Assignment 2
CSE 446: Machine Learning

University of Washington

Due: October 24, 2017

You will submit the following files or documents for this homework, compressed into a gzipped tarball named `A2.tgz`.

- Your **report** as a **pdf file** named `A2.pdf` containing answers to written questions. You must include the plots and explanation for programming questions (if required) in this document only. We *strongly* recommend typesetting your scientific writing using LaTeX. Some free tools that might help: ShareLaTeX (`www.sharelatex.com`), TexStudio (Windows), MacTex (Mac), TexMaker (cross-platform), and Detexify[2] (online). If you want to type, but don't know (and don't want to learn) LaTeX, consider using a markdown editor with real-time preview and equation editing (e.g., `stackedit.io`, `marxi.co`). Writing solutions by hand is fine, as long as they are neat; you will need to scan them into a single pdf.
  Part of the training we aim to give you in this advanced class includes practice with technical writing. Organize your report as neatly as possible, and articulate your thoughts as clearly as possible. We prefer quality over quantity. Do not flood the report with tangential information such as low-level documentation of your code that belongs in code comments or the README. Similarly, when discussing the experimental results, do not copy and paste the entire system output directly to the report. Instead, create tables and figures to organize the experimental results.
- **Python source code** for the programming assignment. Please note that we will not accept Jupyter notebooks. You will submit your code together with a neatly written README file to instruct how to run your code with different settings (if applicable). We assume that you always follow good practice of coding (commenting, structuring), and these factors are not central to your grade.

You are welcome to use any Python libraries for data munging, visualization, and numerical linear algebra. Examples includes Numpy, Pandas, and Matplotlib. You may **not**, however, use Python machine learning libraries such as Scikit-Learn or TensorFlow. If in doubt, email the instructors.

1 Perceptron Exercise [20 points]

Recall that a perceptron learns a linear classifier with weight vector $\mathbf{w}$. It predicts

$$\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x})$$

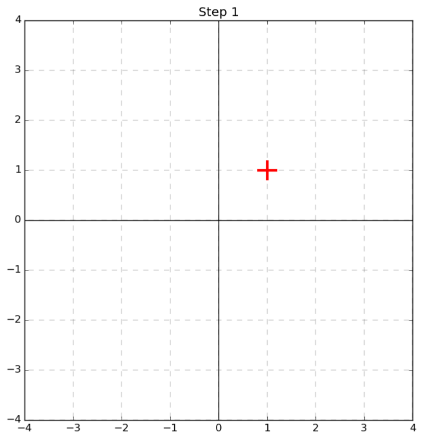
(We assume here that $\hat{y} \in \{+1, -1\}$. Also, note that we are *not* using a bias weight b.) When the perceptron makes a mistake on the nth training example, it updates the weights using the formula

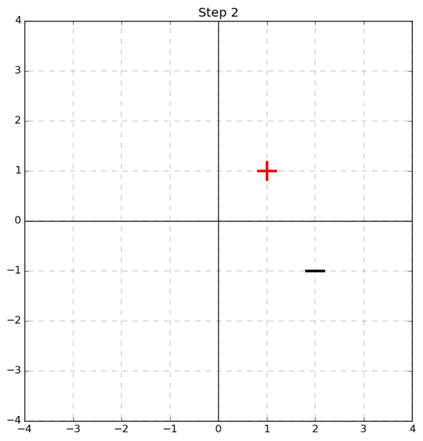
$$\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$$

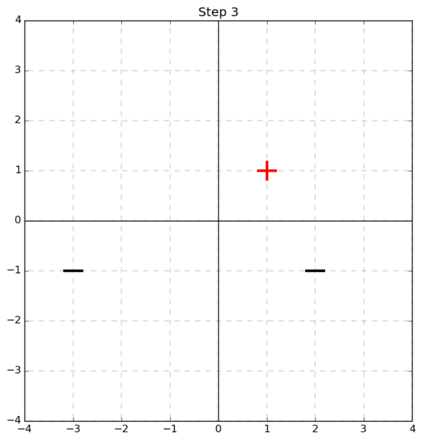
Imagine that we have each $\mathbf{x}_n \in \mathbb{R}^2$, and we encounter the following data points

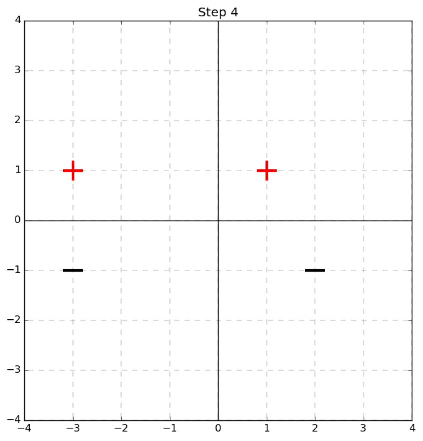
$\mathbf{x}[1]$	$\mathbf{x}[2]$	y
1	1	1
2	-1	-1
-3	-1	-1
-3	1	1

1. Starting with $\mathbf{w} = [0 \quad 0]^\top$, use the perceptron algorithm to learn on the data points in the order from top to bottom. Show the perceptron's linear decision boundary after observing each data point in the graphs below. Be sure to show which side is classified as positive. [10 points]

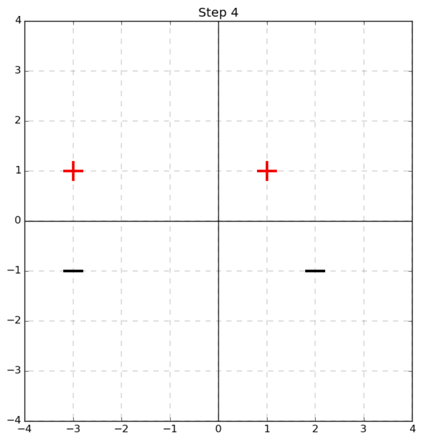








2. Does our learned perceptron maximize the margin between the training data and the decision boundary? If not, draw the maximum-margin decision boundary on the graph below. [5 points]



3. Assume that we continue to collect data and train the perceptron. If all data we see (including the points we just trained on) are linearly separable separable with margin $\gamma = 0.5$ and have maximum norm $\|\mathbf{x}\|_2 \leq 5$, what is the maximum number of mistakes we can make on future data? [5 points]

2 Implementation: Perceptron [70 points]

Implement the perceptron learning algorithm for binary classification, as described in class, from scratch, in Python. All of the files mentioned here are available in the tarball `A2.tgz` available on Canvas.

Implementation details:

- First, read in training data from a single file (we suggest that your program take the filename as a command-line argument) and store it in memory. You can assume for this assignment that you'll never have more than 1000 training examples.
- At the end of each epoch, report the number of mistakes made on that epoch to standard out.
- Test data will always be in a separate file, optionally provided on the command-line. At the end of training, report the number of mistakes on the test data (if there is a test dataset).
- The format for training and testing data will always be one-instance-per-line, formatted as:

  y `tab` $\mathbf{x}[1]$ `tab` $\mathbf{x}[2]$ `tab` $\ldots$ `tab` $\mathbf{x}[d]$ `newline`

  where `tab` is a single tab character and `newline` is a single newline character.
- If there's other diagnostic output you'd like to generate, send it to standard error. This will make it easier in case we need to run your code.

As a sanity check, train the perceptron on the dataset in `A2.1.tsv`, which has 50 positive and 50 negative examples and is separable by the hyperplane corresponding to weights:

$$\mathbf{w}^* = [-1.2 \quad 2.7 \quad -7.6 \quad -2.8 \quad -2.8]^\top$$

	question number			
dataset	2	3	4	5
2				
3				
4				
5				
6				
7				
8				
9				

Table 1: Write your answers on part 2 into a table like this one.

You should be able to converge to a weight vector that achieves zero training-set error in about 20 epochs; your weight vector might not be identical to $\mathbf{w}^*$, of course.

The tarball for this assignment contains eight additional training datasets (`A2.2.train.tsv`, `A2.3.train.tsv`, ..., `A2.9.train.tsv`). Each one has a corresponding test dataset (files `A2.*.test.tsv`). For each dataset [8 points per dataset]:

1. Generate a plot in which the x-axis is the epoch, and the y-axis shows the training error rate in blue and the test error rate in red, at the end of that epoch, where you train on *all of the training data*. We suggest that you train for 1000 or more epochs. You might want to also include a plot that doesn't show all epochs (only earlier ones) so that the "interesting" trends are visible. You might also want to generate a more fine-grained plot that reports training and test error rates after every iteration of the *inner* loop (i.e., after each prediction the perceptron makes on a training instance); don't include such a plot in your writeup unless it shows something interesting we couldn't see in the first plots.
2. Do you believe that the training dataset is linearly separable? If so, give a lower bound on the margin.
3. Do you believe that your learning algorithm overfit the training data? If so, how do you know?
4. Do you suspect that any features are noise? Which ones, and why? (Please index features from 1 to 20 in the order they appear in the data files.)
5. Carve out a development dataset from the training data and use it to tune the perceptron's hyperparameter(s). Report training, development, and test data accuracy, as well as the hyperparameter(s) you tuned and the value you selected for the hyperparameter(s).

Present your answers to the non-plotting questions in a table formatted like Table 1.

Surprise! [6 points] You may not have noticed that test sets 6–8 are identical (but they are). The three corresponding training datasets increase in size from 6 to 7 and 7 to 8, but: (i) they don't overlap and (ii) they do come from the same source. What do you notice when you compare your findings for these three datasets? Can you use this information to get better performance on test set 6 (equivalently, test set 7, equivalently test set 8), still using your perceptron implementation? If so, go for it and give answers to the same questions as above (1–5).

3 Beat the Perceptron [10 points]

Choose one of the datasets (2–9) where the perceptron's test-set performance was not strong, and try to improve using any of the following:

- Decision stump
- K-nearest neighbors
- Voted perceptron

If there are hyperparameters, you should tune them on your development data, **not the test data**.

Tell us which dataset (2–9) you focused on and why. Fully (and concisely) explain the exploration and choices you made as you tuned hyperparameters. Report on how well you were able to perform on the test set, making a fair comparison to the perceptron. It's fine to test more than one of the methods listed above, but you must implement everything yourself and you must report on all of the algorithms you tried (no tuning on test data, even to select the algorithm!). You are only required to test one of them.

4 Bonus [up to 10 extra points]

For the dataset you considered in part 3, reduce its dimensionality using PCA and use the resulting features instead of the original ones, with both the perceptron and the other algorithm(s) you implemented. *For this problem only,* you may use an existing implementation of PCA. For any hope of the maximal number of extra points, you must give a convincing argument that PCA offered some benefit (and explain the benefit), or that it could not help. Be sure to explain how you chose K. Plots that help make your argument more clear are encouraged.

Hints. Remember that PCA is an unsupervised learning algorithm; if you use the ys, you are doing it wrong and will get zero extra points. Also remember to center your data.