COMS 4721 Spring 2022 Homework 3

Due at 11:59 PM on April 1, 2022

Grade for a *late submission* is scaled by factor of $(0.75)^{(\text{\# days late, rounded up to an integer})}$

About this assignment. This homework assignment is a mix of some problems that involve programming and data analysis using Python, as well as some problems that do not involve programming but may require a bit of mathematical reasoning, simple derivations, and/or "calculations" (usually done by hand).

Data files needed for the programming/data analysis parts are available on Courseworks (under "Files").

Working in pairs is explicitly permitted on this assignment (subject to the conditions detailed in the syllabus).

Submission. For the programming and data analysis portions, please create and use a Jupyter Notebook to solve these problems. Include all of the code you write for this assignment in the notebook, and add appropriate comments so that the code is easy to understand. Make sure your answers to the various questions are clearly stated and labeled in separate Markdown cells (even if the answer also appears as an output of a code cell). You will need to submit the Jupyter Notebook on Courseworks as part of your homework submission.

You will also need to create a PDF document containing your solutions to *all problems* (including those involving programming and data analysis). **This PDF document should also contain the code and results from the Jupyeter Notebook.** Please read the homework policy (in the syllabus) for information about how to write-up your solutions, and please see the instructions for properly converting a Jupyter Notebook to PDF. You will need to submit this document on Gradescope.

- For redundancy purposes, please make sure your name and UNI appear prominently on the first page of the PDF. If you are working in pairs, both students' names and UNIs should appear in this manner.
- On Gradescope, for each problem, you need to **select all of the pages that contain your solutions** for that problem. If part of your solution is on an unselected page, we will consider it missing. See the instructions for submitting a PDF on Gradescope.
- On Gradescope, if you are working in pairs, then only one student in the pair should make the submission. That student must make sure to **add both students in the pair to the submission**. This is something that has to be done at the time of submission within the Gradescope submission system. See the instructions for adding group members on Gradescope. Do not forget to do this, or else one student will not receive credit for the homework submission!
- On Courseworks, submit only the Jupyter Notebook (a `.ipynb` file). Do not include any of the data files—we already have those `:-)`
- On Courseworks, if you are working in pairs, then only one student in the pair should make the submission of the Jupyter Notebook. (We are only using this to collect your Jupyter Notebooks; the grading will happen on Gradescope.) If both students submit, we'll only look at one of the submissions (and we'll pick one arbitrarily and ignore the other).
- Please submit both your write-up and Jupyter Notebook by the deadline. Do not wait until the last minute to make your submissions!

Please consult the Gradescope Help Center if you need further help with submitting on Gradescope, and the Canvas Guides if you need further help with submitting on Courseworks.

1 Ridge regression and gradient descent

In this section, $A \in \mathbb{R}^{n \times d}$ is a $n \times d$ matrix, and $\vec{b} \in \mathbb{R}^n$ is an n-dimensional vector.

In lecture, we claimed that the minimizer of the ridge regression objective function

$$J(\vec{w}) := \|A\vec{w} - \vec{b}\|_2^2 + \lambda\|\vec{w}\|_2^2$$

(when $\lambda > 0$) is always unique, even when $A^\intercal A$ is not invertible. The reason we gave for this was that $A^\intercal A + \lambda I$ is always invertible. Why is this the case?

Problem 1.1 (1 point). Explain why $A^\intercal A$ is a symmetric matrix, and also explain what that implies about the eigenvalues of $A^\intercal A$.

Problem 1.2 (1 point). Explain why the eigenvalues of $A^\intercal A$ are non-negative. (Avoid using the phrase "positive semi-definite" in your explanation.)

Problem 1.3 (1 point). If the d eigenvalues of $A^\intercal A$ are $\lambda_1, \ldots, \lambda_d$, then what are the d eigenvalues of $A^\intercal A + \lambda I$? Explain your answer.

Problem 1.4 (1 point). Verify the properties/explanations from the previous three problems for the eigenvalues of $A^\intercal A + \lambda I$, where $\lambda = 1$ and

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & -2 \end{bmatrix}.$$

Your answer should explicitly state the eigenvalues of $A^\intercal A$ and of $A^\intercal A + \lambda I$, as well as a discussion of the properties/explanations from the previous three problems.

Problem 1.5 (1 point). Write a formula for $(A^\intercal A + \lambda I)^{-1}$ (assuming $\lambda > 0$) in terms of the eigenvalues $\lambda_1, \ldots, \lambda_d$ of $A^\intercal A$, the corresponding eigenvectors $\vec{v}_1, \ldots, \vec{v}_d$, and the parameter λ. You may assume, without loss of generality, that $\vec{v}_1, \ldots, \vec{v}_d$ form an orthonormal basis for $\mathbb{R}^d$. Verify your formula for the example in the previous problem.

The ridge regression objective can be approximately minimized using gradient descent. The first step to implement gradient descent for this objective is to derive a formula for the gradient of the objective.

Problem 1.6 (1 point). Write a formula for the gradient of J, evaluated at an arbitrary point $\vec{w}_0 \in \mathbb{R}^d$. Your formula should be given in terms of A, $\vec{b}$, λ, and $\vec{w}_0$.

The next step is to implement the gradient formula.

Problem 1.7 (1 point). Recall that, in terms of training examples $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$, we typically think of the matrix A as having $\vec{x}_i^\intercal$ in its i-th row, and the vector $\vec{b}$ as having y_i in its i-th entry. Write pseudocode for implementing the t-th iterate of the gradient descent update rule for minimizing J in terms of these training examples. Assume that the previous iterate $\vec{w}^{(t-1)}$ is already stored in memory in a variable called `w`, and refer to $\vec{x}_i$ and y_i, respectively, by `x[i]` and `y[i]`. Also assume that the regularization parameter λ and step size η, respectively, are given in variables `lambda` and `eta`. Your pseudocode may use the following types of operations (in addition to standard control structures like for-loops, and standard scalar arithmetic):

- `zero(d)`: return a new vector of dimension `d`, with every entry initialized to zero
- `scale(w,c)`: scale the vector `w` by a real number `c` (i.e., `w := c * w`)
- `add(w,c,v)`: add `c` times the vector `v` to the vector `w` (i.e., `w := w + c * v`)
- `ip(u,v)`: return the inner product between vectors `u` and `v`

Problem 1.8 (1 point). In general, each of `new`, `scale`, `add`, and `ip` requires $O(d)$ basic operations (when dealing with vectors in $\mathbb{R}^d$). How does the running time of gradient descent scale with n, d, and the number of iterations T? Give your answer using "Big-O" notation. Briefly explain your answer.

Problem 1.9 (1 point). Implement gradient descent for the ridge regression objective in Python using the `numpy` library. (Feel free to consult the `numpy` documentation.) Use your implementation with the prostate cancer data set from HW2 (`prostate-train.csv`) to find (an approximation to) the best fit affine function of the features (`lcavol`, `lweight`, `age`, `lbph`, `svi`, `lcp`, `gleason`, `pgg45`) for predicting `lpsa` (as in HW2). Here, you should use $\lambda = 0$ so that ridge regularization is "turned off". And also remember to use affine feature expansion. Initialize gradient descent with the zero vector; use step size $\eta = 2.5 \times 10^{-6}$; and stop after 10^6 iterations.

(a) Report the coefficients in the weight vector and the intercept term.
(b) Report the SSE achieved by the solution from the previous problem.
(c) Also, plot the SSE's achieved by the intermediate iterates as a function of the iteration number. As usual, make sure axes are labeled and the plot is given an appropriate caption/title.

2 Cost-weighted logarithmic losses

Let $(\vec{X}, Y)$ denote a random example for binary classification. This means $\vec{X}$ takes values in $\mathbb{R}^d$, and Y takes values in $\{0, 1\}$. Let $\eta(\vec{x}) = \Pr(Y = 1 \mid \vec{X} = \vec{x})$ denote the conditional probability function for $(\vec{X}, Y)$. Throughout this section, we assume the cost of a false positive mistake is $c_{\mathrm{FP}} > 0$, the cost of a false negative mistake is $c_{\mathrm{FN}} > 0$, and correct predictions incur zero cost. Assume that $c_{\mathrm{FP}} \neq c_{\mathrm{FN}}$.

The modified training objective we considered in lecture can be thought of as an empirical version of the following objective:

$$R(p) := \mathbb{E}[c(Y) \times \ell_{\log}(Y, p(\vec{X}))].$$

Here, we think of R as a function of a predictor $p \colon \mathbb{R}^d \to [0, 1]$. Moreover, $\ell_{\log}$ is the logarithmic loss, $c(0) = c_{\mathrm{FP}}$, and $c(1) = c_{\mathrm{FN}}$.

Problem 2.1 (1 point). Consider the function $f(t) := -\ln(t)$, which is the logarithmic loss for a prediction of t when the correct label is 1. It turns out that f is a convex function over the positive real numbers. To see why this is the case, we'll show that it satisfies the definition of convexity for differentiable functions. Let t and t_0 denote arbitrary positive numbers. We can write $f(t)$ as

$$f(t) = -\ln(t_0) - \ln(t/t_0) = f(t_0) - \ln(t/t_0). \tag{1}$$

Therefore, to establish that f is convex, it is enough to establish that

$$\ln(t/t_0) \leq -f'(t_0) \times (t - t_0), \tag{2}$$

where f' denotes the derivative of f. This is because combining Equations (1) and (2) gives the inequality

$$f(t) \geq f(t_0) + f'(t_0) \times (t - t_0),$$

which shows that the definition of convexity is satisfied by f. So for this problem, your task is to explain why Equation (2) is true for arbitrary positive numbers t and t_0. You can either use an analytic argument, or simply make a graphical plot to establish an inequality that implies Equation (2). You should, of course, explain any plot you make and what implications you draw from it. (Any plots should be included directly in your write-up.)

As you might expect, the logarithmic loss for a prediction when the correct label is 0 is also a convex function of the prediction. Because a non-negative linear combination of convex functions is, itself, a convex function, it turns out that for any fixed $\vec{x} \in \mathbb{R}^d$, the function

$$g_{\vec{x}}(t) := \mathbb{E}[c(Y) \times \ell_{\log}(Y, t) \mid \vec{X} = \vec{x}]$$

is convex as well. This means that any value t that satisfies $g'_{\vec{x}}(t) = 0$ (where $g'_{\vec{x}}$ denotes the derivative of $g_{\vec{x}}$) must be a minimizer of $g_{\vec{x}}$.

Problem 2.2 (1 point). Describe the function $p \colon \mathbb{R}^d \to [0, 1]$ that minimizes the objective $R(p)$. Actually, instead of describing the function p directly, describe the following function based on p:

$$\ln\left(\frac{p(\vec{x})}{1 - p(\vec{x})}\right).$$

Your answer should be given in terms of the conditional probability function and the costs c_{FP} and c_{FN}. Explain how you derived your answer.

Problem 2.3 (1 point). Let p be the function that minimizes R. Suppose $\vec{x} \in \mathbb{R}^d$ is such that $p(\vec{x}) > 1/2$. What does this imply about the value of $\eta(\vec{x})$? Explain your answer.

Problem 2.4 (1 point). Consider the iris classification problem from lecture; the data is available in the file `iris-train.csv`. Treat the first and second species of irises together as "class 0", and treat the third species of irises as "class 1", so the classification problem is now a binary classification problem. (Note that this is different from what was done in lecture.) Assume that false negative mistakes are twenty times as costly as false positive mistakes (i.e., $c_{\mathrm{FN}} = 20c_{\mathrm{FP}}$). Use gradient descent to approximately minimize the empirical version of $R(p)$, where $p(\vec{x})$ is the conditional probability function from a logistic regression model with affine feature expansion. You should initialize gradient descent with all parameter values (both the weight vector and the intercept parameter) at zero. Experiment with the step size and number of iterations needed to obtain convergence in the objective value; and explain, in words, what you did as part of your answer. Report the final number of false positive mistakes and the number of false negative mistakes, both computed just on the training data.

You may use the automatic differentiation facilities in PyTorch if you like, but it is not required. However, you should not use any library functions that directly implement gradient descent (or other optimization procedures).

Problem 2.5 (1 point). You fit a logistic regression model in the previous problem, so you can obtain estimates of the conditional probabilities by composing the logistic function with the affine function $\vec{x} \cdot \vec{w} + \theta$ you learned, i.e., $\hat{p}(\vec{x}) = \mathrm{logistic}(\vec{x} \cdot \vec{w} + \theta)$. Are these conditional probabilities approximately calibrated on the training data? To assess this, divide the interval $[0, 1]$ into ten equal length subintervals $B_1 = [0, 0.1)$ $B_2 = [0.1, 0.2)$, etc.; then, for each subinterval B_i, determine the fraction of training examples with label 1 among all training examples whose conditional probability estimate falls in the interval B_i. Show the results in a table.

3 Classifying restaurant reviews

The Yelp restaurant reviews dataset is comprised of user-contributed reviews posted to Yelp for restaurants in Pittsburgh collected several years ago. Each review is accompanied by a binary indicator of whether or not the reviewer-assigned rating is at least four (on a five-point scale). The text of the reviews has been processed to replace all non-alphanumeric symbols with whitespace, and all letters have been changed to lowercase. The prediction problem we consider here is predicting the binary indicator from the review text.

The training data is contained in the file `reviews_tr.csv`, and the test data is contained in the file `reviews_te.csv`. The following code can be used to load the training data.

```python
from csv import DictReader

vocab = {}
vocab_size = 0
examples = []
with open('reviews_tr.csv', 'r') as f:
    reader = DictReader(f)
    for row in reader:
        label = row['rating'] == '1'
        words = row['text'].split(' ')
        for word in words:
            if word not in vocab:
                vocab[word] = vocab_size
                vocab_size += 1
        examples.append((label, [vocab[word] for word in words]))
```

This code assigns every word that appears in training data a unique non-negative integer, and the mapping is provided in the dictionary `vocab`. Each training example is represented as a binary label paired with a list of non-negative integers (which we'll call "word IDs") corresponding to the words that appear in the review. Note that the list may contain repetitions of any given word ID (since a review may contain repetitions of a word). The examples are collected in the array `examples`. Throughout the rest of this section, we'll refer to the variables defined by this code.

Of course, the test data can be loaded in a similar way. Note that the test data may contain words that do not appear in the training data.

Different examples may have different number of words, so the lists of word IDs may have different lengths. It may not be obvious how they can be used with common machine learning algorithms. However, it is relatively easy to transform a list of word IDs into a "bag-of-words" vector (also called "term frequency" vector), which is a vector $\vec{x}$ where the j-th entry is the number of times word ID j appears in the list. **This is the representation you should use for the feature vectors throughout this section.** The dimension of the vector is equal to the vocabulary size (and the vector indices, like the word IDs, start from 0). Such a vector representation is readily usable by many machine learning algorithms. The following code applies this transformation to the first training example.

```python
from numpy import zeros

def bag_of_words_rep(word_ids, dim):
    bow_vector = zeros(dim) # creates a numpy.ndarray of shape (dim,)
    for word_id in word_ids:
        bow_vector[word_id] += 1
    return bow_vector

first_bow_vector = bag_of_words_rep(examples[0][1], vocab_size)
```

Note that applying this transformation to all training examples would result in a large collection of high-dimensional vectors (each of dimension `vocab_size`), ultimately consuming a lot of memory. So it may be better to only *implicitly* use the bag-of-words representation.

Problem 3.1 (1 point). Give a *rough estimate* of the amount of memory (in bytes) that would be used by the training data if all examples were represented using the bag-of-words representation generated by the code shown above. How does it compare to the total memory used by `examples`? Explain how you arrived at your estimates.

3.1 Online Perceptron

The *Online Perceptron* algorithm is a variant of the standard Perceptron algorithm. Recall that the standard Perceptron algorithm maintains a weight vector $\vec{w}$, and repeatedly updates the weight vector with any training example that is misclassified by the linear classifier with parameter $\vec{w}$. This continues until there are no misclassified training examples. Of course, if the training data is not linearly separable, this will go on forever — some examples will be used to update the weight vector infinitely often. The Online Perceptron, shown below, rectifies this by only considering each training example exactly once, in some fixed ordering over the training examples, and then halting.[1]

- Input: training examples $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$
- Initialize $\vec{w}_0 := \vec{0}$
- For $i = 1, 2, \ldots, n$:
  - If $(\vec{x}_i, y_i)$ is misclassified by the linear classifier with parameter $\vec{w}_{i-1}$, then

$$\vec{w}_i := \begin{cases} \vec{w}_{i-1} + \vec{x}_i & \text{if } y_i = \text{true} \\ \vec{w}_{i-1} - \vec{x}_i & \text{if } y_i = \text{false} \end{cases}$$

  - Else $\vec{w}_i := \vec{w}_{i-1}$
- Return: $\vec{w}_n$

Here, we have assumed that each label is either "true" or "false", and that a linear classifier with parameter $\vec{w}$ predicts "true" on an input feature vector $\vec{x}$ if and only if $\vec{w} \cdot \vec{x} > 0$. (Don't bother with affine expansion.)

Problem 3.2 (2 points). Implement the Online Perceptron algorithm and apply it to the training data (in the order that the examples are already given in `reviews_tr.csv`). What is the training error rate of the linear classifier returned by Online Perceptron? And what is the test error rate?

Recall, that the test data may contain words that do not appear in the training data. It may seem that this could be somewhat cumbersome to deal with. However, you should be able to handle this issue in a relatively simple way by using the fact that the initial weight vector in Online Perceptron is the zero vector.

Problem 3.3 (1 point). To get a sense of the behavior of the linear classifier that you obtained above, determine the 10 words that have the highest (i.e., most positive) weights in the weight vector, and also determine the 10 words that have the lowest (i.e., most negative) weights in the weight vector. Report the actual words, not the word IDs. You may find it helpful to invert the mapping given in `vocab`.

3.2 Averaged Perceptron

The *Averaged Perceptron* algorithm is a variant of Online Perceptron. It is exactly the same as Online Perceptron, except that the weight vector returned is $\vec{w}_{\text{avg}} := \frac{1}{n}\sum_{i=1}^{n} \vec{w}_i$ (instead of just $\vec{w}_n$). This is more "stable" than Online Perceptron, since the average of the weight vectors is much less sensitive to a single training example than the current weight vector maintained by Online Perceptron.

Problem 3.4 (1 point). Suppose instead of returning $\vec{w}_{\text{avg}}$, your implementation of Averaged Perceptron returns $\vec{w}_{\text{sum}} := \sum_{i=1}^{n} \vec{w}_i$. Why would this be okay?

[1] We say that Online Perceptron makes a (single) *pass* through the training data.

Problem 3.5 (2 points). Implement the Averaged Perceptron algorithm and apply it to the training data (in the order that the examples are already given in `reviews_tr.csv`). What are the training error rate and the test error rate of the linear classifier returned by Averaged Perceptron?

Problem 3.6 (1 point). Repeat Problem 3.3 for the linear classifier obtained in Problem 3.5.

3.3 Improvements

There are many ways to easily improve Online Perceptron and Averaged Perceptron. Below, we discuss two possible ways.

First, you may consider improving the feature representation. So far, we have only used the bag-of-words representation of the reviews. It is called bag-of-words because it only depends on the number of times words appear in the review; the order of the words doesn't matter at all. One way to take the order of words into account (in a very limited way) is to also consider the number of times *bigrams* appear in the review. A bigram is a pair of words that appear consecutively. For example, in "a rose is a rose", the bigrams that appear are: (a, rose), which appears twice; (rose, is); and (is, a). If there are d words in a vocabulary, then there are d^2 possible bigrams, which can be enormous when d is even just moderately large. This makes it even more important to avoid explicitly forming the bag-of-words-and-bigrams feature vectors.

Second, you may consider making multiple passes through the training data (instead of just a single pass). Often even just one additional pass through the training data can yield improvements. Of course, increasing the number of passes also increases the training time. If using Averaged Perceptron, it may also be better to only average the weights from the final pass through the training data.

Problem 3.7 (1 point). Implement one of the above suggested improvements, or any other improvement you can think of. (If you come up with your own improvement, give a high-level description of it and explain its motivation. If you one of the suggested improvements from above, please describe it in your own words at a high-level.) Apply the new algorithm to the training data. What is the test error rate of the classifier you obtain?