Labs
Machine Learning Course
Fall 2020

EPFL
School of Computer and Communication Sciences
Nicolas Flammarion & Martin Jaggi
www.epfl.ch/labs/mlo/machine-learning-cs-433

Problem Set 7, Oct 27, 2020 (SVM)

Goals. The goal of this exercise is to

- Implement and debug Support Vector Machine (SVM) using SGD and coordinate descent.
- Derive updates for the coordinate descent algorithm for the dual optimization problem for SVM.
- Implement and debug the coordinate descent algorithm.
- Compare it to the primal solution.

Setup, data and sample code. Obtain the folder `labs/ex07` of the course github repository

github.com/epfml/ML_course

We will finally depart from using the height-weight dataset and instead use a toy dataset from scikit-learn. We have provided sample code templates that already contain useful snippets of code required for this exercise.

1 Support Vector Machines using SGD

Until now we have implemented linear and logistic regression to do classification. In this exercise we will use the Support Vector Machine (SVM) for classification. As we have seen in the lecture notes, the original optimization problem for the Support Vector Machine (SVM) is given by

$$\min_{\boldsymbol{w}\in\mathbb{R}^D} \sum_{n=1}^{N} \ell(y_n \boldsymbol{x}_n^\top \boldsymbol{w}) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 \tag{1}$$

where $\ell : \mathbb{R} \to \mathbb{R}$, $\ell(z) := \max\{0, 1 - z\}$ is the *hinge loss* function. Here for any n, $1 \leq n \leq N$, the vector $\boldsymbol{x}_n \in \mathbb{R}^D$ is the n^{th} data example, and $y_n \in \{\pm 1\}$ is the corresponding label.

Problem 1 (SGD for SVM):

Implement stochastic gradient descent (SGD) for the original SVM formulation (1). That is in every iteration, pick one data example $n \in [N]$ uniformly at random, and perform an update on $\boldsymbol{w}$ based on the (sub-)gradient of the n^{th} summand of the objective (1). Then iterate by picking the next n.

1. Fill in the notebook functions `calculate_accuracy(y, X, w)` which computes the accuracy on the training/test dataset for any $\boldsymbol{w}$ and `calculate_primal_objective(y, X, w, lambda_)` which computes the total primal objective (1).
2. Derive the SGD updates for the original SVM formulation and fill in the notebook function `calculate_stochastic_gradient()` which should return the stochastic gradient of the total cost function (loss plus regularizer) with respect to $\boldsymbol{w}$. Finally, use `sgd_for_svm_demo()` provided in the template for training.

2 Support Vector Machines using Coordinate Descent

As seen in class, another approach to train SVMs is by considering the dual optimization problem given by

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^N} \ \boldsymbol{\alpha}^\top \mathbf{1} - \tfrac{1}{2\lambda}\boldsymbol{\alpha}^\top \boldsymbol{Y}\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{Y}\boldsymbol{\alpha} \qquad \text{such that} \quad 0 \le \alpha_n \le 1 \ \forall n \in [N] \tag{2}$$

where $\mathbf{1}$ is the vector of size N filled with ones, $\boldsymbol{Y} := \text{diag}(\boldsymbol{y})$, and $\boldsymbol{X} \in \mathbb{R}^{N\times D}$ again collects all N data examples as its rows, as usual. In this approach we optimize over the dual variables $\boldsymbol{\alpha}$ and map the solutions back to the primal vector $\boldsymbol{w}$ via the mapping we have seen in class: $\boldsymbol{w}(\boldsymbol{\alpha}) = \frac{1}{\lambda}\boldsymbol{X}^T\boldsymbol{Y}$.

Problem 2 (Coordinate Descent for SVM):

In this part we will derive the coordinate descent (or rather ascent in our case) algorithm seen in class to solve the dual (2) of the SVM formulation. That is, at every iteration, we pick a coordinate $n \in [N]$ uniformly at random, and fully optimize the objective (2) **with respect to that coordinate alone**. Hence one step of coordinate ascent corresponds to solving, for a given coordinate $n \in \big[N\big]$ and our current vector $\boldsymbol{\alpha} \in [0,1]^N$, the one dimensional problem:

$$\max_{\gamma\in\mathbb{R}} f(\boldsymbol{\alpha} + \gamma \boldsymbol{e}_n) \qquad \text{such that} \quad 0 \le \alpha_n + \gamma \le 1 \tag{3}$$

where $f(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2\lambda}\boldsymbol{\alpha}^\top \boldsymbol{Y}\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{Y}\boldsymbol{\alpha}$ and $\boldsymbol{e}_n = [0, \cdots, 1, \cdots, 0]^\top$ (all zero vector except at the n^{th} position). We denote by γ^* the value of γ which maximises problem 3. The coordinate update is then $\boldsymbol{\alpha}^{\text{new}} = \boldsymbol{\alpha} + \gamma^* \boldsymbol{e}_n$.

1. Solve problem 3 and give a closed form solution for the update $\boldsymbol{\alpha}^{\text{new}} = \boldsymbol{\alpha} + \gamma^* \boldsymbol{e}_n$, this update should only involve $\boldsymbol{\alpha}$, λ, $\boldsymbol{x}_n$, y_n and $\boldsymbol{w}(\alpha)$ (*Hint:* Notice that $\gamma \mapsto f(\alpha + \gamma \boldsymbol{e}_n)$ is polynomial and **don't forget the constraints !**). Convince yourself that this update can be computed in $O(D)$ time.
2. Find an efficient way of updating the corresponding $\boldsymbol{w}(\boldsymbol{\alpha}^{\text{new}})$. This should be computable in $O(D)$ time.
3. Fill in the notebook functions `calculate_coordinate_update()` which should compute the coordinate update for a single desired coordinate and `calculate_dual_objective()` which should return the objective (loss) for the dual problem (2) .
4. Finally train your model using coordinate descent (here ascent) using the given function `sgd_for_svm_demo()` in the template. Compare to your SGD implementation. Which one is faster? (Compare the training objective values (1) for the $\boldsymbol{w}$ iterates you obtain from each method). Is the gap going to 0?

Theory Excercises

Problem 3 (Kernels):

In class we have seen that many kernel functions $k(\boldsymbol{x}, \boldsymbol{x}')$ can be written as inner products $\phi(\boldsymbol{x})^\top \phi(\boldsymbol{x}')$ for a suitably chosen feature map $\phi(\cdot)$. Let us say that such a kernel function is *valid*. We further discussed many operations on valid kernel functions that result again in valid kernel functions. Here are two more.

1. Let $k_1(\boldsymbol{x}, \boldsymbol{x}')$ be a valid kernel function. Let f be a polynomial with positive coefficients. Show that $k(\boldsymbol{x}, \boldsymbol{x}') = f(k_1(\boldsymbol{x}, \boldsymbol{x}'))$ is a valid kernel.
2. Show that $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(k_1(\boldsymbol{x}, \boldsymbol{x}'))$ is a valid kernel assuming that $k_1(\boldsymbol{x}, \boldsymbol{x}')$ is a valid kernel. *Hint*: The limit of valid kernels is still a valid kernel.