Homework 1

571: Probabilistic Machine Learning

Due: January 24 10.00pm

Please submit your report in a single PDF file (you may use LaTeX, MS Office or any other editor of your choice) and any code you have produced. You may use any programming language, however make sure to comment your code. Please read the Course Outline for the homework rules and policy.

1 Perceptron Algorithm and Convergence Analysis (40 points)

1. Perceptrons can be used to represent many Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$ in n-dimensional space.

(a) Provide a two-input Boolean function $y = f(x_1, x_2)$, where $y, x_1, x_2 \in \{0,1\}$, that can be represented by a single perceptron. Plot the points (x_1, x_2) that represent all possible pair values (e.g. $(0,0), (0,1), (1,0), (1,1)$) and provide a separating hyperplane on the same figure.

(b) Provide a two-input Boolean function $y = f(x_1, x_2)$ that can not be represented by a single perceptron. Briefly explain why.

(c) Provide a three-input Boolean function $y = f(x_1, x_2, x_3)$, where $y, x_1, x_2, x_3 \in \{0,1\}$, that can be represented by a single perceptron. Plot the points (x_1, x_2, x_3) that represent all possible pair values and provide a separating hyperplane on the same figure.

2. For a response variable $y \in \{-1, 1\}$ and a linear classification function $f(x) = \beta_0 + \beta^T x$, suppose that we classify according to $sign(f(x))$. Show that the signed Euclidean Distance of the point x with label y to the decision boundary is given by

$$\frac{1}{||\beta||_2} y f(x)$$

3. Suppose we have n points $x_i \in \mathbb{R}^p$ with class labels $y_i \in \{-1, 1\}$. Points are correctly classified by a hyperplane $w^{sep^T} x = 0$. Assuming $y_i w^{sep^T} x_i \geq 1$ for all i, show that the perceptron algorithm converges to a separating hyperplane in no more than $||w^{(0)} - w^{sep}||_2^2$ steps.

2 Programming Assignment (60 points)

In this problem set, you will implement two algorithms – Perceptron and Balanced Winnow – in order to classify the 4 and 9 handwritten digits from the MNIST dataset. Each algorithm will have 784 inputs (image pixels represented as a column vector) and one output.

For the MNIST dataset (http://yann.lecun.com/exdb/mnist/) filter out all observations that are not labeled **4** or **9**. In order to keep the weights small make sure data values are scaled

between 0 and 1. If you are using a random number generator, set the seed of the generator to 2018. Split the data into training and testing datasets.

1. Write a function called `perceptron` that takes as an input a training set $S = \{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$, $i = 1..n$ and a maximum number of epochs I. The weights w are initialized as $w = (0, ..., 0)$. The output of the function should be a weight vector w after convergence to a separating hyperplane or after a maximum number of epochs is reached. Algorithm 1 contains pseudo code for the perceptron algorithm.

(a). Run the function `perceptron` on the training set and plot the evolution of the accuracy versus the epoch counter. What do you observe? What will happen if you increase or decrease the maximum number of epochs?

(b). Plot the evolution of testing dataset accuracy versus the epoch counter (use the same figure as in part (a)). What do you observe? How does it compare to the previous curve?

(c). The *confusion matrix* is a table that allows to visualize the performance of a classifier on a testing dataset and reports the number of false positives, false negatives, true positives, and true negatives. In particular one can compute entries of a confusion matrix as follows:

	Actual yes: $y = +1$	Actual no: $y = -1$
Predicted yes: $\hat{y} = +1$	TP	FP
Predicted no: $\hat{y} = -1$	FN	TN

Report the accuracy and confusion matrix of the perceptron algorithm on the testing set after the last epoch.

(d). Start running the perceptron algorithm and stop after one third of the dataset has been processed. Save the current weight vector w'. Continue running algorithm till the convergence or a maximum number of epochs is reached and save the weight vector w^*. Use w' and w^* to create two ROC curves on the training data and display them on the same figure. Based on ROC curves provide an argument to support a claim that weight vector w^* leads to a better decision boundary.

You may create the ROC curve by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold b setting, where

$$TPR = \frac{TP}{TP + FN} \quad FPR = \frac{FP}{FP + TN}$$

To compute the (FPR, TPR) pairs you should modify a classification rule. In particular, algorithm with an unknown weight vector w and known parameter b classifiers a data point (x, y) correctly if $y(w^T x - b) > 0$.

Please do not use build-in functions for ROC computations.

(e). Use any approximation technique (e.g. Riemann sum, trapezoidal rule) to approximate AUC. Report AUC values for w^* and w'. How do the values you received correspond to the ROC curves in part (c)?

2. Write a function called `Balanced Winnow` that takes as an input a training set $S = \{(x_i, y_i)\}_{i=1}^n$, where $x_i \in R^d$ and $y_i \in \{-1, 1\}$, $i = 1..n$ and a maximum number of epochs I.

Algorithm 1 Perceptron

Input: $\{(x_i, y_i)\}_{i=1}^n$
Initialize: $w = (0, ..., 0)$, $I = 100$
for e=1 to I **do**
  for i=1 to N **do**
    if $y_i w^T x_i \leq 0$ **then**
      $w = w + y_i x_i$
    end if
  end for
end for

The weights $w = [w_p, w_n]$ are initialized as $w = (\frac{1}{2p}, ..., \frac{1}{2p})$. The output of the function should be a weight vector w after convergence to a separating hyperplane or after a maximum number of epochs is reached. Algorithm 2 contains pseudo code for the Balanced Winnow algorithm.

Algorithm 2 Balanced Winnow

Input: $\{(x_i, y_i)\}_{i=1}^n$
Initialize: $w_p = (1/2p, ..., 1/2p)$, $w_n = (1/2p, ..., 1/2p)$, I
for e=1 to I **do**
  for i=1 to N **do**
    if $y_i(w_p^T x_i - w_n^T x_i) \leq 0$ **then**
      $w_p = w_p e^{\eta y_i x_i}$
      $w_n = w_n e^{-\eta y_i x_i}$
      $s = \sum_j w_n^j + w_p^j$
      $w_p = w_p / s$
      $w_n = w_n / s$
    end if
  end for
end for

(a). Run the function `Balanced Winnow` on the training set and plot the evolution of the accuracy versus the epoch counter. Report the accuracy and confusion matrix on the test set.

(b). Based on your experiments, is there a value of the parameter η that is better than the others? What technique would you use to tune the value of the parameter η?