CSE 353 – Homework I

Dr. Ritwik Banerjee

Due by Mar 29, 2019 [11:59 pm]

1 Theory

I: A "warm up" problem **5 points**

Consider instances of $\mathcal{X}$ drawn from the uniform distribution $\mathcal{D}$ on $[-1, 1]$. Let f denote the actual labeling function mapping each instance to its label $y \in \{-1, 1\}$ with probabilities

$$Pr(y = \ \ 1|x > 0) = 0.9$$
$$Pr(y = -1|x > 0) = 0.1$$
$$Pr(y = \ \ 1|x \leq 0) = 0.1$$
$$Pr(y = -1|x \leq 0) = 0.9$$

The hypothesis h predicts the label for each instance as defined below:

$$h(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Measure the success of this predictor by calculating the training error for h.

II: Bayes Optimal Predictor **5 points**

Show that for every probability distribution $\mathcal{D}$, the Bayes optimal predictor $f_{\mathcal{D}}$ is, in fact, optimal. That is, show that for every classifier $g : \mathcal{X} \to \{0, 1\}$,

$$\mathcal{L}_{\mathcal{D}}(f_{\mathcal{D}}) \leq \mathcal{L}_{\mathcal{D}}(g).$$

III: Perceptron with a Learning Rate **5 points**

Let us modify the perceptron algorithm as follows:

In the update step, instead of setting $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i\mathbf{x}_i$ every time there is a misclassification, we set $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta y_i\mathbf{x}_i$ instead, for some $0 < \eta < 1$. Show that this modified perceptron will

(a) perform the same number of iterations as the original, and
(b) converge to a vector that points to the same direction as the output of the original perceptron.

IV: Unidentical Distributions **5 points**

Let $\mathcal{X}$ be a domain and let $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_m$ be a sequence of distributions over $\mathcal{X}$. Let $\mathcal{H}$ be a finite class of binary classifiers over $\mathcal{X}$ and let $f \in \mathcal{H}$. Suppose we are getting a sample $\mathcal{S}$ of m examples such that the instances are independent but *not* identically distributed, the i^{th} instance is sampled from $\mathcal{D}_i$ and then y_i is set to be $f(\mathbf{x}_i)$. Let $\overline{\mathcal{D}}_m$ denote the average, i.e., $\overline{\mathcal{D}}_m = (\mathcal{D}_1 + \ . \ . \ . \ + \mathcal{D}_m)/m$.

Fix an accuracy parameter $\epsilon \in (0, 1)$. Show that

$$Pr\left[\exists\, h \in \mathcal{H} \text{ s.t. } L_{(\overline{\mathcal{D}}_m, f)}(h) > \epsilon \text{ and } L_{(S,f)}(h) = 0\right] \leq |\mathcal{H}|e^{-\epsilon m}$$

2 Programming

In this section, you will be working with a Breast Cancer Prediction dataset. Diagnosis of breast cancer is performed when an abnormal lump is found (from self-examination or x-ray) or a tiny speck of calcium is seen (on an x-ray). In this dataset, there are 569 observations, with each observation consisting of 5 features (mean radius, mean texture, mean perimeter, mean area, mean smoothness). The last column is the diagnosis – 0 indicates that the finding was benign, and 1 indicates that it was malignant.

I: Perceptron **25 points**

The first task is to write your own code in Java or Python to implement the perceptron learning algorithm. Your goal is minimize the empirical risk (i.e., no need to do validation or structural risk minimization) for the perceptron to learn the diagnosis.

As you can tell, this is a binary classification task. For this task, submit your code for the perceptron learning algorithm. This code should be able to read the dataset, and eventually provide the learned weight vector as its final output.

II: A modified Perceptron algorithm **15 points**

The second task is to modify the perceptron algorithm to exploit the observations that do not lead to updated weights. This modified perceptron algorithm does the following: for each weight vector $\mathbf{w}$, (i) it keeps count of the number of consecutive correctly classified observations until the next update, and (ii) it always keeps the weight vector $\mathbf{w}$ that had the longest such streak. This variation is called the **pocket algorithm** because it keeps the best solution seen so far "in the pocket", and returns this rather than the last solution (see Alg. 1).

Your submission for this task should be your code for the pocket algorithm. Just like the original perceptron code, this code too should be able to read the dataset, and eventually provide the learned weight vector as its final output. You may write a totally separate code for the pocket algorithm, or you may choose to incoroporate it as a 'version' of the perceptron algorithm. An example of the latter approach could be something that looks like

```
$ python perceptron.py --version naive --dataset /path/to/data/filename.csv
$ python perceptron.py --version pocket --dataset /path/to/data/filename.csv
```

III: Linear Regression for Classification **25 points**

The third and final programming task is to use linear regression for classification. In class, we saw how logistic regression has a nice probabilistic interpretation that can be used for binary classification. Linear regression, however, does not.

What you are required to do, instead, is divide the dataset into two sub-datasets according to whether the diagnosis is 0 or 1. Next, run linear regression on each of these two sub-datasets to obtain two linear subspaces that fit the benign findings and the malignant findings, respectively. Once you have obtained these two subspaces (both through ERM), the actual binary classfication will be done by computing the Euclidean distance of each observation from the two subspaces, and assigning the label of the closer fit. That is, once you obtain the two vectors $\mathbf{w}_0$ (for benign) and

$\mathbf{w}_1$ (for malignant) as output of the two linear regressions, an observation will be classified as 0 or 1 depending on which of these two weight vectors it is closer to.

Your submission for this task should be your code for linear regression. This code should be able to perform the whole task defined above, and not require the user to manually divide the dataset and call the 'usual' linear regression code multiple times. For example, the user should be able to do something like

```
$ python linearregressionclassifier.py --dataset /path/to/data/filename.csv
```

and obtain the classifier's performance.

IV: Final Report **15 points**

Along with your code, you are also required to submit a short report (no more than 2 pages)[1]. The report must contain the following:

- The performance of each model (i.e., the empirical risk).
- A brief discussion (at most a half-page) about what you observed regarding the runtime and termination of the original perceptron algorithm, and what you did if/when the algorithm did not stop after some reasonable amount of time. In this discussion, also investigate why the algorithm did or did not stop and justify your reasoning.
- A brief discussion (at most a half-page) about what you observed regarding the runtime and termination of the pocket algorithm, especially when compared to the original perceptron algorithm.
- A brief discussion (at most a half-page) about using linear regression in this way and what might be the potential benefits and/or drawbacks of the approach.
- A README section explaining how to run your code. Keep in mind that the grader will only run the code, and not do any manual work (e.g., placing the dataset in the same folder as your code, or divide the dataset into two, etc. etc.) to make sure that your code runs "as is"!

The report will be graded based on (a) performance details, (b) replicability of experiments, (c) explanation of how to run your code, and (d) the quality of the discussions about each algorithm.

Notes

What programming languages are allowed?

1. Java (JDK 1.8 or above), Python (3.x).
2. You are NOT allowed to use any data science and/or machine learning library for this assignment. If you have doubts about whether the use of a library is acceptable, please ask before assuming that it can be used!
3. You can use libraries for the matrix multiplication and eigenvalue decomposition steps required in linear regression. Beyond that, though, the ERM code for linear regression must be your own original code.

What should we submit, and how?

Submit a single `.zip` archive containing (i) one folder simply called "code", containing all your code, (ii) a PDF document for your report, and (iii) a PDF document for the theory part of your submission. Please do **NOT** handwrite the solutions and scan or take pictures. For the PDF documents, use either LaTeXor MS Word to write the solutions, and export to PDF. Anything else will not be graded.

[1] For the report, please stick to single line spacing.

Input:
Training data: $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, $\mathcal{X} = 1 \times \mathbb{R}^d$, $\mathcal{Y} = \{-1, 1\}$.
Learning rate parameter: $0 < \eta < 1$
Maximum number of steps: $\max_{iter}$
Initialization:
$\mathbf{w} \leftarrow \mathbf{0}$
$\mathbf{w}_{\text{pocket}} \leftarrow \mathbf{0}$
$\text{run} \leftarrow 0$
$\text{run}_{\text{pocket}} \leftarrow 0$
while $\mathbf{w}$ *has updated* $\wedge$ *run* $< \max_{iter}$ **do**
  draw the next labeled example $(\mathbf{x}, y)$ from $\mathcal{S}$;
  if $(\langle \mathbf{w}, \mathbf{x} \rangle \geq 0 \wedge y = -1) \vee (\langle \mathbf{w}, \mathbf{x} \rangle < 0 \wedge y = 1)$ **then**
    if *run* $>$ *run*$_{pocket}$ **then**
      $\mathbf{w}_{\text{pocket}} \leftarrow \mathbf{w}$;
      $\text{run}_{\text{pocket}} \leftarrow \text{run}$;
    end
    $\mathbf{w} \leftarrow \mathbf{w} + \eta\mathbf{x}$;
    $\text{run} \leftarrow 0$;
  else
    $\text{run} \leftarrow \text{run} + 1$;
  end
end
Output: $\mathbf{w}_{\text{pocket}} \in \mathbb{R}^{d+1}$.

Algorithm 1: Pocket Algorithm