COMS 4771-2 F18 Homework 1 (due September 26, 2018)

Instructions

Submit your write-up on Gradescope as a neatly typeset (not scanned nor handwritten) PDF document by 11:59 PM of the due date.

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:

- https://gradescope.com/help#help-center-section-student-workflow

(If you don't select the 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.

Source code

Please combine all requested source code files into a *single* ZIP file[1], 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 reason about these things, you must be able to make *clear* and *precise* claims and arguments about them.

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. And 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. Moreover, a time-economical attempt will be made to interpret such answers/arguments, and the grade you receive will be based on this interpretation.

[1]See https://en.wikipedia.org/wiki/Zip_(file_format).

Problem 1 (30 points)

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

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 $(x_1, y_1), \dots, (x_n, y_n)$ (for $n = 60000$), where each $x_i \in \mathbb{R}^d$ and each $y_i \in \{0, \dots, 9\}$.

The *nearest neighbor classifier* $\hat{f} \colon \mathbb{R}^d \to \{0, \dots, 9\}$ is a function defined in terms of the 60000 training examples as follows. On input $x \in \mathbb{R}^d$, let $i(x) := \arg\min_{i \in \{1,\dots,n\}} \|x - x_i\|_2$, breaking ties arbitrarily (i.e., $i(x)$ is any $i \in \{1, \dots, n\}$ for which $\|x - x_{i(x)}\|_2 \leq \min_{i \in \{1,\dots,n\}} \|x - x_i\|_2$); and return $y_{i(x)}$.[2]

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 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. If in doubt what is okay to use, 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 $X \in \mathbb{R}^{n \times d}$, and the m test feature vectors as the rows of another matrix $T \in \mathbb{R}^{m \times d}$. If the i-th row of T is $t_i^\top$ and the j-th row of X is $x_j^\top$, then the squared Euclidean distance between t_i and x_j is

$$\|t_i - x_j\|_2^2 = (t_i - x_j)^\top (t_i - x_j) = t_i^\top t_i - 2t_i^\top x_j + x_j^\top x_j.$$

Can you leverage this identity to speed-up computation of the nearest neighbor classifier?[3]

[2]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 (or if there are none!).

[3]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.

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.[4]
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 the 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.

What to submit in your write-up:

(a) A brief description of how you implemented the nearest neighbor classifier (especially the "$\arg\min$"). Note that it is fine to use `numpy.argmin`, but you should explain precisely how you use it in your code to efficiently implement the nearest neighbor classification of new (test) points.
(b) Learning curve plot. (Please include the plot directly in the PDF write-up.)
(c) What would the learning curve look like if, instead of test error rate, you computed *training error rate*?

Please submit your source code on Courseworks.

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

Problem 2 (35 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 the first two 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 $\Theta = \{\theta \in \mathbb{R} : \theta > 0\}$.

In each of the next two cases, specific values of the data are given as follows:

$$(X_1, \ldots, X_n) = (0, 1, 1, 0, 2, 2, 1, 2, 1, 0, 2, 0, 2, 2, 0, 3, 0, 0, 0, 2).$$

Derive the actual MLE value of θ given this data (as opposed to an abstract formula), and briefly justify each step of the derivation.

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

$$f_\theta(x) \propto e^{-(x-\theta)^2/(2\theta^2)} \quad \text{for all } x \in \mathbb{R}.$$

The parameter space is $\Theta = \{\theta \in \mathbb{R} : \theta \neq 0\}$ (i.e., all real numbers θ such that $\theta \neq 0$).

(d) $X_1 \sim P_\theta$ has probability mass function p_θ satisfying

$$p_\theta(0) = \frac{1}{2}, \quad p_\theta(1) = e^\theta, \quad p_\theta(2) = 2e^\theta, \quad p_\theta(3) = \frac{1}{2} - 3e^\theta.$$

The parameter space is $\Theta = \{\theta \in \mathbb{R} : \theta < -\ln(6)\}$.

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

Problem 3 (35 points)

In this problem, you will implement a learning algorithm based on generative models for classification, apply the learning algorithm to a simple data set, and quantitatively and qualitatively study the learned classifiers.

Download the "20 Newsgroups data set" `news.mat` from Courseworks. The training feature vectors/labels and test feature vectors/labels are stored as `data/labels` and `testdata/testlabels`. Each data point corresponds to a message posted to one of 20 different newsgroups (i.e., message boards). The representation of a message is a (sparse) binary vector in $\mathcal{X} := \{0,1\}^d$ (for $d := 61188$) that indicates the words that are present in the message. If the j-th entry in the vector is 1, it means the message contains the word that is given on the j-th line of the text file `news.vocab`. The class labels are $\mathcal{Y} := \{1, 2, \ldots, 20\}$, where the mapping from classes to newsgroups is in the file `news.groups` (which we won't actually need).

In this problem, you'll develop a classifier based on a Naive Bayes generative model. Here, we use class conditional distributions with probability mass functions of the form $p_\mu(x) = \prod_{j=1}^d \mu_j^{x_j}(1-\mu_j)^{1-x_j}$ for $x = (x_1, x_2, \ldots, x_d) \in \mathcal{X}$. The parameter vector is $\mu = (\mu_1, \mu_2, \ldots, \mu_d)$ must come from the parameter space $[0,1]^d$. Since there are 20 classes, the generative model is actually parameterized by 20 such vectors, $\mu_y = (\mu_{y,1}, \mu_{y,2}, \ldots, \mu_{y,d})$ for each $y \in \mathcal{Y}$, as well as the class prior parameters, π_y for each $y \in \mathcal{Y}$. The class prior parameters, of course, must satisfy $\pi_y \in [0,1]$ for each $y \in \mathcal{Y}$ and $\sum_{y \in \mathcal{Y}} \pi_y = 1$.

(a) Give the formula for the MLE of the parameter $\mu_{y,j}$ based on training data $\{(x_i, y_i)\}_{i=1}^n$. (Remember, each unlabeled point is a vector: $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,d}) \in \{0,1\}^d$.)

(b) MLE is not a good estimator for the class conditional parameters if the estimate turns out to be zero or one. An alternative is the following estimator based on a technique called *Laplace smoothing*:

$$\hat{\mu}_{y,j} := \left(1 + \sum_{i=1}^n \mathbb{1}\{y_i = y\} x_{i,j}\right) \Big/ \left(2 + \sum_{i=1}^n \mathbb{1}\{y_i = y\}\right) \in (0,1).$$

Write code for training and testing a classifier based on the Naive Bayes generative model described above. Use Laplace smoothing to estimate class conditional distribution parameters, and MLE for class prior parameters. You should *not* use or look at any existing implementation (e.g., those that may be provided as library functions). Using your code, train and test a classifier with the data from `news.mat`.

What to submit: training and test error rates.

(c) Consider the *binary* classification problem, where newsgroups $\{1, 16, 20\}$ comprise the "negative class" (class 0), and newsgroups $\{17, 18, 19\}$ comprise the "positive class" (class 1). Newsgroups $\{1, 16, 20\}$ are "religious" topics, and newsgroups $\{17, 18, 19\}$ are "political" topics. Modify the data in `news.mat` to create the training and test data sets for this problem. Using these data and your codes from part (b), train and test a Naive Bayes classifier. Report the training and test error rates. Save the learned classifier for part (d).

What to submit: training and test error rates.

(d) The classifier you learn is ultimately an affine classifier, which means it has the following form:

$$x \mapsto \begin{cases} 0 & \text{if } \alpha_0 + \sum_{j=1}^d \alpha_j x_j \leq 0 \\ 1 & \text{if } \alpha_0 + \sum_{j=1}^d \alpha_j x_j > 0 \end{cases}$$

for some real numbers $\alpha_0, \alpha_1, \ldots, \alpha_d$. Determine the values of these α_j's for your learned classifier from part (c). Then, report the vocabulary words whose indices $j \in \{1, 2, \ldots, d\}$ correspond to the 20 largest (i.e., most positive) α_j value, and also the vocabulary words whose indices $j \in \{1, 2, \ldots, d\}$ correspond to the 20 smallest (i.e., most negative) α_j value. Don't report the indices j's, but rather the actual vocabulary words (from `news.vocab`).

What to submit: two ordered lists (clearly labeled) of 20 words each.

Please also submit your source code on Courseworks.