CSE 6740 A/ISyE 6740, Fall 2023, Georgia Tech

CSE 6740: Homework 1

Due Sept 14, '23 (11:59 pm ET) on Gradescope

Cite any sources and collaborators; do not copy. See syllabus for policy.

1 Linear Regression [30 pts]

In class, we derived a closed form solution (normal equation) for linear regression problem: $\hat{w} = (X^T X)^{-1} X^T Y$. A probabilistic interpretation of linear regression tells us that we are relying on an assumption that each data point is actually sampled from a linear hyperplane, with some noise. The noise follows a zero-mean Gaussian distribution with constant variance. Specifically,

$$Y^i = w^T X^i + \epsilon^i \tag{1}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, $w \in \mathbb{R}^d$, and $\{X^i, Y^i\}$ is the i-th data point, with $1 \leq i \leq m$. In other words, we are assuming that each every point is independent to each other and that every data point has same variance.

(a) Using the normal equation, and the model (Eqn. 1), derive the expectation $\mathbb{E}[\hat{w}]$. Note that here X is fixed, and only Y is random, i.e. "fixed design" as in statistics. [6 pts]

(b) Similarly, derive the variance $\text{Var}[\hat{w}]$. [6 pts]

(c) Under the white noise assumption above, does $\hat{w}$ follow a Gaussian distribution with mean and variance in (a) and (b), respectively? Why or why not? [8 pts]

(d) **Weighted linear regression:**

Suppose we keep the independence assumption but remove the same variance assumption. In other words, data points would be still sampled independently, but now they may have different variance σ_i. Thus, the variance (the covariance matrix) of ϵ would be still diagonal, but with different values:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_m^2 \end{bmatrix}. \tag{2}$$

Derive the estimator $\hat{w}$ (similar to the normal equations) for this problem using matrix-vector notations with Σ. [10 pts]

2 Ridge Regression [15 pts]

For linear regression, it is often assumed that $y_i = w^\top x_i + \epsilon$ where $w, x \in \mathbb{R}^d$ by absorbing the constant term (bias) in an affine hypothesis into w, and $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is a Gaussian random variable. Given m i.i.d samples $z_i = (x_i, y_i), 1 \leq i \leq m$, we define $Y = (y_1, ..., y_m)^\top$ and $X = (x_1, ..., x_m)^\top$. Thus, we have $Y \sim \mathcal{N}(Xw, \sigma^2 I)$. Show that the ridge regression estimate is the mean of the posterior distribution under a Gaussian prior $w \sim \mathcal{N}(0, \tau^2 I)$. Find the explicit relation between the regularization parameter λ in the ridge regression estimate of the parameter w, and the variances σ^2, τ^2.

3 Lasso estimator

The LASSO regression problem can be shown to be the following optimization problem:

$$\min_{w \in \mathbb{R}^d} \sum_{i=1}^{m} (w^\top x_i - y_i)^2 \quad \text{subject to} \|w\|_1 \leq \lambda, \tag{3}$$

where $\lambda > 0$ is a regularization parameter. Here, we develop a stochastic gradient descent (SGD) algorithm for this problem, which is useful when we have $m >> d$, where d is the dimension of the parameter space.

(a) Write $w = w^+ - w^-$, where $w^+, w^- \geq 0$ are the positive and negative parts of w respectively. That is, when the jth component, w_j, of w, is less than 0, $w_j^+ = 0$ and $w_j^- = -w_j$. Similarly, when $w_j > 0$, $w_j^+ = w_j$ and $w_j^- = 0$. Find a quadratic function, Q, of w^+ and w^- such that

$$\min_{w^+, w^- \geq 0} \lambda \sum_{j=1}^{m} Q(w^+, w^-) \tag{4}$$

is equivalent to the above LASSO problem in (3). Explain the equivalence. [10 pts]

(b) [Mohri et al Ex. 11.10] Derive a stochastic gradient descent algorithm for the quadratic program (with affine constraints) in part (a). [10 pts]

(c) Suppose $X = [x_1, \cdots, x_m]^\top$ is orthonormal and there exists a solution w for $Xw = Y$, where $Y = [y_1, \cdots, y_m]^\top$ with no more than k non-zero elements. Can the SGD algorithm get arbitrarily close to w? Explain why or why not. [10 pts]

4 Logistic Regression

Logistic regression is named after the log-odds of success (the logit of the probability) defined as below:

$$\ln \left(\frac{P[Y = 1 | X = x]}{P[Y = 0 | X = x]} \right)$$

where

$$P[Y = 1|X = x] = \frac{\exp(w_0 + w^T x)}{1 + \exp(w_0 + w^T x)}$$

(a) Show that log-odds of success is a linear function of X. [6 pts]

(b) Show that the logistic loss $L(z) = \log(1 + \exp(-z))$ is a convex function. [9 pts]

5 Programming: Recommendation System [40 pts]

Personalized recommendation systems are used in a wide variety of applications such as electronic commerce, social networks, web search, and more. Machine learning techniques play a key role to extract individual preference over items. In this assignment, we explore this popular business application of machine learning, by implementing a simple matrix-factorization-based recommender using gradient descent.

Suppose you are an employee in Netflix. You are given a set of ratings (from one star to five stars) from users on many movies they have seen. Using this information, your job is implementing a personalized rating predictor for a given user on unseen movies. That is, a rating predictor can be seen as a function $f : \mathcal{U} \times \mathcal{I} \to \mathbb{R}$, where $\mathcal{U}$ and $\mathcal{I}$ are the set of users and items, respectively. Typically the range of this function is restricted to between 1 and 5 (stars), which is the the allowed range of the input.

Now, let's think about the data representation. Suppose we have m users and n items, and a rating given by a user on a movie. We can represent this information as a form of matrix, namely rating matrix M. Suppose rows of M represent users, while columns do movies. Then, the size of matrix will be $m \times n$. Each cell of the matrix may contain a rating on a movie by a user. In $M_{15,47}$, for example, it may contain a rating on the item 47 by user 15. If he gave 4 stars, $M_{15,47} = 4$. However, as it is almost impossible for everyone to watch large portion of movies in the market, this rating matrix should be very sparse in nature. Typically, only 1% of the cells in the rating matrix are observed in average. All other 99% are missing values, which means the corresponding user did not see (or just did not provide the rating for) the corresponding movie. Our goal with the rating predictor is estimating those missing values, reflecting the user's preference learned from available ratings.

Our approach for this problem is matrix factorization. Specifically, we assume that the rating matrix M is a low-rank matrix. Intuitively, this reflects our assumption that there is only a small number of factors (e.g, genre, director, main actor/actress, released year, etc.) that determine like or dislike. Let's define r as the number of factors. Then, we learn a user profile $U \in \mathbb{R}^{m \times r}$ and an item profile $V \in \mathbb{R}^{n \times r}$. (Recall that m and n are the number of users and items, respectively.) We want to approximate a rating by an inner product of two length r vectors, one representing user profile and the other item profile. Mathematically, a rating by user u on movie i is approximated by

$$M_{u,i} \approx \sum_{k=1}^{r} U_{u,k} V_{i,k}. \tag{5}$$

We want to fit each element of U and V by minimizing squared reconstruction error over all training data points. That is, the objective function we minimize is given by

$$E(U,V) = \sum_{(u,i)\in M} (M_{u,i} - U_u^T V_i)^2 = \sum_{(u,i)\in M} (M_{u,i} - \sum_{k=1}^{r} U_{u,k} V_{i,k})^2 \tag{6}$$

where U_u is the uth row of U and V_i is the ith row of V. We observe that this looks very similar to the linear regression. Recall that we minimize in linear regression:

$$E(w) = \sum_{i=1}^{m} (Y^i - w^T x^i)^2 = \sum_{i=1}^{m} (Y^i - \sum_{k=1}^{r} w_k x_k^i)^2 \tag{7}$$

where m is the number of training data points. Let's compare (6) and (7). $M_{u,i}$ in (6) corresponds to Y^i in (7), in that both are the observed labels. $U_u^T V_i$ in (6) corresponds to $w^T x^i$ in (7), in that both are our estimation with our model. The only difference is that both U and V are the parameters to be learned in (6), while only w is learned in (7). This is where we personalize our estimation: with linear regression, we apply the same w to any input x^i, but with matrix factorization, a different profile U_u are applied depending on who is the user u.

As U and V are interrelated in (6), there is no closed form solution, unlike linear regression case. Thus, we need to use gradient descent:

$$U_{v,k} \leftarrow U_{v,k} - \mu \frac{\partial E(U,V)}{\partial U_{v,k}}, \qquad V_{j,k} \leftarrow V_{j,k} - \mu \frac{\partial E(U,V)}{\partial V_{j,k}}, \tag{8}$$

where μ is a hyper-parameter deciding the update rate. It would be straightforward to take partial derivatives of $E(U,V)$ in (6) with respect to each element $U_{v,k}$ and $V_{j,k}$. Then, we update each element of U and V using the gradient descent formula in (8).

(a) Derive the update formula in (8) by solving the partial derivatives. [10 pts]

(b) To avoid overfitting, we usually add regularization terms, which penalize for large values in U and V. Redo part (a) using the regularized objective function below. [5 pts]

$$E(U,V) = \sum_{(u,i)\in M} (M_{u,i} - \sum_{k=1}^{r} U_{u,k} V_{i,k})^2 + \lambda \sum_{u,k} U_{u,k}^2 + \lambda \sum_{i,k} V_{i,k}^2$$

(λ is a hyper-parameter controlling the degree of penalization.)

(c) Implement `myRecommender.py` by filling the gradient descent part. You are given a skeleton code `myRecommender.py`. Using the training data `rateMatrix`, you will implement your own recommendation system of rank `lowRank`. The only file you need to edit and submit is `myRecommender.py`. In the gradient descent part, repeat your update formula in (b), observing the average reconstruction error between your estimation and ground truth in training set. You need to set a stopping

criteria, based on this reconstruction error as well as the maximum number of iterations. You should play with several different values for μ and λ to make sure that your final prediction is accurate.

Formatting information is here:

5.1 Input

- **rateMatrix**: training data set. Each row represents a user, while each column an item. Observed values are one of $\{1, 2, 3, 4, 5\}$, and missing values are 0.
- **lowRank**: the number of factors (dimension) of your model. With higher values, you would expect more accurate prediction.

5.2 Output

- **U**: the user profile matrix of dimension user count $\times$ low rank.
- **V**: the item profile matrix of dimension item count $\times$ low rank.

5.3 Evaluation [15 pts]

Upload your `myRecommender.py` implementation file. (Do not copy and paste your code in your report. Be sure to upload your `myRecommender.py` file.)

To test your code, try to run `homework1.py`. You may have noticed that the code prints both training and test error, in RMSE (Root Mean Squared Error), defined as follows:

$$\sum_{(u,i)\in M} (M_{u,i} - f(u,i))^2$$

where $f(u, i)$ is your estimation, and the summation is over the training set or testing set, respectively. For the grading, we will use another set-aside testing set, which is not released to you. If you observe your test error is less than 1.00 without cheating (that is, training on the test set), you may expect to see the similar performance on the unseen test set as well.

Note that we provide `homework1.py` just to help you evaluate your code easily. You are not expected to alter or submit this to us. In other words, we will not use this file when we grade your submission. The only file we take care of is `myRecommender.py`.

Grading criteria:

- Your code should output U and V as specified. The dimension should match to the specification.

- We will test your output on another test dataset, which was not provided to you. The test RMSE on this dataset should be at least 1.05 to get at least partial credit.
- We will measure elapsed time for learning. If your implementation takes longer than 3 minutes for rank 5, you should definitely try to make your code faster or adjust parameters. Any code running more than 5 minutes is not eligible for credit.
- Your code should not crash. Any crashing code will be not credited.

5.4 Report [10 pts]

In your report, show the performance (RMSE) both on your training set and test set, with varied `lowRank`. (The default is set to 1, 3, and 5, but you may want to vary it further.) Discuss what you observe with varied low rank. Also, briefly discuss how you decided your hyper-parameters (μ, λ).

5.5 Note

- Do not print anything in your code (e.g, iteration 1 : err=2.4382) in your final submission.
- Do not alter input and output format of the skeleton file. (E.g, adding a new parameter without specifying its defalut value) Please make sure that you returned all necessary outputs according to the given skeleton.
- Please do not use additional file. This task is simple enough that you can fit in just one file.
- Submit your code with the best parameters you found. We will grade without modifying your code. (Applying cross-validation to find best parameters is fine, though you do not required to do.)
- Please be sure that your program finishes within a fixed number of iterations. Always think of a case where your stopping criteria is not satisfied forever. This can happen anytime depending on the data, not because your code is incorrect. For this, we recommend setting a maximum number of iteration in addition to other stopping criteria.