Homework 2

10-605/10-805: Machine Learning with Large Datasets

Due Wednesday, September 27th at 11:59 PM Eastern Time

Submit your solutions via Gradescope, **with your solution to each subproblem on a separate page**, i.e., following the template below.

IMPORTANT: Be sure to highlight where your solutions are for each question when submitting to Gradescope otherwise you will be marked 0 and will need to submit regrade request for every solution un-highlighted in order for fix it!

Note that Homework 2 consists of two parts: this written assignment and a programming assignment. Remember to fill out the collaboration section found at the end of this homework as per the course policy.

All students are required to complete the following:

1. Written Section 1.1 (a) and (b) *[6 points]*
2. Written Section 1.2 (a), (b), (c), (d), and (e) *[24 points]*
3. Programming Section *[70 points]*

All students are required to complete **all** sections of the homework. Your homework score will be calculated as a percentage over the maximum points you can earn, which is 100.

1 Written Section [30 Points]

1.1 Ridge Regression [6 Points]

Suppose we want to use *ridge regression* to fit a model to our data using the following optimization problem:

$$\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_2^2 \ ,$$

where $\mathbf{X} \in \mathbb{R}^{n\times k}$ represents the data matrix, $\mathbf{y} \in \mathbb{R}^n$ stores the labels, and $\mathbf{w} \in \mathbb{R}^k$ is the model. In this question we will compare the cost of computing the closed-form solution of this objective vs. computing the update rule for gradient descent. Assume in the question that all work is being executed on a single machine (i.e., not in a parallel/distributed setting).

(a) *[3 points]* What is the closed-form solution for $\mathbf{w}$? In big-O notation, what is the *computational cost* for computing this closed-form solution? The closed form solution is:

$$w = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_k)^{-1}\mathbf{X}^T y$$

The time complexity is:

$$O(nk^2 + k^3)$$

The nk^2 comes from calculating the matrix multiplication $\mathbf{X}^T\mathbf{X}$. The k^3 term comes from matrix multiplication between the $k \times k$ matrix $(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_k)^{-1}$ and the $k \times n$ matrix $\mathbf{X}^T$.

(b) *[3 points]* What is the gradient descent update for the objective at iteration $i+1$? In big-O notation, what is the *computational cost* for performing this one iteration/update of gradient descent? Assume that the step size is α.

The update rule is:

$$w_{i+1} = w_i - \alpha_i \nabla f(w_i)$$
$$w_{i+1} = w_i - \alpha_i \sum_{j=1}^{n}((w_i^T x^{(j)} - y^{(j)})x^{(j)}) - 2\lambda w$$

where α_i is the step size at time i. The computational complexity is $O(nk)$ which comes from calculating $w_i^T x^{(j)}$, n times. Calculating $w_i^T x^{(j)}$ takes k multiplication steps and this has to be done n times over the summation, thus big-O computational cost $O(nk)$.

1.2 Nyström Method [24 Points]

Suppose that we have a kernel matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$, where $\mathbf{X} \in \mathbb{R}^{n \times m}$. Define the following block representation of the kernel matrix:

$$\mathbf{K} = \begin{bmatrix} \mathbf{W} & \mathbf{K}_{21}^\top \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix} \quad \text{and} \quad \mathbf{C} = \begin{bmatrix} \mathbf{W} \\ \mathbf{K}_{21} \end{bmatrix}.$$

Recall that the Nyström method uses $\mathbf{W} \in \mathbb{R}^{r \times r}$, $\mathbf{C} \in \mathbb{R}^{n \times r}$ to form the approximation $\widetilde{\mathbf{K}} = \mathbf{C}\mathbf{W}^{-1}\mathbf{C}^\top \approx \mathbf{K}$. In this question we will explore several aspects of the Nyström method to get a better understanding of the quality of the approximation that it makes and the computational benefits it provides. Throughout all the subquestions, we will assume that $\mathbf{W}$ is full rank (i.e., $\mathbf{W}^{-1}$ exists).

(a) *[5 points]* First, we will quantify how well $\widetilde{\mathbf{K}}$ approximates the matrix kernel matrix $\mathbf{K}$. Show that $\left\|\mathbf{K} - \widetilde{\mathbf{K}}\right\|_F = \left\|\mathbf{K}_{22} - \mathbf{K}_{21}\mathbf{W}^{-1}\mathbf{K}_{21}^\top\right\|_F$, where $\|.\|_F$ is the Frobenius norm.

$$\left\|\mathbf{K} - \widetilde{\mathbf{K}}\right\|_F = \left\| \begin{bmatrix} \mathbf{W} & \mathbf{K}_{21}^\top \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix} - \begin{bmatrix} \mathbf{W} \\ \mathbf{K}_{21} \end{bmatrix} \mathbf{W}^{-1} \begin{bmatrix} \mathbf{W}^T & \mathbf{K}_{21}^T \end{bmatrix} \right\|_F$$

$$\left\|\mathbf{K} - \widetilde{\mathbf{K}}\right\|_F = \left\| \begin{bmatrix} \mathbf{W} & \mathbf{K}_{21}^\top \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix} - \begin{bmatrix} \mathbf{I}_r \\ \mathbf{K}_{21}\mathbf{W}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{W}^T & \mathbf{K}_{21}^T \end{bmatrix} \right\|_F$$

$$\left\|\mathbf{K} - \widetilde{\mathbf{K}}\right\|_F = \left\| \begin{bmatrix} \mathbf{W} & \mathbf{K}_{21}^\top \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix} - \begin{bmatrix} \mathbf{W}^T & \mathbf{K}_{21}^T \\ \mathbf{K}_{21}\mathbf{W}^{-1}\mathbf{W}^T & \mathbf{K}_{21}\mathbf{W}^{-1}\mathbf{K}_{21}^T \end{bmatrix} \right\|_F$$

$$\left\|\mathbf{K} - \widetilde{\mathbf{K}}\right\|_F = \left\| \begin{bmatrix} \mathbf{W} & \mathbf{K}_{21}^\top \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix} - \begin{bmatrix} \mathbf{W}^T & \mathbf{K}_{21}^T \\ \mathbf{K}_{21}\mathbf{I}_r & \mathbf{K}_{21}\mathbf{W}^{-1}\mathbf{K}_{21}^T \end{bmatrix} \right\|_F$$

$$\left\|\mathbf{K} - \widetilde{\mathbf{K}}\right\|_F = \left\| \begin{bmatrix} \mathbf{W} & \mathbf{K}_{21}^\top \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix} - \begin{bmatrix} \mathbf{W}^T & \mathbf{K}_{21}^T \\ \mathbf{K}_{21} & \mathbf{K}_{21}\mathbf{W}^{-1}\mathbf{K}_{21}^T \end{bmatrix} \right\|_F$$

$$\left\|\mathbf{K} - \widetilde{\mathbf{K}}\right\|_F = \left\| \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{K}_{22} - \mathbf{K}_{21}\mathbf{W}^{-1}\mathbf{K}_{21}^T \end{bmatrix} \right\|_F$$

$$\left\|\mathbf{K} - \widetilde{\mathbf{K}}\right\|_F = \left\|\mathbf{K}_{22} - \mathbf{K}_{21}\mathbf{W}^{-1}\mathbf{K}_{21}^T\right\|_F$$

(b) *[2 points]* As we mentioned in lecture, there is an important connection between the compact SVD and low-rank matrix approximation. Let $\mathbf{X}' \in \mathbb{R}^{r \times m}$ be the first r rows of $\mathbf{X}$. From the definition of the Compact SVD, define $\mathbf{X}' = \mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{V}_{X'}^{\top}$, where $\mathbf{U}_{X'}$ is an $r \times t$ matrix with orthogonal columns, $\mathbf{\Sigma}_{X'}$ is a $t \times t$ diagonal matrix, $\mathbf{V}_{X'}$ is an $m \times t$ matrix with orthogonal columns. Assuming that the rank of $\mathbf{X}'$ is r, then what is the relationship between r and t?

$r = t$ because the number of non-zero singular values in SVD is equal to its rank of the matrix being decomposed. Since there are r singular values in $\mathbf{\Sigma}$, there must also be r non-zero right-singular vectors.

(c) *[8 points]* Let $\mathbf{X'} \in \mathbb{R}^{r \times m}$ be the first r rows of $\mathbf{X}$. Assuming that the rank of $\mathbf{X'}$ is r, show that $\widetilde{\mathbf{K}} = \mathbf{X}\mathbf{P}_{V_{X'}}\mathbf{X}^\top$, where $\mathbf{P}_{V_{X'}} = \mathbf{V}_{X'}\mathbf{V}_{X'}^\top$ is the orthogonal projection onto the span of the right singular vectors of $\mathbf{X'}$, i.e, the span of $\mathbf{V}_{X'}$. (Hint: Express $\mathbf{C}$ and $\mathbf{W}$ in terms of $\mathbf{X}$ and $\mathbf{X'}$)

$$\mathbf{K} = \begin{bmatrix} \mathbf{W} & \mathbf{K}_{21}^T \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix} = \mathbf{X}\mathbf{X}^T$$

$$\text{Let } \mathbf{X} = \begin{bmatrix} \mathbf{X}^{'} \\ \mathbf{X}^{''} \end{bmatrix}$$

$$\mathbf{X}\mathbf{X}^T = \begin{bmatrix} \mathbf{X}^{'} \\ \mathbf{X}^{''} \end{bmatrix} \begin{bmatrix} \mathbf{X}^{'T} & \mathbf{X}^{''T} \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{X}^{'}\mathbf{X}^{'T} & \mathbf{X}^{'}\mathbf{X}^{''T} \\ \mathbf{X}^{''}\mathbf{X}^{'T} & \mathbf{X}^{''}\mathbf{X}^{''T} \end{bmatrix}$$

$$\mathbf{W} = \mathbf{X}^{'}\mathbf{X}^{'T}$$

$$\mathbf{C} = \begin{bmatrix} \mathbf{X}^{'}\mathbf{X}^{'T} \\ \mathbf{X}^{''}\mathbf{X}^{'T} \end{bmatrix} = \begin{bmatrix} \mathbf{X}^{'} \\ \mathbf{X}^{''} \end{bmatrix} \mathbf{X}^{'T} = \mathbf{X}\mathbf{X}^{'T}$$

$$\widetilde{\mathbf{K}} = \mathbf{C}\mathbf{W}^{-1}\mathbf{C}^\top$$
$$\widetilde{\mathbf{K}} = \mathbf{X}\mathbf{X}^{'T}(\mathbf{X}^{'}\mathbf{X}^{'T})^{-1}(\mathbf{X}\mathbf{X}^{'T})^T$$
$$\widetilde{\mathbf{K}} = \mathbf{X}\mathbf{X}^{'T}(\mathbf{X}^{'}\mathbf{X}^{'T})^{-1}\mathbf{X}^{'}\mathbf{X}^T$$
$$\widetilde{\mathbf{K}} = \mathbf{X}(\mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{V}_{X'}^T)^T(\mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{V}_{X'}^T(\mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{V}_{X'}^T)^T)^{-1}\mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{V}_{X'}^T\mathbf{X}^T$$
$$\widetilde{\mathbf{K}} = \mathbf{X}(\mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{V}_{X'}^T)^T(\mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{V}_{X'}^T\mathbf{V}_{X'}\mathbf{\Sigma}_{X'}^T\mathbf{U}_{X'}^T)^{-1}\mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{V}_{X'}^T\mathbf{X}^T$$
$$\widetilde{\mathbf{K}} = \mathbf{X}(\mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{V}_{X'}^T)^T(\mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{\Sigma}_{X'}^T\mathbf{U}_{X'}^T)^{-1}\mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{V}_{X'}^T\mathbf{X}^T$$
$$\widetilde{\mathbf{K}} = \mathbf{X}\mathbf{V}_{X'}\mathbf{\Sigma}_{X'}^T\mathbf{U}_{X'}^T(\mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{\Sigma}_{X'}^T\mathbf{U}_{X'}^T)^{-1}\mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{V}_{X'}^T\mathbf{X}^T$$
$$\widetilde{\mathbf{K}} = \mathbf{X}\mathbf{V}_{X'}\mathbf{\Sigma}_{X'}^T\mathbf{U}_{X'}^T\mathbf{U}_{X'}^{T^{-1}}\mathbf{\Sigma}_{X'}^{T^{-1}}\mathbf{\Sigma}_{X'}^{-1}\mathbf{U}_{X'}^{-1}\mathbf{U}_{X'}\mathbf{\Sigma}_{X'}\mathbf{V}_{X'}^T\mathbf{X}^T$$
$$\widetilde{\mathbf{K}} = \mathbf{X}\mathbf{V}_{X'}\mathbf{\Sigma}_{X'}^T\mathbf{\Sigma}_{X'}^{T^{-1}}\mathbf{\Sigma}_{X'}^{-1}\mathbf{\Sigma}_{X'}\mathbf{V}_{X'}^T\mathbf{X}^T$$
$$\widetilde{\mathbf{K}} = \mathbf{X}\mathbf{V}_{X'}\mathbf{V}_{X'}^T\mathbf{X}^T$$
$$\widetilde{\mathbf{K}} = \mathbf{X}\mathbf{P}_{V_{X'}}\mathbf{X}^\top$$

(d) *[4 points]* Recall that the kernel matrix $\mathbf{K}$ is a symmetric positive semidefinite (SPSD) matrix. Is $\widetilde{\mathbf{K}}$ also SPSD? Please provide justification of your answer.

Yes.

$$\widetilde{\mathbf{K}} = \mathbf{X}\mathbf{V}_{X'}\mathbf{V}_{X'}^T\mathbf{X}^T$$

$$\text{Let } \mathbf{A} = \mathbf{V}_{X'}^T\mathbf{X}^T$$

$$\widetilde{\mathbf{K}} = A^T A$$

$\widetilde{\mathbf{K}}$ is symmetric because $\mathbf{A}^T\mathbf{A}$ is symmetric by definition.

Let z be some vector that $\neq \overrightarrow{0}$. Then $z^T\mathbf{A}^T\mathbf{A}z = (\mathbf{A}z)^T(\mathbf{A}z) = \|Az\| \geq 0$. Thus $\widetilde{\mathbf{K}}$ is symmetric and semi-definite.

(e) *[5 points]* If $n = 20$ million and $\mathbf{K}$ is a dense matrix, how much space (in terabytes) is required to store $\mathbf{K}$ if each entry is stored as a double (8 bytes)? How much space (in terabytes) is required by the Nyström method if $r = 10,000$? (For Nyström method, consider the cost needed to store the approximation in factored form, i.e., before computing $\widetilde{\mathbf{K}}$)

If $n = 20$ million and K is dense, then the space required to store k is
8 bytes $\times$ 20e6 $\times$ 20e6 $= 3,200$e12 bytes $= 3,200$ terabytes

To store $\widetilde{K}$ the space required is 8 bytes(20e6 $\times$ 1e4 $+$ 1e4 $\times$ 1e4) $=$ 8 bytes(20e10 $+$ 1e8) $=$
8 bytes(2.001e11) $=$ 16.008e11 $=$ 1.6008 terabytes.

2 Programming Section [70 Points]

2.1 Introduction

This assignment involves understanding the basics of building machine learning pipelines and techniques in training linear regression models. The assignment will also involve using principal component analysis (PCA) and feature-based aggregation for exploratory data analysis.

This assignment consists of two major parts. The first part is to train a linear regression model to predict the release year of a song given a set of audio features. The second part of the assignment is to apply PCA to a light-sheet imaging dataset and complete a feature-based aggregation to find a moving visual pattern of neural activity.

2.2 Logistics

We provide the code template for this assignment in *two* Jupyter notebooks. What you need to do is to follow the instructions in the notebooks and implement the missing parts marked with '`<FILL_IN>`' or '`# YOUR CODE HERE`'. Most of the '`<FILL_IN>/YOUR CODE HERE`' sections can be implemented in just one or two lines of code.

2.3 Getting lab files

You can obtain the notebooks `'hw2_part1.ipynb'` and `'hw2_part2.ipynb'` in the homework 2 handout .zip file.

Next, import the notebooks into your Databricks account, which provides you a well-configured Spark environment and will definitely save your time (see the next section for details).

2.4 Preparing for submission

We provide several public tests via `assert` in the notebook. You may want to pass all those tests before submitting your homework. You can individually submit a notebook for debugging but **make sure to submit both notebooks for your final submission to receive full credit.**

You will also have to

2.5 Submission

1. Export both solution notebooks as IPython notebook files on Databricks via `File -> Export -> IPython Notebook`
2. Submit both completed notebooks and deliverables via Gradescope (you can select both notebooks when uploading your solutions).

2.6 Setting up environments on Databricks

We provide step-by-step instructions on how to configure your Databricks platform. The recitations slides related to PySpark and Databricks setup can be found here.

1. Sign up for the **Community Edition** of Databricks here: https://databricks.com/try-databricks.
2. Import the notebook file we provide on your homepage: `Workspace -> Users -> Import`
3. Create a cluster: `Clusters -> Create Cluster`. You can use any cluster name as you like. When configuring your cluster, make sure to choose **runtime version** `13.2`. Note: It may take a while to launch the cluster, please wait for its status to turn to '`active`' before start running.

4. Installing third-party packages that will be used in the homework on Databricks: `Clusters -> Cluster name -> Libraries -> Install New`. Then select `PyPI`, enter the package name as `nose`. Finally click `Install` to install it.

5. You can start to play with the notebook now!

Note: Databricks Community Edition only allows you to launch one 'cluster'. If the current cluster is 'terminated', then you can either (1) delete it, and then create a new one, or (2) activate and attach to the existing cluster when running the notebook. Make sure to install nose.

2.7 Linear Regression on the Million Song Dataset

This section covers a common supervised learning pipeline, using a subset of the Million Song Dataset from the UCI Machine Learning Repository. Our goal is to train a linear regression model to predict the release year of a song given a set of audio features.

In this section, you will be implementing a common supervised learning pipeline, using a subset of the Million Song Dataset from the UCI Machine Learning Repository, to train a linear regression model to predict the release year of a song given a set of audio features.

In this part, we will cover

- Part 1: Reading and parsing the Million Song dataset
- Part 2: Creating and evaluating a baseline model
- Part 3: Training (via gradient descent) and evaluating a linear regression model
- Part 4: Training using SparkML and tune hyperparameters via grid search
- Part 5: Adding interactions between features

See the notebook for detailed descriptions and instructions of each question.

2.8 Principal Component Analysis (PCA)

This section delves into exploratory analysis of neuroscience data, specifically using principal component analysis (PCA) and feature-based aggregation. We will use a dataset of light-sheet imaging recorded by the Ahrens Lab at Janelia Research Campus.

Our dataset is generated by studying the movement of a larval zebrafish, an animal that is especially useful in neuroscience because it is transparent, making it possible to record activity over its entire brain using a technique called light-sheet microscopy. Specifically, we'll work with time-varying images containing patterns of the zebrafish's neural activity as it is presented with a moving visual pattern. Different stimuli induce different patterns across the brain, and we can use exploratory analyses to identify these patterns.

In this section you will learn about PCA, and then compare and contrast different exploratory analyses of the same data set to identify which neural patterns they best highlight. You will also demonstrate the Johnson–Lindenstrauss lemma by performing random projections of the zebrafish dataset.

In this part, we will cover:

- Part 1: Working through the steps of PCA on a sample dataset
- Part 2: Writing a PCA function and evaluating PCA on sample datasets
- Part 3: Parsing, inspecting, and preprocessing neuroscience data then perform PCA
- Part 4: Feature-based aggregation and PCA
- Part 5: Random-projection and Johnson–Lindenstrauss lemma

See the notebook for detailed descriptions and instructions of each question.

3 Collaboration Questions

1. (a) Did you receive any help whatsoever from anyone in solving this assignment?

   Yes

   (b) If you answered 'yes', give full details (e.g. "Jane Doe explained to me what is asked in Question 3.4")
   Ethan Gaskin helped me start on the 1.2c to represent c in terms of X and X'

2. (a) Did you give any help whatsoever to anyone in solving this assignment?

   No

   (b) If you answered 'yes', give full details (e.g. "I pointed Joe Smith to section 2.3 since he didn't know how to proceed with Question 2")

3. (a) Did you find or come across code that implements any part of this assignment?

   No

   (b) If you answered 'yes', give full details (book & page, URL & location within the page, etc.).