{th} feature.

(a) [5 points] How many mistakes will the algorithm **CON** from class make on any function from this concept class?

In general, **CON** will make at most $|\mathcal{C}| - 1$ mistakes on any concept class $\mathcal{C}$.
For this concept class $\mathcal{C}_2$, the algorithm **CON** will make at most $n - 1$ mistakes since there are n functions in the concept class.

(b) [5 points] How many mistakes will the Halving algorithm make on any function from this concept class?

In general, the Halving algorithm will make at most $\log_2 |\mathcal{C}|$ mistakes on any concept class $\mathcal{C}$.
For this concept class $\mathcal{C}_2$, the Halving algorithm will make at most $\log_2 n$ mistakes since there are n functions in the concept class.

4 The Perceptron Algorithm and its Variants

For this question, you will experiment with the Perceptron algorithm and some variants on a data set.

4.1 The task and data

We will be using the Diabetic Retinopathy dataset from the UCI Machine Learning repository [1]. The dataset consists of features extracted from images and the goal is to predict whether an image contains signs of diabetic retinopathy or not. Using this labeled data, we want to build a classifier that identifies whether a new retinal image shows signs of diabetic retinopathy or not.

The data has been preprocessed into the same format we used for the previous homework. Use the training/development/test files called `diabetes.train.csv`, `diabetes.dev.csv` and `diabetes.test.csv`. For details about the data format, check README.txt in the dataset file provided to you.

4.2 Algorithms

You will implement several variants of the Perceptron algorithm. Note that each variant has different hyper-parameters, as described below. Use 5-fold cross-validation to identify the best hyper-parameters as you did in the previous homework. To help with this, we have split the training set into five parts `train0.data.csv`–`train4.data.csv` in the folder `CVSplits`.

[1] `https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set`

1. **Simple Perceptron:** Implement the simple batch version of Perceptron as described in the class. Use a fixed learning rate η chosen from $\{1, 0.1, 0.01\}$. An update will be performed on an example $(\mathbf{x}, y)$ if $y(\mathbf{w}^T\mathbf{x} + b) < 0$ as:

   $$\mathbf{w} \leftarrow \mathbf{w} + \eta y \mathbf{x},$$
   $$b \leftarrow b + \eta y.$$

   Hyper-parameter: Learning rate $\eta \in \{1, 0.1, 0.01\}$

   Two things bear additional explanation.

   (a) First, note that in the formulation above, the bias term b is explicitly mentioned. This is because the features in the data do not include a bias feature. Of course, you could choose to add an additional constant feature to each example and not have the explicit extra b during learning. (See the class lectures for more information.) However, here, we will see the version of Perceptron that explicitly has the bias term.

   (b) Second, in this specific case, if $\mathbf{w}$ and b are initialized with zero, then the fixed learning rate will have no effect. To see this, recall the Perceptron update from above.

   Now, if $\mathbf{w}$ and b are initialized with zeroes and a fixed learning rate η is used, then we can show that the final parameters will be equivalent to having a learning rate 1. The final weight vector and the bias term will be scaled by η compared to the unit learning rate case, which does not affect the sign of $\mathbf{w}^T\mathbf{x} + b$.

   To avoid this, you should initialize the all elements of the weight vector $\mathbf{w}$ and the bias to a small random number between -0.01 and 0.01.

2. **Decaying the learning rate:** Instead of fixing the learning rate, implement a version of the Perceptron algorithm whose learning rate decreases as $\frac{\eta_0}{1+t}$, where η_0 is the starting learning rate, and t is the time step. Note that t should keep increasing across epochs. (That is, you should initialize t to 0 at the start and keep incrementing it after each epoch.)

   Hyper-parameter: Initial learning rate $\eta_0 \in \{1, 0.1, 0.01\}$

3. **Margin Perceptron:** This variant of Perceptron will perform an update on an example $(\mathbf{x}, y)$ if $y(\mathbf{w}^T\mathbf{x} + b) < \mu$, where μ is an additional positive hyper-parameter, specified by the user. Note that because μ is positive, this algorithm can update the weight vector even when the current weight vector does not make a mistake on the current example. You need to use the decreasing learning rate as before.

   Hyper-parameters:

   (a) Initial learning rate $\eta_0 \in \{1, 0.1, 0.01\}$

   (b) Margin $\mu \in \{1, 0.1, 0.01\}$

Note: When there is more than one hyper-parameter to cross-validate, you need to consider all combinations of the hyper-parameters. In this case, you will need to perform cross-validation for all pairs (η_0, μ) from the above sets.

4. **Averaged Perceptron** Implement the averaged version of the original Perceptron algorithm from the first question. Recall from class that the averaged variant of the Perceptron asks you to keep two weight vectors (and two bias terms). In addition to the original parameters $(\mathbf{w}, b)$, you will need to update the averaged weight vector $\mathbf{a}$ and the averaged bias b_a as:

   (a) $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{w}$

   (b) $b_a \leftarrow b_a + b$

   This update should happen once for every example in every epoch, *irrespective of whether the weights were updated or not for that example.* In the end, the learning algorithm should return the averaged weights and the averaged bias.

   (Technically, this strategy can be used with any of the variants we have seen here. For this homework, we only ask you to implement the averaged version of the original Perceptron. However, you are welcome to experiment with averaging the other variants.)

5. **Aggressive Perceptron with Margin, (For 6350 Students)** This algorithm is an extension of the margin Perceptron and performs an aggressive update as follows:

   If $y(\mathbf{w}^T\mathbf{x}) \leq \mu$ then
   (a) Update $\mathbf{w}_{new} \leftarrow \mathbf{w}_{old} + \eta y \mathbf{x}$

   Unlike the standard Perceptron algorithm, here the learning rate η is given by

   $$\eta = \frac{\mu - y(\mathbf{w}^T\mathbf{x})}{\mathbf{x}^T\mathbf{x} + 1}$$

   As with the margin Perceptron, there is an additional positive parameter μ.

   Explanation of the update. We call this the aggressive update because the learning rate is derived from the following optimization problem:

   When we see that $y(\mathbf{w}^T\mathbf{x}) \leq \mu$, we try to find new values of $\mathbf{w}$ such that $y(\mathbf{w}^T\mathbf{x}) = \mu$ using

   $$\begin{array}{cc} \min\limits_{\mathbf{w}_{new}} & \frac{1}{2}||\mathbf{w}_{new} - \mathbf{w}_{old}||^2 \\ \text{such that} & y(\mathbf{w}^T x) = \mu. \end{array}$$

   That is, the goal is to find the smallest change in the weights so that the current example is on the right side of the weight vector.

By substituting (a) from above into this optimization problem, we will get a single variable optimization problem whose solution gives us the η defined above. You can think of this algorithm as trying to tune the weight vector so that the current example is correctly classified right after the update.

Implement this aggressive Perceptron algorithm.

Hyper-parameters: $\mu \in \{1, 0.1, 0.01\}$

4.3 Experiments

For all 5 settings above (4 for undergraduate students), you need to do the following things:

1. Run cross validation for **ten** epochs for each hyper-parameter combination to get the best hyper-parameter setting. Note that for cases when you are exploring combinations of hyper-parameters (such as the margin Perceptron), you need to try out all combinations.

2. Train the classifier for **20** epochs. At the end of each training epoch, you should measure the accuracy of the classifier on the development set. For the averaged Perceptron, use the average classifier to compute accuracy.

3. Use the classifier from the epoch where the development set accuracy is highest to evaluate on the test set.

4.4 What to report

1. [5 points] Briefly describe the design decisions that you have made in your implementation. (E.g, what programming language, how do you represent the vectors, etc.)

2. [2 points] *Majority baseline:* Consider a classifier that always predicts the most frequent label. What is its accuracy on test and development set?

3. [10 points per variant] For each variant above (5 for 6350 students, 4 for 5350 students), you need to report:

   (a) The best hyper-parameters

   (b) The cross-validation accuracy for the best hyperparameter

   (c) The total number of updates the learning algorithm performs on the training set

   (d) Development set accuracy

   (e) Test set accuracy

   (f) Plot a *learning curve* where the x axis is the epoch id and the y axis is the dev set accuracy using the classifier (or the averaged classifier, as appropriate) at the end of that epoch. Note that you should have selected the number of epochs using the learning curve (but no more than 20 epochs).

Experiment Submission Guidelines

1. The report should detail your experiments. For each step, explain in no more than a paragraph or so how your implementation works. Describe what you did. Comment on the design choices in your implementation. For your experiments, what algorithm parameters did you use? Try to analyze this and give your observations.

2. Your report should be in the form of a *pdf* file, LaTeX is recommended.

3. *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.

4. Please do not hand in binary files! We will *not* grade binary submissions.

5. Please look up the late policy on the course website.