CS 5350/6350: Machine Learining Fall 2016

Homework 2

Gopal Menon

September 17, 2016

1 Warm up: Feature expansion

1. Consider the following function that maps the examples in $\Re^2$ into a higher space $\Re^3$:

$$\phi : (x_1, x_2) \rightarrow (x_1, x_2, f_r(x_1, x_2))$$

This has the effect of raising the positively labelled examples in the newly introduced dimension. I'm not sure about the mathematical notation, but my intention is to map the function from two-dimensional to three-dimensional space.

2. In order to verify that this achieves a linear separation between the positive and negative examples, I will define a weight vector w which includes a bias term as follows:

$$w = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The dot product of the weight vector transpose and an example (here the example has a dimension in addition to the added third dimension in order to accommodate the bias) will be

$$\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ f_r(x_1, x_2) \end{bmatrix} = \begin{cases} +1 & 4x_1^4 + 16x_2^4 \leq r; \\ -1 & \text{otherwise} \end{cases}$$

This means that the weight vector w can be used to separate out the positive and negative labelled examples. The hyperplane that separates the positive and negative labelled examples will pass through the origin (since the bias is zero) and will be perpendicular to the weight vector w. The dot product of the weight vector and any example will give the distance of the example point from the linearly separating plane

(since it's a projection of the example vector on the weight vector) that separates the positive and negative examples, we can see that the dot product is $+1$ for positively labelled examples and is -1 for negatively labelled examples. Hence the weight vector w separates out the examples with different labels.

2 Mistake Bound Model of Learning

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_r in $\mathcal{C}$ is defined by a radius r (with $100 \leq r \leq 6400$) as follows:

$$f_r(x_1, x_2) = \begin{cases} +1 & x_1^2 + x_2^2 \leq r^2; \\ -1 & \text{otherwise} \end{cases} \tag{1}$$

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

1. 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.

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

Questions

1. [5 points] Determine $|\mathcal{C}|$, the size of concept class.

2. [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_r 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 r that checks whether the current hypothesis f_r has made a mistake.

3. [10 points] Next, we need to specify how we will update a hypothesis if there is an error. Since f_r is completely defined in terms of r, we only need to update r. How will you update r if there is an error? Consider errors for both positive and negative examples.

4. [20 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?

5. (**For 6350 students**)[15 points total] We have seen the Halving algorithm in class. The Halving algorithm will maintain a set of hypotheses consistent with all the examples seen so far and predict using the most frequent label among this set. Upon making a mistake, the algorithm prune at least half of this set. In this question, you will design and analyze a Halving algorithm for this particular concept space.

   a. [5 points] The set of hypotheses consistent with all examples seen so far can be defined by storing only two integers. How would you do this?

   b. [5 points] How would you check if there is an error for an example (x_1^t, x_2^t) that has the label y^t?

   c. [5 points] Write the full Halving algorithm for this specific concept space. (Do not write the same Halving algorithm we saw in class. You need to tailor it to this problem.) What is its mistake bound?

3 The Perceptron Algorithm and Its Variants

3.1 The Task and Data

Imagine you have access to information about people such as age, gender and level of education. Now, you want to predict whether a person makes over $\$50K$ a year or not using these features.

We will use Adult data set from the UCI Machine Learning repository[1] to study this. As in homework 1, you may use Java, Python, Matlab, C/C++ for this assignment. If you want to use a different language, you must contact the instructor first. Any other language you may want to use **MUST** run on the CADE machines.

The original Adult data set has 14 features, among which 6 are continuous and 8 are categorical. In order to make it easier to use, we will use a pre-processed version (and subset) of the original Adult data set, created by the makers of the popular LIBSVM tool. From the LIBSVM website: "In this data set, the continuous features are discretized into quantiles, and each quantile is represented by a binary feature. Also, a categorical feature with m categories is converted to m binary features."

Use the training/test files called `'a5a.train'` and `'a5a.test'`, available on the assignments page of the class website.[2] This data is 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.

[1] Look for information about the Adult data set at `https://archive.ics.uci.edu/ml/datasets/Adult`

[2] These are the same as `a5a` and `a5a.t` available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`

3.2 Algorithms

You will implement two variants of the Perceptron algorithm. Note that each variant has different hyper-parameters, as described below. For this homework, you are not required to use cross-validation to find good hyperparameters. However, if you want, you may choose to do so.

- **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 $y(\mathbf{w}^T\mathbf{x} + b) < 0$.

  Hyper-parameters: The learning rate r

  Two things bear additional explanation.

  First, note that in the formulation above, the bias term b is explicitly mentioned. This is because the features in the Adult 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.

  Second, if $\mathbf{w}$ and b are initialized with zero, then the learning rate will have no effect. To see this, recall the Perceptron update:

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

  Now, if $\mathbf{w}$ and b 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, which does not affect the sign of $\mathbf{w}^T\mathbf{x} + b$).

  For this assignment, unless otherwise specified, 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 $y(\mathbf{w}^T\mathbf{x} + b) < \mu$, 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.

  Hyper-parameters: Learning rate r and the margin μ.

  We recommend setting the value of μ between 0 and 5.0.

3.3 Experiments

1. [Sanity check, 10 points] Run the simple Perceptron algorithm on the data in the file `table2` from the dataset (one pass only) and report the weight vector that the algorithm returns. How many mistakes does it make?

   For this question alone, you should initialize the weights with zero.

2. [Online setting, 15 points] Run both the Perceptron algorithm and the margin Perceptron on the Adult data 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.

   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. Once again, we recommend that you use cross-validation for choosing them, but you do not have to do so.

3. [Using online algorithms in the batch setting, 20 points] The third experiment is to evaluate the algorithms in a more realistic setting, where the algorithms perform multiple passes over the training data. This means that there is an *additional* hyper-parameter: the number of epochs.

   Run the algorithms for three and five epochs and report the number of updates made, and the accuracies of the final weight vectors on the training and test data.

   It is often important to shuffle the training data before starting each epoch. Report the results of the above experiments when you shuffle the data at the start of each epoch. Briefly explain your results.

4. (**For 6350 Students**) [Aggressive Perceptron with Margin, 10 points] Implement an extension of the margin Perceptron which performs an *aggressive* update as follows:

   If $y(\mathbf{w}^T\mathbf{x} + b) < \mu$, then update

   (a) $\mathbf{w}_{new} \leftarrow \mathbf{w}_{old} + \eta y x$

   (b) $b_{new} \leftarrow b + \eta y$,

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

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

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

   We call this the aggressive update because the update can be derived from the following optimization problem. When we see that $y(\mathbf{w}^T\mathbf{x} + b) < \mu$, we try to find new values of $\mathbf{w}$ and b such that $y(\mathbf{w}^T\mathbf{x} + b) = \mu$ using

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

   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) and (b) 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.

Repeat the batch experiments with the aggressive update. You should report two sets of results (one with shuffling and one without).

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.

   You may provide the results as a table or a graph.

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.