CSCI 4360/6360 Data Science II
Department of Computer Science
University of Georgia

Homework 3: I Heard You Like Theory

DUE: Thursday, October 21 by 11:59:59pm

Out October 5, 2021

Questions

This homework assignment focuses on machine learning theory, PAC learning, and graph theoretic methods.

1 Regularization in Linear Regression [30pts]

Recall our high-dimensional linear regression equation from the previous homework assignment, where we needed to find the β that minimized the squared-error loss function:

$$J(\beta) = \sum_{i=1}^{n} (y_i - \vec{x}_i^T \beta)^2,$$

or more simply in matrix form:

$$J(\beta) = (X\beta - Y)^T (X\beta - Y), \tag{1}$$

When the number of features m is much larger than the number of training examples n, or very few of the features are non-zero (as we saw in Homework 1), the matrix $X^T X$ is not full rank, and therefore cannot be inverted. This wasn't a problem for logistic regression which didn't have a closed-form solution anyway; for "vanilla" linear regression, however, this is a show-stopper.

Instead of minimizing our original loss function $J(\beta)$, we minimize a new loss function $J_R(\beta)$ (where the R is for "regularized" linear regression):

$$J_R(\beta) = \sum_{i=1}^{n}(Y_i - X_i^T\beta)^2 + \lambda\sum_{j=1}^{m}\beta_j^2,$$

which can be rewritten as:

$$J_R(\beta) = (X\beta - Y)^T(X\beta - Y) + \lambda||\beta||^2 \tag{2}$$

[5pts] Explain what happens as $\lambda \to 0$ and $\lambda \to \infty$ in terms of J, J_R, and β.

[15pts] Rather than viewing β as a fixed but unknown parameter (i.e. something we need to solve for), we can consider β as a random variable. In this setting, we can specify a prior distribution $P(\beta)$ on β that expresses our prior beliefs about the types of values β should take. Then, we can estimate β as:

$$\beta_{\text{MAP}} = \text{argmax}_\beta \prod_{i=1}^{n} P(Y_i|X_i;\beta)P(\beta), \tag{3}$$

where MAP is the *maximum a posteriori* estimate.

(aside: this is different from the MLE, which is the frequentist strategy for solving for a parameter. think of MAP as the Bayesian version.)

Show that maximizing Equation 3 can be expressed as *minimizing* Equation 2 with the assumption of a Gaussian prior on β, i.e. $P(\beta) \sim \mathcal{N}(0, I\sigma^2/\lambda)$. In other words, show that the L_2-norm regularization term in Equation 2 is effectively imposing a Gaussian prior assumption on the parameter β.

Hint #1: Start by writing out Equation 3 and filling in the probability terms.

Hint #2: Logarithms nuke pesky terms with exponents without changing linear relationships.

Hint #3: Multiplying an equation by -1 will switch from "argmin" to "argmax" and vice versa.

[10pts] What is the probabilistic interpretation of $\lambda \to 0$ under this model? What about $\lambda \to \infty$? Take note: this is asking a related but *different* question than the first part of this problem!

Hint: Consider how the prior $P(\beta)$ is affected by changing λ.

2 Spectral Clustering **[40pts]**

The general idea behind spectral clustering is to construct a mapping of data points to an eigenspace of a graph-induced affinity matrix A, with the hope that the points are well-separated in the eigenspace to the point where something simple like k-means will work well on the embedded data.

A very simple affinity matrix can be constructed as follows:

$$A_{i,j} = A_{j,i} = \begin{cases} 1 & \text{if } d(\vec{x}_i, \vec{x}_j) \le \Theta \\ 0 & \text{otherwise} \end{cases}$$

where $d(\vec{x}_i, \vec{x}_j)$ denotes Euclidean distance between points $\vec{x}_i$ and $\vec{x}_j$.

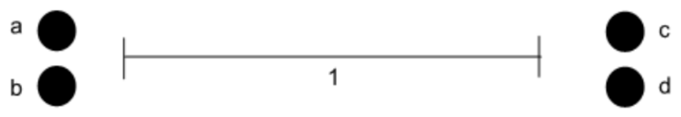

Figure .1: Simple toy dataset.

As an example, consider forming an affinity matrix for the dataset in Figure 1 using the affinity equation above, using $\Theta = 1$. Then we get the affinity matrix in Figure 2.

$$A = \left[\begin{array}{c|cccc} & a & b & c & d \\ \hline a & 1 & 1 & 0 & 0 \\ b & 1 & 1 & 0 & 0 \\ c & 0 & 0 & 1 & 1 \\ d & 0 & 0 & 1 & 1 \end{array}\right] \qquad \tilde{A} = \left[\begin{array}{c|cccc} & a & c & b & d \\ \hline a & 1 & 0 & 1 & 0 \\ c & 0 & 1 & 0 & 1 \\ b & 1 & 0 & 1 & 0 \\ d & 0 & 1 & 0 & 1 \end{array}\right]$$

(a) (b)

Figure .2: Affinity matrices of Fig. 1 with $\Theta = 1$.

For this particular example, the clusters $\{a, b\}$ and $\{c, d\}$ show up as nonzero blocks in the affinity matrix. This is, of course, artificial since we could have constructed the matrix A using any ordering of $\{a, b, c, d\}$. For example, another possible affinity matrix A could have been as in Figure 2(b).

The key insight here is that the eigenvectors of both A and $\tilde{A}$ have the same entries, just permuted. The eigenvectors with nonzero eigenvalues of A are $\vec{e}_1 = [0.7, 0.7, 0, 0]^T$ and $\vec{e}_2 = [0, 0, 0.7, 0.7]^T$. Likewise, the nonzero eigenvectors of $\tilde{A}$ are $\vec{e}_1 = [0.7, 0, 0.7, 0]^T$ and $\vec{e}_2 = [0, 0.7, 0, 0.7]^T$.

Spectral clustering embeds the original data points in a new space by using the coordinates of these eigenvectors. Specifically, it maps the point $\vec{x}_i$ to the point $[e_1(i), e_2(i), ..., e_k(i)]$, where $\vec{e}_1, ..., \vec{e}_k$ are the top k eigenvectors of A. We refer to this mapping as the *spectral embedding*. See Figure 3 for an example.

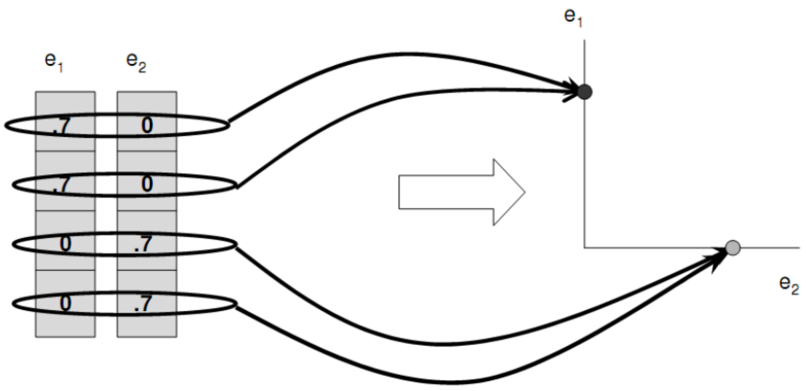

Figure .3: Using the eigenvectors of A to embed the data points. Notice that the points $\{a, b, c, d\}$ are tightly clustered in this space.

In this problem, we'll analyze how spectral clustering works on the simple dataset shown in the next figure.

[5pts] For the dataset in Figure 4, assume that the first cluster has m_1 points in it, and the second one has m_2 points. If we use the affinity equation from before to compute the affinity matrix A, what Θ value would you choose and why?

[10pts] The second step is to compute the first k dominant eigenvectors of the affinity matrix, where k is the number of clusters we want to have. For the dataset in the above figure, and the affinity matrix defined by the previous equation, is there a value of Θ for which you can analytically compute the first two eigenvalues and eigenvectors? If not, explain why not. If yes, compute and record these eigenvalues and eigenvectors. What are the other $((m_1 + m_2) - k)$ eigenvalues? Explain briefly.

Spectral clustering algorithms often make use a graph Laplacian matrix, L. A favorite variant is the *normalized* graph Laplacian, $L = D^{-1/2}AD^{-1/2}$, as this formulation has many convenient properties (D is a diagonal matrix whose i^{th} diagonal element, d_{ii}, is the sum of the i^{th} row of A).

[15pts] Show that a vector $\vec{v} = \left[\sqrt{d_{11}}, \sqrt{d_{22}}, ..., \sqrt{d_{nn}}\right]^T$ is an eigenvector of L with corresponding eigenvalue $\lambda = 1$.

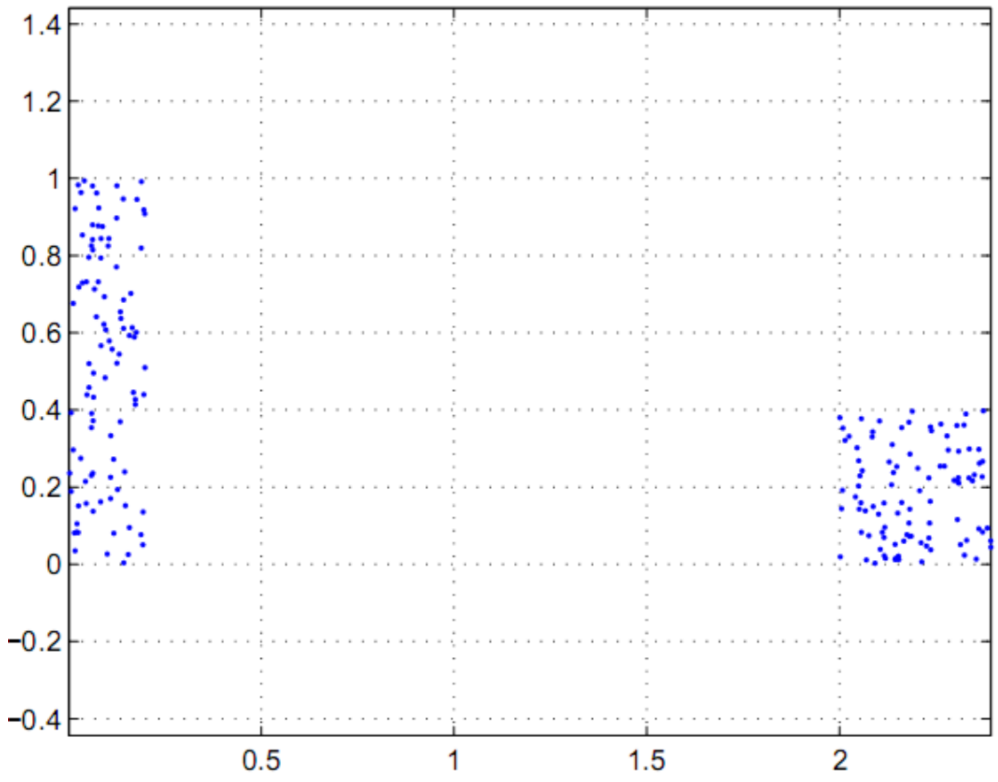

Figure .4: Dataset with rectangles.

One of the convenient properties of normalized graph Laplacians is the eigenvalue λ_1 of the leading eigenvector is, at most, 1; all other eigenvalues $\lambda_2, ..., \lambda_n$ have values strictly smaller than 1.

Consider a matrix P, where $P = D^{-1}A$, where A is our affinity matrix and D is the diagonal matrix. Each $p_{ij} = a_{ij}/d_{ii}$. Note the intuition of this operation: we are normalizing each edge by the total degree of the incoming vertex, essentially creating a "transition probability" p_{ij} of transitioning from vertex i to vertex j. In other words, each row of P sums to 1, so it is therefore a valid probability transition matrix. Hence, P^t is a matrix whose $\{i, j\}^{th}$ element shows the probability of being at vertex j after t number of steps, if one started at vertex i.

[10pts] Show that $P^\infty = D^{-1/2}\vec{v}_1\vec{v}_1^T D^{1/2}$. This property shows that if points are viewed as vertices according to a transition probability matrix, then $\vec{v}_1$ is the only eigenvector needed to compute the probability distribution over P^∞.

3 Genetic Programming [30pts]

[30pts] In this part, you'll re-implement your logistic regression code from the Homework 1 to use a simple genetic algorithm to learn the weights, instead of gradient descent.

Your script `homework3.py` should accept the following required arguments:

1. a file containing training data (same as Homework 1)
2. a file containing training labels (same as Homework 1)
3. a file containing testing data (same as Homework 1)

It should also be able to accept the following *optional* arguments:

- `-n`: a population size (default: 200)
- `-s`: a per-generation survival rate (default: 0.3)
- `-m`: a mutation rate (default: 0.05)
- `-g`: a maximum number of generations (default: 50)
- `-r`: a random seed (default: -1)

The handout on Autolab contains a skeleton script with the command-line parsing ready to go. It also contains subroutines that ingest and parse out the data files into NumPy arrays. You'll use the same dataset as before: the training set for your evolutionary algorithm to learn good weights, and the testing set to evaluate the weights.

Your evolutionary algorithm for learning the weights should have a few core components:

Random population initialization. You should initialize a full array of weights *randomly* (don't use all 0s!); this counts as a single "person" in the full population. Consequently, initialize n arrays of weights randomly for your full population. You'll evaluate each of these weights arrays independently and pick the best-performing ones to carry on to the next generation.

Fitness function. This is a way of evaluating how "good" your current solution is. Fortunately, we have this already: the objective function! You can use the weights to predict the training labels (as you did during gradient descent); the fitness for a set of weights is then the *average classification accuracy.*

Reproduction. Once you've evaluated the fitness of your current population, you'll use that information to evolve the "strongest." You'll first take the top s%–the ns arrays of weights with the highest fitness scores–and set them aside as the "parents" of the next generation. Then, you'll "breed" random pairs of these parents parents to produce "children" until you have n arrays of weights again. The breeding is done by simply averaging the two sets of parent weights together.

Mutation. Each individual weight has a mutation rate of m. Once you've computed the "child" weight array from two parents, you need to determine where and how many of the elements in the child array will mutate. First, flip a coin that lands on heads (i.e., indicates mutation) with probability m (the mutation rate) for each weight w_i. Then, for each mutation, you'll generate the new w_i by sampling from a Gaussian distribution with mean and variance set to be the empirical mean and variance of *all* the w_i weights of the *previous* generation. So if W_p is the $n \times |\beta|$ matrix of the previous population of weights, then we can define $\mu_i = W_p\left[:, i\right].\text{mean}()$ and $\sigma_i^2 = W_p\left[:, i\right].\text{var}()$. Using these quantities, we can then draw our new weight $w_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$.

Generations. You'll run the fitness evaluation, reproduction, and mutation repeatedly for g generations, after which you'll take the set of weights from the final population with the highest fitness and evaluate these weights against the testing dataset.

The parent and child populations should be kept *distinct* during reproduction, and only the children should undergo mutation!

Your script should be able to be invoked as follows:

```
> python homework3.py train.data train.label test.data
```

with the optional parameters then able to be stated at the end. The data files (`train.data` and `test.data`) contain three numbers on each line:

```
<document_id> <word_id> <count>
```

Each row of the data files contains the count of how often a given word (identified by ID) appears in certain documents (also identified by ID). The corresponding labels for the data has only one number per row in the file: the label, 1 or 0, of the document with ID corresponding to the row of the label in the label file. For example, a 0 on the 27th line of the label file means the document with ID 27 has the label 0.

After you've found your final weights and used them to make predictions on the test set, your code should print a predicted label (0 or 1) by itself on a single line, *one for each document*–this means a single line of output per unique document ID (or per line in one of the `.label` files). The output will be used to autograde your GA on AutoLab. For example, if the following `test.data` file has four unique document IDs in it, your program should print out four lines, each with a 1 or 0 on it, e.g.:

```
> python homework3.py train.data train.label test.data
```

```
0
0
1
1
```

Evolutionary programs **will take longer** than logistic regression's gradient descent. I strongly recommend staying under a population size of 300, with no more than about 300 generations. **Make liberal use of NumPy vectorized programming** to ensure your program is running as efficiently as possible. The Autolab autograder timeout will be extended to about 10 minutes, but you should be able to get reasonable training performance without having to go even half that long.

4 BONUS 1: Law of Large Numbers [15pts]

As Dr. Ceren explained in his guest lecture, the Hoeffding inequality has the form:

$$P[|\mu_{emp} - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N} \tag{4}$$

and provides a probabilistic bound on how often the empirical mean μ_{emp} of some sample of size N deviates form the true mean μ of some random process by more than a certain tolerance ϵ.

The importance of this inequality comes from its uniform (non-asymptotic) bound (i.e., N does not need to approach ∞ for it to be true).

[10pts] Design and implement an experiment showing this inequality holds in practice. Show that for means of samples from a random variable $X \sim Bernoulli(p = 0.6)$ abide by the Hoeffding inequality over a large range of sizes ($N = 1$ to 1000), with a fixed tolerance of $\epsilon = 0.1$. You can provide pseudocode in your write-up as long as it is *provably* correct. One way to do that would be to provide full code (e.g., Python) embedded in your write-up.

[5pts] Plot your trials against the bound provided by the inequality. You will need to run the same trial for each N multiple times (e.g., 10) and take the fraction of times of deviation as your $P[|\mu_{emp} - \mu| > \epsilon]$. Include this plot in your write-up. Make sure you are plotting both the fraction of deviations AND the bound for each N, as a function of the current iteration.

5 BONUS 2: Generalization Bounds [20pts]

In Probably Approximately Correct (PAC) Learning, concentration bounds like Eq(4) play a pivotal role for providing guarantees of generalising to unseen data-points. The inequality provides bounds on how a certain candidate solution $\mathbf{w}$ will behave over the entire domain of data-points compared to how many you tested on (N).

$$P[|\mathcal{R}_{emp}(\mathbf{w}) - \mathcal{R}(\mathbf{w})| > \epsilon] \leq 2e^{-2\epsilon^2 N} \tag{5}$$

$$\mathcal{R}_{emp}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} loss(f_{\mathbf{w}}(x), y_i)$$

$$\mathcal{R}(\mathbf{w}) = \int loss(f_{\mathbf{w}}(x), y) dP(x, y)$$

where $\mathcal{R}_{emp}(\mathbf{w})$ represents the error on the finite set of points N available to you, and $\mathcal{R}(\mathbf{w})$ is the error on all possible data-points in the domain (unavailable to you). $f_{\mathbf{w}}$ is your classifier/regression model parametrized by $\mathbf{w}$.

[15pts] Inequality 5 provides bounds for a specific $\mathbf{w}$. However, in data science techniques, we will be searching for a candidate solution over a whole family of solutions. If the family of solutions has 3 possible candidate solutions $\mathbf{w}_1$,$\mathbf{w}_2$,$\mathbf{w}_3$, what will be a bound over ANY of $\mathcal{R}_{emp}(\mathbf{w}_1)$, $\mathcal{R}_{emp}(\mathbf{w}_2)$, $\mathcal{R}_{emp}(\mathbf{w}_3)$ deviating from $\mathcal{R}(\mathbf{w}_1)$, $\mathcal{R}(\mathbf{w}_2)$, $\mathcal{R}(\mathbf{w}_3)$ respectively?

In other words, provide a tight bound for:

$$P[|\mathcal{R}_{emp}(\mathbf{w}_1) - \mathcal{R}(\mathbf{w}_1)| > \epsilon \cup |\mathcal{R}_{emp}(\mathbf{w}_2) - \mathcal{R}(\mathbf{w}_2)| > \epsilon \cup |\mathcal{R}_{emp}(\mathbf{w}_3) - \mathcal{R}(\mathbf{w}_3)| > \epsilon]$$

[5pts] What does that mean for a family that has infinite candidate solutions (e.g., all possible weights and biases for a logistic regression classifier)?

Administration

1 Submitting

All submissions will go to **AutoLab**. You can access AutoLab at:

- https://autolab.cs.uga.edu

You can submit deliverables to the **Homework 3** assessment that is open. When you do, you'll submit two files:

1. `homework3.py`: the Python script that implements your algorithms, and
2. `homework3.pdf`: the PDF write-up with any questions that were asked

These should be packaged together in a tarball; the archive can be named whatever you want when you upload it to AutoLab, but the files in the archive should be named **exactly** what is above. Deviating from this convention could result in the autograder failing!

To create the tarball archive to submit, run the following command (on a *nix machine):

```
> tar cvf homework3.tar homework3.py homework3.pdf
```

This will create a new file, `homework3.tar`, which is basically a zip file containing your Python script and PDF write-up. Upload the archive to AutoLab. There's no penalty for submitting as many times as you need to, but keep in mind that swamping the server at the last minute may result in your submission being missed; AutoLab is programmed to close submissions *promptly* at 11:59pm on October 21, so give yourself plenty of time! A late submission because the server got hammered at the deadline will *not* be acceptable (there is a *small* grace period to account for unusually high load at deadline, but I strongly recommend you avoid the problem altogether and start early).

Also, to save time while you're working on the coding portion, you are welcome to create a tarball archive of just the Python script and upload that to AutoLab. Once you get the autograder score you're looking for, you can then include the PDF in the folder, tarball everything, and upload it. AutoLab stores the entire submission history of every student on every assignment, so your autograder (code) score will be maintained and I can just use your most recent submission to get the PDF.

2 Reminders

- If you run into problems, ping the `#questions` room of the Discord server. If you still run into problems, ask me. But please please please, **do NOT** ask Google to give you the code you seek! I will be on the lookout for this (and already know some of the most popular venues that might have solutions or partial solutions to the questions here).
- Prefabricated solutions (e.g. `scikit-learn`, OpenCV) are NOT allowed! You have to do the coding yourself! But you **can** use the pairwise metrics in scikit-learn, as well as the vector norm in SciPy.
- If you collaborate with anyone, just mention their names in a code comment and/or at the top of your homework writeup.
- Cite any external and/or non-course materials you referenced in working on this assignment.