MICAS901 - Introduction to Optimization: Homework 2

Please submit your solution by email before November 22nd, 2020

Hadi Ghauch, Telecom Paris,
email: hadi.ghauch@telecom-paristech.fr

All the steps and derivations are needed for full credit. The solution for each homework should be submitted individually by each student (one student per homework). Regarding the questions that need programming or coding, feel free to use any programing language you are comfortable with. The homework is worth 20% of total course grade.

Part I: Closed-form solutions vs iterative algorithms

Consider the following linear regression problem:

$$\boldsymbol{w}^\star = \min_{\boldsymbol{w}\in\mathbb{R}^d} \frac{1}{N} \sum_{i\in[N]} \|\boldsymbol{w}^T\boldsymbol{x}_i - y_i\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2 \tag{1}$$

where $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ is the training set.

a) Derive a closed-form solution for $\boldsymbol{w}^\star$ in (1).
Derive an approximation of the computational complexity (in terms of $\mathcal{O}(-)$ analysis) for this solution.

b) Consider "Communities and Crime" dataset ($N = 1994$, $d = 128$) and find the optimal linear regressor from the closed-form expression.
What is the complexity in that case for this value of N ?

c) Repeat b) for "Individual household electric power consumption" dataset ($N = 2075259$, $d = 9$). Observe the scalability issue of the closed-form expression.

d) Derive a simple gradient algorithm to solve (1), iteratively.
How do you select the 'best' stepsize ? argue your choice
Derive an approximation of the computational complexity (in terms of $\mathcal{O}(-)$ analysis) for this method. Compare it with that of the closed form solution.
Implement this algorithm using the "Individual household electric power consumption"
Plot the training error (using ℓ_2 loss function) vs the number of iterations, against the optimal closed-form solution.
Verify that the gradient descent eventually converges to the optimal solution. Argue the reason for that.

Part II: Deterministic vs Stochastic algorithms

Now consider logistic ridge regression

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} f(\boldsymbol{w}) = \frac{1}{N}\sum_{i\in[N]} f_i(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2, \quad \text{where} \quad f_i(\boldsymbol{w}) = \log\left(1+\exp\{-y_i\boldsymbol{w}^T\boldsymbol{x}_i\}\right)$$

for the "Individual household electric power consumption" dataset ($N = 2075259$, $d = 9$). This is supervised learning problem, specifically, a classification problem, i.e., the training set is defined as: $\{\ (\boldsymbol{x}_i, y_i)\ \}_{i=1}^N$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is a feature vector, and $y_i \in \mathbb{B}$ is a *binary label*. Solve the optimization problem using GD, plain stochastic GD, and SVRG .

0-a) Is f Lipschitz continuous? If so, find a small Lipschitz constant L?

0-b) Is f strongly convex? If so, find a large μ?

0-c) Can you use the same "trick" as Part III of HW1 to find L and μ ?

0-d) What can you say about the ratio L/μ ? is it a 'good' or 'bad' setup for a plain GD method, for the dataset considered here ?

a) Write/Derive the update equations for each algorithm, and implement them

b) Tune the hyper-parameters for algorithm (including λ), using a held out cross validation test set.

c) Compare all these methods in terms complexity of hyper-parameter tunning, convergence time, convergence rate (in terms of # outer-loop iterations), and memory requirement.

d) Compare the training error (using ℓ_2 loss function) of all these algorithms. Comment on these results: are they expected or is there any discrepancy?

e) Compare the test error (using ℓ_2 loss function) of all these algorithms. Comment on these results: are they expected or is there any discrepancy?

Part III: Matrix Factorization for Recommendation System

Consider the low-rank matrix factorization (MF) for recommendation systems. Read carefully the problem setting in second lecture on non-convex optimisation (lecture 7)

- Data matrix $\boldsymbol{X}$ of size $M \times N$. M = number of users, N = number of items
- $[\boldsymbol{X}]_{u,i}$ is rating (score) that user u gave to item i
- In recommendation sys model rating as: $[\boldsymbol{X}]_{u,i} = \boldsymbol{p}_i^T\boldsymbol{q}_u$,
  $\boldsymbol{p}_i \in \mathbb{R}^d$ vector of **item characteristics** , $\boldsymbol{q}_u \in \mathbb{R}^d$ vector of **user preferences**
- factorize $\boldsymbol{X}$ with low-rank matrices: $\boldsymbol{P} \in \mathbb{R}^{M\times d}$, $\boldsymbol{Q} \in \mathbb{R}^{N\times d}$, $d \ll (M, N)$ such that $\boldsymbol{X} = \boldsymbol{P}\boldsymbol{Q}^T$
- every col. of $\boldsymbol{Q}^T$ (resp. row of $\boldsymbol{P}$) is user prefs (resp item characteristics)

We will build on that example by considering a realistic implementation using popular movie recommendation datasets, e.g., the MovieLens dataset. You will be solving the problem posed in Lecture 7, namely, the "Low-rank MF (supervised learning)" slide: please make sure you read the problem that is formulated in that slide.

We denote by $\mathcal{K}$ the set of **rated entries** in $\boldsymbol{X}$, i.e., entries of the rating matrix which have a score/rating by at least one user. Then the low-rank MF model is learned from the set of rated entries, $\mathcal{K}$, i.e.,

$$\underset{\{\boldsymbol{p}_i\ ,\ \boldsymbol{q}_u\}_{(u,i)\ \in\mathcal{K}}}{\operatorname{argmin}} \sum_{(u,i)\in\mathcal{K}} \left[\left([\boldsymbol{X}]_{u,i} - \boldsymbol{p}_i^T\boldsymbol{q}_u\right)^2 + \lambda_i\|\boldsymbol{p}_i\|_2^2 + \mu_u\|\boldsymbol{q}_u\|_2^2\right] \tag{2}$$

In general, $\{\lambda_i > 0, \mu_u > 0\}_{(u,i)\in\mathcal{K}}$ are regularization parameters to prevent overfitting, tuned via cross-validation. However, here we will focus on the training part (rather than the testing) and assume that all these regularization parameters are **strictly positive** and **given** to us. Due to the presence of coupling in the cost function, Block-Coordinate Descent (BCD) methods are prevalent for solving (2). We thus propose to use the following standard methods to solve (2): Block-Coordinate Descent (BCD) with closed-form solution, and BCD with Gradient Descent (GD).

0a) Starting from (2), apply the BCD method to derive the subproblem for each block of coordinates, In other words, derive the subproblem corresponding to each of the optimization variables in (2), $\{\boldsymbol{p}_i\ ,\ \boldsymbol{q}_u\}_{(u,i)\ \in\mathcal{K}}$

0b) show that each subproblem is strongly convex.

1) **BCD with closed-form solution:** also called alternating least-squares
- derive the update for each of the subproblems (part 0a)) in **closed-form solution**
- show that algorithm which results from these updates **converges monotonically to a stationary point** of (2)
- derive an estimate of the **computational complexity** per BCD iteration (in $\mathcal{O}()$ notation)

2) **BCD with GD:** also called block-gradient descent
- In part 1) you derived a closed-form solution for each of the subproblems. For this method, do not solve each subproblem optimally, but rather take T **GD steps** (T is a small positive integer) with respect to the block of coordinates being optimized for that subproblem. Derive the update equations that result from this strategy, as a function T.
- what happens as $T \to \infty$?
- can you show the **convergence to a stationary point** of the resulting algorithm using the framework of BCD ?
- derive an estimate of the **computational complexity** per BCD iteration, as a function T (in $\mathcal{O}()$ notation)

3 Implement all your algorithms using the MovieLens 100K dataset [1].
- pick a value for model complexity, d, and plot the training error (empirical risk in (2)) using the ℓ_2 loss function, for each of the algorithm.
- plot the training error of BCD with GD for increasing value of T. what are the pros and cons of increasing T ?

[1] You download that dataset along with its description here `https://grouplens.org/datasets/movielens/100k/`