Convex Optimization for Machine Learning and Computer Vision

Lecture: Dr. Tao Wu
Exercises: Yuesong Shen, Zhenzhang Ye
Winter Semester 2018/19

Computer Vision Group
Institut für Informatik
Technische Universität München

Weekly Exercises 5

Room: 01.09.014
Wednesday, 05.12.2018, 12:15-14:00
Submission deadline: Monday, 03.12.2018, 16:15, Room 01.09.014

Proximal operator (10+6 Points)

Exercise 1 (4 Points)**.** Assume function $J : \mathbb{R}^n \to \mathbb{R}$ is convex and subdifferentiable on its domain. Show that u^* minimizes J if and only if $u^* = \text{prox}_J(u^*)$.

Exercise 2 (4 Points)**.** Prove following properties of proximal operator:

- If $J(u) = \alpha f(u) + b$, with $\alpha > 0$, then $\text{prox}_{\lambda J}(v) = \text{prox}_{\alpha\lambda f}(v)$.
- If $J(u) = f(Qu)$, where Q is an orthogonal matrix, then $\text{prox}_{\lambda J}(v) = Q^\top \text{prox}_{\lambda f}(Qv)$

Exercise 3 (4 Points)**.** Show that the ℓ_1-norm proximal operator of $v \in \mathbb{R}^n$ is given as

$$\text{prox}_{\lambda\|\cdot\|_1}(v) = u \in \mathbb{R}^n, \quad u_i := \begin{cases} v_i + \lambda & \text{if } v_i < -\lambda \\ 0 & \text{if } v_i \in [-\lambda, \lambda] \\ v_i - \lambda & \text{if } v_i > \lambda. \end{cases}$$

Exercise 4 (4 points)**.** Compute the proximal operator of the $1,2$-norm, i.e.

$$\text{prox}_{\tau||X||_{1,2}},$$

where $X \in \mathbb{R}^{m\times n}$ is a matrix .

Multinomial Logistic Regression (Due:17.12) (16 Points)

Exercise 5 (16 Points)**.** In this exercise you are asked to train a linear model for a multiclass classification task with Logistic regression. The idea is as follows: You are given a set of training samples $\mathcal{I} = \{1, \dots, N\}$ that are represented by their feature vectors $x_i \in \mathbb{R}^d$, for $i \in \mathcal{I}$. Each training sample i is associated with a class label $y_i \in \{1, \dots, C\}$. The aim is to estimate a linear classifier parameterized by $W^* \in \mathbb{R}^{d \times C}, b^* \in \mathbb{R}^C$ so that $y_i = \text{argmax}_{1 \le j \le C} \, x_i^\top W_j^* + b_j^*$ for most training samples i. Once you have obtained this "optimal" classifier the hope is, that you are able to classify new unseen and unlabeled samples $x \in \mathbb{R}^d$. In machine learning this is called generalization. For this task you may query your trained model via the classifier rule

$$y = \text{argmax}_{1 \le j \le C} \, x^\top W_j^* + b_j^* \tag{1}$$

and y probably is the true class label of x if your model generalizes well. In order to estimate the model we solve an optimization problem of the form

$$\min_{W \in \mathbb{R}^{d \times C}, b \in \mathbb{R}^C} \; \frac{1}{N} \sum_{i=1}^{N} \; \ell(W, b, x_i, y_i) + \frac{\lambda_1}{2} \|W\|_2^2 + \frac{\lambda_2}{2} \|b\|_2^2, \tag{2}$$

where

$$\ell(W, b, x_i, y_i) = -\log \left(\frac{\exp(\langle W_{y_i}, x_i \rangle + b_{y_i})}{\sum_{j=1}^{C} \exp(\langle W_j, x_i \rangle + b_j)} \right) \tag{3}$$

is called the softmax loss. Note that the above problem is smooth and strongly convex and can be solved with gradient descent. In practice however, it may happen, that some features (i.e. components of the vector x_i) do not contain any information about the true class labels, i.e. components that are just noise. In order to filter out the useless features we modify the norm on W. So we have

$$\min_{W \in \mathbb{R}^{d \times C}, b \in \mathbb{R}^C} \; \frac{1}{N} \sum_{i=1}^{N} \; \ell(W, b, x_i, y_i) + \frac{\lambda_1}{2} \|W\|_{1,2} + \frac{\lambda_2}{2} \|b\|_2^2 \tag{4}$$

You are asked to do the following:

- Download the toy data template from the homepage
- Implement a proximal gradient descent algorithm to optimize above objective function (4) (Avoid for-loops)
- Make sure that your objective monotonically decreases. Plot the objective values. Stop your code if the difference of two successive iterates is less than 10^{-12}.

- In order to ensure that your derivative is computed correctly you may first optimize the fully differentiable model (2) with MATLABs *fminunc* with the options *'GradObj'*, *'On'* and *'DerivativeCheck'*, *'On'*. (Python: check out e.g. `scipy.optimize.grad_check`. This step is optional.)

- Iteratively compute the test error in percent, i.e. how many test samples are not classified correctly via the rule (1).

- Play around with different parameter settings for λ_1, λ_2. Can you identify the useless features? Explain why the model generalizes better to unseen test data if you use $1, 2$-norm on W^* (answer by comment at the end of your code).

- You may apply your code to the MNIST dataset `http://yann.lecun.com/exdb/mnist/` and see that your are now able to classify handwritten digits.