CS 5350/6350: Machine Learining Fall 2015

Homework 2

Handed out: Sep 17, 2015
Due date: Oct 1, 2015

General Instructions

- 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 ask questions about 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 will not be accepted.
- The homework is due by midnight of the due date. Please submit the homework on Canvas.
- 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, you are welcome to do the question too, but you will not get any credit for it.

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.

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.

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. [7 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.

2 Mistake-bound learning

1. Let us define a concept class C_n of Boolean functions from the n dimensional Boolean space $\{0,1\}^n$ to $\{0,1\}$. Every function in this class is an indicator function for a particular element in the input space. That is, for every $\mathbf{z} \in \{0,1\}^n$, there is a function $f_{\mathbf{z}}$ in C_n defined as follows:

$$f_{\mathbf{z}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{z} = \mathbf{x} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

   (a) [2 points] How many functions does C_n contain?

   (b) [10 points] Suppose we use the Halving algorithm to learn this concept class. How many mistakes will the algorithm make? Write a short proof for your answer. (Hint: The bound we proved in class doesn't directly apply.)

   (c) [1 point] Is Halving a mistake-bound algorithm for this concept class?

2. [12 points] Recall from class that the Halving algorithm assumes that there is the true (hidden) function is in the concept class C with N elements and tries to find it. In this setting, we know the number of mistakes made by the algorithm is $O(\log N)$.

   Another way to think about this setting is that we are trying to predict with expert advice. That is, we have a pool of N so called experts, only one of whom is perfect. As the halving algorithm proceeds, it cuts down this pool by at least half each time a mistake is made.

   Suppose, instead of one perfect expert, we have M perfect experts in our pool. Show that the mistake bound of the same Halving algorithm in this case is $O(\log \frac{N}{M})$.

   (*Hint*: To show this, consider the stopping condition of the algorithm. At what point, will the algorithm stop making mistakes?)

3 The Perceptron Algorithm and its Variants

For this question, you will experiment with the Perceptron algorithm and some variants on a large collection of data sets.

Data

The training and test data sets are available on the assignments page of the class website. There are two collections of data, named `data0` and `data1`. Both have files with the following naming convention: `trainI.K` and `testI.K`, where K represents the number of dimensions/features in each data point. So for example `train0.10` is a training set in `data0` with 11 features (ten features plus a constant bias feature) and the corresponding test data set is `test0.10`.

One popular format for representing labeled data is the `libSVM` format (described below). All the data for this assignment is presented in this format. Each row in the file is a single example. The format of the each row in the data is

`<label> <index1>:<value1> <index2>:<value2>` $\cdots$

Here `<label>` denotes the label for that example. The rest of the elements of the row is a sparse vector denoting the feature vector. For example, if the original feature vector is $[0, 0, 1, 2, 0, 3]$, this would be represented as `3:1 4:2 6:3`. That is, only the non-zero entries of the feature vector are stored.

Algorithms

You will implement two variants of the Perceptron algorithm. Note that each variant has different hyper-parameters, as described below.

- **Perceptron**: This is the simple version of Perceptron as described in the class. An update will be performed on an example $(\mathbf{x}, y)$ if $y(\mathbf{w}^T\mathbf{x}) \leq 0$.

  Hyper-parameters: The learning rate r.

  Two things bear additional explanation. First, note that in the formulation above, the bias term b is *not* explicitly mentioned. This is because the features in the data folder include a bias feature. (See the class lectures for more information.) Second, if all elements of $\mathbf{w}$ and the bias term are initialized with zero, then the learning rate will have no effect. To see this, recall the Perceptron update:

$$\mathbf{w}_{new} \leftarrow \mathbf{w}_{old} + ry\mathbf{x}$$

  Now, if $\mathbf{w}$ are initialized with zeroes and a learning rate r 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 r compared to the unit learning rate case.

  For this assignment, you should initialize the weight vector and the bias randomly and tune the learning rate parameter. We recommend trying small values less than one. (eg. 1, 0.1, 0.01, etc.)

- **Margin Perceptron:** This variant of Perceptron will perform an update on an example$(\mathbf{x}, y)$ if $y(\mathbf{w}^T\mathbf{x}) \leq \mu$, 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.

  Hyper-parameters: Learning rate r and the margin μ. We recommend setting the value of μ between 0 and 5.0.

Experiments

For all your experiments, you may choose whatever hyper-parameters you like, but we suggest that you informally experiment with them before submitting the results. (You can use cross-validation to find the hyper-parameters as you did in the previous homework. Note that we did not partition the data into parts, so you should do that if you want to find hyper-parameters using cross-validation.)

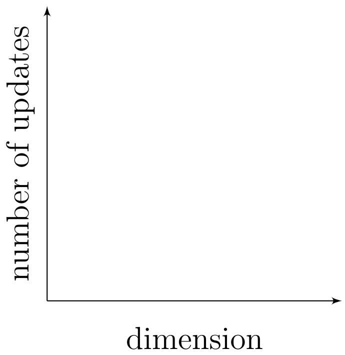
1. [Sanity check, 5 points] Run the simple Perceptron algorithm on the data in Table 2 (one pass only, without shuffling) and report the weight vector that the algorithm returns. How many mistakes does it make?

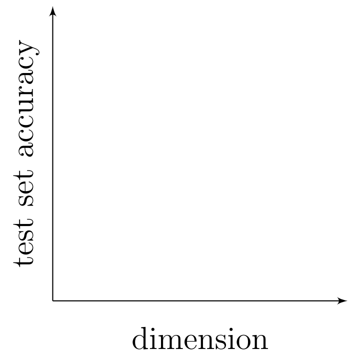
2. [Online Perceptron, 15 points] Choose the 10 dimensional training set (`train0.10`) from the `data0` folder and its corresponding test dataset. Run both the Perceptron algorithm and the margin Perceptron on this dataset for one pass. Report the number of updates (or equivalently mistakes) made by each algorithm and the accuracy of the final weight vector on both the training and the test set. Once again, you will require some playing with the algorithm hyper-parameters. You will see that the hyper-parameters will make a difference and so try out different values.

3. [Batch Perceptron on all datasets, 40 points] Now, on each train/test dataset in the `data0` and `data1` folders, run the Perceptron and margin Perceptron algorithms for ten epochs. *Do not forget to randomly shuffle the data at the start of each epoch.*

   Record the the number of updates on the train datasets and the accuracy on the test data sets.

   So, for example, use the `data0/train0.10` file to train your Perceptron and test it on `data0/test0.10` and record the number of updates/mistakes and accuracy. Then, repeat this for all other datasets. This constitutes one experiment. Once you have the above data, plot two sets of graphs for each experiment (`data0`/`data1` + Perceptron/margin Perceptron):





4. **(For 6350 Students)** [Aggressive Perceptron with Margin, 15 points] This algorithm is an extension of the margin Perceptron which performs an aggressive update as follows:

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

   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.

   Repeat the experiment 3 with the aggressive update. Note that You should report two sets of results (one for `data0` and one for `data1`).

   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

$$\min_{\mathbf{w}_{new}} \quad \frac{1}{2}||\mathbf{w}_{new} - \mathbf{w}_{old}||^2$$
$$\text{such that} \quad y(\mathbf{w}^T x) = \mu.$$

   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.

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.

2. *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.

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