CS 189 / 289A Introduction to Machine Learning
Fall 2022 Jennifer Listgarten, Jitendra Malik **HW4**

Due Tuesday 11/1 at 11:59pm

- Homework 4 consists of written and programming 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.
- You have slightly over a week to complete this homework.

Deliverables:

Submit a PDF of your homework 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. If you worked by yourself, state you worked by yourself. 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 underneath. If you are using LaTeX, you must type your full name underneath instead. 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 Eigenfaces (11 points)

In this question we will perform and analyze PCA on a dataset consisting of images of faces. Each datapoint, $\mathbf{x}$, consists of a flattened 62×47 pixel image (i.e. $\mathbf{x} \in \mathbb{R}^{2914}$). The dataset can be downloaded using the *sklearn* library as follows:

```
dataset = sklearn.datasets.fetch_lfw_people()
X = dataset['data']
```

This may take a few minutes to run. The file sizes are 250Mb so be sure that you have enough space on your computer to store the images. The *sklearn* library should only be used for downloading the data. For the rest of the problem you may use *matplotlib*, *numpy.**, *numpy.linalg.eig* and *numpy.linalg.svd*.

(a) (2 points) Plot the first 20 images to get familiar with the dataset.

Note: when plotting the images, be sure to reshape them to be a matrix of size 62 × 47. Images can be plotted with matplotlib.pyplot.imshow. The argument cmap=matplotlib.pyplot.cm.gray provides the best colormap to view the images

(b) (1 point) Recall that in order to perform PCA, we must first center our data. Compute the average face of the dataset, center the data, and plot the average face.

(c) (3 points) Perform PCA on the dataset. Plot the first 20 images reconstructed after being projected onto the top 10 principal component directions (PCDs). Do the same after projecting onto the top 100 PCDs and the top 1000 PCDs.

(d) (1 point) For this dataset, we refer to the PCDs as "eigenfaces". Plot the top 20 eigenfaces.

(e) (2 points) Recall from lecture that we can compute the percent variance explained by a certain number of principal components by using the eigenvalues of the covariance matrix. Specifically, letting $\Sigma = \frac{1}{n}\tilde{\mathbf{X}}^\top\tilde{\mathbf{X}}$ where $\tilde{\mathbf{X}} \in \mathbb{R}^{n\times d}$ is the mean-centered data matrix and $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_d \geq 0$ the eigenvalues of Σ, λ_i is the variance of the ith principal component. The percent variance explained by the first k principal components is

$$\frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^d \lambda_i} \times 100\%.$$

Note that $\lambda_i = \sigma_i^2$, where σ_i is the ith singular value of $\frac{1}{\sqrt{n}}\tilde{\mathbf{X}}$.

Plot the percent variance explained as a function of k and determine how many principal components are needed to explain 95% of the variance.

(f) (2 points) Use the first 80% of the dataset as your training set and the remaining 20% as the test set. We will use the training set to compute the PCDs and we will evaluate our reconstruction loss on both the training and test set. The reconstruction loss is defined as

$$\frac{1}{nd}\|\tilde{\mathbf{X}}\mathbf{V}_k\mathbf{V}_k^\top - \tilde{\mathbf{X}}\|_F^2$$

where $\mathbf{V}_k$ contains the k principal component directions and n and d are the dimensions of $\tilde{\mathbf{X}}$.

For the following number of PCDs, $[10, 20, 50, 100, 500, 1000, 2914]$, perform PCA using the training set and compute the average reconstruction loss for both the training and test set. Plot the error for both the training and test set as a function of the number of PCDs.

2 Running Time of k-Nearest Neighbor and Kernelized Nearest Neighbor Search Methods (6 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, $\mathcal{D} = \{x_1 \dots, x_n\} \subseteq \mathbb{R}^d$ and a test point $q \in \mathbb{R}^d$, there are 2 steps to decide what class to predict:

1. Find the k training points nearest q.
2. Return the class with the most votes from the k training points.

(a) (2 points) We can find the k nearest neighbors of q using the following procedure:

- (i) Insert the first k training points into a max heap, ordered by distance from q. The top (root) of the heap will be the furthest training point seen from q so far.
- (ii) For each remaining training point, if the distance to q exceeds the largest distance in the max-heap, skip it. Otherwise, pop off the root of the max-heap and insert the new training point into the heap.

What is the runtime to classify a newly given test point q, using Euclidean distance, using the above procedure to find its k-nearest neighbors? You may express your answer in big-O notation.

(b) (1 point) Let us now 'lift' the points into a higher dimensional space by adding all possible monomials of the original features with degree at most p. For example, for $d = 2$ and $p = 3$, we lift a point $[x, y]^\top$ to the point $[1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3]^\top$. 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? You may express your answers in big-O notation.

(c) (3 points) Decades of research have focused on devising a way of preprocessing the data so that the k-nearest neighbors for each query can be found efficiently. "Efficient" means the time complexity of finding the k-nearest neighbors is lower than that of the naïve exhaustive search algorithm—meaning that the complexity must be *sublinear* in n.

Many efficient algorithms for k-nearest neighbor search rely on a divide-and-conquer strategy known as space partitioning. The idea is to divide the feature space into cells and maintain a data structure that keeps track of the points that lie in each. Then, to find the k-nearest neighbors of a query, these algorithms look up the cell that contains the query and obtain the subset of points in $\mathcal{D}$ that lie in the cell and adjacent cells. Adjacent cells must be included in case the query point is in the corner of its cell. Then, exhaustive search is performed on this subset to find the k points that are the closest to the query.

For simplicity, we'll consider the special case of $k = 1$ in the following questions, but note that this can be easily extended to the setting with arbitrary k. We first consider a simple partitioning scheme, where we place a Cartesian grid (a rectangular grid consisting of hypercubes) over the feature space.

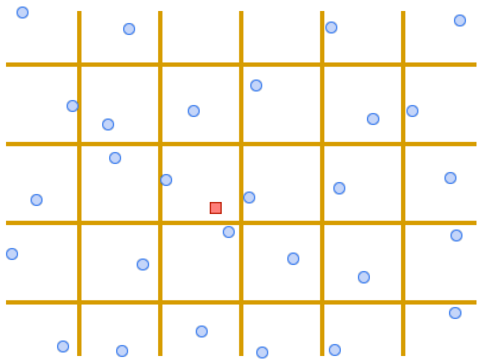

Figure 1: Illustration of the space partitioning scheme we consider. The data points are shown as blue circles and the query is shown as the red square. The cell boundaries are shown as gold lines.

If we search over a cell and its neighboring cells, how many cells do we search if the data points are one-dimensional? Two-dimensional? d-dimensional? If each cell contains one data point, what is the time complexity for finding the 1-nearest neighbor in terms of d, assuming accessing any cell takes constant time?

3 Entropy and Bagging (13 points)

This problem helps build intuition behind the core techniques that make random forests work: entropy and bagging. We will first clarify the concepts of surprise and entropy. Recall that entropy is one of the standards for us to split the nodes in decision trees until we reach a certain level of homogeneity.

(a) (1 point) Suppose you have a bag of balls, all of which are black. How surprised are you if you take out a black ball?

(b) (1 point) With the same bag of balls, how surprised are you if you take out a white ball?

(c) (1 point) Now we have 10 balls in the bag, each of which is black or white. Under what color distribution(s) is the entropy of the bag minimized? And under what color distribution(s) is the entropy maximized? Calculate the entropy in each case.
Recall: letting p_B denote the fraction of balls that are black, the entropy of a random draw from the bag is

$$H_b(p_B) = -p_B \log p_B - (1 - p_B)\log(1 - p_B)$$

where $H_b(p)$ denotes the entropy of a Bernoulli(p) distribution.

(d) (1 point) Draw the graph of entropy $H_b(p_c)$ when there are only two classes C and D, with $p_d = 1 - p_c$. Is the entropy function strictly concave, concave, strictly convex, or convex?

(e) (2 points) Suppose in a binary classification decision tree node, $P(Y = 1)$, the proportion of data points labeled 1, is equal to p. We split the points in the node based on whether the jth feature is greater than v. Defining $X_{j,v} = \mathbf{1}\{X_j < v\}$, this results in two child nodes with label proportions $P(Y = 1|X_{j,v} = 1) = q_1$ and $P(Y = 1|X_{j,v} = 0) = q_2$. The information gain is defined as the decrease in entropy:

$$I(X_{j,v}; Y) = H(Y) - H(Y|X_{j,v}),$$

where $H(Y)$ denotes the entropy of Y:

$$H(Y) = -\sum_{i\in\{0,1\}} P(Y = i)\log P(Y = i) = H_b(P(Y = 1))$$

and $H(Y|X_{j,v})$ denotes the conditional entropy of Y given $X_{j,v}$:

$$H(Y|X_{j,v}) = \sum_{i\in\{0,1\}} P(X_{j,v} = i)H_b(P(Y = 1|X_{j,v} = i)).$$

Show that p is a weighted average of q_1 and q_2, i.e. $p = \lambda q_1 + (1-\lambda)q_2$ for some $\lambda \in [0, 1]$, and that the information gain of the split is positive if both child nodes are nonempty and $q_1 \neq q_2$.

(f) (2 points) **Ensemble Learning - the motivation of averaging.** 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. 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.)

(g) (3 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 B random subsamples of size n' by sampling with replacement. Within an individual subsample, some points may be chosen multiple times, while some may not be chosen at all. For large n, if $n' = n$, around 63% of the points are chosen, and the remaining 37% are called out-of-bag (OOB) samples.

(i) (2 points) Show that when $n \geq 25$, the probability a point is not chosen for a particular subsample is in $[0.36, \frac{1}{e} \approx 0.368]$.

(ii) (1 point) If we use bagging to train our model, what is a practical way to choose the hyperparameter B? Typically in random forests, a few hundred to several thousand trees are used, depending on the size and nature of the training set.

(h) (2 points) In part (f), we see that averaging reduces variance for uncorrelated classifiers. Real world prediction will of course not be completely uncorrelated, but reducing correlation 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 Machine Learning on the Diabetes Dataset (20 points)

In this question, you will apply the following machine learning techniques to a dataset involving diabetes patients: t-SNE, PCA, ordinary least squares (OLS), ridge regression, and K-nearest neighbors regression. Fetch the dataset from the *sklearn* library using

```
diabetes = sklearn.datasets.load_diabetes()
```

The data matrix $\mathbf{X}$, labels $\mathbf{y}$, and a dataset description can be accessed under the `data`, `target`, and `DESCR` keys. This dataset contains 442 data points, where each data point has 10 input features and a real-valued target. Thus $\mathbf{X} \in \mathbb{R}^{442\times 10}$ and $\mathbf{y} \in \mathbb{R}^{442}$.

The input features include age, sex, BMI, blood pressure, and six blood serum measurements. The target is a quantitative measure of diabetes one year after the previous measurements were taken.

(a) (2 points) Perform a two-dimensional t-SNE on the input data. You may use `sklearn.manifold.TSNE` for this purpose. Visualize the dimension-reduced data points in a scatter plot, and color code each input data point according to its target value. This can be done using the `c` and `cmap` arguments of `matplotlib.pyplot.scatter`. Also, include a colorbar in your plot.

(b) (1 point) You should obtain two oblong clusters in the above t-SNE plot. If you do not, try running it again. What feature do these clusters correspond to? Re-plot the t-SNE plot, but with the data points colored according to that feature.

(c) (2 points) Run t-SNE on the data matrix with the feature corresponding to the two clusters removed. What change do you see? Explain why the change occurs, and what it would mean if after removing the feature the change did not occur.

(d) (1 points) Run a two-dimensional PCA to generate a scatter-plot color coding it by label like the t-SNE plot in part (a). If you are using `numpy.linalg.svd`, remember to center your data! Otherwise if you use `sklearn.decomposition.PCA`, the package will do it for you.

(e) (3 points) For the next three parts, you will use the first 100 points of the data `X_train = X[:100]` as your training set and the last 342 points as your test set. Compute the test MSE and test C-index of the OLS predictor with bias:

$$(\mathbf{w}^*_{\text{OLS}}, b^*_{\text{OLS}}) = \underset{\mathbf{w}\in\mathbb{R}^{10}, b\in\mathbb{R}}{\arg\min} \|\mathbf{X}_{\text{train}}\mathbf{w} + b\mathbf{1} - \mathbf{y}_{\text{train}}\|_2^2.$$

The test MSE equals

$$\frac{1}{n_{\text{test}}}\|\mathbf{X}_{\text{test}}\mathbf{w}^*_{\text{OLS}} + b^*_{\text{OLS}}\mathbf{1} - \mathbf{y}_{\text{test}}\|_2^2.$$

In scenarios with hard-to-predict-labels, the MSE can become very large and as a result, hard to interpret. The C-index metric gets around this issue, since it compares how data points are ranked relative to each other, rather than their absolute label values. Letting $\hat{\mathbf{y}}_{\text{test}} = \mathbf{X}_{\text{test}}\mathbf{w}^*_{\text{OLS}} + b^*_{\text{OLS}}\mathbf{1}$ denote our test predictions, the C-index equals

$$\text{C-index} = \frac{\text{number of concordant pairs}}{\text{number of concordant pairs + number of discordant pairs}}.$$

A concordant pair is a pair $1 \leq i \neq j \leq n_{\text{test}}$ such that $(\mathbf{y}_{\text{test}})_i > (\mathbf{y}_{\text{test}})_j$ and $\hat{\mathbf{y}}_i > \hat{\mathbf{y}}_j$. A discordant pair is a pair $1 \leq i \neq j \leq n_{\text{test}}$ such that $(\mathbf{y}_{\text{test}})_i > (\mathbf{y}_{\text{test}})_j$ and $\hat{\mathbf{y}}_i < \hat{\mathbf{y}}_j$. Given a random pair of data points where one has a higher label than the other, the C-index is the probability we predict a higher value for the data point with a higher label. A random predictor would get a C-index of 0.5 and a perfect predictor would get a C-index of 1.

For the following parts, you will be doing cross validation on three different methods: ridge regression, PCA regression, and k-nearest neighbors regression. It is recommended that you write one cross validation function that can take in each type of method and run cross validation on it, to shorten your code.

(f) (4 points) By dividing `X_train` into 10 equal parts, run 10-fold cross validation on ridge regression

$$\min_{\mathbf{w}\in\mathbb{R}^{10}, b\in\mathbb{R}} \|\mathbf{X}_{\text{CV\_train}}\mathbf{w} + b\mathbf{1} - \mathbf{y}_{\text{CV\_train}}\|_2^2 + \lambda\|\mathbf{w}\|_2^2$$

using the λ parameters $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1]$. Plot the average validation MSE and average train MSE for each λ parameter on the same plot vs. $\log_{10}(\lambda)$ on the x-axis. Choose the λ^* with the lowest average validation MSE, find the ridge solution on the full train set with this λ^*, and compute the MSE and C-index of the resulting solution on the test set.

(g) (3 points) Run 10-fold cross validation on PCA regression

$$\min_{\mathbf{w}\in\mathbb{R}^{k}, b\in\mathbb{R}} \|\tilde{\mathbf{X}}_{\text{CV\_train}}\mathbf{V}_k\mathbf{w} + b\mathbf{1} - \mathbf{y}_{\text{CV\_train}}\|_2^2 \tag{1}$$

varying the value of k (number of components) from the integers 1 to 10 inclusive. $\mathbf{V}_k \in \mathbb{R}^{10\times k}$ contains the first k right singular vectors of the mean-centered cross validation training matrix $\tilde{\mathbf{X}}_{\text{CV\_train}}$. To make a prediction on a new data point $\mathbf{x}_{\text{new}} \in \mathbb{R}^{10}$, we need to use the saved mean $\boldsymbol{\mu}_{\text{CV\_train}} = \frac{1}{n_{\text{CV\_train}}}\mathbf{X}^\top_{\text{CV\_train}}\mathbf{1} \in \mathbb{R}^{10}$ and $\mathbf{V}_k$ to compute

$$\hat{y}_{\text{new}} = (\mathbf{x}_{\text{new}} - \boldsymbol{\mu}_{\text{CV\_train}})^\top\mathbf{V}_k\mathbf{w} + b.$$

The `sklearn.decomposition.PCA` class helpfully saves the mean vector and principal directions computed from the dataset it was applied to.

Plot the average validation MSE and average train MSE for each value of k on the same plot vs. k on the x-axis. Choose the k^* with the lowest average validation MSE, find the PCA regression solution on the full training set with k^*, and compute the MSE and C-index of the resulting solution on the test set.

(h) (2 points) Repeat the previous three parts, minus the plots, now using the first $n_{\text{train}} = 300$ data points as the training set and the latter 142 data points as the test set. How does the test performance of ridge and PCA regression with the best cross-validated hyperparameters λ^* and k^* compare to the test performance of OLS, when $n_{\text{train}} = 100$ and when $n_{\text{train}} = 300$? Does this agree with what you expect?

(i) (2 points) With the same train and test sets as in the previous part, run 10-fold cross validation on k-nearest neighbors regression, varying the value of k in $[1, 2, 3, 5, 10, 20, 30, 50, 100]$. No need to plot anything. You are encouraged to use `sklearn.neighbors.KNeighborsRegressor`. Using the k^* with the lowest validation MSE, train a KNN regressor on the full train set and compute its test MSE and test C-index.