COMS 4771-1 F19 Homework 3 (due November 6, 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** appear 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 (30 points)

In this problem, you will qualitatively compare the inductive biases of different learning methods for linear regression.

Download the synthetic data sets from Courseworks (`hw3p1_train.csv`, `hw3p1_test.csv`) for this problem. In the training data ($n = 32$ data points), the first column represents the (scalar) input feature, and the second column represents the corresponding label. In the test data ($m = 2079$ data points), only the input feature is provided. (You'll qualitatively compare the learned functions by plotting their predictions on these test data.)

You will use the feature map $\boldsymbol{\varphi}\colon \mathbb{R} \to \mathbb{R}^{2L}$, with $L = 128$, given by

$$\boldsymbol{\varphi}(x) = \begin{bmatrix} \cos(x) & \sin(x) & \cos(2x) & \sin(2x) & \cdots & \cos(Lx) & \sin(Lx) \end{bmatrix}^{\mathsf{T}},$$

along with the following learning methods.

1. *Minimum Euclidean norm solution.* Given training data $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R} \times \mathbb{R}$, return the solution $\hat{\boldsymbol{w}}_{\text{Euclid}}$ to the following optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{w} \in \mathbb{R}^{2L}} \ \|\boldsymbol{w}\|_2^2 \\ &\text{subject to } \frac{1}{n}\sum_{i=1}^{n} (y_i - \boldsymbol{\varphi}(x_i)^{\mathsf{T}}\boldsymbol{w})\boldsymbol{\varphi}(x_i) = \mathbf{0}. \end{aligned}$$

   We have seen that this method is a form of *spectral regularization* akin to ridge regression, principal component regression, and gradient descent for the least squares objective.
2. *Minimum weighted Euclidean norm solution.* Given training data $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R} \times \mathbb{R}$, return the solution $\hat{\boldsymbol{w}}_{\text{weighted}}$ to the following optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{w} \in \mathbb{R}^{2L}} \ \sum_{j=1}^{2L} \lceil j/2 \rceil^2 w_j^2 \\ &\text{subject to } \frac{1}{n}\sum_{i=1}^{n} (y_i - \boldsymbol{\varphi}(x_i)^{\mathsf{T}}\boldsymbol{w})\boldsymbol{\varphi}(x_i) = \mathbf{0}. \end{aligned}$$

   Above, $\lceil r \rceil$ denotes the smallest integer z such that $z \geq r$.
3. *(Weighted) Dantzig selector.* Given training data $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R} \times \mathbb{R}$, return the solution $\hat{\boldsymbol{w}}_{\text{Dantzig}}$ to the following optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{w} \in \mathbb{R}^{2L}} \ \sum_{j=1}^{2L} \lceil j/2 \rceil |w_j| \\ &\text{subject to } \frac{1}{n}\sum_{i=1}^{n} (y_i - \boldsymbol{\varphi}(x_i)^{\mathsf{T}}\boldsymbol{w})\boldsymbol{\varphi}(x_i) = \mathbf{0}. \end{aligned}$$

   The (original) Dantzig selector is a close cousin of the Lasso.

Minimum Euclidean norm solution

First, write a Python function that takes as input $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{b} \in \mathbb{R}^n$ and returns the minimum Euclidean norm solution to the system of linear equations $\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A}\boldsymbol{w} = \boldsymbol{A}^{\mathsf{T}}\boldsymbol{b}$. Then use this Python function to compute $\hat{\boldsymbol{w}}_{\text{Euclid}}$ for the given training data, and apply the corresponding function $\hat{f}_{\text{Euclid}}(x) = \boldsymbol{\varphi}(x)^{\mathsf{T}}\hat{\boldsymbol{w}}_{\text{Euclid}}$ to the given test data $\tilde{x}_1, \ldots, \tilde{x}_m$. Plot the points $(\tilde{x}_i, \hat{f}_{\text{Euclid}}(\tilde{x}_i))$ for $i = 1, \ldots, m$, with lines connecting consecutive points for illustrative purposes.

Minimum weighted Euclidean norm solution

Explain (in words and/or pseudocode) how to use the Python function from above to compute $\hat{\boldsymbol{w}}_{\text{weighted}}$. You should be able to do this simply by modifying the input (specifically, the data matrix $\boldsymbol{A}$ and label vector $\boldsymbol{b}$ that you supply to the Python function to compute $\hat{\boldsymbol{w}}_{\text{Euclid}}$), calling the Python function, and then suitably modifying the output. Use this approach to compute $\hat{\boldsymbol{w}}_{\text{weighted}}$ for the given training data, and apply the corresponding function $\hat{f}_{\text{weighted}}(x) = \boldsymbol{\varphi}(x)^{\mathsf{T}}\hat{\boldsymbol{w}}_{\text{weighted}}$ to the given test data $\tilde{x}_1, \ldots, \tilde{x}_m$. Plot the points $(\tilde{x}_i, \hat{f}_{\text{weighted}}(\tilde{x}_i))$ for $i = 1, \ldots, m$, with lines connecting consecutive points for illustrative purposes.

Dantzig selector

The (weighted) Dantzig selector seeks the minimum *weighted* ℓ_1 norm solution to the normal equations. It can be obtained as the solution to a *linear program*, i.e., an optimization problem where we seek to minimize a linear function of the optimization variables subject to affine equality and inequality constraints on the optimization variables:

$$\begin{aligned}
\min_{(\boldsymbol{v},\boldsymbol{w})\in\mathbb{R}^{2L}\times\mathbb{R}^{2L}} \quad & \sum_{j=1}^{2L} \lceil j/2 \rceil v_j \\
\text{subject to } & \frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{\varphi}(x_i)^{\mathsf{T}}\boldsymbol{w})\boldsymbol{\varphi}(x_i) = \boldsymbol{0} \\
& -v_j + w_j \leq 0 \quad \text{for all } j = 1, \ldots, 2L \\
& -v_j - w_j \leq 0 \quad \text{for all } j = 1, \ldots, 2L \\
& -v_j \leq 0 \quad \text{for all } j = 1, \ldots, 2L.
\end{aligned}$$

Here, we have introduced "dummy variables" $\boldsymbol{v} \in \mathbb{R}^{2L}$ that are intended to represent the absolute values of ("real") optimization variables $\boldsymbol{w} \in \mathbb{R}^{2L}$. The linear constraint $-v_j \leq 0$ ensures that v_j is non-negative. The pair of linear constraints $-v_j + w_j \leq 0$ and $-v_j - w_j \leq 0$ ensures that $|w_j| \leq v_j$. Under these constraints, minimizing $\sum_{j=1}^{2L} \lceil j/2 \rceil v_j$ (a linear function of the optimization variables $(\boldsymbol{v}, \boldsymbol{w})$) is the same as minimizing $\sum_{j=1}^{2L} \lceil j/2 \rceil |w_j|$, which is what we wanted.

Typical linear programming solvers will require you to supply the following as input:

- a linear objective function (represented as a vector);
- a system of equality constraints (represented by a matrix and a vector);
- a system of inequality constraints (also represented by a matrix and a vector);
- upper- and lower-bounds on the optimization variables (some of which can be $+\infty$ or $-\infty$).

The linear program described above has $4L$ optimization variables, so the matrices you supply to the solver will each have $4L$ columns. There are $2L$ equality constraints, which correspond to the normal equations. The remaining constraints are inequality constraints, although the ones of the form $-v_j \leq 0$ can be enforced using the lower-bounds on the variables.

Use a linear programming solver to compute $\hat{\boldsymbol{w}}_{\text{Dantzig}}$ for the given training data, and apply the corresponding function $\hat{f}_{\text{Dantzig}}(x) = \boldsymbol{\varphi}(x)^{\mathsf{T}}\hat{\boldsymbol{w}}_{\text{Dantzig}}$ to the given test data $\tilde{x}_1, \ldots, \tilde{x}_m$. Plot the points $(\tilde{x}_i, \hat{f}_{\text{weighted}}(\tilde{x}_i))$ for $i = 1, \ldots, m$, with lines connecting consecutive points for illustrative purposes.

Also, compute the number of non-zero entries in each of $\hat{\boldsymbol{w}}_{\text{Euclid}}$, $\hat{\boldsymbol{w}}_{\text{weighted}}$, and $\hat{\boldsymbol{w}}_{\text{Dantzig}}$. The weight vector $\hat{\boldsymbol{w}}_{\text{Dantzig}}$ should be much sparser than the others.

What to submit in your write-up

(a) Brief description of your Python function for computing the minimum Euclidean norm solution to the normal equations.
(b) Plot of $(\tilde{x}_i, \hat{f}_{\text{Euclid}}(\tilde{x}_i))$ for $i = 1, \ldots, m$.
(c) Brief description of how to use the Python function from Part (a) to compute $\hat{\boldsymbol{w}}_{\text{weighted}}$.
(d) Plot of $(\tilde{x}_i, \hat{f}_{\text{weighted}}(\tilde{x}_i))$ for $i = 1, \ldots, m$.
(e) Plot of $(\tilde{x}_i, \hat{f}_{\text{Dantzig}}(\tilde{x}_i))$ for $i = 1, \ldots, m$.
(f) Number of non-zero entries in each of $\hat{\boldsymbol{w}}_{\text{Euclid}}$, $\hat{\boldsymbol{w}}_{\text{weighted}}$, and $\hat{\boldsymbol{w}}_{\text{Dantzig}}$.
(g) (Optional.) Try to explain the difference between these plots (at least for f_{Euclid} and f_{weighted}). How would you explain the inductive biases of the different methods? Does the Fourier spectrum reveal anything of interest?

For Parts (a) and (c):

- First, give a 2-3 sentence high-level overview of the method.
- Then give more details about the subroutines used, their purpose, etc. Feel free to enumerate them in a list.
- Conclude with any citations for external software libraries used.
- For Part (c), also very briefly explain the correctness of your procedure.

For Parts (b), (d), and (e):

- Make sure your plots are clearly labeled, and include them directly in your write-up.

Please submit your source code on Courseworks. For this problem, we recommend using the `numpy.linalg` library, as well as `scipy.optimize.linprog` for linear programming. You are welcome to use other software libraries if you like, provided that you include appropriate citations in your write-up.

Problem 2 (30 points)

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

20 Newsgroups

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`. You can load this data in Python as follows:

```
from scipy.io import loadmat
news = loadmat('news.mat')
# news['data'] is the matrix of training feature vectors
# news['labels'] is the vector of training labels
# news['testdata'] is the matrix of test feature vectors
# news['testlabels'] is the vector of test labels
```

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 $\mathcal{Y} := \{1, \ldots, 20\}$, where the mapping from classes to newsgroups is in the file `news.groups` (which we won't actually need).

Naive Bayes model

In this problem, you'll develop a classifier based on the *Naive Bayes model*, a particular statistical model for classification data.[1] In this model, the training data are treated as iid random variables $(\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n)$ from a probability distribution $P_{\boldsymbol{\theta}}$; here $\boldsymbol{\theta} = (\boldsymbol{\pi}, (\boldsymbol{\mu}_y)_{y \in \mathcal{Y}})$ is the parameter vector, with $\boldsymbol{\pi} = (\pi_y)_{y \in \mathcal{Y}} \in [0,1]^{|\mathcal{Y}|}$ and $\boldsymbol{\mu}_y = (\mu_{y,1}, \ldots, \mu_{y,d}) \in [0,1]^d$ for each $y \in \mathcal{Y}$. The label Y_i of the i-th training example has distribution given by

$$P_{\boldsymbol{\theta}}(Y_i = y) = \pi_y, \quad \text{for all } y \in \mathcal{Y}.$$

Furthermore, for each $y \in \mathcal{Y}$, the conditional distribution of the feature vector $\boldsymbol{X}_i$ given $Y_i = y$ is given by

$$P_{\boldsymbol{\theta}}(\boldsymbol{X}_i = \boldsymbol{x} \mid Y_i = y) = \prod_{j=1}^{d} \mu_{y,j}^{x_j} (1 - \mu_{y,j})^{1 - x_j}, \quad \text{for all } \boldsymbol{x} = (x_1, \ldots, x_d) \in \mathcal{X}.$$

The parameter vector $\boldsymbol{\theta}$ must, of course, satisfy $\sum_{y \in \mathcal{Y}} \pi_y = 1$.

Aside: The Naive Bayes model is unlike other statistical models we have encountered in this course, in the sense that it explicitly describes the probability distribution of the $\boldsymbol{X}_i$'s (specifically, the conditional distribution of $\boldsymbol{X}_i$ given Y_i). In contrast, the logistic regression model only describes the conditional distribution of Y_i given $\boldsymbol{X}_i$.

[1] Technically, *Naive Bayes* refers to a broader class of statistical models for classification data where the features are regarded as conditional independent random variables given the class label.

Parameter estimation for Naive Bayes

Given training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n) \in \mathcal{X} \times \mathcal{Y}$, the MLE for the parameter vector $\boldsymbol{\theta}$ is given by

$$\hat{\pi}_y := \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{y_i = y\}}, \quad \text{for all } y \in \mathcal{Y};$$
$$\hat{\mu}_{y,j} := \frac{\sum_{i=1}^{n} \mathbb{1}_{\{y_i = y\}} x_{i,j}}{\sum_{i=1}^{n} \mathbb{1}_{\{y_i = y\}}}, \quad \text{for all } y \in \mathcal{Y} \text{ and } j = 1, \ldots, d;$$

However, in many applications, it is problematic if some of the parameters turn out to be zero or one (and this will be the case when this model is fit to the 20 Newsgroup data using MLE). So it is common to use a different estimator for the $\mu_{y,j}$ parameters based on a technique called *Laplace smoothing*:

$$\hat{\mu}_{y,j} := \frac{1 + \sum_{i=1}^{n} \mathbb{1}_{\{y_i = y\}} x_{i,j}}{2 + \sum_{i=1}^{n} \mathbb{1}_{\{y_i = y\}}}, \quad \text{for all } y \in \mathcal{Y} \text{ and } j = 1, \ldots, d.$$

(We could also use Laplace smoothing to estimate the π_y parameters, but we forego it in this assignment.) Laplace smoothing has a Bayesian interpretation.

Naive Bayes classifiers

Given an estimate $\hat{\boldsymbol{\theta}}$ of the parameter vector $\boldsymbol{\theta}$, we can form a classifier based on the *plug-in principle*. This means we should form the classifier $\hat{f} \colon \mathcal{X} \to \mathcal{Y}$ such that

$$\hat{f}(\boldsymbol{x}) = \operatorname*{arg\,max}_{y \in \mathcal{Y}} \; P_{\hat{\boldsymbol{\theta}}}(Y = y \mid \boldsymbol{X} = \boldsymbol{x}).$$

Using Bayes' rule, we have

$$\begin{aligned} P_{\hat{\boldsymbol{\theta}}}(Y = y \mid \boldsymbol{X} = \boldsymbol{x}) &= \frac{P_{\hat{\boldsymbol{\theta}}}(Y = y \wedge \boldsymbol{X} = \boldsymbol{x})}{P_{\hat{\boldsymbol{\theta}}}(\boldsymbol{X} = \boldsymbol{x})} \\ &= \frac{P_{\hat{\boldsymbol{\theta}}}(Y = y) \cdot P_{\hat{\boldsymbol{\theta}}}(\boldsymbol{X} = \boldsymbol{x} \mid Y = y)}{P_{\hat{\boldsymbol{\theta}}}(\boldsymbol{X} = \boldsymbol{x})} \\ &= \frac{\hat{\pi}_y \cdot \prod_{j=1}^{d} \hat{\mu}_{y,j}^{x_j} (1 - \hat{\mu}_{y,j})^{1 - x_j}}{P_{\hat{\boldsymbol{\theta}}}(\boldsymbol{X} = \boldsymbol{x})}. \end{aligned}$$

The denominator $P_{\hat{\boldsymbol{\theta}}}(\boldsymbol{X} = \boldsymbol{x})$ is irrelevant for determining the $y \in \mathcal{Y}$ for which this conditional probability is highest. Therefore,

$$\begin{aligned} \hat{f}(\boldsymbol{x}) &= \operatorname*{arg\,max}_{y \in \mathcal{Y}} \; P_{\hat{\boldsymbol{\theta}}}(Y = y \mid \boldsymbol{X} = \boldsymbol{x}) \\ &= \operatorname*{arg\,max}_{y \in \mathcal{Y}} \; \hat{\pi}_y \cdot \prod_{j=1}^{d} \hat{\mu}_{y,j}^{x_j} (1 - \hat{\mu}_{y,j})^{1 - x_j} \\ &= \operatorname*{arg\,max}_{y \in \mathcal{Y}} \; \ln \left(\hat{\pi}_y \cdot \prod_{j=1}^{d} \hat{\mu}_{y,j}^{x_j} (1 - \hat{\mu}_{y,j})^{1 - x_j} \right) \\ &= \operatorname*{arg\,max}_{y \in \mathcal{Y}} \; \ln \hat{\pi}_y + \sum_{j=1}^{d} x_j \cdot \ln \hat{\mu}_{y,j} + (1 - x_j) \cdot \ln(1 - \hat{\mu}_{y,j}). \end{aligned}$$

This classifier $\hat{f}$ is called a Naive Bayes classifier.

Fitting the Naive Bayes model to the "20 Newsgroups" data

Write code for training and testing a classifier based on the Naive Bayes model described above. Use Laplace smoothing to estimate the $\mu_{y,j}$ parameters, and MLE for the π_y 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`. Compute the training and test error rates.

Binary classification and inspection

Consider the *binary* classification problem, where newsgroups $\{1, 16, 20\}$ comprise the "negative class" (class -1), 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. The training and test data sets for this problem can be downloaded from Courseworks (`news_binary.mat`). Using these data and your codes from the previous part, train and test a Naive Bayes classifier. Compute the training and test error rates.

The binary classifier you learn is ultimately an affine classifier (i.e., a linear classifier with affine feature expansion), which means it has the following form:

$$\boldsymbol{x} \mapsto \begin{cases} -1 & \text{if } w_0 + \sum_{j=1}^{d} w_j x_j \leq 0 \\ +1 & \text{if } w_0 + \sum_{j=1}^{d} w_j x_j > 0 \end{cases}$$

for some real numbers $w_0, w_1, \ldots, w_d$. Determine the values of these w_j's for your learned classifier. Then, report the vocabulary words whose indices $j \in \{1, \ldots, d\}$ correspond to the 20 largest (i.e., most positive) w_j value, and also the vocabulary words whose indices $j \in \{1, \ldots, d\}$ correspond to the 20 smallest (i.e., most negative) w_j value. Don't report the indices j's, but rather the actual vocabulary words (from `news.vocab`).

What to submit

(a) Training and test error rates of the 20-class Naive Bayes classifier $\hat{f}$ on the full 20 Newsgroup data set.
(b) Training and test error rates of the binary classifier on the modified 20 Newsgroup data set.
(c) Two ordered lists (clearly labeled) of 20 words each as described above.

Please submit your source code on Courseworks.

Problem 3 (30 points)

In this problem, you will implement gradient descent for logistic regression, and apply it to text classification (same data set as in Problem 2).

Gradient in logistic regression

Recall the logistic regression objective function, i.e., the empirical logistic loss risk of a linear function (weight vector) on training examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n) \in \mathbb{R}^d \times \{-1, +1\}$:

$$\widehat{\mathcal{R}}(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} \ell_{\log}(y_i \boldsymbol{x}_i^\mathsf{T} \boldsymbol{w})$$

where

$$\begin{aligned} \ell_{\log}(z) &= -\ln \sigma(z) && \text{(logistic loss)}, \\ \sigma(z) &= \frac{1}{1 + \exp(-z)} && \text{(sigmoid function)}. \end{aligned}$$

The derivative of $\ell_{\log}$ is given by

$$\begin{aligned} \frac{\mathrm{d}\ell_{\log}(z)}{\mathrm{d}z} &= -\frac{1}{\sigma(z)} \cdot \frac{\mathrm{d}\sigma(z)}{\mathrm{d}z} \\ &= -\frac{1}{\sigma(z)} \cdot \sigma(z) \cdot \sigma(-z) \\ &= -\sigma(-z). \end{aligned}$$

Therefore, by linearity and the chain rule, the negative gradient of $\widehat{\mathcal{R}}$ with respect to $\boldsymbol{w}$ is

$$\begin{aligned} -\nabla \widehat{\mathcal{R}}(\boldsymbol{w}) &= -\frac{1}{n} \sum_{i=1}^{n} \nabla\, \ell_{\log}(y_i \boldsymbol{x}_i^\mathsf{T} \boldsymbol{w}) \\ &= -\frac{1}{n} \sum_{i=1}^{n} \left. \frac{\mathrm{d}\ell_{\log}(z)}{\mathrm{d}z} \right|_{z = y_i \boldsymbol{x}_i^\mathsf{T} \boldsymbol{w}} \cdot \nabla \left(y_i \boldsymbol{x}_i^\mathsf{T} \boldsymbol{w} \right) \\ &= \frac{1}{n} \sum_{i=1}^{n} \sigma(-y_i \boldsymbol{x}_i^\mathsf{T} \boldsymbol{w}) \cdot y_i \boldsymbol{x}_i. \end{aligned}$$

Now suppose $\boldsymbol{A} = [\boldsymbol{x}_1 | \cdots | \boldsymbol{x}_n]^\mathsf{T} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{b} = [y_1 | \cdots | y_n]^\mathsf{T} \in \mathbb{R}^n$. (Notice that we have omitted the $1/\sqrt{n}$ scaling that we had for least squares linear regression.) Then the negative gradient of $\widehat{\mathcal{R}}$ can be written as

$$-\nabla \widehat{\mathcal{R}}(\boldsymbol{w}) = \frac{1}{n} \boldsymbol{A}^\mathsf{T} (\boldsymbol{b} \odot \sigma(-\boldsymbol{b} \odot (\boldsymbol{A}\boldsymbol{w}))),$$

where $\boldsymbol{u} \odot \boldsymbol{v} \in \mathbb{R}^n$ is the coordinate-wise product of vectors $\boldsymbol{u} \in \mathbb{R}^n$ and $\boldsymbol{v} \in \mathbb{R}^n$, and $\sigma(\boldsymbol{v}) \in \mathbb{R}^n$ is the coordinate-wise application of the sigmoid function to $\boldsymbol{v} \in \mathbb{R}^n$.

Gradient descent for logistic regression

Write a Python function executes gradient descent to approximately minimize the empirical logistic loss risk for a given set of training examples for binary classification. The function should take as input the labeled training examples (e.g., as a data matrix $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ and label matrix $\boldsymbol{b} \in \{-1, +1\}^n$), step size parameter $\eta > 0$, and the number of iterations T. It should execute gradient descent on the empirical logistic loss risk objective starting from an initial weight vector of $\boldsymbol{w}^{(0)} = \boldsymbol{0}$, for T iterations with step size $\eta > 0$, and return the final weight vector $\boldsymbol{w}^{(T)}$.

Text classification with logistic regression

It turns out the training data for the binary classification task in `news_binary.mat` is linearly separable. Use your gradient descent implementation, with step size $\eta := 1$, to find such a linear separator based on the training data. Experimentally determine the number of iterations T needed so that the *training error rate* (i.e., empirical risk with zero-one loss, not logistic loss) of $\boldsymbol{w}^{(T)}$ is zero. What is the test error rate of $\boldsymbol{w}^{(T)}$?

Report the vocabulary words whose indices $j \in \{1, \ldots, d\}$ correspond to the 20 largest (i.e., most positive) $w_j^{(T)}$ value, and also the vocabulary words whose indices $j \in \{1, \ldots, d\}$ correspond to the 20 smallest (i.e., most negative) $w_j^{(T)}$ value. Don't report the indices j's, but rather the actual vocabulary words (from `news.vocab`).

What to submit

(a) Number of iterations T needed to reach a training error rate zero.
(b) Test error rate of $\boldsymbol{w}^{(T)}$.
(c) Two ordered lists (clearly labeled) of 20 words each as described above.

Please submit your source code on Courseworks.

Postscript

You should find that the training error rate of the linear classifier obtained via Naive Bayes (in Problem 2) is non-zero, whereas the training error rate for the linear classifier you find using gradient descent in this problem is zero. Moreover, the corresponding test error rates will have a similar comparison. We say that the linear classifier from Naive Bayes is *under-fit* to the data.

Note that both methods are based on some underlying statistical model for the data (the Naive Bayes model and the logistic regression model). Under-fitting is a sign that an underlying statistical model may not be well-suited for the data. Here is a very good paper that discusses some possible improvements to the Naive Bayes model.

Problem 4 (10 points)

In this problem, you will discover why the non-zero eigenvalues of $\boldsymbol{A}^\top \boldsymbol{A}$ are the same as those of $\boldsymbol{A}\boldsymbol{A}^\top$, and then derive the singular value decomposition.

Suppose a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ is given.

(a) Suppose λ is a non-zero eigenvalue of $\boldsymbol{A}^\top \boldsymbol{A}$ with corresponding eigenvector $\boldsymbol{v} \neq \boldsymbol{0}$. Prove that λ is an eigenvalue of $\boldsymbol{A}\boldsymbol{A}^\top$.

(b) Suppose 0 is an eigenvalue of $\boldsymbol{A}^\top \boldsymbol{A}$ with corresponding eigenvector $\boldsymbol{v} \neq \boldsymbol{0}$. Prove that $\boldsymbol{v}$ is in the null space of $\boldsymbol{A}$.

(c) Explain why the facts in Parts (a) and (b) imply that $\boldsymbol{A}^\top \boldsymbol{A}$ and $\boldsymbol{A}\boldsymbol{A}^\top$ have the same non-zero eigenvalues.

Now let $\lambda_1 \geq \cdots \geq \lambda_d \geq 0$ be the eigenvalues of $\boldsymbol{A}^\top \boldsymbol{A}$, and let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d$ be corresponding orthonormal eigenvectors. Let r denote the number of λ_j's that are (strictly) positive. Define, for each $j = 1, \ldots, r$,

$$\boldsymbol{u}_j := \frac{1}{\|\boldsymbol{A}\boldsymbol{v}_j\|_2} \boldsymbol{A}\boldsymbol{v}_j.$$

(d) Prove that $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r$ are orthonormal.

(e) Prove the following statement. For any two matrices $\boldsymbol{M} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{M'} \in \mathbb{R}^{n \times d}$ and any collection of d linearly independent vectors $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_d \in \mathbb{R}^d$, if $\boldsymbol{M}\boldsymbol{q}_i = \boldsymbol{M'}\boldsymbol{q}_i$ for all $i = 1, \ldots, d$, then $\boldsymbol{M} = \boldsymbol{M'}$.

(f) Use the strategy suggested in the claim from Part (e) to prove that

$$\boldsymbol{A} = \sum_{j=1}^{r} \sqrt{\lambda_j} \boldsymbol{u}_j \boldsymbol{v}_j^\top.$$

Note: Answers to most of the parts of this problem can be found in the reading material. It is fine to use the reading material to help you with this problem, although we recommend you attempt it on your own first. In any case, you must write your answers in your own words, without referencing the reading material.

Postscript

The decomposition of $\boldsymbol{A}$ in Part (f) is called the *singular value decomposition (SVD)* of $\boldsymbol{A}$. The vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r$ are the *left singular vectors* of $\boldsymbol{A}$, the vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r$ are the *right singular vectors* of $\boldsymbol{A}$, and the positive scalars $\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_r}$ are the *singular values* of $\boldsymbol{A}$ (usually denoted by $\sigma_1, \ldots, \sigma_r$). Every matrix has such a decomposition!

We can "extend" the decomposition beyond the rank of $\boldsymbol{A}$ to have $\min\{n, d\}$ components as follows. We extend the orthonormal bases $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r$ to $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r$ to $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d$ (both via Gram-Schmidt orthogonalization), and then define $\sigma_j := 0$ for $j > r$. Then we can write

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$$

where $\boldsymbol{U} = [\boldsymbol{u}_1 | \cdots | \boldsymbol{u}_n] \in \mathbb{R}^{n \times n}$, $\boldsymbol{V} = [\boldsymbol{v}_1 | \cdots | \boldsymbol{v}_d] \in \mathbb{R}^{d \times d}$, and $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times d}$ is the matrix whose (j, j)-th entry is σ_j for $j = 1, \ldots, \min\{n, d\}$, and all other entries are zero. Since $\boldsymbol{V}$ and $\boldsymbol{U}$ are orthogonal matrices, we have $\boldsymbol{A}^\top \boldsymbol{A} = \boldsymbol{V}(\boldsymbol{\Sigma}^\top \boldsymbol{\Sigma})\boldsymbol{V}^\top$ and $\boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{U}(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^\top)\boldsymbol{U}^\top$; this reveals the eigendecompositions of $\boldsymbol{A}^\top \boldsymbol{A}$ and $\boldsymbol{A}\boldsymbol{A}^\top$, respectively since $\boldsymbol{\Sigma}^\top \boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}\boldsymbol{\Sigma}^\top$ are diagonal matrices.

The SVD is useful for understanding some regularization methods for linear regression. For any $\boldsymbol{b} \in \mathbb{R}^n$:

- Ridge regression (with hyperparameter $\lambda > 0$):

$$\hat{\boldsymbol{w}}_{\text{ridge},\lambda} = \sum_{j=1}^{r} \frac{\sigma_j}{\sigma_j^2 + \lambda} \boldsymbol{v}_j \boldsymbol{u}_j^\top \boldsymbol{b}.$$

- Principal component regression (with hyperparameter $\lambda > 0$):

$$\hat{\boldsymbol{w}}_{\text{pcr},\lambda} = \sum_{j=1}^{r} \mathbb{1}_{\{\sigma_j^2 \geq \lambda\}} \cdot \frac{1}{\sigma_j} \boldsymbol{v}_j \boldsymbol{u}_j^\top \boldsymbol{b}.$$

- Minimum Euclidean norm solution to normal equations:

$$\hat{\boldsymbol{w}}_{\text{min. Euclidean norm}} = \sum_{j=1}^{r} \frac{1}{\sigma_j} \boldsymbol{v}_j \boldsymbol{u}_j^\top \boldsymbol{b}.$$

This final expression can be written as

$$\hat{\boldsymbol{w}}_{\text{min. Euclidean norm}} = \boldsymbol{A}^\dagger \boldsymbol{b}$$

where

$$\boldsymbol{A}^\dagger := \sum_{j=1}^{r} \frac{1}{\sigma_j} \boldsymbol{v}_j \boldsymbol{u}_j^\top$$

is called the *Moore-Penrose pseudoinverse* of $\boldsymbol{A}$. (You might want to use this fact on Problem 1. In NumPy, it is `numpy.linalg.pinv`.)

Problem 5 (10 points)

In this problem, you will practice Bayesian inference with multivariate Gaussian distributions.

Consider the following statistical model for n scalar observations, $Y_1, \ldots, Y_n$:

$$\boldsymbol{Y} := (Y_1, \ldots, Y_n) \sim \mathrm{N}(\boldsymbol{\theta}, \boldsymbol{I}),$$

where $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n) \in \mathbb{R}^n$ is the parameter vector for the model.

(a) Give a formula for the MLE of $\boldsymbol{\theta}$ given data $y_1, \ldots, y_n \in \mathbb{R}$. Your answer should be given in terms of $y_1, \ldots, y_n$. *Hint: It is very simple.*

(b) Let us now take $\mathrm{N}(\mathbf{0}, \tau \boldsymbol{I})$ to be a prior distribution for the parameter $\boldsymbol{\theta}$, for some fixed positive constant $\tau > 0$. Show that, for any data $y_1, \ldots, y_n \in \mathbb{R}$, the posterior distribution of $\boldsymbol{\theta}$ given $\boldsymbol{Y} = (y_1, \ldots, y_n)$ is a multivariate Gaussian distribution, and give explicit formulas for the mean vector and covariance matrix of that multivariate Gaussian distribution in terms of the data $y_1, \ldots, y_n$ and the prior parameter τ. *Hint: It suffices to determine the posterior distribution of θ_i given $Y_i = y_i$.*

(c) Suppose you take the prior belief from Part (b) very seriously, and instead assume the data is generated as follows. First, we draw $\boldsymbol{\theta}$ from $\mathrm{N}(\mathbf{0}, \tau \boldsymbol{I})$. Then, conditioned on the value of $\boldsymbol{\theta}$, we draw $\boldsymbol{Y}$ from $\mathrm{N}(\boldsymbol{\theta}, \boldsymbol{I})$; i.e., $\boldsymbol{Y} \mid \boldsymbol{\theta} \sim \mathrm{N}(\boldsymbol{\theta}, \boldsymbol{I})$. Show that the *marginal distribution* of $\boldsymbol{Y}$ is a multivariate Gaussian distribution with mean $\mathbf{0}$ and covariance matrix $(1 + \tau)\boldsymbol{I}$. *Hint: It suffices to show that the marginal distribution of Y_i is $\mathrm{N}(0, 1 + \tau)$.*

Postscript

Assume $n \geq 3$. It turns out that under the statistical model for $\boldsymbol{Y}$ from Part (c), we have

$$\mathbb{E}\left[\frac{n-2}{\|\boldsymbol{Y}\|_2^2}\right] = \frac{1}{1+\tau}.$$

The *James-Stein* estimator for $\boldsymbol{\theta}$ (in the original model $\boldsymbol{Y} \sim \mathrm{N}(\boldsymbol{\theta}, \boldsymbol{I})$) is given by

$$\left(1 - \frac{n-2}{\|\boldsymbol{y}\|_2^2}\right)\boldsymbol{y}$$

for any $\boldsymbol{y} = (y_1, \ldots, y_n)$. (One motivation for this estimator comes from the answer to Part (b).) This estimator is superior to the MLE (i.e., the answer in Part (a)) in a very surprising sense.

About grading

This assignment will be graded out of 100 points, so you may earn up to 10 points extra credit. (Ignore the placeholder point totals on Gradescope.)