Homework 1

10-405/10-605: Machine Learning with Large Datasets

Due Wednesday, February 17th 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. Be sure to highlight where your solutions are for each question when submitting to Gradescope! Note that Homework 1 consists of two parts: this written assignment, and a programming assignment. Each part (written and programming) is worth 50% of your total HW1 grade. Remember to fill out the collaboration section found at the end of this homework as per the course policy.

1 Written Section [50 Points]

1.1 Sparse PCA

Sparse PCA is a variant of PCA that can be used to better interpret which of the k features are most important for the various principal components. In sparse PCA, the first principal component $\boldsymbol{p}$ is the solution to the following constrained optimization problem:

$$\begin{aligned} \max_{\boldsymbol{p}} \quad & \boldsymbol{p}^T \mathbf{C}_{\mathbf{X}} \boldsymbol{p} \\ \text{subject to} \quad & ||\boldsymbol{p}||_2 = 1 \text{ and } \quad ||\boldsymbol{p}||_0 \leq s, \end{aligned} \tag{1}$$

where $\mathbf{X}$ is an $n \times k$ matrix of n data points and $\mathbf{C}_{\mathbf{X}}$ is the covariance matrix of $\mathbf{X}$. The L0 norm $||\boldsymbol{p}||_0$ represents the number of non-zero elements in the vector $\boldsymbol{p}$, and we want this to be at most s, where s is a fixed constant such that $1 \leq s \leq k$.

(a) *[10 points]* The second constraint of the above optimization problem, $||\boldsymbol{p}||_0 \leq s$, is not convex. Show this by proving that the L0 norm $||\boldsymbol{p}||_0$ is not a convex function.

(b) *[10 points]* Being a non-convex problem, (1) is hard to solve directly. Instead, we can use the following brute-force optimization method. Fix a set $S \subset \{1, 2, \ldots, k\}$, such that $|S| = s$ corresponding to s locations of non-zero elements in $\boldsymbol{p}$. For each S we solve the following convex problem:

$$\begin{aligned} \max_{\boldsymbol{p}} \quad & \boldsymbol{p}^{\boldsymbol{T}} \boldsymbol{C}_{\boldsymbol{X}} \boldsymbol{p} \\ \text{subject to} \quad & ||\boldsymbol{p}||_2 = 1 \text{ and } \quad p_i = 0 \text{ for } i \notin S, \end{aligned} \tag{2}$$

where p_i is the i-th element of $\boldsymbol{p}$. To solve the original problem (1), we can solve (2) for all possible sets S and choose the solution that gives the minimum objective value.

How many different sets S will we have to iterate over in the above method? Why is this brute-force method computationally expensive for high-dimensional data?

(c) *[5 points]* Instead of the brute-force method suggested in (b) which yields an exact solution, we may want to consider approximate solutions that do not necessarily satisfy $||\boldsymbol{p}||_0 \leq s$, but impose some sparsity in $\boldsymbol{p}$. Do you have any suggestions for an approximate solution which can be more computationally tractable?

1.2 Laplacian Eigenmaps

Several non-linear methods for dimensionality reduction assume that high-dimensional data lie on or near a low-dimensional non-linear manifold embedded in the input space. These so-called manifold learning techniques aim to learn this manifold structure. Global methods, e.g,. Isomap, aim to preserve distances along the manifold between all pairs of data points. In contrast, local methods, like Laplacian Eigenmaps focus only on preserving local neighborhood relationships in the high-dimensional space. In particular, given a dataset of n data points, Laplacian Eigenmaps works by first finding t nearest neighbors for each point; then constructing $\mathbf{W}$, the sparse, symmetric $n \times n$ nearest neighborhood adjacency matrix, where $\mathbf{W}_{ij} = \exp\big(-||\mathbf{x}_i - \mathbf{x}_j||_2^2/\sigma^2\big)$ if $(\mathbf{x}_i, \mathbf{x}_j)$ are neighbors[1], 0 otherwise, and σ is a scaling parameter; and finally, finding the r-dimensional representation that minimizes the following objective:

$$\mathbf{Y} = \text{argmin}_{\mathbf{Y}'} \sum_{i,j} \mathbf{W}_{ij} ||\mathbf{y}'_i - \mathbf{y}'_j||_2^2 \,, \tag{3}$$

subject to appropriate constraints on $\mathbf{Y}'$. Note that $\mathbf{Y}' \in \mathbb{R}^{r \times n}$ and $\mathbf{y}'_i \in \mathbb{R}^r$ is the ith column of $\mathbf{Y}'$.

(a) *[5 points]* Describe in words how the objective in Eq (3) aims to preserve local neighborhood relationships. What is the role of the weight matrix $\mathbf{W}$?

[1] To ensure that $\mathbf{W}$ is symmetric, we define $(\mathbf{x}_i, \mathbf{x}_j)$ as neighbors if either data point is a t nearest neighbor of the other.

(b) *[10 points]* Assume we seek a one-dimensional representation $\mathbf{y}$, i.e., $r = 1$. Show that (3) is equivalent to

$$\mathbf{y} = \text{argmin}_{\mathbf{y}'} \, \mathbf{y}'^{\top} \mathbf{L} \mathbf{y}', \tag{4}$$

where $\mathbf{D}$ is the diagonal matrix with $\mathbf{D}_{ii} = \sum_j \mathbf{W}_{ij}$, and $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is the graph Laplacian.

(c) *[10 points]* From (3) and (4), we can see that the graph Laplacian is positive semidefinite (take a minute to figure out why this is). The solution to the problem is anything proportional to the ones vector. Explain why this is the solution and why it might be bad. What constraint can you put on $\mathbf{y}'$ to stop the ones vector from being the solution?

2 Programming [50 points]

This assignment involves understanding some basics of distributed computing, the MapReduce programming model, Spark, and an example of data cleaning.

This assignment consists of two major parts. The first part is to build a simple word count application, and the second part is on entity resolution, a common type of data cleaning.

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. Also, keep in mind to delete the '`raise NotImplementedError()` once you are done implementing a function.

NOTE 1: Please **DO NOT** add or delete any cells in the Jupyter notebooks we provide. The autograder may not be able to parse you solution correctly if you add or delete new cells. Also, please **DO NOT** change cells with only assertions in them.

NOTE 2: Please make sure your files to be graded are named as '`hw1_coding_1.ipynb`' and '`hw1_coding_2.ipynb`' before submitting to gradescope.

2.1 Getting lab files

You can obtain the notebooks '`hw1_coding_1.ipynb`' and '`hw1_coding_2.ipynb`' after downloading and unzipping `hw1.zip` at https://github.com/10605/released-hws/releases/tag/s21-hw1.

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).

2.2 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.

In order to enable auto-grading, please do not change any function signatures (e.g., function name, parameters, etc) or delete any cells. If you do delete any of the provided cells (even if you re-add them), the autograder will fail to grade your homework. If you do this, you will need to re-download the homework files and fill in your answers again and resubmit.

2.3 Submission

1. Export your solution notebook as a IPython notebook file on Databricks via `File -> Export -> IPython Notebook`
2. Submit your solution via Gradescope.

2.4 Setting up environments on Databricks

We provide step-by-step instructions on how to configure your Databricks platform. We also introduced it in detail during the recitation on February 5th. The recitations slides 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** `7.5`. 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.

2.5 Entity Resolution

Entity Resolution, or "Record linkage" is the term used by statisticians, epidemiologists, and historians, among others, to describe the process of joining records from one data source with another that describe the same entity. Other terms with the same meaning include, "entity disambiguation/linking", "duplicate detection", "deduplication", "record matching", "(reference) reconciliation", "object identification", "data/information integration", and "conflation".

Entity Resolution (ER) refers to the task of finding records in a dataset that refer to the same entity across different data sources (e.g., data files, books, websites, databases). ER is necessary when joining datasets based on entities that may or may not share a common identifier (e.g., database key, URI, National identification number), as the case may be due to differences in record shape, storage location, and/or curator style or preference. A dataset that has undergone ER may be referred to as being cross-linked. In this homework, we break the entity resolution problem into four parts:

- **Part 0 – Preliminaries:** Load in the dataset into pair RDDs where the key is the mapping ID, and the value is a string consisting of the name/title, description, and manufacturer of the corresponding record.
- **Part 1 – ER as Text Similarity - Bags of Words:** Build components for bag-of-words text analysis, and then compute record similarity. Bag-of-words is a conceptually simple yet powerful approach for text analysis. The idea is treating strings, a.k.a. documents, as unordered collections of words or tokens, i.e., as bags of words.
- **Part 2 – ER as Text Similarity - Weighted Bag-of-Words using TF-IDF:** In this part we compute the TF-IDF weight for each record. Bag-of-words comparisons do not perform well when all

tokens are treated in the same way. In real world scenarios, some tokens are more important than the others. Weights give us a way to specify which tokens could have higher "importance". With weights, when we compare documents, instead of counting common tokens, we sum up the weights of common tokens. A good heuristic for assigning weights is called "Term-Frequency/Inverse-Document-Frequency," or TF-IDF for short. **TF** rewards tokens that appear many times in the same document. It is computed as the frequency of a token in a document. **IDF** rewards tokens that are rare overall in a dataset. The intuition is that it is more significant if two documents share a rare word than a common one.

- **Part 3 – ER as Text Similarity - Cosine Similarity:** Compute the cosine similarity of the tokenized strings based on the TF-IDF weights.
- **Part 4 – Scalable ER:** Use the inverted index data structure to scale ER. The ER algorithm above is quadratic in two ways. First, we did a lot of redundant computation of tokens and weights, since each record was reprocessed every time it was compared. Second, we made quadratically many token comparisons between records. In reality, most records have nothing (or very little) in common. Moreover, it is typical for a record in one dataset to have at most one duplicate record in the other dataset (this is the case assuming each dataset has been de-duplicated against itself). In this case, the output is linear in the size of the input and we can hope to achieve linear running time. An inverted index is a data structure that will allow us to avoid making quadratically many token comparisons. It maps each token in the dataset to the list of documents that contain the token. So, instead of comparing, record by record, each token to every other token to see if they match, we will use inverted indices to look up records that match on a particular token.
- **Part 5 – Analysis:** Determine duplicate entities based on the similarity scores, and compute evaluation metrics. Now we have an authoritative list of record-pair similarities, but we need a way to use those similarities to decide if two records are duplicates or not. The simplest approach is to pick a threshold. Different thresholds correspond to different false positives and false negatives, which will result in different precision and recall scores.

2.6 Machine Learning Pipeline

Following what we have talked about in class, we will work through a minimal machine learning pipeline. The steps include understanding the problem to obtaining a machine learning model. We break this down into 7 parts.

- **Part 1 – Business Understanding:** The first step is to understand the problem we are trying to solve. This could involve conversations with stakeholders as well as readings describing the problem at hand.
- **Part 2 – Data Extract-Transform-Load:** We need to take the raw data we are given and make it usable for machine learning purposes, potentially pulling it from multiple sources.
- **Part 3 – Explore Your Data:** In this part we use SQL embedded in notebooks to take a look at the underlying database containing the data.
- **Part 4 – Visualize Your Data:** In this section, we use the notebook to create a few visualizations of the data we have.
- **Part 5 – Data Preparation:** We have to group up the relevant variables we want to actually use in our model.

- **Part 6 – Data Modeling:** Create a machine learning model and train it.
- **Part 7 – Tuning and Evaluation:** Try out different hyperparameter settings and different model types.

See the notebooks 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?

   (b) If you answered 'yes', give full details (e.g. "Jane Doe explained to me what is asked in Question 3.4")

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

   (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?

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