CSE 417T: Homework 5

Due: Dec 4 (Monday), 2023

Notes:

- Please submit your homework via Gradescope and check the submission instructions.
- Please download the following files for this homework.
  `https://wustl.instructure.com/courses/113022/files/folder/hw5`
- Homework is due **by 11:59 PM on the due date.** Remember that you may not use more than 2 late days on this homework, and you only have a budget of 5 in total.
- Make sure you **specify the pages for each problem correctly**. You **will not get points** for problems that are not correctly connected to the corresponding pages.
- Please keep in mind the collaboration policy as specified in the course syllabus. If you discuss questions with others you **must** write their names on your submission, and if you use any outside resources you **must** reference them. **Do not look at each others' writeups, including code.**
- Please comment your code properly.
- There are 6 problems on 2 pages in this homework.

Problems:

1. (30 points) The purpose of this problem is to implement AdaBoost and observe its performance on training and test errors. Please use the same handwritten digit recognition dataset as in homework 4. Again focus on the same two binary classification problems – distinguishing between the digit one and the digit three, and distinguishing between the digit three and the digit five. You need to report the results for both problems (1 versus 3 and 3 versus 5).

   - Code (Complete/submit `hw5.py`): Use decision stumps learned using information gain as the weak learner[1]. You may use `sklearn.tree.DecisionTreeClassifier`[2] to help learn decision stumps. Read the document carefully to understand how to learn a decision stump from a *weighted* dataset.
   - Report: For each of the binary classification problems, graphically report the training set error and the test set error as a function of the number of weak hypotheses (from $t = 1$ to 200). You might want to include training/testing error curves in the same

[1] We choose information gain as the criteria for consistency reasons. You can consider it as an *approximation* of the weak learner that minimizes the weighted in-sample error.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

plot (i.e., one plot for each classification problem) for easy comparison. Summarize and interpret your results. You should at least comment on the trend of training/test errors and whether AdaBoost is overfitting as the number of weak learners increases.

2. (5 points) Suppose your input data consists of the following (x, y) pairs:

   $(3, 5); (5, 6); (7, 9); (2, 11); (3, 8)$

   What value of y would you predict for a test example where $x = 3.2$ using the 3-nearest neighbors regression?

3. (15 points) (From Russell & Norvig) Construct a support vector machine that computes the XOR function. $\vec{x} = (x_1, x_2)$ denotes the two inputs and y denotes the output. Use $+1$ and -1 to represent boolean variables in this question (so the notations are consistent with our lectures). Map the input (x_1, x_2) into a space consisting of x_1 and x_1x_2. Draw the four input points in this space, and the maximal margin separator. What is the margin? Now draw the separating line back in the original Euclidean input space.

4. (15 points) The key point of the so-called "kernel trick" in SVMs is to learn a classifier that effectively separates the training data in a higher dimensional space without having to explicitly compute the representation $\Phi(\vec{x})$ of every point $\vec{x}$ in the original input space. Instead, all the work is done through the kernel function that computes dot products $K(\vec{x}_i, \vec{x}_j) = \Phi(\vec{x}_i)^T\Phi(\vec{x}_j)$.

   For any two points $\vec{x}_i$ and $\vec{x}_j$, show how to compute the *squared Euclidean distance* of the two points in the projected space, $\Phi(\vec{x}_i)$ and $\Phi(\vec{x}_j)$, without explicitly computing the Φ mapping. Instead, write down the squared Euclidean distance using the kernel function K.

5. (20 points) Construct by hand a neural network with only one hidden layer (of any number of units) that implements XOR(AND$(x_1, x_2), x_3)$. You can use sign function as the activation function. Draw your network, and show all weights of each unit. (You may find it useful to first simplify the Boolean formula using common Boolean identities, such as De Morgan's law.)

6. We will discuss the initialization of gradient descent for Neural Networks in this question.

   a (10 points) Adapted from LFD Exercise 7.7 (from e-Chapter 7):
   For the *sigmoidal perceptron*, i.e., $h(\vec{x}) = \tanh(\vec{w}^T\vec{x})$, let the in-sample error be $E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N}(\tanh(\vec{w}^T\vec{x}_n) - y_n)^2$. Show that

   $$\nabla E_{in}(\vec{w}) = \frac{2}{N}\sum_{n=1}^{N}(\tanh(\vec{w}^T\vec{x}_n) - y_n)(1 - \tanh^2(\vec{w}^T\vec{x}_n))\vec{x}_n$$

   If $\vec{w}$ consists of very, very large weights, what happens to the gradient (hint: consider the extreme case that the weights approach ∞)? Based on your answer, when using gradient descent to optimize the weights of a sigmoidal perceptron, is it a good idea to initialize the weights to be very, very large?

   b (5 points) LFD Problem 7.10 (from e-Chapter 7).