Assignment 3, CSCI 471/571 Fall 2020

Your name here

Due: November 2, 11:59 p.m.

Group work following the guidelines in the Syllabus is okay.
Github username: user_name
List your collaborators.

Total points: 58
Problems 2 & 3 are adapted from Kevin Jamieson and Jamie Morgenstern's course at UW.

The following section should be computed by hand. Show your work.

Coordinate descent

1. *[5 points]* (Coordinate descent for least-squares) Coordinate descent is an optimization method that minimizes over one coordinate at a time: rather than try to update all coordinates in the vector $\vec{w}$ at once, it updates w_0, w_1, etc. sequentially and loops until convergence. In pseudocode, the basic coordinate descent algorithm is:

Data: cost function $C : \mathbb{R}^d \to \mathbb{R}$, initial guess $\vec{w} \in \mathbb{R}^d$
Result: $\vec{w}^*$ approximate solution to $\min_{\vec{w} \in \mathbb{R}^d} C(\vec{w})$
while *not converged* **do**
  for $i = 1, \ldots, d$ **do**
    Update $w_i = \arg\min_{w_i} C(\vec{w})$, where w_j for $j \neq i$ are fixed
  end
end

Algorithm 1: Coordinate descent

In general, Algorithm 1 is most useful when the minimization over a single coordinate can be solved very easily. We will show that this is the case for least squares. Take $C(\vec{w}) = \|X\vec{w} - \vec{y}\|^2$ to be the least-squares loss, with data $X \in \mathbb{R}^{n \times d}$ and $\vec{y} \in \mathbb{R}^n$ as usual. Derive a formula for the update step in the coordinate descent algorithm.

(Hints: This is just a one-variable optimization problem for w_i. Set $\frac{\partial C}{\partial w_i} = 0$ and solve for w_i, treating the other w_j for $j \neq i$ as constants. This equation should be of the form $w_i a_i - b_i = 0$, which has the solution $w_i = b_i / a_i$. Feel free use what you already know about the gradient of C from past homework and in-class notes.)

Lasso

For the following, you should program in Python and may use the `numpy` package and `matplotlib` for plotting, and `pandas` for loading data, but *you may not use any of the built-in least-squares solvers or anything from* `sklearn`. Any output from your program (displays of matrices or vectors and plots) must

be included in your pdf submission. Your code must be submitted as described in the Syllabus. All plots should be legible with axes labeled and legends if there are multiple things plotted.

Coordinate descent is very useful for solving for the Lasso solution

$$\vec{\beta}_{\text{lasso}} = \arg\min_{\vec{\beta}} \sum_{i=1}^{n} \left(\sum_{j=1}^{d} X_{ij}\beta_j + \beta_0 - y_i \right)^2 + \lambda \sum_{i=1}^{d} |\beta_i|.$$

Note that the penalty term does not affect the intercept β_0. It turns out that one can analytically derive an update formula for β_i that takes into account the penalty term. This leads to the following algorithm that you will be implementing:

Data: covariates X (without adding the column of 1s), labels $\vec{y}$, initial guess $\vec{\beta}$, penalty $\lambda \geq 0$
Result: $\vec{\beta}^*$ approximate Lasso solution
while *not converged* **do**
  $\beta_0 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \sum_{j=1}^{d} X_{ij}\beta_j)$
  for $i = 1, \ldots, d$ **do**
    $a_i = 2\sum_{j=1}^{n} X_{ji}^2$
    $b_i = 2\sum_{j=1}^{n} X_{ji}\left(y_j - \beta_0 - \sum_{k\neq i} X_{jk}\beta_k\right)$
    Update $\beta_i = T_\lambda(b_i)/a_i$.
  end
end

Algorithm 2: Coordinate descent for Lasso

The update is written in terms of the *soft-thresholding* function:

$$T_\lambda(z) = \begin{cases} z + \lambda & \text{if } z < -\lambda \\ 0 & \text{if } z \in [-\lambda, \lambda] \\ z - \lambda & \text{if } z > \lambda \end{cases}.$$

2. (Coordinate descent math for Lasso)

   1. *[2 points]* Draw a picture (by hand or computer) of the soft thresholding function $T_\lambda(z)$. Be sure to show the effect of λ.

   2. *[2 points]* When $\lambda = 0$, show that you recover the coordinate descent update equations for least-squares from problem 1.

Implement a Lasso solver via the coordinate descent method detailed in Algorithm 2. Start from the code included. You have two test conditions that should help you debug any errors:

- $\lambda = 0$: In this case, your algorithm should converge to the least-squares solution $\vec{\beta}_\lambda = \vec{\beta}_{\text{OLS}}$.
- $\lambda \geq \lambda_{\max}$: In this case, your algorithm should converge to the solution $\vec{\beta} = 0$. The value of this constant is

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

We will test your algorithm with these values of λ as well as an intermediate value of λ to determine whether is is working. Your grade, however, will be dependent on the applications below.
Some other useful hints for implementing this method:

- You may precompute a_i since this never changes throughout the iterations.

- Vectorize your code rather than relying on loops for summation, like when computing b_i. In python, loops are slow, whereas numpy is highly optimized.
- Print the cost after every iteration (end of inner for loop) and ensure that this is decreasing. It also should be decreasing with every update inside the for loop.
- If you have solved Lasso for $\lambda = 1$ and want to solve it for a nearby λ, say $\lambda = 2$, a good approach is to initialize with the solution you got for $\lambda = 1$.
- For a convergence criterion, use $\|\vec{\beta} - \vec{\beta}_{\text{old}}\|_\infty = \max_i |\beta_i - (\beta_{\text{old}})_i| \leq \delta$, where $\vec{\beta}_{\text{old}}$ was the previous iteration of the outer (while) loop. Don't neglect to store the older iterate before you start updating $\vec{\beta}$ so that you can evaluate the convergence criterion. If your algorithm is taking forever to converge, it could be that your δ is too small.
- When you apply your algorithm to real and fake data in the next two questions, set δ small enough so that you are getting to convergence.

3. (Testing Lasso on fake data) In class, we saw that the Lasso is good at finding sparse solutions, which is a great prior if we have reason to believer that only a few features are enough to predict y. Let $\vec{x} \in \mathbb{R}^d, y \in \mathbb{R}$, and $k < d$ be a sparsity parameter. We will generate the data $(\vec{x}_i, y_i)_{i=1}^n$ with the model $y_i = \vec{x}_i^T \vec{w} + \epsilon_i$, where

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

the noise $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ are iid Gaussian noise with mean 0 and variance σ^2. Note that we can generate a vector of noise with these characteristics by `epsilon = np.random.randn(n) * sigma`.

For this problem, set $n = 500, d = 1000, k = 100$, and $\sigma = 1$. Generate the data by setting $X_{ij} \sim \mathcal{N}(0, 1)$ and use the model for $\vec{y}$.

1. *[10 points]* Take your synthetic data and solve mutiple Lasso problems on a regularization path starting from $\lambda_{\max}$ and descreasing lambda to until nearly all features are selected (i.e. until the number of nonzeros in the solution $\|\vec{\beta}\|_0 = 0.99d$). Decrease each value of λ by the constant ratio 1.5, so that you end up with a exponential range $\lambda_{\max}, \frac{\lambda_{\max}}{1.5}, \frac{\lambda_{\max}}{(1.5)^2}$, etc. In plot 1, plot the number of nonzeros on the y-axis versus λ on the x-axis. Use logarithmic scaling on the x-axis: `plt.xscale('log')`.
2. *[10 points]* For each value of λ, record the false discovery rate (FDR), defined as the number of incorrect nonzeros in your estimate $\vec{\beta}$ divided by $\|\vec{\beta}\|_0$, along with the true positive rate (TPR), defined as the number of correct nonzeros in your estimate $\vec{\beta}$ divided by $\|\vec{\beta}\|_0$. In plot 2, plot TPR (y-axis) versus FDR (x-axis). Make both axes go from 0 to 1 and color your points by the value of λ. The best possible situation would be to have high TPR and low FDR, which is in the upper-left corner of the plot.
3. *[5 points]* Talk about the effect of λ in these two plots.

4. (Testing Lasso on real data) Now we will put this to work on some real data, stored in "crime-train.txt" and "crime-test.txt". You can read these files with:

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

This loads the data into two Pandas DataFrame objects. DataFrame objects are similar to data frames used in the statistical programming language R. You can think of them as objects similar to a CSV of data. The include an "index" for every row and labels for every column and allow the storage of columns of multiple types. In our dataset, however, all the types are floats, which is good since we're going to use this dataset for regression. 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]              # Get the first 3 rows and cols like Numpy.
```

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 orthe percentage of children that graduate high school. The data have been standardized and split into a training and testset with 1,595 and 399 entries, respectively.

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 $\vec{\beta}$ use 0. Then, cut shrink λ by a factor of 2 and run again, but this time pass in the values of $\vec{\beta}$ from your previous solution as your initial weights. This is faster than initializing with 0 weights each time. Continue the process of shrinking by a factor of 2 until $\lambda < 0.01$. For all plots, use log-scaling for the x-axis which represents λ.

1. *[4 points]* Plot the number of nonzeros versus λ (same as 3.1).
2. *[4 points]* Plot the regularization paths (values of the coefficients in β as a function of λ, like in ISLR Figure 6.6) for just the coefficients corresponding to 'gePct12t29', 'pctWSocSec', 'pctUrban', 'agePct65up', and 'householdsize'.
3. *[4 points]* Plot the mean squared error on the training and test data versus λ.
4. *[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 $\lambda = 30$.W hich 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`.
5. *[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?
6. *[4 points]* "Predictive policing" is when police departments use various datasets to try and predict where crimes will occur so that they can focus their resources in those areas. Discuss (1–2 paragraphs) whether or not you agree with the approach, and possible pitfalls you might see.