CIS 472/572, Winter 2018
Homework 3 (Written): Nearest Neighbors
DUE DATE: Submit via Gradescope
by Wednesday, February 7th at 11:00pm.

1. An analyst wants to classify a number of customers based on some given attributes: total number of accounts, credit utilization (amount of used credit divided by total credit amount), percentage of on-time payments, age of credit history, and inquiries (number of time that the customer requested a new credit account, whether accepted or not). The analyst acquired some labeled information as shown in the following table:

ID	Total Accounts	Utilization	Payment History	Age of History (days)	Inquiries	Label
1	8	15%	100%	1000	5	GOOD
2	15	19%	90%	2500	8	BAD
3	10	35%	100%	500	10	BAD
4	11	40%	95%	2000	6	BAD
5	12	10%	99%	3000	6	GOOD
6	18	15%	100%	2000	5	GOOD
7	3	21%	100%	1500	7	BAD
8	14	4%	100%	3500	5	GOOD
9	13	5%	100%	3000	3	GOOD
10	6	25%	94%	2800	9	BAD

Consider the following three accounts to be labeled:

Total Accounts	Utilization	Payment History	Age of History (days)	Inquiries	Label
20	50%	90%	4500	12	P1
8	10%	100%	550	4	P2
9	13%	99%	3000	6	P3

   (a) Before using nearest neighbor methods to make predictions, how would you recommend processing or transforming the data? Why? Make any changes you think appropriate to the data before continuing on to the next two parts.

   (b) What are the predicted labels P1, P2, and P3 using 1-NN with L_1 distance? Assume that percentages are represented as their corresponding decimal numbers, so 95% = 0.95. Show your work.

   (c) Keep the information of customers 7, 8, 9, and 10 as validation data, and find the best K value for the K-NN algorithm. If the best value of K is not equal to 1, find the new predictions for P1, P2, and P3. Show your work.

2. This exercise demonstrates how the performance of nearest neighbor degrades as you add more dimensions. Please do the following in a programming language of your choice:

    1. Generate a "training" dataset with 100 points in 10 dimensions. For 50 of the points, the first dimension (x_1) should be 0; for the other 50, the first dimension should be 1. All other dimensions (x_2 through x_{10}) should be either 0 or 1, selected randomly with equal probability.
    2. Follow the same procedure from step 1 to generate a random "test" dataset.
    3. Count the fraction of test points that would be correctly labeled using 1-nearest neighbor with this training data. (One way to do this is by checking if the closest training point to each test point has the same value for the first dimension, x_1.) This is the accuracy of nearest neighbor on this synthetic data.
    4. Repeat steps 1-3 10 times, and average the results. This is the average accuracy of nearest neighbor in 10 binary dimensions, with 1 perfectly informative attribute and 9 random ones.
    5. Repeat step 4 with 5, 10, 20, 50, and 100 dimensions. As before, every dimension should be random except for the first (x_1). Report the average accuracy for each number of dimensions.

    If you're familiar with numpy and Python list comprehensions, it's possible to do this exercise with just 30 lines of well-structured code. If you do everything from scratch or use a more verbose language, it's a few more lines of code but it still shouldn't be too difficult.

    For your answer to this question: please include:

    (a) the average accuracies obtained in step 5,

    (b) a graph showing the accuracy versus the number of dimensions, and

    (c) a printout of the code you used to help answer this question. (Your code must be your own work.)

3. When evaluating nearest neighbor models, we can compute the accuracy on the training data, holding out one training point at a time so that a point is not considered its own neighbor. This is often referred to as "leave-one-out cross-validation" (LOOCV).

    (a) Construct a dataset where 1-NN has an LOOCV accuracy of 0% but 3-NN has an LOOCV accuracy of 100%.

    (b) Describe a dataset with n points (for some $n \geq 10$ of your choice) where $(n-1)$-NN achieves LOOCV accuracy of 100%.

4. **(Grad students only.)** The multiclass perceptron maintains a weight vector and bias for each class: $(w_1, b_1), (w_2, b_2), \ldots, (w_k, b_k)$. When it makes an incorrect prediction, it adjusts the weight vectors of both the predicted class $\hat{y}$ and the correct class y:

$$\begin{aligned} w_y &\leftarrow w_y + x \\ b_y &\leftarrow b_y + 1 \\ w_{\hat{y}} &\leftarrow w_{\hat{y}} - x \\ b_{\hat{y}} &\leftarrow b_{\hat{y}} - 1 \end{aligned}$$

   Prove that the standard perceptron is equivalent to the multiclass perceptron when there are only 2 classes. In other words, show that the two methods always make the same predictions when given the same sequence of training examples.

5. **(Grad students only.)** Given a set of data points $x_1, \ldots, x_n$, we can define the convex hull to be the set of all points x given by

$$x = \sum_{i=1}^{n} \alpha_i x_i$$

   where $\alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i = 1$. Consider a second set of points $z_1, \ldots, z_n$ together with their corresponding convex hull. By definition, the two sets of points will be linearly separable if there exists a vector w and a scalar b such that $w^T x_i + b > 0$ for all x_i, and $w^T z_i + b < 0$ for all z_i.

   Prove that two sets of points are linearly separable if their convex hulls do not intersect.

   (**BONUS:** Prove that two sets of points are linearly separable *if and only if* their convex hulls do not intersect.)