PROEA-821: Machine Learining Spring 2018

Homework 2

Handed out: April 13th, 2018
Due date: April 27th, 2018

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.
- Handwritten solutions will not be accepted.
- The homework is due by midnight of the due date. Please submit the homework on Canvas.
- You should submit two files: a separate pdf or word file for the homework report, and a zip file containing everything else, e.g. code, data, etc.

1 The Perceptron Algorithm and its Variants

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

1.1 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.

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

1.2 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 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. **Perceptron with dynamic learning rate:** Implement a Perceptron 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 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 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 $\mu \in \{1, 0.1, 0.01\}$

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.)

1.3 Experiments

For all 4 settings above, 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.

1.4 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, 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. You may provide the results for the final step as a table or a graph.

2. You should include a shell script, `run.sh`, that will execute your source code. Your code should produce similar output to what you include in your report. It is suggested (but not required) to include a ReadMe file describing how to run your code.

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