Homework #2

CSE 446/546: Machine Learning
Prof. Kevin Jamieson and Prof. Jamie Morgenstern
Due: **Tuesday** May 12, 2020 11:59 PM

Please review all homework guidance posted on the website before submitting to Gradescope. Reminders:

- Please provide succinct answers along with succinct reasoning for all your answers. Points may be deducted if long answers demonstrate a lack of clarity. Similarly, when discussing the experimental results, concisely create tables and/or figures when appropriate to organize the experimental results. In other words, all your explanations, tables, and figures for any particular part of a question must be grouped together.
- When submitting to gradescope, please link each question from the homework in gradescope to the location of its answer in your homework PDF. Failure to do so may result in point deductions. For instructions, see `https://www.gradescope.com/get_started#student-submission`.
- Please recall that B problems, indicated in $\boxed{\text{boxed text}}$ are only graded for 546 students, and that they will be weighted at most 0.2 of your final GPA (see website for details). In Gradescope there is a place to submit solutions to A and B problems seperately. You are welcome to create just a single PDF that contains answers to both, submit the same PDF twice, but associate the answers with the individual questions in gradescope.
- If you collaborate on this homework with others, you must indicate who you worked with on your homework. Failure to do so may result in accusations of plagiarism.

Conceptual Questions *[10 points]*

A0. The answers to these questions should be answerable without referring to external materials. Briefly justify your answers with a few words.

a. *[2 points]* Suppose that your estimated model for predicting house prices has a large positive weight on 'number of bathrooms'. Does it implies that if we remove the feature "number of bathrooms" and refit the model, the new predictions will be strictly worse than before? Why?

b. *[2 points]* Compared to L2 norm penalty, explain why a L1 norm penalty is more likely to result in a larger number of 0s in the weight vector or not?

c. *[2 points]* In at most one sentence each, state one possible upside and one possible downside of using the following regularizer: $\left(\sum_i |w_i|^{0.5}\right)$

d. *[1 points]* True or False: If the step-size for gradient descent is too large, it may not converge.

e. *[2 points]* In your own words, describe why SGD works.

f. *[2 points]* In at most one sentence each, state one possible advantage of SGD (stochastic gradient descent) over GD (gradient descent) and one possible disadvantage of SGD relative to GD.

Convexity and Norms *[30 points]*

A1. A *norm* $\|\cdot\|$ over $\mathbb{R}^n$ is defined by the properties: i) non-negative: $\|x\| \geq 0$ for all $x \in \mathbb{R}^n$ with equality if and only if $x = 0$, ii) absolute scalability: $\|a\,x\| = |a|\,\|x\|$ for all $a \in \mathbb{R}$ and $x \in \mathbb{R}^n$, iii) triangle inequality: $\|x+y\| \leq \|x\| + \|y\|$ for all $x, y \in \mathbb{R}^n$.

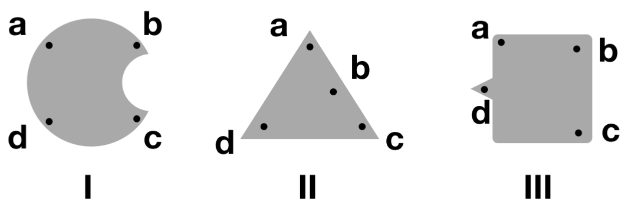
a. *[3 points]* Show that $f(x) = \left(\sum_{i=1}^n |x_i|\right)$ is a norm. (Hint: begin by showing that $|a+b| \leq |a| + |b|$ for all $a, b \in \mathbb{R}$.)

b. *[2 points]* Show that $g(x) = \left(\sum_{i=1}^n |x_i|^{1/2}\right)^2$ is not a norm. (Hint: it suffices to find two points in $n = 2$ dimensions such that the triangle inequality does not hold.)

Context: norms are often used in regularization to encourage specific behaviors of solutions. If we define $\|x\|_p := \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$ then one can show that $\|x\|_p$ is a norm for all $p \geq 1$. The important cases of $p = 2$ and $p = 1$ correspond to the penalty for ridge regression and the lasso, respectively.

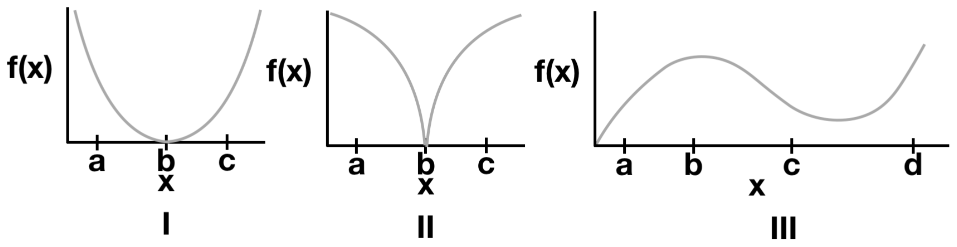
B1. *[6 points]* For any $x \in \mathbb{R}^n$, define the following norms: $\|x\|_1 = \sum_{i=1}^n |x_i|$, $\|x\|_2 = \sqrt{\sum_{i=1}^n |x_i|^2}$, $\|x\|_\infty := \lim_{p\to\infty} \|x\|_p = \max_{i=1,\dots,n} |x_i|$. Show that $\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1$.

A2. *[3 points]* A set $A \subseteq \mathbb{R}^n$ is *convex* if $\lambda x + (1-\lambda)y \in A$ for all $x, y \in A$ and $\lambda \in [0,1]$.



For each of the grey-shaded sets above (I-III), state whether each one is convex, or state why it is not convex using any of the points a, b, c, d in your answer.

A3. *[4 points]* We say a function $f : \mathbb{R}^d \to \mathbb{R}$ is convex on a set A if $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$ for all $x, y \in A$ and $\lambda \in [0,1]$.



For each of the grey-colored functions below (I-III), state whether each one is convex on the given interval or state why not with a counterexample using any of the points a, b, c, d in your answer.

a. Function in panel I on $[a, c]$

b. Function in panel II on $[a, c]$

c. Function in panel III on $[a, d]$

d. Function in panel III on $[c, d]$

B2. Use just the definitions above and let $\|\cdot\|$ be a norm.

a. *[3 points]* Show that $f(x) = \|x\|$ is a convex function.

b. *[3 points]* Show that $\{x \in \mathbb{R}^n : \|x\| \leq 1\}$ is a convex set.

c. *[2 points]* Draw a picture of the set $\{(x_1, x_2) \ : \ g(x_1, x_2) \leq 4\}$ where $g(x_1, x_2) = \left(|x_1|^{1/2} + |x_2|^{1/2}\right)^2$. (This is the function considered in 1b above specialized to $n = 2$.) We know g is not a norm. Is the defined set convex? Why not?

Context: It is a fact that a function f defined over a set $A \subseteq \mathbb{R}^n$ is convex if and only if the set $\{(x, z) \in \mathbb{R}^{n+1} : z \geq f(x), x \in A\}$ is convex. Draw a picture of this for yourself to be sure you understand it.

B3. For $i = 1, \ldots, n$ let $\ell_i(w)$ be convex functions over $w \in \mathbb{R}^d$ (e.g., $\ell_i(w) = (y_i - w^\top x_i)^2$), $\|\cdot\|$ is any norm, and $\lambda > 0$.

a. *[3 points]* Show that

$$\sum_{i=1}^{n} \ell_i(w) + \lambda \|w\|$$

is convex over $w \in \mathbb{R}^d$ (Hint: Show that if f, g are convex functions, then $f(x) + g(x)$ is also convex.)

b. *[1 points]* Explain in one sentence why we prefer to use loss functions and regularized loss functions that are convex.

Lasso *[45 points]*

Given $\lambda > 0$ and data $(x_1, y_1), \ldots, (x_n, y_n)$, the Lasso is the problem of solving

$$\arg \min_{w \in \mathbb{R}^d, b \in \mathbb{R}} \sum_{i=1}^{n} (x_i^T w + b - y_i)^2 + \lambda \sum_{j=1}^{d} |w_j|$$

λ is a regularization tuning parameter. For the programming part of this homework, you are required to implement the coordinate descent method of Algorithm 1 that can solve the Lasso problem.
You may use common computing packages (such as NumPy or SciPy), but do not use an existing Lasso solver (e.g., of scikit-learn).
Before you get started, here are some hints that you may find helpful:

Algorithm 1: Coordinate Descent Algorithm for Lasso

```
while not converged do
    $b \leftarrow \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{d} w_j x_{i,j}\right)$
    for $k \in \{1, 2, \cdots d\}$ do
        $a_k \leftarrow 2 \sum_{i=1}^{n} x_{i,k}^2$
        $c_k \leftarrow 2 \sum_{i=1}^{n} x_{i,k} \left(y_i - (b + \sum_{j \neq k} w_j x_{i,j})\right)$
        $w_k \leftarrow \begin{cases} (c_k + \lambda)/a_k & c_k < -\lambda \\ 0 & c_k \in [-\lambda, \lambda] \\ (c_k - \lambda)/a_k & c_k > \lambda \end{cases}$
    end
end
```

- For-loops can be slow whereas vector/matrix computation in Numpy is very optimized; exploit this as much as possible.
- The pseudocode provided has many opportunities to speed up computation by precomputing quantities like a_k before the for loop. These small changes can speed things up considerably.
- As a sanity check, ensure the objective value is nonincreasing with each step.
- It is up to you to decide on a suitable stopping condition. A common criteria is to stop when no element of w changes by more than some small δ during an iteration. If you need your algorithm to run faster, an easy place to start is to loosen this condition.
- You will need to solve the Lasso on the same dataset for many values of λ. This is called a regularization path. One way to do this efficiently is to start at a large λ, and then for each consecutive solution, initialize the algorithm with the previous solution, decreasing λ by a constant ratio (e.g., by a factor of 2) until finished.
- The smallest value of λ for which the solution $\widehat{w}$ is entirely zero is given by

  $$\lambda_{max} = \max_{k=1,\dots,d} 2\left|\sum_{i=1}^{n} x_{i,k}\left(y_i - \left(\frac{1}{n}\sum_{j=1}^{n} y_j\right)\right)\right|$$

  This is helpful for choosing the first λ in a regularization path.

A4. We will first try out your solver with some synthetic data. A benefit of the Lasso is that if we believe many features are irrelevant for predicting y, the Lasso can be used to enforce a sparse solution, effectively differentiating between the relevant and irrelevant features. Suppose that $x \in \mathbb{R}^d, y \in \mathbb{R}, k < d$, and pairs of data (x_i, y_i) for $i = 1, \dots, n$ are generated independently according to the model $y_i = w^T x_i + \epsilon_i$ where

$$w_j = \begin{cases} j/k & \text{if } j \in \{1, \dots, k\} \\ 0 & \text{otherwise} \end{cases}$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is some Gaussian noise (in the model above $b = 0$). Note that since $k < d$, the features $k + 1$ through d are unnecessary (and potentially even harmful) for predicting y.
With this model in mind, let $n = 500, d = 1000, k = 100$, and $\sigma = 1$. Generate some data by choosing $x_i \in \mathbb{R}^d$, where each component is drawn from a $\mathcal{N}(0, 1)$ distribution and y_i generated as specified above.

a. *[10 points]* With your synthetic data, solve multiple Lasso problems on a regularization path, starting at λ_{max} where 0 features are selected and decreasing λ by a constant ratio (e.g., 1.5) until nearly all the features are chosen. In plot 1, plot the number of non-zeros as a function of λ on the x-axis (Tip: use `plt.xscale('log')`).

b. *[10 points]* For each value of λ tried, record values for false discovery rate (FDR) (number of incorrect nonzeros in $\widehat{w}$/total number of nonzeros in $\widehat{w}$) and true positive rate (TPR) (number of correct nonzeros in $\widehat{w}$/k). In plot 2, plot these values with the x-axis as FDR, and the y-axis as TPR and note that in an ideal situation we would have an (FDR,TPR) pair in the upper left corner, but that can always trivially achieve $(0, 0)$ and $(\frac{d-k}{d}, 1)$.

c. *[5 points]* Comment on the effect of λ in these two plots.

A5. Now we put the Lasso to work on some real data. Download the training data set "crime-train.txt" and the test data set "crime-test.txt" from the website under Homework 2. Store your data in your working directory and read in the files with:

```
import pandas as pd
df_train = pd.read_table("crime-train.txt")
df_test = pd.read_table("crime-test.txt")
```

This stores the data as Pandas DataFrame objects. DataFrames are similar to Numpy arrays but more flexible; unlike Numpy arrays, they store row and column indices along with the values of the data. Each column of a DataFrame can also, in principle, store data of a different type. For this assignment, however, all data are floats. Here are a few commands that will get you working with Pandas for this assignment:

```
df.head()                    # Print the first few lines of DataFrame df.
df.index                     # Get the row indices for df.
df.columns                   # Get the column indices.
df[``foo'']                  # Return the column named ``foo''.
df.drop(``foo'', axis = 1)   # Return all columns except ``foo''.
df.values                    # Return the values as a Numpy array.
df[``foo''].values           # Grab column foo and convert to Numpy array.
df.iloc[:3,:3]               # Use numerical indices (like Numpy) to get 3 rows and cols.
```

The data consist of local crime statistics for 1,994 US communities. The response y is the crime rate. The name of the response variable is `ViolentCrimesPerPop`, and it is held in the first column of `df_train` and `df_test`. There are 95 features. These features include possibly relevant variables such as the size of the police force or the percentage of children that graduate high school. The data have been split for you into a training and test set with 1,595 and 399 entries, respectively[1].

We'd like to use this training set to fit a model which can predict the crime rate in new communities and evaluate model performance on the test set. As there are a considerable number of input variables, overfitting is a serious issue. In order to avoid this, use the coordinate descent LASSO algorithm you just implemented in the previous problem.

Begin by running the LASSO solver with $\lambda = \lambda_{\max}$ defined above. For the initial weights, just use 0. Then, cut λ down by a factor of 2 and run again, but this time pass in the values of $\hat{w}$ from your $\lambda = \lambda_{\max}$ solution as your initial weights. This is faster than initializing with 0 weights each time. Continue the process of cutting λ by a factor of 2 until the smallest value of λ is less than 0.01. For all plots use a log-scale for the λ dimension (Tip: use `plt.xscale('log')`).

a. *[4 points]* Plot the number of nonzeros of each solution versus λ.

b. *[4 points]* Plot the regularization paths (in one plot) for the coefficients for input variables agePct12t29, pctWSocSec, pctUrban, agePct65up, and householdsize.

c. *[4 points]* Plot the squared error on the training and test data versus λ.

d. *[4 points]* Sometimes a larger value of λ performs nearly as well as a smaller value, but a larger value will select fewer variables and perhaps be more interpretable. Inspect the weights (on features) for $\lambda = 30$. Which feature variable had the largest (most positive) Lasso coefficient? What about the most negative? Discuss briefly. A description of the variables in the data set can be found here: `http://archive.ics.uci.edu/ml/machine-learning-databases/communities/communities.names`.

e. *[4 points]* Suppose there was a large negative weight on agePct65up and upon seeing this result, a politician suggests policies that encourage people over the age of 65 to move to high crime areas in an effort to reduce crime. What is the (statistical) flaw in this line of reasoning? (Hint: fire trucks are often seen around burning buildings, do fire trucks cause fire?)

Logistic Regression

Binary Logistic Regression *[30 points]*

A6. Let us again consider the MNIST dataset, but now just binary classification, specifically, recognizing if a digit is a 2 or 7. Here, let $Y = 1$ for all the 7's digits in the dataset, and use $Y = -1$ for 2. We will use

[1]The features have been standardized to have mean 0 and variance 1.

regularized logistic regression. Given a binary classification dataset $\{(x_i, y_i)\}_{i=1}^n$ for $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$ we showed in class that the regularized negative log likelihood objective function can be written as

$$J(w, b) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i(b + x_i^T w))) + \lambda ||w||_2^2$$

Note that the offset term b is not regularized. For all experiments, use $\lambda = 10^{-1}$. Let $\mu_i(w, b) = \frac{1}{1+\exp(-y_i(b+x_i^T w))}$.

a. *[8 points]* Derive the gradients $\nabla_w J(w, b)$, $\nabla_b J(w, b)$ and give your answers in terms of $\mu_i(w, b)$ (your answers should not contain exponentials).

b. *[8 points]* Implement gradient descent with an initial iterate of all zeros. Try several values of step sizes to find one that appears to make convergence on the training set as fast as possible. Run until you feel you are near to convergence.

   (i) For both the training set and the test, plot $J(w, b)$ as a function of the iteration number (and show both curves on the same plot).

   (ii) For both the training set and the test, classify the points according to the rule $\text{sign}(b + x_i^T w)$ and plot the misclassification error as a function of the iteration number (and show both curves on the same plot).

   Note that you are only optimizing on the training set. The $J(w, b)$ and misclassification error plots should be on separate plots.

c. *[7 points]* Repeat (b) using stochastic gradient descent with a batch size of 1. Note, the expected gradient with respect to the random selection should be equal to the gradient found in part (a). Take careful note of how to scale the regularizer.

d. *[7 points]* Repeat (b) using stochastic gradient descent with batch size of 100. That is, instead of approximating the gradient with a single example, use 100. Note, the expected gradient with respect to the random selection should be equal to the gradient found in part (a).

B4.

Multinomial Logistic Regression *[25 points]*

We've talked a lot about binary classification, but what if we have $k > 2$ classes, like the 10 digits of MNIST? Concretely, suppose you have a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{1, \ldots, k\}$. Like in our least squares classifier of homework 1 for MNIST, we will assign a separate weight vector $\mathbf{w}^{(\ell)}$ for each class $\ell = 1, \ldots, k$; let $W = [\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(k)}] \in \mathbb{R}^{d \times k}$. We can generalize the binary classification probabilistic model to multiple classes as follows: let

$$\mathbb{P}_W(y_i = \ell | W, \mathbf{x}_i) = \frac{\exp(\mathbf{w}^{(\ell)} \cdot \mathbf{x}_i)}{\sum_{j=1}^k \exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)}$$

The negative log-likelihood function is equal to

$$\mathcal{L}(W) = -\sum_{i=1}^{n} \sum_{\ell=1}^{k} \mathbf{1}\{y_i = \ell\} \log \left(\frac{\exp(\mathbf{w}^{(\ell)} \cdot \mathbf{x}_i)}{\sum_{j=1}^k \exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)} \right)$$

Define the `softmax`$(\cdot)$ operator to be the function that takes in a vector $\theta \in \mathbb{R}^d$ and outputs a vector in $\mathbb{R}^d$ whose ith component is equal to $\frac{\exp(\theta_i)}{\sum_{j=1}^d \exp(\theta_j)}$. Clearly, this vector is nonnegative and sums to one. If for any i we have $\theta_i \gg \max_{j \neq i} \theta_j$ then `softmax`(θ) approximates $\mathbf{e}_i$, a vector of all zeros with a one in the ith component.

For each y_i let $\mathbf{y}_i$ be the one-hot encoding of y_i (i.e., $\mathbf{y}_i \in \{0, 1\}^k$ is a vector of all zeros aside from a 1 in the y_ith index).

a. *[5 points]* If $\widehat{\mathbf{y}}_i^{(W)} = \texttt{softmax}(W^\top \mathbf{x}_i)$, show that $\nabla_W \mathcal{L}(W) = -\sum_{i=1}^n \mathbf{x}_i(\mathbf{y}_i - \widehat{\mathbf{y}}_i^{(W)})^\top$.

b. *[5 points]* Recall *Ridge Regression on MNIST* problem in Homework 1 and define $J(W) = \frac{1}{2}\sum_{i=1}^n \|\mathbf{y}_i - W^\top \mathbf{x}_i\|_2^2$. If $\widetilde{\mathbf{y}}_i^{(W)} = W^\top \mathbf{x}_i$ show that $\nabla_W J(W) = -\sum_{i=1}^n \mathbf{x}_i(\mathbf{y}_i - \widetilde{\mathbf{y}}_i^{(W)})^\top$. Comparing the least squares linear regression gradient step of this part to the gradient step of minimizing the negative log likelihood of the logistic model of part a may shed light on why we call this classification problem *logistic regression.*

c. *[15 points]* Using the original representations of the MNIST flattened images $\mathbf{x}_i \in \mathbb{R}^d$ ($d = 28 \times 28 = 764$) and all $k = 10$ classes, implement gradient descent (or stochastic gradient descent) for both $J(W)$ and $\mathcal{L}(W)$ and run until convergence on the training set of MNIST. For each of the two solutions, report the classification accuracy of each on the training and test sets using the most natural $\arg\max_j \mathbf{e}_j W^\top \mathbf{x}_i$ classification rule.

We highly encourage you to use `PyTorch` for this problem! The base object in `PyTorch` is the `torch.tensor`, which is essentially a `numpy.ndarray` but with some powerful new features. Namely, tensors have accelerator support (GPU, TPU) and high-performance autodifferentiation. Don't worry too much about the details of `PyTorch`! We will discuss `PyTorch` and the `torch.autograd` package in greater detail once we get to neural networks! At a high-level though, `torch.autograd` allows us to automatically calculate the gradients of our model parameters with minimal additional cost. Yep, that's right! Your days of writing out gradients by hand are numbered! :D

We include some starter pseudocode for the negative log-likelihood + softmax portion of this question. You are expected to find good hyperparameters. You can install the library at `https://pytorch.org/` and access the relevant beginner tutorial **here**.

```
import torch

W = torch.zeros(784, 10, requires_grad=True)
for epoch in range(epochs):
    y_hat = torch.matmul(X_train, W)
    # cross entropy combines softmax calculation with NLLLoss
    loss = torch.nn.functional.cross_entropy(y_hat, y_train)
    # computes derivatives of the loss with respect to W
    loss.backward()
    # gradient descent update
    W.data = W.data - step_size * W.grad
    # .backward() accumulates gradients into W.grad instead
    # of overwriting, so we need to zero out the weights
    W.grad.zero_()
```