CS 5350/6350: Machine Learning Fall 2020

Homework 2

Handed out: 17 September, 2020
Due date: 01 October, 2020

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

2 Mistake Bound Model of Learning

For all the questions in this section, assume that we are working with n-dimensional Boolean instances. Individual features are denoted by $x_1, x_2, \cdots, x_n$.

1. Consider the hypothesis space $\mathcal{P}_k$ consisting of *parity functions* with at most k variables. Each parity function in this set is defined by selecting at most k of the n variables, and taking the XOR of them. For example, if $k = 3$, the set $\mathcal{P}_3$ would contain functions such as x_1 xor x_{12}, x_{n-2} xor x_{n-1} xor x_n, and so on.

   (a) [5 points] What is the number of functions in the set $\mathcal{P}_k$?

   (b) [5 points] If we use the Halving algorithm we saw in class, what is the upper bound on the number of mistakes that the algorithm would make? You can use the result we derived in class and state your answer in terms of n and k.

2. Consider the hypothesis space $\mathcal{H}$ consisting of *indicator functions* for Boolean vectors in the n-dimensional space. That is, for every vector $\mathbf{z} \in \{0,1\}^n$, the set $\mathcal{H}$ will contain a function $f_{\mathbf{z}}$ defined as:

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

   Note that each function $f_{\mathbf{z}}$ is defined by a specific vector $\mathbf{z}$, and produces the value 1 only when the input is that vector $\mathbf{z}$. For all other inputs, it produces 0.

   (a) [5 points] What is the number of functions in $\mathcal{H}$?

   (b) [20 points] Suppose you ran the Halving algorithm for this hypothesis space. How many mistakes will the algorithm make? You will need to justify your answer with a proof.

   (To get an intuition, instantiate the problem for a small n, say 2, and enumerate all functions in the hypothesis space. Now, pick one of these functions as the oracle function, and step through the Halving algorithm for some sequence of examples.)

   (c) [5 points] Is the mistake bound you derive above the same as what we saw in class for the general Halving algorithm? If so, construct a sequences of examples that forces the algorithm to make those many mistakes for any dimensionality. If not, explain why there appears to be a discrepancy between the two mistake bounds?

3. [5 points] Now, consider a different hypothesis space $\mathcal{C}$ that is defined very similarly to $\mathcal{H}$ above, except that it switches the 0's and 1's in the output of the functions. That is, the set $\mathcal{C}$ consists of functions $g_{\mathbf{z}}$ of the form:

$$g_{\mathbf{z}}(\mathbf{x}) = \begin{cases} 0, & \mathbf{x} = \mathbf{z}, \\ 1, & \text{otherwise.} \end{cases} \tag{2}$$

Note that the output of the function $g_{\mathbf{z}}$ is the negation of the output of the function $f_{\mathbf{z}}$ in $\mathcal{H}$ in the previous question.

Is the mistake bound for the hypothesis space $\mathcal{C}$ the same as the one you derived above for $\mathcal{H}$? Prove your claim.

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

We will experiment with the `Badges` task discussed in Lecture 1. We have created a new version of the data based on names of authors from the International Conference on Machine Learning (ICML) 2018 and 2019.

We have labeled these names as -1 or $+1$ using a certain (hidden) function, and the goal of this question is to train linear classifiers that attempt to mimic the hidden function.

To help with this, we have extracted 206 features from each of these names. The data has been preprocessed into two standard formats: LIBSVM, and CSV.

1. LIBSVM: Use the training/test files called `train` and `test` in the directory `libSVM-format/`. 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.

2. CSV: Use the training/test files called `train.csv` and `test.csv` in the directory `csv-format/`. This format contains the full feature vector representation - all 206 features - instead of a sparse representation. Again, each row is a single training example, where the first column is the label followed by 206 values for features.

In case you are curious (and perhaps want to play with additional features), we have also provided the actual names on the badges in our dataset. These can be found in directory `raw-data/`.

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 `fold1`–`fold5` in the folder `CVFolds` (inside both the `csv-format/` and `libSVM-format/` directories).

(If you use a random number generator to initialize weights or shuffle data during training, please set the random seed in the beginning to a fixed number so that different runs produce the same results. Each language has its own way to set the random seed. For example, in `python`, you can set the seed to be, say the number 42, using `random.seed(42)`.)

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. **Decaying the learning rate:** Instead of fixing the learning rate, implement a version of the Perceptron algorithm 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 increasing it by one each time an epoch is completed.)

   Hyper-parameter: Initial learning rate $\eta_0 \in \{1, 0.1, 0.01\}$

3. **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.

   Since we are applying averaging to the original Perceptron algorithm, we will have the same hyper-parameters, namely the learning rate η. Use the same set as above for this exploration.

   (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.)

4. **Margin Perceptron (For 6350 Students):** 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 $\mu \in \{1, 0.1, 0.01\}$

   Note: When there is more than one hyper-parameter to cross-validate, you need to consider all combinations of the hyper-parameters. In this case, you will need to perform cross-validation for all pairs (η_0, μ) from the above sets.

5. **Feature transformation (Extra credit for everyone)**: This is not a variant of the learning algorithm, but the feature representation. Construct a new feature space by taking the products of all pairs of features. As an illustration, if for some example, your original feature was the 3-dimensional vector $[1.0, 2.0, 3.0]$, your transformed feature vector would be the 6-dimensional vector with elements $[1.0 \times 1.0, 1.0 \times 2.0, 1.0 \times 3.0, 2.0 \times 2.0, 2.0 \times 3.0, 3.0 \times 3.0] = [1.0, 2.0, 3.0, 4.0, 6.0, 9.0]$.

   Pre-process the training, testing and cross-validation datasets into these transformed representations. Run the averaged Perceptron using the same set of hyperparameters as above. (You can choose to do the pre-processing once and save the transformed data to disk, or you could do the pre-processing within the learning loop whenever the vectors are used by the learner.)

3.3 Experiments

For all 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 training set. For the averaged Perceptron, use the average classifier to compute accuracy.

3. Use the classifier from the epoch where the training 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 training set?

3. [10 points per variant] For each variant above (4 for 6350 students, 3 for 5350 students, plus one extra credit), 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) Training 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 training 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.