APM 523
Quadratic Programming
Interior Point Methods

Matt Kinsinger, Hongjun Choi, Adarsh Akkshai

November 15, 2017

1 Interior Point Methods

The interior-point approach can be applied to a convex quadratic programs by extending the linear-programming algorithm. We can represent the standard form of a convex quadratic programs with inequality constraints as follows :

$$\min_x \quad q(x) = \frac{1}{2}x^T Gx + x^T c, \tag{1.1}$$

where G is symmetric and positive semidefinite matrix.

$$\begin{aligned} &\textit{subject to } Ax \geq b, \\ A_{m\times n} = [a_i]_{i\in\mathcal{I}}, \quad b = [b_i]_{i\in\mathcal{I}}, \quad &\mathcal{I} = 1, 2, ...m. \end{aligned} \tag{1.2}$$

From above standard form, we can rewrite the first order **KKT conditions** by adding the slack vector $y \geq 0$.

$$Gx - A^T\lambda + c = 0, \tag{1.3}$$

$$Ax - y - b = 0, \tag{1.4}$$

$$y_i\lambda_i = 0, \quad i = 1, 2, ..., m \tag{1.5}$$

$$(y, \lambda) \geq 0. \tag{1.6}$$

These KKT conditions are satisfied with not only necessary conditions but also sufficient conditions in that G is positive semidefinite. Hence we can solve the convex quadratic program (1.1)-(1.2) by solving the KKT conditions.
As seen in chapter 14, we derive path-following, primal-dual methods by considering the perturbed KKT conditions defined by

$$F(x, y, \lambda; \sigma\mu) = \begin{bmatrix} Gx - A^T\lambda + c \\ Ax - y - b \\ \mathcal{Y}\Lambda e - \sigma\mu\, e \end{bmatrix} = 0, \tag{1.7}$$

$$\mathcal{Y} = diag(y_1, y_2, ...y_n), \quad \Lambda = diag(\lambda_1, \lambda_1, ...m\lambda_m), \quad e = (1, 1, ..., 1)^T \tag{1.8}$$

where $\sigma \in [0, 1]$. By applying Newton's method to (1.7), we obtain the linear system

$$\begin{bmatrix} G & 0 & -A^T \\ A & -I & 0 \\ 0 & \Lambda & \mathcal{Y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -\Lambda\mathcal{Y}e + \sigma\mu\, e \end{bmatrix}. \tag{1.9}$$

where

$$r_d = Gx - A^T\lambda_c, \quad r_p = Ax - y - b. \tag{1.10}$$

Then we obtain the next iteration by setting

$$(x^+, y^+, \lambda^+) = (x, y, \lambda) + \alpha(\Delta x, \Delta y, \Delta\lambda) \tag{1.11}$$

where α is selected to retain the inequality $(y^+, \lambda^+) > 0$ and possible to satisfy various other conditions.

1.1 Solving the Primal-Dual System

Computing the solution of system (1.9) can be much more costly to factor than the matrix (14.9) in Chapter 14 since the computation of the Hessian matrix G give rise to non-singular matrix. So we would introduce the direct factorization algorithm by exploiting the structure (1.9), or by choosing an appropriate preconditioner for an an iterative solver.// As we mentioned in linear programming, the system (1.9) restate more condensing form. The "augmented system" is

$$\begin{bmatrix} G & -A^T \\ A & \Lambda^{-1}\mathcal{Y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p + (-y + \sigma\mu\Lambda^{-1}e) \end{bmatrix}. \tag{1.12}$$

After a simple transformation to symmetric form, a symmetric indefinite factorization scheme can be applied to the coefficient matrix in this system. Then the "normal equation" form is

$$(G + A^T\Lambda^{-1}\Delta A)\Delta x = -r_d + A^T\mathcal{Y}^{-1}\Delta[-r_p - y + \sigma\mu\Delta^{-1}e], \tag{1.13}$$

which can be solved by means of a modified Cholesky algorithm. This approach have two advantages :

1: effectiveness if the term $A^T\Lambda^{-1}\Delta A$ is not too dense compared with G

2: more smaller (1.12) if there are many inequality constraints.

1.2 Step Length

We already mentioned in linear programming that interior-point method for linear programming are more efficient if different step length α^{pri},α^{dual} are used for the primal and dual variables.In quadratic programming, the situation is different. We can define the new iterate as

$$(x^+, y^+) = (x, y) + \alpha^{pri}(\Delta x, \Delta y), \quad \lambda^+ = \lambda + \alpha^{dual}\Delta\lambda. \tag{1.14}$$

where α^{pri} and α^{dual} are step lengths that ensure the positivity of (y^+, λ^+). By using (1.9) and (1.10), we represent the new residuals relations:

$$r_p^+ = (1 - \alpha^{pri})r_p \tag{1.15}$$

$$r_d^+ = (1 - \alpha^{dual})r_d + (\alpha^{pri} - \alpha^{dual})G\Delta x. \tag{1.16}$$

In case that $\alpha^{pri} = \alpha^{dual} = \alpha$, both residuals decrease linearly for all $\alpha \in (0, 1)$, otherwise, the dual residual r_d^+ my increase for certain choices of α^{pri},α^{dual}, possibly causing divergence of the interior-point iteration.
One option is to use equal step length, and to set $\alpha = min(\alpha_\tau^{pri}, \alpha_\tau^{dual})$ such that

$$\alpha_\tau^{pri} = max\alpha \in (0, 1] : y_\alpha \Delta y \geq (1 - \tau)y, \tag{1.17}$$

$$\alpha_\tau^{dual} = max\alpha \in (0, 1] : \lambda_\alpha \Delta\lambda \geq (1 - \tau)\lambda; \tag{1.18}$$

the parameter $\tau \in (0, 1)$ controls how far we back off from the maximum step for which the conditions $y + \alpha\Delta y \geq 0$ and $\lambda + \alpha\Delta y \geq 0$ are satisfied. However, numerical experience has shown that using different step length in the primal and dual variables often leads to faster convergence.

1.3 A Practical primal-Dual Method

The most popular interior-point method for convex QP is based on Mehrotra's predictor-corrector algorithm in Chapter 14. From the predictor-corrector algorithm in linear programming, we can easily extend to quadratic programing, as we show.
First, we compute an affine scaling step $(\Delta x^{aff}, \Delta y^{aff}, \Delta\lambda^{aff})$ by setting $\sigma = 0$ in (1.9). Then, we compute centering parameter σ using following equation

$$\sigma = (\frac{\mu_{\text{aff}}}{\mu})^3. \tag{1.19}$$

The total step is obtained by solving the following system.

$$\begin{bmatrix} G & 0 & -A^T \\ A & -I & 0 \\ 0 & \Lambda & \mathcal{Y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_p \\ -\Lambda\mathcal{Y}e - \Delta\Lambda^{\text{aff}}\Delta\mathcal{Y}^{\text{aff}}e + \sigma\mu\, e \end{bmatrix}. \tag{1.20}$$

For simplicity, we will assume in our description that equal step lengths are used in the primal and dual variables though, as noted above, unequal step length can give slightly faster convergence.

1.4 Experiment for Example 16.4

We tested out algorithm on the example 16.4 and obtained the optimal solution of x^*= (1.4, 1.7), $f(x^*) = 0.8$. This is the same solution shown in the textbook. See **test_qp.py** for the python code implementing the algorithm for this example.

Algorithm 1 Predictor-Corrector Algorithm for QP

Calculate (x_0, y_0, λ_0) with $(y_0, \lambda_0) > 0$;
while k=0,1,2... **do**
  Set $(x, y, \lambda) = (x_k, y_k, \lambda_k)$ and solve (1.9) with $\sigma = 0$ for $(\Delta x^{\text{aff}}, \Delta y^{\text{aff}}, \Delta \lambda^{aff})$;
  Calculate $\mu = y^T \lambda / m$;
  Calculate $\hat{\alpha}_{aff} = max\alpha \in (0, 1] | (y, \lambda) + \alpha(\Delta y^{\text{aff}}, \Delta \lambda^{aff}) \geq 0$;
  Calculate $\mu_{aff} = (y + \hat{\alpha}^{aff} \Delta y^{aff})^T (\lambda + \hat{\alpha}^{aff} \Delta \lambda^{aff}) / m$;
  Set centering parameter to $\sigma = (\mu_{aff}/\mu)^3$
  Solve (1.20) for $(\Delta x, \Delta y, \Delta \lambda)$;
  Choose $\tau_k \in (0, 1)$ and set $\hat{\alpha} = min(\alpha_k^{pri}, \alpha_k^{dual})$ (see (1.17)-(1.18))
  Set $(x_{k+1}, y_{k+1}, \lambda_{k+1}) = (x_k, y_k, \lambda_k) + \hat{\alpha}(\Delta x, \Delta y, \Delta \lambda)$;
end(while)

2 Support Vector Machine

In machine learning, support vector machine (**SVM**) are supervised learning models with associated learning algorithm that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in spaces, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to category based on which side of the gap they fall.

2.1 Simplest Formulation

Given data points $x_1, x_2, ..., x_m$ in $\mathbb{R}^n$ with labels $y_i = \pm 1$, $i = 1, 2, ..., m$, find a vector $w \in \mathbb{R}^n$ and a scalar γ such that the hyperplane defined by $w^T x + \gamma$ separates the data. That is,

$$\begin{aligned} y_i = +1 \quad &\Rightarrow \quad w^T x_i + \gamma \geq +1, \\ y_i = -1 \quad &\Rightarrow \quad w^T x_i + \gamma \leq -1, \end{aligned} \tag{2.1}$$

Then, we can show that w with this property that minimizes $\|w\|_2$ gives the maximal margin between the data. Hence, we can formulate the problem as a QP:

$$\begin{aligned} \min_{w,\gamma} \quad & \frac{1}{2} w^T w, \\ \text{subject to} \quad & y_i(w^T x_i + \gamma) \geq 1 \\ & i = 1, 2 \dots m \end{aligned} \tag{2.2}$$

2.2 Soft-Margin Classifiers

For non-separable data sets this problem is infeasible. Therefore, we can modify by penalizing for points on the wrong side of the plane and introducing slack variables for violated constrained in which slack variables, ζ can measures the amount of the data point that we've violated the margin constraint: What we need to solve the system is the following equations:

$$\begin{aligned} \min_{w,\zeta,\gamma} \quad & \frac{1}{2} w^T w + C e^T \zeta \\ \text{s.t.} \quad & y_i(w^T x_i + \gamma) \geq 1 - \zeta_i \\ & \zeta_i \geq 0 \quad i = [1 \dots m] \end{aligned} \tag{2.3}$$

where $e = (1 \dots 1)^T$ and C is a scalar penalty parameter for violated constraints which equal to 1. If C is chosen to be large number, we'll pay attention to making sure no data violate the margin. On the other hand, C is a small number, we'll try to maximize the margin for most data, but allow some of them to violate it. An easier way is to solve the dual of this problem. The slack variable ζ_i is greater than or equal to zero. The slack is closer to 0, then approach closely "hard-margin". If the slack variable has the higher value, then the algorithm has more soft the margin. In case of slack variable equal to 0, then we'd have a typical hard-margin classifier.
We can derive the dual problem using duality theorem for quadratic programming. First, the Lagrange function with Lagrange multipliers α_n and β_n is defined as following:

$$\begin{aligned} \mathcal{L}(b, w, \zeta, \alpha, \beta) = \frac{1}{2} w^T w + C \sum_{i=1}^{m} \zeta_i \\ + \sum_{i=1}^{m} \alpha_n (1 - \zeta_i - y_i(w^T x_i + b)) + \sum_{i=1}^{m} \beta_i (-\zeta_i) \end{aligned} \tag{2.4}$$

We want Lagrange dual which can be represented by equation (2.5).

$$\begin{aligned} \max_{\alpha \geq 0, \beta \geq 0} \quad & (\min_{b,w,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{m} \zeta_i + \sum_{i=1}^{m} \alpha_i (1 - \zeta_i - y_i(w^T x_i + b)) \\ & + \sum_{i=1}^{m} \beta_i (-\zeta_i)) \end{aligned} \tag{2.5}$$

1: Take derivative w.r.t $\zeta \rightarrow \frac{\partial \mathcal{L}}{\partial \zeta} = 0 = C - \alpha_i - \beta_i$

2: no loss of optimality if solving with implicit constraint $\beta_i = C - \alpha_i$ and explicit constraint $0 \leq \alpha_i \leq C$

Then, from above the first equation, we can plug it into equation (2.5). We have

$$\max_{0 \geq \alpha_i \leq C, \beta_i = C - \alpha_i} \quad (\min_{b,w} \frac{1}{2} w^T w + \sum_{i=1}^{m} \alpha_i (1 - y_i(w^T x_i + b))) \tag{2.6}$$

1: $\frac{\partial \mathcal{L}}{\partial b} = 0$: no loss of optimality if solving with constraint $\sum_{i=1}^{m} \alpha_i y_i = 0$

2: $\frac{\partial \mathcal{L}}{\partial w_i} = 0 : w = \sum_{i=1}^{m} \alpha_i y_i x_i$

Hence, we can derive standard soft-margin SVM model by substituting above result into equation (2.6)

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{i=1}^{m} \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{m} y_i \alpha_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad \text{for} \quad i = 1, 2, ...m \end{aligned} \tag{2.7}$$

Our approach to solve C-SVM problem is two things. First, the dual problem was proposed for solving C-SVM problem by changing primal problem as above mentioned (CSVM(dual). However, we could not get right answer compared to primal problem. We omitted detailed matrix form in this section (see **CSVM_dual.py** for the Python code attempting to implement this dual method).
We now propose the primal method following section.

2.3 Primal Method for the CSVM

The challenge is to formulate the problem statement of the CSVM in the standard QP form. This is how we accomplish this:

$$\begin{aligned} \min \quad & \frac{1}{2} \begin{bmatrix} w^T & \gamma & \zeta \end{bmatrix} G \begin{bmatrix} w \\ \gamma \\ \zeta \end{bmatrix} + \begin{bmatrix} 0 & \dots & 0 & | & 0 & | & C & \dots & C \end{bmatrix} \begin{bmatrix} w \\ \gamma \\ \zeta \end{bmatrix} \\ s.t. \quad & A \begin{bmatrix} w \\ \gamma \\ \zeta \end{bmatrix} - \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ & \zeta - t = 0 \\ & s, t \geq 0 \end{aligned}$$

Where

- $w \in \mathbb{R}^n$, $\gamma \in \mathbb{R}$, ζ, t, s $\in \mathbb{R}^m$ (n = 9, m = 679)
- $G \in \mathbb{R}^{(n+1+m)\times(n+1+m)}$ contains the identity matrix I_n in its upper left section and zeros everywhere else.

- $A \in \mathbb{R}^{(2m)\times(n+1+m)}$ such that

$$A = \begin{bmatrix} y_1\left(x_1^T|1\right) & \\ y_2\left(x_2^T|1\right) & I_m \\ \vdots & \\ y_m\left(x_m^T|1\right) & \\ & \\ zero & I_m \\ & \end{bmatrix}$$

- We build the matrix K to solve the system (1.9)

$$K = \begin{bmatrix} G & 0_{(n+1+m)\times(2m)} & -A^T \\ A & -I_{2m} & 0_{2m} \\ 0_{(2m)\times(n+1+m)} & \Lambda & S \end{bmatrix}$$

where Λ = diag(λ_1, ..., λ_{2m}), contains the Lagrange multipliers and S = diag(s_1,..,s_m,t_1,..,t_m), contains the slack variables. This a huge matrix (3405×3405) and each iteration of the algorithm took 90 seconds because we had to invert this matrix. We utilized the *scipy.sparse* package to take advantage of the sparsity of the matrix K.

Solution, Codes and Data We ran our algortihm for 20 iterations and it appeared that it had converged to a solution (csvm_data_iterations.txt shows the convergence of the objective function). Our solution was:

$$w^* = \begin{bmatrix} 0.23536287 \\ -0.02280066 \\ 0.17137626 \\ 0.11211598 \\ 0.09457784 \\ 0.17768162 \\ 0.18014265 \\ 0.09104447 \\ 0.18103805 \end{bmatrix} \tag{2.8}$$

and

$$\gamma = -4.27453685. \tag{2.9}$$

We tested our solution vector w^* on on a collection of the data vectors that were used by the algorithm. The relations

$$y_i = +1 \Rightarrow w^T x_i + \gamma \geq +1,$$

$$y_i = -1 \Rightarrow w^T x_i + \gamma \leq -1,$$

held for a majority (maybe 8 out of 10) of the data vectors x_i. In the original problem statement it was said the a CSVM problem is infeasible,

hence maybe our solution is correct even though the separation relations did not hold true for every data vector.

Python Codes and Data Files for Primal Method

- **csvm_primal.py** : The code for solving the primal QP version of the CSVM problem.
- **data_vectors.txt**, **length_data.txt**, **num_vectors.txt** : Data and parameters used by csvm_primal.py to solve the CSVM problem
- **csvm_data_iterations.txt**, **csvm_data_output.txt** : The output of the algorithm. The first shows the convergence of the algorithm to an objective value for each iteration, the second in the solution of the problem. It contains $w \in \mathbb{R}^9$, $\gamma \in \mathbb{R}$, and $\zeta \in \mathbb{R}^{679}$.