CS189: Introduction to Machine Learning

Homework 6

Due: 12:00 pm noon, Thursday December 1st, 2016

- 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.

Submission Instructions

You will submit your PDF writeup and code to **Gradescope**. There will also be two **Kaggle** competitions.

In your submission to **Gradescope**, include separately:

1. A PDF writeup with answers to all the questions. Include at the end of the pdf a copy of all your code.
2. A zip file of all your code.

Submitting to **Kaggle**

3. Submit a csv file with your best predictions for the examples in the test set to Kaggle, just like in previous homeworks.

Note: The Kaggle invite links and more instructional details will be posted on Piazza. Good luck!

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

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

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`.

[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.

(i) Run both algorithms on MNIST with $k = 5, 10, 20$ cluster centers and visualize the centers, viewing each coordinate as a pixel.

(ii) Use the image data at `mnist_data/images.mat`. There 60,000 unlabeled images. Each image contains 28×28 pixels.

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

2 Principle component analysis (PCA)

In this problem, we fix an $n \times n$ positive semi-definite matrix A. We will derive, from first principles, the best rank-1 approximation to A, and then (optionally) derive an algorithm for computing such an approximation. Recall the following metric of approximation: for any integer $1 \le r \le n$, we define the best rank-r approximation as any minimizer A_r of

$$\min_{M \in \mathbb{R}^{n \times n}} \|A - M\|_F : \operatorname{rank}(M) \le r \ .$$

Note that such a minimizer is not necessarily unique (why?). Since A is positive semi-definite, A_r must be as well: you make take this fact for granted. In this problem, we focus on the special case of $r = 1$. Also assume $A \neq 0$ to avoid any uninteresting, degenerate cases.

(a) Show that $A_1 = mm^\mathsf{T}$, where m is any minimizer of the following *unconstrained* optimization problem

$$\min_{m \in \mathbb{R}^n} \|A - mm^\mathsf{T}\|_F \ .$$

(b) Define the function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ as $f(m) = \|A - mm^\mathsf{T}\|_F^2$. Compute $\nabla f(m)$.

(c) Set $\nabla f(m) = 0$ and conclude that all minimizers m of f must satisfy $m = \sqrt{\lambda} v$ where $\lambda \ge 0$ is an eigenvalue of A and v is the corresponding (unit normalized) eigenvector of A.

(d) Argue from part (c) that $A_1 = \lambda_1 v_1 v_1^\mathsf{T}$, where λ_1 is the maximum eigenvalue of A and v_1 is a corresponding (unit normalized) eigenvector.

(e) **(Optional)** Now let us devise an algorithm to compute A_1. We will do this by computing v_1. Suppose that we have the gap condition $\lambda_1 > \lambda_2$, where λ_2 is the second largest eigenvalue of A. Consider the following algorithm (known as *power iteration*): start with an $x_0 \sim N(0, I)$, and iterate

$$x_{k+1} = \frac{Ax_k}{\|Ax_k\|_2} \ , \ \ k = 0, 1, 2, \dots \ . \tag{3}$$

Show that for some $\sigma \in \{-1, +1\}$, with probability 1,

$$\lim_{k \to \infty} \|x_k - \sigma v_1\|_2 = 0 \ .$$

Hint: Observe that $x_k = \frac{A^k x_0}{\|A^k x_0\|_2}$. Express x_0 in the orthonormal eigenbases of A. The fact that $x_0 \sim N(0, I)$ means that with probability 1, $\langle x_0, v_i \rangle \neq 0$ for every eigenvector v_i. Now use the gap condition.

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

3 Singular value decomposition (SVD)

In this question, we will study a few simple properties of singular values and singular vectors. Given a matrix A, we order the singular values $\sigma_1(A) \geq \sigma_2(A) \geq \ldots \geq \sigma_r(A) > 0$, where $r = \operatorname{rank}(A)$. We define $\sigma_{\max}(A) = \sigma_1(A)$ and $\sigma_{\min} = \sigma_r(A)$. For this question, given a matrix A, we define three matrix norms

$$\|A\| = \sigma_1(A) \, , \ \ \|A\|_F = \|\operatorname{vec}(A)\|_2 \, , \ \ \|A\|_* = \sum_{i=1}^{r} \sigma_i(A) \, .$$

(a) Show that the singular values are *unitarily invariant*. That is, given an $n \times m$ matrix A, show that $\sigma(A) = \sigma(Q_1 A) = \sigma(A Q_2)$, where Q_1 is any $n \times n$ orthonormal matrix and Q_2 is any $m \times m$ orthonormal matrix, and $\sigma(\cdot)$ denotes the set of singular values.

(b) Show that $\|A\|_F = \sqrt{\sum_{i=1}^{r} \sigma_i(A)^2}$. Hence, by part (a) the Frobenious norm is also unitarily invariant.

(c) Conclude that for any A of rank r,

$$\|A\| \leq \|A\|_F \leq \|A\|_* \leq \sqrt{r}\|A\|_F \leq r\|A\| \, .$$

You may use without proof the equivalence of norms on $\mathbb{R}^d$.

(d) Show the following equivalences,

$$\|A\| \leq 1 \iff I - A^{\mathsf{T}} A \succeq 0 \iff I - A A^{\mathsf{T}} \succeq 0 \iff \begin{bmatrix} I & A \\ A^{\mathsf{T}} & I \end{bmatrix} \succeq 0 \, .$$

Hint: For the last equivalence, you make look up and use without proof Schur complements, or you can supply an alternative proof that does not directly use Schur complements.

(e) Suppose that A has full column rank. Prove that

$$\|AB\| \geq \sigma_{\min}(A)\|B\| \, ,$$

for any size conforming B.

(f) **(Optional)** Use the singular value decomposition of A to diagonalize the following Hermitian lift matrix,

$$\phi(A) = \begin{bmatrix} 0 & A \\ A^{\mathsf{T}} & 0 \end{bmatrix} \, .$$

Hint: First, show that the eigenvalues of $\phi(A)$ are $\pm$ the singular values of A. Then use the left and right singular vectors of A to build an orthonormal matrix.

4 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.

4.1 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 4.3).

4.2 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.

1. Replace all missing values by zero. Then use singular value decomposition (SVD) to learn a lower dimensional vector representation 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.

2. Evaluate the learnt vector representations by mean squared error:

$$\text{MSE} = \sum_{(i,j)\in S} (\langle u_i, v_j\rangle - R_{ij})^2 \quad \text{where } S := \{(i,j) : R_{ij} \neq \texttt{NaN}\}. \tag{4}$$

   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.

3. 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. 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 (4) 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.

4. Compare the resulting MSE and the prediction error with Step 2. Note that you need to tune the hyper-parameter λ to optimize the performance.

4.3 Recommending Jokes

1. 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.

2. (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.