COMPSCI 671D Fall 2020
Homework 3

Due 10:15 PM EST, October 21st

For coding questions, please make sure your results and code derivations are the finalized version in Jupyter notebook before converting it into a PDF file and upload to Gradescope.

1 Statistical Learning: Hoeffding's Inequality

(a) The Chernoff Bound (which was not actually due to Chernoff) can be defined as:

$$Pr(X \geq \mu_X + t) \leq \min_{\lambda \geq 0} M_{X-\mu_X}(\lambda) e^{-\lambda t}, \tag{1}$$

where X is a random variable and $\mu_X = \mathbb{E}[X]$ is the mean and $M_X(\lambda) = \mathbb{E}[e^{\lambda X}]$ is the moment generating function, prove that, for any $t \geq 0$ equation (1) holds. Hint: you are permitted to use Markov's inequality (which was not actually due to Markov).

(b) Let's define Hoeffding's Lemma: Let X be a bounded random variable with $X \in [a, b]$. Then

$$\mathbb{E}[e^{\lambda(X-\mu_X)}] \leq \exp\left(\frac{\lambda^2(b-a)^2}{8}\right), \text{ for all } \lambda \in \mathbb{R}.$$

Use the Chernoff Bound and Hoeffding's Lemma to prove the powerful Hoeffding's Inequality

$$Pr\left(\frac{1}{n}\sum_{i=1}^{n}(X_i - \mu_{X_i}) \geq t\right) \leq \exp\left(-\frac{2nt^2}{(b-a)^2}\right), \text{ for all } t \geq 0.$$

where $X_1, ..., X_n$ are independent random variables with $X_i \in [a, b]$ for all i.
Just so you know, most of statistical learning theory revolves around Hoeffding's inequality (in various forms).

(c) Hoeffding's inequality can be very loose in certain cases (i.e., the left hand side of the inequality is very small compared to the right hand side). Please give a simple distribution of X_i where this is the case (*Hint:* Hoeffding's inequality only depends on the upper and lower bounds of X_i).

2 SVM and the power of kernels

The kernel trick states that if we have an algorithm, like in this particular problem the SVM, where the examples/observations appear only in inner products, you can freely replace the inner product with a different one. In other words, we replace $x_i^T x_l$ (i.e., $\langle x_i, x_l \rangle_{\mathbb{R}^p}$) with $k(x_i, x_l)$, where k (kernel) happens to be an inner product in some feature space, $k(x_i, x_l) = \langle \Phi(x_i), \Phi(x_l) \rangle_{\mathbb{H}_k}$.

(a) Suppose we have a dataset $\{x_i, y_i\}_{i=1}^n, x_i \in \mathbb{R}^p, y_i \in \{-1, 1\}$ with no conflicting labels, where a conflicting label means $\exists i, j \quad s.t. \quad x_i = x_j, y_i \neq y_j$. Prove that there exists an SVM classifier with an **rbf** kernel that can achieve 0 training error. (You may use asymptotic arguments if you are more comfortable with those. You must show that the predictions $\hat{y}_i$ for each point are correct.)

(b) Solve the 1d separable case manually. Put positives on the left, negatives on the right. Use a as the position of the rightmost positive point. Use b as the position of the leftmost negative point. Provide the formulas for (i) the dual variables, as well as $f(x)$, as a function of a and b and (ii) the position of the decision boundary. (iii) In the process of this calculation, you will prove that the number of support vectors is not equal to 1 in this case. Can it ever be 1 in any case? Please provide your logic.

(c) For this part of the problem, you are allowed to use built-in functions in the the programming language you are using (i.e., *Pandas* or *Scikit-Learn* if you are using *Python*, but beware of the solver in *Scikit-Learn*). Train an SVM classifier with the **rbf** kernel function on 9/10ths of the credit card data set (Please set seed for the random number generator as 671 and choose randomly which tenth to leave for testing). Try several values of γ (i.e, $\gamma \in [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]$ and for the purpose of this problem keep $C = 1$ for all your SVM classifiers) and calculate the training and test error rates (you can present these values in a table or plot them). Does the training error increase or decrease as γ increases?. Why this is the case? Finally, in the bias variance tradeoff, does small γ correspond to high bias or high variance?.

3 Perceptron: Theoretical

Consider applying the perceptron algorithm (through the origin) to a small training set containing three points: $\mathbf{x}_1 = [-1, 1]$, $\mathbf{x}_2 = [0, -1]$, and $\mathbf{x}_3 = [1.5, 1]$ with labels $y_1 = 1$, $y_2 = -1$, and $y_3 = 1$. Given that the algorithm starts with $\mathbf{w} \equiv \mathbf{0}$, the first point that the algorithm sees is always considered a mistake. Today, the algorithm starts with one data point and then cycles through the data *in order* until it makes no further mistakes. When it gets to the end of the dataset, it starts from the beginning again.

(a) How many mistakes does the algorithm make until convergence if the algorithm starts with data point $\mathbf{x}_1$? How many mistakes does the algorithm make if it starts with data point $\mathbf{x}_2$? Draw a diagram showing the progression of the decision boundary as the algorithm cycles. (The drawing can be somewhat approximate to still get full credit. You can draw it on a piece of paper and take a picture of the picture with your phone or you could do it in powerpoint or another drawing software.)

(b) We change the position of $\mathbf{x}_3$ so that it is $\mathbf{x}_3 = [10, 1]$. How many mistakes does the algorithm make until convergence if cycling starts with data point $\mathbf{x}_1$? How many if it starts with data point $\mathbf{x}_2$? Draw a diagram showing the progression of the plane as the algorithm cycles.

(c) Briefly describe an adversarial procedure for selecting the order of labeled data points so as to maximize (or at least make large) the number of mistakes the perceptron algorithm makes before converging. Assume that the data is indeed linearly separable, by a hyperplane that you (but not the algorithm) know. General ideas are fine here, you do not need to program

anything or compute anything.

4 Perceptron and Balanced Winnow: Programming

In this problem, 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/ and also uploaded to hw3 folder in Sakai) Split the data into training and testing datasets so you have 60,000 images and 10,000 respectively and 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 671.

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 ≤ 0 then
            w = w + y_i x_i
        end if
    end for
end for
```

(a) Write a function called `perceptron` that takes as an input a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$, $i = 1..n$ and a maximum number of epochs I. The weights $\mathbf{w}$ are initialized as $\mathbf{w} = (0, ..., 0)$. The output of the function should be (1) a weight vector $\mathbf{w}$, after convergence to a separating hyperplane or after a maximum number of epochs is reached and (2) the training accuracy. Algorithm 1 contains pseudocode for the perceptron algorithm.

(b) Run the function `perceptron` on the training set ($I = 100$) and plot the evolution of the training accuracy versus the epoch counter. What do you observe? What will happen if you increase or decrease the maximum number of epochs?

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

(d) Write a function called `Balanced Winnow` that takes as an input a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$, $i = 1..n$, a maximum number of epochs I and a hyperparameter η. The weights $\mathbf{w} = [\mathbf{w}_p, \mathbf{w}_n]$ are initialized as $\mathbf{w} = (\frac{1}{2p}, ..., \frac{1}{2p})$. The output of the function should be (1) a weight vector $\mathbf{w}$, after convergence to a separating hyperplane or after a maximum number of epochs is reached and (2) the training accuracy. Algorithm 2 contains pseudocode for the Balanced Winnow algorithm.

(e) Run the function `Balanced Winnow` on the training set (as before) ($\eta = 0.1$ and $I = 30$) and plot the evolution of both the train and test accuracy versus the epoch counter. How many epochs do we need for the algorithm to reach a stable state?

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