CS 189 Introduction to Machine Learning
Spring 2022 Marvin Zhang

HW4

Due 4/5/22 at 11:59pm

- Homework 4 consists of both written and coding questions.
- We prefer that you typeset your answers using LaTeX or other word processing software. If you haven't yet learned LaTeX, one of the crown jewels of computer science, now is a good time! Neatly handwritten and scanned solutions will also be accepted for the written questions.
- In all of the questions, **show your work**, not just the final answer.

Deliverables:

1. Submit a PDF of your homework ,**with an appendix listing all your code**, to the Gradescope assignment entitled "HW4 Write-Up". **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.
    - In your write-up, please state with whom you worked on the homework. This should be on its own page and should be the first page that you submit.
    - In your write-up, please copy the following statement and sign your signature next to it. (Mac Preview and FoxIt PDF Reader, among others, have tools to let you sign a PDF file.) We want to make it *extra* clear so that no one inadvertently cheats. *"I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."*
    - **Replicate all your code in an appendix**. Begin code for each coding question in a fresh page. Do not put code from multiple questions in the same page. When you upload this PDF on Gradescope, *make sure* that you assign the relevant pages of your code from appendix to correct questions.

1 Administrivia (2 points)

1. Please fill out the Check-In Survey if you haven't already. Please write down the 10 digit alphanumeric code you get after completing the survey.

2. **Declare and sign the following statement:**

   "I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted."

   Signature: ______________

   While discussions are encouraged, *everything* in your solution must be your (and only your) creation. Furthermore, all external material (i.e., anything outside lectures and assigned readings, including figures and pictures) should be cited properly. We wish to remind you that consequences of academic misconduct are particularly severe

2 Running Time of k-Nearest Neighbor and Kernelized Nearest Neighbor Search Methods (10 points)

The method of k-nearest neighbors is a fundamental conceptual building block of machine learning. A classic example is the k-nearest neighbor classifier, which is a non-parametric classifier that finds the k closest examples in the training set to the test example, and then outputs the most common label among them as its prediction. Generating predictions using this classifier requires an algorithm to find the k closest examples in a possibly large and high-dimensional dataset, which is known as the k-nearest neighbor search problem. More precisely, given a set of n points, $\mathcal{D} = \{\mathbf{x}_1 \ldots, \mathbf{x}_n\} \subseteq \mathbb{R}^d$ and a test point $\mathbf{q} \in \mathbb{R}^d$, there are 2 steps to decide what class to predict:

1. Find the k training points nearest $\mathbf{q}$.
2. Return the class with the most votes from the k training points.

k-nearest neighbors takes no time to train, as we simply use the data points that we are given to make inferences. Since there is no training time, it is reasonable to question whether the classification time is absurdly large. In this problem, we will analyze the runtime of k-nearest neighbors to assess its viability in practical applications.

(a) **(4 points)** What is the runtime to classify a newly given test point q, using Euclidean distance? (**Hint:** Use heaps to keep track of the k smallest distances.)

(b) **(2 points)** Let us now 'lift' the points into a higher dimensional space by adding all possible mononomials of the original features with degree at most p. What dimension would this space be? What would the new runtime for classification of a test point q be, in terms of n, d, k, and p?

(c) **(4 points)** Instead, we can use the polynomial kernel to compute the distance between 2 points in the 'lifted' $O(d^p)$-dimensional space without having to move all of the points into the higher dimensional space. Using the polynomial kernel, $k(x, y) = (x^T y + \alpha)^p$ instead of Euclidean distance, what is the runtime for k-NN to classify a test point q?

3 Random Forest Motivation (10 points)

Ensemble learning is a general technique to combat overfitting, by combining the predictions of many varied models into a single prediction based on their average or majority vote.

(a) **(4 points) The motivation of averaging.** Consider a set of uncorrelated random variables $\{Y_i\}_{i=1}^n$ with mean μ and variance σ^2. Calculate the expectation and variance of their average. (In the context of ensemble methods, these Y_i are analogous to the prediction made by classifier i.)

(b) **(4 points) Ensemble Learning – Bagging.** In lecture, we covered bagging (Bootstrap AGGregatING). Bagging is a randomized method for creating many different learners from the same data set.

Given a training set of size n, generate T random subsamples, each of size n', by sampling with replacement. Some points may be chosen multiple times, while some may not be chosen at all. If $n' = n$, around 63% are chosen, and the remaining 37% are called out-of-bag (OOB) samples.

  (a) Why 63%?

  (b) If we use bagging to train our model, How should we choose the hyperparameter T? Recall, T is the number of subsamples, and typically, a few dozen to several thousand trees are used, depending on the size and nature of the training set.

(c) **(2 points)** In part (a), we see that averaging reduces variance for uncorrelated classifiers. Real-world prediction will of course not be completely uncorrelated, but reducing correlation among decision trees will generally reduce the final variance. Reconsider a set of correlated random variables $\{Z_i\}_{i=1}^n$. Suppose $\forall i \neq j$, $\mathrm{Corr}(Z_i, Z_j) = \rho$. Calculate the variance of their average.

4 Decision Trees for Spam Classification (26 points)

In this problem, you will implement a decision tree for classification on the spam dataset to detect spam emails. The data is with the assignment. You are not allowed to use an off-the-shelf implementation for your decision tree. To implement the decision tree, you are only allowed to use numpy.

In your submission, make sure to attach a pdf copy of your jupyter notebook so that we can examine your code and grade your submission.

(a) **(5 points) Entropy Criterion** Decision trees use a criterion to determine which feature to split on at each node. For this exercise, we will be using entropy as the decision criterion (rather than gini impurity). Fill in the code for the `entropy()` and `information_gain()` functions

(b) **(2 points) Splitting Data** Complete the `feature_split()` function to split the data into two halves based on a specific feature at a specific threshold. The process for this should be similar to the splitting process in information gain.

(c) **(5 points) Finding Splits** Next, we want to use the information gain to determine which feature to split on at each node, and at what threshold value. Fill in the code for the `find_feature_thresh()` and `find_best_feature_split()` functions.

For `find_feature_thresh()`, choose the threshold from 10 evenly spaced values between $(min\_value + eps)$ and $(max\_value - eps)$ for the feature. We need to include the eps terms because without it, there is a chance that the training algorithm will split the data on the min or max value, which does not yield a meaningful node split.

(d) **(10 points) Fit the Decision Tree** Fill in the code for the `fit()` function. If implemented correctly, your decision tree should make the same predictions as the sklearn decision tree, and have roughly 80% train accuracy and 79% validation accuracy.

The fit function for the decision tree should be recursive. In other words, inside the `fit()` function you should be calling `left_subtree.fit() and right_subtree.fit()` to continue building the decision tree.

(e) **(4 Points) Depth vs. Performance** How does the depth of the decision tree affect the training and validation accuracy? Train multiple decision trees with depths ranging from 1 to 39, and plot the training/validation accuracy with respect to tree depth. Comment on your findings