COMS 4771-1 F19 Homework 1 (due September 23, 2019)

Instructions

Submit your write-up on Gradescope as a neatly typeset (not scanned nor hand-written) PDF document by 2:30 PM of the due date. (Feel free to use Word, LaTeX, etc.—whatever you like.)

On Gradescope, be sure to select the pages containing your answer for each problem. More details can be found on the Gradescope Student Workflow help page.

(If you don't select pages containing your answer to a problem, you'll receive a zero for that problem.)

Make sure **your name and your UNI** appears prominently on the first page of your write-up.

You are welcome and encouraged to discuss homework assignments with each other in small groups (two to three people). You must **list all discussants in your homework write-up**: also do this prominently on the first page of your write-up.

Remember, discussion about homework assignments may include brainstorming and verbally discussing possible solution approaches, but **must not go as far as one person telling others how to solve a problem**. In addition, **you must write-up your solutions by yourself**, and **you may not look at another student's homework write-up/solutions (whether partial or complete)**. The academic rules of conduct can be found in the course syllabus.

Source code

Please combine all requested source code files into a **single** ZIP file, along with a plain text file called `README` that contains your name and briefly describes all of the other files in the ZIP file. **Do not include the data files.** Submit this ZIP file on Courseworks.

Clarity and precision

One of the goals in this class is for you to learn to reason about machine learning problems and algorithms. To demonstrate this reasoning, you must be able to make **clear** and **precise** arguments. A clear and precise argument is not the same as a long, excessively detailed argument. Unnecessary details and irrelevant side-remarks often make an argument less clear. Non-factual statements also detract from the clarity of an argument.

Points may be deducted for answers and arguments that lack sufficient clarity or precision. We will grade your answer/argument based only on a time-economical attempt to understand it.

Problem 1 (10 points)

Name one or two of your own personal, academic, or career values. Briefly explain how you hope machine learning can be of service to those values.

Problem 2 (20 points)

In this problem, you will implement and evaluate nearest neighbor classifiers.

MNIST data set

Download the MNIST data set of handwritten digit images `ocr.mat` from the course website, and load it into Python:

```
from scipy.io import loadmat
ocr = loadmat('ocr.mat')
```

The 60000 unlabeled training data (i.e., *feature vectors*) are contained in a matrix called `data` (one data point per row), and the corresponding labels are in a vector called `labels`. The 10000 test feature vectors and labels are in, respectively, `testdata` and `testlabels`.

In Python, you can view an image (say, the first one) with the following commands:

```
import matplotlib.pyplot as plt
from matplotlib import cm
plt.imshow(ocr['data'][0].reshape((28,28)), cmap=cm.gray_r)
plt.show()
```

Nearest neighbor classifier

Let the training examples be denoted by $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$ (for $n = 60000$), where each $\boldsymbol{x}_i \in \mathbb{R}^d$ and each $y_i \in \{0, \ldots, 9\}$.

The *nearest neighbor classifier* $\hat{f}\colon \mathbb{R}^d \to \{0, \ldots, 9\}$ is a function defined in terms of the 60000 training examples as follows. On input $\boldsymbol{x} \in \mathbb{R}^d$, let $i(\boldsymbol{x}) := \arg\min_{i \in \{1,\ldots,n\}} \|\boldsymbol{x} - \boldsymbol{x}_i\|_2$, breaking ties arbitrarily (i.e., $i(\boldsymbol{x})$ is any $i \in \{1, \ldots, n\}$ for which $\|\boldsymbol{x} - \boldsymbol{x}_{i(\boldsymbol{x})}\|_2 \leq \min_{i \in \{1,\ldots,n\}} \|\boldsymbol{x} - \boldsymbol{x}_i\|_2$); and return $y_{i(\boldsymbol{x})}$.[1]

Write a program in Python that implements the nearest neighbor classifier. Your program should take as input a matrix of training feature vectors `X` and a vector of corresponding labels `Y`, as well as a matrix of test feature vectors `test`. The output of the program should be a vector of the predicted labels `preds` for all test points, as provided by the nearest neighbor classifier $\hat{f}$ based on the examples in `X` and `Y`.

You should not use (or look at the source code for) any library functions for computing all pairwise distances between entire sets of vectors, or for computing nearest neighbor queries; that would, of course, defeat the purpose of this assignment. However, you can use functions for computing inner products between vectors and norms of vectors, as well as other basic matrix and vector operations. It is also fine to use `numpy.argmin` and `numpy.argmax`. If in doubt about what is okay, just ask.

In order to make your code efficient and not take a long time to run, you should use matrix/vector operations (rather than, say, a bunch of explicit for-loops). Think of the n training feature vectors as being stacked as the rows of a matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, and the m test feature vectors as the rows of

[1] The expression "$\arg\min_{a \in A} f(a)$" denotes the set of minimizers of the function f among all elements in the set A. On occasion, as we do here, we'll be a little sloppy and write expressions of the form "$\hat{a} := \arg\min_{a \in A} f(a)$". This will always means that $\hat{a}$ is *some* minimizer of f within A, rather than the set of minimizers. For this to really make sense, though, it must be made clear (either explicitly or implicitly) what we mean in the case that there are multiple minimizers—i.e., what the tie-breaking rule is—or if there are no minimizers!

another matrix $\boldsymbol{T} \in \mathbb{R}^{m \times d}$. If the i-th row of $\boldsymbol{T}$ is $\boldsymbol{t}_i^\mathsf{T}$ and the j-th row of $\boldsymbol{X}$ is $\boldsymbol{x}_j^\mathsf{T}$, then the squared Euclidean distance between $\boldsymbol{t}_i$ and $\boldsymbol{x}_j$ is

$$\|\boldsymbol{t}_i - \boldsymbol{x}_j\|_2^2 = (\boldsymbol{t}_i - \boldsymbol{x}_j)^\mathsf{T}(\boldsymbol{t}_i - \boldsymbol{x}_j) = \boldsymbol{t}_i^\mathsf{T}\boldsymbol{t}_i - 2\boldsymbol{t}_i^\mathsf{T}\boldsymbol{x}_j + \boldsymbol{x}_j^\mathsf{T}\boldsymbol{x}_j.$$

Can you leverage this identity to speed-up computation of the nearest neighbor classifier?[2]

Evaluating the nearest neighbor classifier

Instead of using your nearest neighbor program with `data` and `labels` as the training data, do the following. For each value $n \in \{1000, 2000, 4000, 8000\}$:

1. Draw n points from `data`, together with their corresponding labels, uniformly at random. Use `sel = random.sample(range(60000,n)` (after `import random`), `ocr['data'][sel].astype('float')`, and `ocr['labels'][sel]` to select the examples.[3]
2. Use these n points as the training data and `testdata` as the test points; compute the fraction of test examples on which the nearest neighbor classifier predicts the label incorrectly (i.e., the *test error rate*).

A plot of test error rate (on the y-axis) as a function of n (on the x-axis) is called a *learning curve*.

Carry out the (random) process described above ten times, independently. Produce a learning curve plot using the average of these test error rates (that is, averaging over ten repetitions). Add error bars to your plot that extend to one standard deviation above and below the means. Ensure the plot axes are properly labeled.

K-fold cross validation

K-fold cross validation is an alternative to the "hold-out validation" method for "hyperparameter selection" (also called "model selection").

Before describing K-fold cross validation, we first recall the hold-out validation method:

- Start with training data S.
- Partition S into two parts, S_1 and S_2 (randomly).
- For each hyperparameter $h \in H$:
  - Train a classifier using S_1; call it $\hat{f}_h$.
  - Compute the test error rate $\mathrm{err}(\hat{f}_h, S_2)$ of $\hat{f}_h$ on S_2. This is the hold-out error rate for h.
- Select the hyperparameter $h \in H$ with smallest hold-out error rate; call this $\hat{h}$.
- (Now use all of S with hyperparameter $\hat{h}$ to train a classifier $\hat{f}$.)

Above, H is the set of possible hyperparameters you want to try, such as the possible values of k for the k-NN method.

Hold-out validation mimics the act of splitting data into training data and test data, training the classifier using the training data, and then evaluate its performance on the test data. But, of course, the important thing is that only the training data is used in hold-out validation.

[2] In order to take advantage of matrix/vector operations, your Python installation needs to be using optimized linear algebra libraries, such as ATLAS, MKL, or the Accelerate/vecLib framework on OS X. If you installed Python using Anaconda, then you should already have MKL.

[3] The data are stored as unsigned integers, so you need to convert them to floating point or else you may get integer overflows.

Note, however, that the hold-out validation method is asymmetric—S_1 and S_2 are used for very different purposes. To "symmetrize" the method, we can swap the roles of S_1 and S_2 to produce another hold-out error rate for each hyperparameter.[4] So now we have two hold-out error rates for each hyperparameter, and we can choose the hyperparameter for which the average of the two hold-out error rates is smallest.

- Start with training data S.
- Partition S into two parts, S_1 and S_2 (randomly).
- For each hyperparameter $h \in H$:
  - Train a classifier using S_1; call it $\hat{f}_h$.
  - Train a classifier using S_2; call it $\hat{f}'_h$.
  - Compute $\frac{1}{2}(\text{err}(\hat{f}_h, S_2) + \text{err}(\hat{f}'_h, S_1))$. This is the symmetrized hold-out error rate for h.
- Select the hyperparameter $h \in H$ with smallest symmetrized hold-out error rate; call this $\hat{h}$.
- (Now use all of S with hyperparameter $\hat{h}$ to train a classifier $\hat{f}$.)

K-fold cross validation is a generalization of this "symmetrized hold-out validation" method, where instead of just splitting the training data S into two parts, we split into K parts (of equal size):

- Start with training data S.
- Partition S into K equal-sized parts, $S_1, \ldots, S_K$ (randomly).
- For each hyperparameter $h \in H$:
  - For $i = 1, \ldots, K$:
    - Train a classifier using $\bigcup_{j \neq i} S_j$ (i.e., all the training data *except* S_i); call it $\hat{f}_{h,i}$.
  - Compute $\frac{1}{K} \sum_{i=1}^{K} \text{err}(\hat{f}_{h,i}, S_i)$. This is the K-fold cross validation error rate for h.
- Select the hyperparameter $h \in H$ with smallest K-fold cross validation hold-out error rate; call this $\hat{h}$.
- (Now use all of S with hyperparameter $\hat{h}$ to train a classifier $\hat{f}$.)

Use K-fold cross validation, with $K = 10$, to select the value of k in k-NN using the MNIST training data. The values of k to try are $\{1, 2, \ldots, 10\}$. You are welcome to use a software library (e.g., scikit-learn) to implement K-fold cross validation, provided that (i) you verify yourself that it exactly implements K-fold cross validation as described above, and (ii) you cite the software library that you use in your write-up.

What to submit in your write-up

(a) A brief description of how your nearest neighbor implementation works.
(b) Learning curve plot. (Please include the plot directly in the PDF write-up.)
(c) 10-fold cross validation error rates for $k \in \{1, 2, \ldots, 10\}$. (Please put the results in a table.) Also report the test error rate of the k-NN classifier using the value of k that has the smallest 10-fold cross validation error rate.

Please submit all of your source code on Courseworks.

[4] There is still some asymmetry, because often in hold-out validation, S_1 and S_2 are of different sizes. For example, we often use more data for fitting a model than for evaluating it. However, the generalization to K-fold cross validation will address this issue.

Problem 3 (15 points)

In this problem, you will analyze an approximate nearest neighbor method.

Suppose you want to speed-up the 1-NN method by replacing the nearest neighbor search with an "approximate nearest neighbor" search. What might be a good notion of "approximate nearest neighbor"? In this problem, we shall use the following notion. Given a test point $\boldsymbol{x}$, we say a 5%-approximate nearest neighbor is $\boldsymbol{x}_{i(\boldsymbol{x})}$ such that $\|\boldsymbol{x} - \boldsymbol{x}_{i(\boldsymbol{x})}\|_2$ is among the smallest 5% of distances to training points $\|\boldsymbol{x} - \boldsymbol{x}_1\|_2, \ldots, \|\boldsymbol{x} - \boldsymbol{x}_n\|_2$ (here, n is the number of training examples).

Of course, we can do this by simply computing the distances $\|\boldsymbol{x} - \boldsymbol{x}_i\|_2$ to $\lceil 0.95n \rceil + 1$ of the training points $\boldsymbol{x}_i$, and then returning the arg min just among those distances. This is, indeed, faster than computing the distances to all n training points, but unfortunately not by very much.

Using a random number generator, we can do much better. Pick T numbers $i_1, \ldots, i_T$ uniformly at random (without replacement) from $\{1, 2, \ldots, n\}$. Here, T is a number that we'll determined later. Then compute the T distances $\|\boldsymbol{x} - \boldsymbol{x}_{i_1}\|_2, \ldots, \|\boldsymbol{x} - \boldsymbol{x}_{i_T}\|_2$, and return the arg min among *these* distances.

First, we observe that unless $T \geq \lceil 0.95n \rceil + 1$, then there are data sets on which there is some chance that this algorithm does not return a 5%-approximate nearest neighbor.

(a) Explain why the above statement is true.

How large should T be so that this chance is small—say, probability at most 1%?

(b) Prove that $T = 90$ is sufficient.

This is rather remarkable—a reduction from $\Theta(n)$ distance computations to $\Theta(1)$ distance computations! Unfortunately, I believe this method often does not work very well in practice.

(c) (Optional.) Investigate the behavior of this algorithm on MNIST through some exploratory analysis. Describe your approach and findings. Does it work well? Why or why not?

Problem 4 (10 points)

In this problem, you will practice deriving formulae for maximum likelihood estimators.

Consider a statistical model for iid random variables $X_1, \ldots, X_n$, parameterized by $\theta \in \Theta$. The distribution P_θ of X_1 is specified in each part below.

In each of these cases, derive a simple formula for the MLE for θ (given data $(X_1, \ldots, X_n) = (x_1, \ldots, x_n)$), and briefly justify each step of the derivation.

(a) $X_1 \sim P_\theta$ has probability density function f_θ satisfying

$$f_\theta(x) \propto x^2 e^{-x/\theta} \quad \text{for all } x > 0,$$

and $f_\theta(x) = 0$ for all $x \leq 0$.[5] The parameter space is the positive reals $\Theta = \{\theta \in \mathbb{R} : \theta > 0\}$.

(b) $X_1 \sim P_\theta$ has probability density function f_θ satisfying

$$f_\theta(x) \propto 1 \quad \text{for all } 0 \leq x \leq \theta,$$

and $f_\theta(x) = 0$ for all $x < 0$ and all $x > \theta$. The parameter space is the positive reals $\Theta = \{\theta \in \mathbb{R} : \theta > 0\}$.

[5] The symbol "$\propto$" means "proportional to". Remember, probability density functions should integrate to one, and probability mass functions should sum to one.

Problem 5 (10 points)

In this problem, you will hand-construct decision trees for small data sets with just two features.

Consider decision trees (on $\mathbb{R}^2$) where the "splits" are of the form $(x_1, x_2) \mapsto \mathbb{1}_{\{x_i > \theta\}}$, where $i \in \{1, 2\}$, and θ is an *integer*.[6]

(a) Write a Python function that implements a decision tree with the following properties.
- The splits are of the above form.
- The training error rate is exactly 1/6.
- The training false negative rate is zero.[7]
- The depth of the tree is as small as possible.

(b) Write a Python function that implements a decision tree with the following properties.
- The splits are of the above form.
- The training error rate is zero.
- The test error rate is as high as possible.
- The depth of the tree is as small as possible.

(The order of precedence for these last two stipulations in Part (b) does not matter.)

The training data and test data are shown in the figures below.[8] Red points are negative examples (label 0), and blue points are positive examples (label 1). You can assume the Python function takes as input two real (well, floating point) numbers, `x1` and `x2`, corresponding to the two feature values of the input (`x1` = feature value along horizontal axis; `x2` = feature value along vertical axis).

Please include your code in your write-up, rather than in a separate file. In LaTeX, you might try using the `listings` package.

[6] The notation "$z \mapsto h(z)$" is used to denote an anonymous function that maps the variable z to $h(z)$. The notation $\mathbb{1}_{\{P\}}$ is used to denote the $\{0, 1\}$-valued indicator of the predictate P; it takes value 1 if P is true, and it takes value 0 if P is false.

[7] The *false negative rate* is the proportion of (true) positive examples that are (incorrectly) predicted to be negative.

[8] Plots are courtesy of Matus Telgarsky.

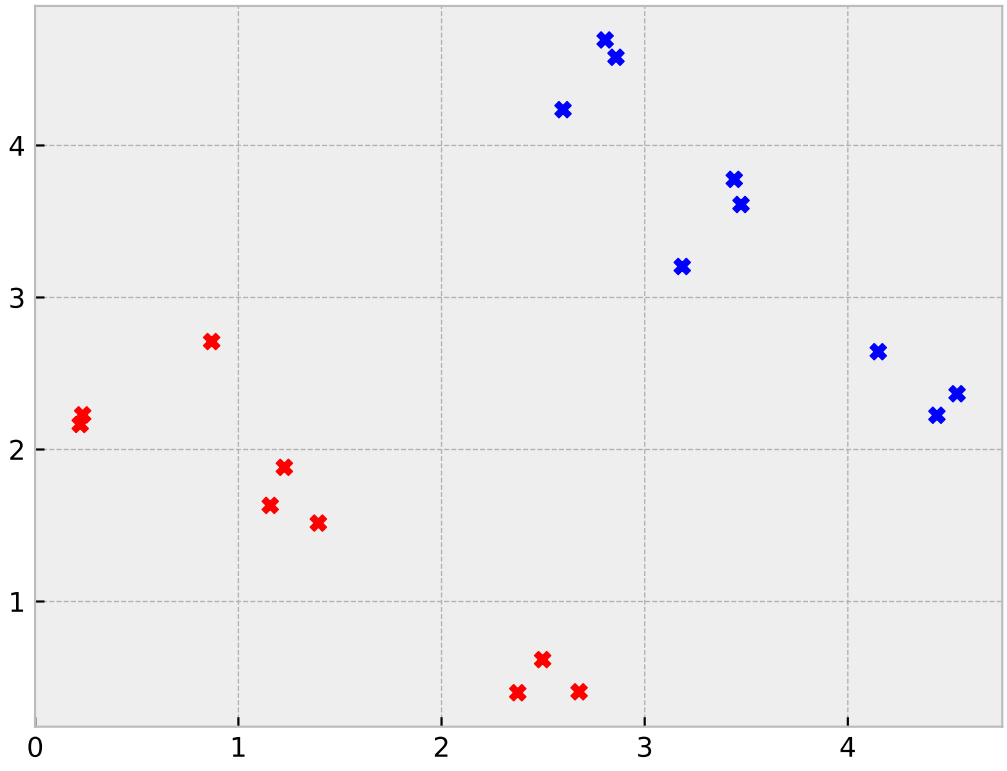

Figure 1: Training data (18 data points)

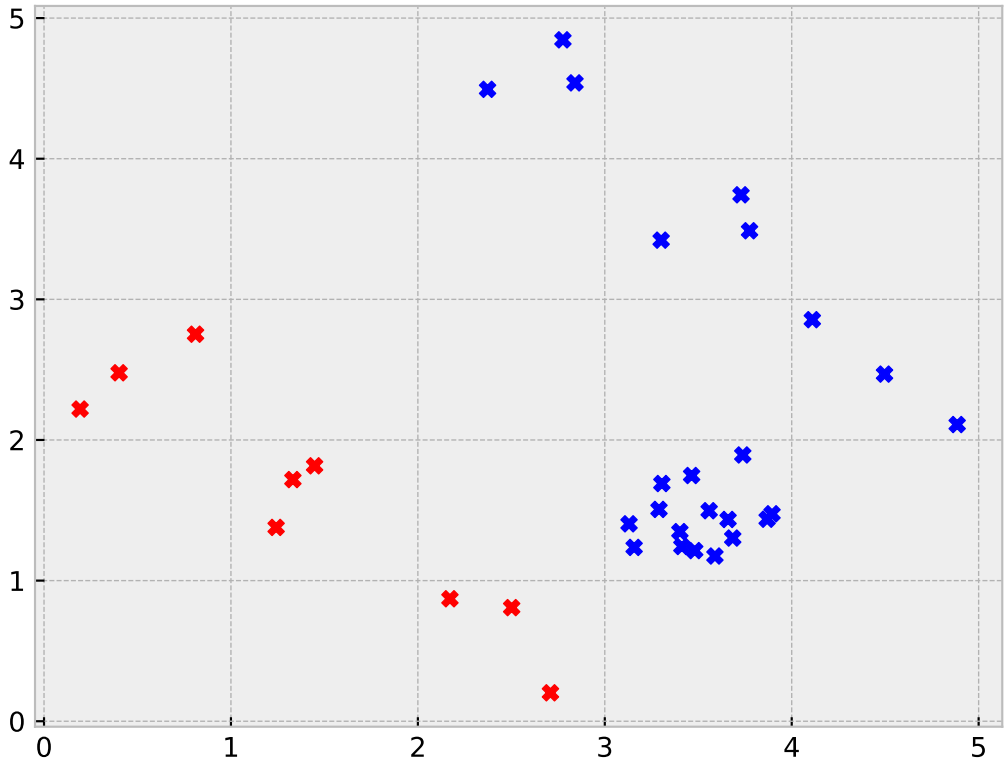

Figure 2: Test data (33 data points)

Problem 6 (20 points)

In this problem, you will consider a different model for collecting data that ensures some kind of privacy guarantee.

Suppose you would like to estimate the proportion of people in a population with a certain property P (e.g., whether or not the person has a particular disease). You could just survey randomly chosen people from the population, and ask them whether or not they have property P. However, people may be uncomfortable sharing that information with you outright.

The *randomized response* method was developed to address this privacy concern. In the randomized response method, surveyed individuals are not asked to directly tell you whether or not they have property P. Instead, you ask each surveyed individual to do the following:

- Toss a fair coin twice (without letting you see the outcomes).
- If at least one toss comes up heads:
  - Respond truthfully (i.e., say 1 if the individual has property P; 0 if not).
- Else:
  - Give the opposite response (i.e., say 0 if the individual has property P; 1 if not).

Because you do not observe the outcomes of the coin tosses, you never definitively learn whether the surveyed individual has property P or not. This is a very strong privacy guarantee for the surveyed individuals. Moreover, the collected information can still be used to obtain a good estimate of the proportion of individuals in the population with property P.

Consider a statistical model for n responses collected in this way, where the responses are regarded as iid $\{0,1\}$-valued random variables $Y_1, \ldots, Y_n$, and all coin tosses are independent.[9] Let $\theta \in [0,1]$ denote the parameter of the model that equals the proportion of individuals in the population with property P.

(a) What is the probability that $Y_1 = 1$? Give your answer in terms of the parameter θ.

(b) What is the log-likelihood of θ given data $y_1, \ldots, y_n \in \{0,1\}$? Write your answer in terms of θ and $y_1, \ldots, y_n$.

(c) Suppose $n = 100$, and the number of y_i that are equal to 1 is 40 (i.e., $\sum_{i=1}^{n} y_i = 40$). Plot the log-likelihood as a function of $\theta \in [0,1]$. What appears to be the θ with highest likelihood? (Please include the plot directly in the PDF write-up.)

[9] When sampling from a finite population, we typically use sampling without replacement. Here, we use sampling with replacement to simplify the problem.

Problem 7 (15 points)

Find and read an article (e.g., from popular science venues like *Scientific American* and *Science*) where supervised machine learning is applied to a "real" problem, but the iid model of data is not appropriate. (Don't pick an article that is very general and does not get into specifics of the real problem and application.)

(a) Describe the application, and summarize how machine learning is used. Also give a full citation for the article (and URL link if available).
(b) Explain why the iid model of data is not appropriate for the application.
(c) Does the author or ML practitioner acknowledge the invalidity of the iid model for the application? (They may, for instance, use a data model that captures dependence.)

For each of parts (a) and (b), a short paragraph response is sufficient.

For part (c), just "yes" or "no" suffices.