CS 189:289A Introduction to Spring 2017 Machine Learning — Homework 7: Unsupervised Due Fri., April 28

Instructions

- We prefer that you typeset your answers using LaTeX or other word processing software. Neatly handwritten and scanned solutions will also be accepted.
- Please make sure to start **each question on a new page**, as grading (with Gradescope) is much easier that way!
- This assignment contains some optional questions. Feel free to work on them. We encourage you to try them out to expand your understanding. They will not be graded and will not count towards your grade.
- Deliverables. Submit a **PDF of your writeup** to the Homework 7 assignment on Gradescope. Include your code in your writeup in the appropriate sections. Submit your **code zip** and a README to the Homework 7 Code assignment on Gradescope. Finally, submit **your predictions** for the test sets to Kaggle. Be sure to include your Kaggle display name and score in your writeup.
- Due **Friday, April 28, 2017 at 11:59 PM.**

1 k-means clustering

Given a dataset $x_1, ..., x_n \in \mathbb{R}^d$ and an integer $1 \leq k \leq n$, recall the following k-means objective function

$$\min_{\pi_1,...,\pi_k} \sum_{i=1}^{k} \sum_{j \in \pi_i} \|x_j - \mu_i\|_2^2 \,, \;\; \mu_i = \frac{1}{|\pi_i|} \sum_{j \in \pi_i} x_j \,. \tag{1}$$

Above, $\{\pi_i\}_{i=1}^k$ is a partition of $\{1, 2, ..., n\}$. The objective (1) is NP-hard[1] to find a global minimizer of. Nevertheless the commonly used heuristic which we discussed in lecture, known as Lloyd's algorithm, typically works well in practice.

(a) Implement Lloyd's algorithm for solving the k-means objective (1). Do not use any off the shelf implementations, such as those found in `scikit-learn`.

(i) Run your algorithm on MNIST with $k = 5, 10, 20$ cluster centers. Use the image data at `mnist_data/images.mat`, which contains 60,000 unlabeled images, and each image contains 28×28 pixels. Each pixel represents one coordinate in the k-means algorithm.

(ii) Visualize the centers you get, viewing each coordinate as a pixel (i.e., each center is represented by an image of 28×28 pixels). What are the differences between results with different numbers of cluster centers?

(iii) **(Optional)** Implement the `kmeans++` initialization scheme[2] for your k-means implementation. Note that this initialization scheme is widely used in practice.

[1] To be more precise, it is both NP-hard in d when $k = 2$ and k when $d = 2$. See the references on the wikipedia page for k-means for more details.

[2] See `http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf`.

2 Low-Rank Approximation

Low-rank approximation tries to find an approximation to a given matrix, where the approximation matrix has a lower rank compared to the original matrix. This is useful for mathematical modeling and data compression. Mathematically, given a matrix D, we try to find $\hat{D}$ in the following optimization problem,

$$\underset{\hat{D}}{\operatorname{argmin}} \|D - \hat{D}\|_F \quad \text{subject to} \quad \operatorname{rank}(\hat{D}) \leq k \tag{2}$$

where $\|A\|_F = \sqrt{\sum_i \sum_j a_{ij}^2}$ represents the Frobenius norm, i.e., sum of squares of all entries in the matrix followed by a square root.

This problem can be solved using Singular Value Decomposition (SVD). Specifically, let $D = U\Sigma V^\top$, where $\Sigma = \operatorname{diag}(\sigma_1, ..., \sigma_n)$. Then a rank-$k$ approximation of D can be written as $\hat{D} = U\hat{\Sigma}V^\top$, where $\hat{\Sigma} = \operatorname{diag}(\sigma_1, ..., \sigma_k, 0, ..., 0)$. In this problem, we aim to perform this approximation method on gray-scale images, which can be thought of as a 2D matrix.

(a) Using the image `low-rank_data/face.jpg`, perform a rank-5, rank-20, and rank-100 approximation on the image. Show the obtained images in your report.

(b) Plot the Mean Squared Error (MSE) of using different rank approximations compared to the original image, ranging from rank 1 to rank 100.

(c) Now perform the same rank-5, rank-20, and rank-100 approximation on `low - rank_data/sky.jpg`. Show the obtained images in your report.

(d) Find the lowest rank approximation that you begin to have a hard time differentiating the original and the approximated images. Compare your results for the face and the sky image. What are the possible reasons for the difference?

3 Joke Recommender System

You will build a personalized joke recommender system. There are $m = 100$ jokes and $n = 24,983$ users. As historical data, every user read a subset of jokes and rated them. The goal is to recommend more jokes, such that the recommended jokes match the individual user's sense of humour.

3.1 Data Format

The historical rating is represented by a matrix $R \in \mathbb{R}^{n \times m}$. The entry R_{ij} represents the user i's rating on joke j. The rating is a real number in $[-10, 10]$: a higher value represents that the user is more satisfied with the joke. If the joke is not rated, then the corresponding entry value is `NaN`.

The directory `joke_data/jokes/` contains the text of all 100 jokes. Read them before you start! In addition, you are provided with three files at `joke_data/`:

- `joke_train.mat` is given as the training data. It contains the matrix R specified above.
- `validation.txt` contains user-joke pairs that doesn't appear in the training set. Each line takes the form "`i, j, s`", where `i` is the user index, `j` is the joke index, `s` indicates whether the user likes the joke. More specifically, `s` = 1 if and only if the user gives a positive rating to the joke.
- `query.txt` contains user-joke pairs that are neither in the training set nor in the validation set. Each line takes the form "`id, i, j`". You are asked to predict if user `i` likes joke `j`. The integer `id` is a unique id for the user-joke pair. Use it to submit the prediction to Kaggle (see Section 3.3).

3.2 Latent Factor Model

Latent factor model is the state-of-the-art method for personalized recommendation. It learns a vector representation $u_i \in \mathbb{R}^d$ for each user and a vector representation $v_j \in \mathbb{R}^d$ for each joke, such that the inner product $\langle u_i, v_j \rangle$ approximates the rating R_{ij}. You will build a simple latent factor model using two different methods in the following.

(a) The first method is using singular value decomposition (SVD) to learn a lower dimensional vector representation for users and jokes.

(i) Start by replacing all missing values in the data matrix by zero. Then apply SVD on the resulting matrix, and compute the corresponding representations for users and jokes. Recall this means to project the data vectors to lower dimensional subspaces of their corresponding spaces, spanned by singular vectors. Refer to the lecture materials on SVD, PCA and dimensionality reduction.

(ii) Evaluate the learnt vector representations by mean squared error:

$$\text{MSE} = \sum_{(i,j) \in S} (\langle u_i, v_j \rangle - R_{ij})^2 \quad \text{where } S := \{(i, j) : R_{ij} \neq \texttt{NaN}\}. \tag{3}$$

Try $d = 2, 5, 10, 20$. How does the MSE vary as a function of d? Use the inner product $\langle u_i, v_j \rangle$ to predict if user i likes joke j. Report prediction accuracies on the validation set.

(b) For sparse data, replacing all missing values by zero is not a completely satisfying solution. A missing value means that the user has not read the joke, but doesn't mean that the rating should be zero. A more reasonable choice is to minimize the MSE only on rated joke.

(i) Let's define a loss function:

$$L\Big(\{u_i\},\{v_j\}\Big) := \sum_{(i,j)\in S} (\langle u_i, v_j\rangle - R_{ij})^2 + \lambda \sum_{i=1}^{n} \|u_i\|_2^2 + \lambda \sum_{j=1}^{m} \|v_j\|_2^2,$$

where set S has the same definition as in equation (3) and $\lambda > 0$ is the regularization coefficient. Implement an algorithm to learn vector representations by minimizing the loss function $L(\{u_i\},\{v_j\})$.

Hint: you may want to employ an alternating minimization scheme. First, randomly initialize $\{u_i\}$ and $\{v_j\}$. Then minimize the loss function with respective to $\{u_i\}$ by treating $\{v_j\}$ as constant vectors, and minimize the loss function with respect to $\{v_j\}$ by treating $\{u_i\}$ as constant vectors. Iterate these two steps until both $\{u_i\}$ and $\{v_j\}$ converge. Note that when one of $\{u_i\}$ or $\{v_j\}$ is given, minimizing the loss function with respect to the other part has closed-form solutions.

(ii) Compare the resulting MSE and the prediction error with (a). Note that you need to tune the hyper-parameter λ to optimize the performance.

3.3 Recommending Jokes

(a) Using the methods you have implemented to predict the users' preference on unread jokes. For each line "`id, i, j`" in the query file, output a line "`id, s`" to a file named `kaggle_submission.txt`. Here, `s = 1` means that user `i` will give a positive rating to joke `j`, while `s = 0` means that the user will give a non-positive rating. The first line of `kaggle_submission.txt` should be the field titles: "`Id, Category`". Submit `kaggle_submission.txt` to Kaggle.

(b) (Optional) Build a joke recommender system for your friends: randomly display 5 jokes to receive their ratings, then recommend 5 best jokes and 5 worst jokes. Report your friends' ratings on the jokes that the system recommends.