# CS 5350/6350: Machine Learining Fall 2017

Homework 2

Handed out: 12 September, 2017
Due date: 26 September, 2017

## 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 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 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. Students who have registered for CS 5350 are welcome to do this question for extra credit.

## 1 Warm up: Linear Classifiers and Boolean Functions

[10 points] Please indicate if each of the following boolean funcitons is linearly separable. If it is linearly separable, write down a linear threshold unit equivalent to it.

1. [2 points] $x_1 \vee \neg x_2 \vee x_3$
2. [2 points] $(x_1 \vee x_2) \wedge (x_2 \vee x_3)$
3. [2 points] $(x_1 \wedge \neg x_2) \vee x_3$
4. [2 points] $x_1$ xor $x_2$ xor $x_3$
5. [2 points] $\neg x_1 \wedge x_2 \wedge \neg x_3$

# 2 Mistake Bound Model of Learning

1. [20 points] Consider an instance space consisting of integer points on the two dimensional plane $(x_1, x_2)$ with $-80 \leq x_1, x_2 \leq 80$. Let $\mathcal{C}$ be a concept class defined on this instance space. Each function $f_l$ in $\mathcal{C}$ is defined by a length $l$ (with $1 \leq l \leq 80$) as follows:

$$f_l(x_1, x_2) = \begin{cases} +1 & |x_1| \leq l \text{ and } |x_2| \leq l \\ -1 & \text{otherwise} \end{cases} \tag{1}$$

Our goal is to come up with a error-driven algorithm that will learn the correct function $f \in \mathcal{C}$ that correctly classifies a dataset.

**Side notes**

(a) Recall that a concept class is the set of functions from which the true target function is drawn and the hypothesis space is the set of functions that the learning algorithm searches over. In this question, both these are the same set.

(b) Assume that there is no noise. That is, assume that the data is separable using the hypothesis class.

**Questions**

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

(b) [5 points] To design an error driven learning algorithm, we should be able to first write down what it means to make a mistake. Suppose our current guess for the function is $f_l$ defined as in Equation 1 above. Say we get an input point $(x_1^t, x_2^t)$ along with its label $y^t$. Write down an expression (an equality or an inequality) in terms of $x_1^t, x_2^t, y^t$ and $l$ that checks whether the current hypothesis $f_l$ has made a mistake.

(c) [5 points] Next, we need to specify how we will update a hypothesis if there is an error. Since $f_l$ is completely defined in terms of $l$, we only need to update $l$. How will you update $l$ if there is an error? Consider errors for both positive and negative examples.

(d) [5 points] Use the answers from the previous two steps to write a mistake-driven learning algorithm to learn the function. Please write the algorithm concisely in the form of pseudocode. What is the maximum number of mistakes that this algorithm can make on any dataset?

2. [20 points] Recall from class that the Halving algorithm assumes that there is the true (hidden) function is in the concept class $\mathcal{C}$ 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 data set.

## 3.1 The task and data

Image you are an network security expert and you are given a collection of URLs. Now, your goal is to build a classifier that identifies which among these are phishing websites.

We will use Phishing data set from the UCI Machine Learning repository[1] to study this. The data has been preprocessed into a standard format. Use the training/development/test files called `phishing.train`, `phishing.dev` and `phishing.test`. These files are in the LIBSVM format, where each row is a single training example. The format of the each row in the data is:

```
<label> <index1>:<value1> <index2>:<value2> ...
```

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.

## 3.2 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 `training00.data`–`training04.data` in the folder `CVSplits`.

1. **Simple Perceptron:** Implement the simple batch version of Perceptron as described in the class. Use a fixed learning rate $\eta$ 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.

[1] `https://archive.ics.uci.edu/ml/datasets/Phishing+Websites`

(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 $\eta$ 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 $\eta$ 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. **Perceptron with dynamic learning rate:** Implement a Perceptron whose learning rate decreases as $\frac{\eta_0}{1+t}$, where $\eta_0$ 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 for each update.)

   **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 $\mu$ is an additional positive hyper-parameter, specified by the user. Note that because $\mu$ 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 $\mu \in \{1, 0.1, 0.01\}$

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 $\eta$ 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 $\mu$.

   **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

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

   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 $\eta$ 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:** $\mu \in \{1, 0.1, 0.01\}$

## 3.3 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.

### 3.4 What to report

1. [8 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.