Assignment 4

Robotics 811, Fall 2015

DUE: Thursday, October 29, 2015

1. Consider the following differential equation over the interval $[0, 1]$:

$$\frac{dy}{dx} = \frac{2}{x^2(1-y)}, \qquad \text{with } y(1) = -1.$$

   (a) Obtain an exact analytic solution $y(x)$ to the differential equation.

   (b) Implement and use Euler's method to solve the differential equation numerically. Use a step size of 0.05. How accurate is your numerical solution? (Compare the numerical solution to the exact solution.)

   (c) Implement and use a fourth-order Runge-Kutta method to solve the differential equation numerically. Again, use a step size of 0.05. Again, how accurate is your numerical solution?

   (d) Finally, implement and use fourth-order Adams-Bashforth for the differential equation. Again, use a step size of 0.05. Initialize the iteration with the following four values:

$$\begin{aligned} y(1.15) &= -0.865009616, \\ y(1.10) &= -0.906925178, \\ y(1.05) &= -0.951800146, \\ y(1) &= -1. \end{aligned}$$

   Once again, how accurate is your numerical solution?

2. Consider the function $f(x, y) = x^3 + y^3 - 2x^2 + 3y^2 - 8$.

   (a) Find all critical points of f by sketching in the (x, y) plane the iso-contours $\frac{\partial f}{\partial x} = 0$ and $\frac{\partial f}{\partial y} = 0$. Then classify the critical points into local minima, local maxima, and saddle points by considering nearby gradient directions.

   (b) Show how steepest descent would behave starting from the point $(x, y) = (1, -1)$. How many steepest descent steps are needed to converge to a local minimum?

3. Let Q be a real symmetric positive definite $n \times n$ matrix.

   (a) Show that any two eigenvectors of Q corresponding to *distinct* eigenvalues of Q are Q-orthogonal. Show this directly from the definition of eigenvector.

   (b) You will now enhance the argument from part (a) to establish Q-orthogonality more generally:

   It is well known that the eigenvectors of Q can be chosen to be orthogonal in the usual sense. Using this fact, show that *any* two such eigenvectors of Q are in fact also Q-orthogonal.

4. (a) Show that in the purely quadratic form of the conjugate gradient method, $d_k^T Q d_k = -d_k^T Q g_k$. Using this show that to obtain x_{k+1} from x_k it is necessary to use Q only to evaluate g_k and Qg_k.

   (See the notes for notation:

   "Purely quadratic" means the conjugate gradient method applied to functions of the form $f(x) = c + b^T x + \frac{1}{2} x^T Q x$, where Q is a real symmetric positive definite $n \times n$ matrix.

   g_k is the gradient of $f(x)$ evaluated at x_k, and d_k is the descent direction leading from x_k to x_{k+1}.)

   (b) Show that in the purely quadratic form Qg_k can be evaluated by taking a unit step from x_k in the direction of the negative gradient and then evaluating the gradient there. Specifically, if $y_k = x_k - g_k$ and $p_k = \nabla f(y_k)$, then $Qg_k = g_k - p_k$.

   (c) Combine the results of parts (a) and (b) to derive a conjugate gradient method for general functions f much in the spirit of the algorithm presented in class, but which does not require knowledge of the Hessian of f or a line search. (Recall that a line search is used to find the minimum of f along a particular direction.)

5. In searching for the minimum of $f(x) = c + b^T x + \frac{1}{2} x^T Q x$, along the n Q-orthogonal directions $\{d_0, d_1, \ldots, d_{n-1}\}$, does the order of the directions matter? Is it necessary to search the same direction more than once? Explain your answers.

   (Assume that each line search locates its minimum exactly, and that Q is real symmetric positive definite.)

6. Find the rectangle of a given perimeter that has the greatest area, by solving the Lagrange multiplier first-order necessary conditions. Verify the second-order sufficiency conditions.

7. Please turn to page 3 for a problem on Support Vector Machines.

   One purpose of this problem is to show you a common way for converting one type of optimization problem into another, using Lagrange multipliers. Another purpose is to give you some tools for solving (part of) a real-world problem posed by the Post Office: recognizing handwritten zip code digits.

   Do not be alarmed by the length of the problem. The reason the problem appears long is that there is much guidance taking you through the math to the application.

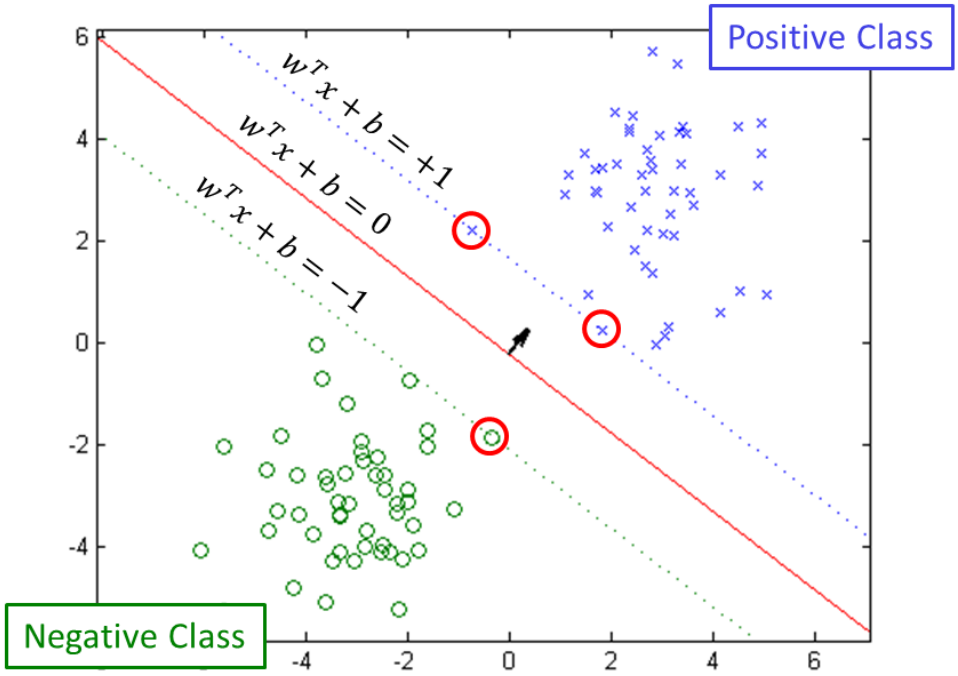


Figure 1: Figure depicting an SVM example

In this problem, you will explore how Lagrange multipliers are used to convert between the primal and dual formulations of the Support Vector Machine (SVM) algorithm. SVMs are a very common classification algorithm used in the machine learning field. The name is derived from the fact that from a whole data set, only a select few points (the "support vectors") are used to determine the decision boundary.

Suppose we have a set of l n-D points, $\{\mathbf{x}_1, \ldots, \mathbf{x}_l \in \mathbb{R}^n\}$. Suppose these points are split into two different classes and that we have the class labels for each of these points: $y_i \in \{-1, +1\}, i = 1, \ldots, l$. We want to draw a decision boundary between these points that can separate the -1 class from the $+1$ class. In particular, we want to draw the boundary such that the margin between the two classes is as large as possible.

You can see an example of this in Figure 1. Here, the blue x's and the green o's represent the positive and negative classes respectively. The decision boundary (red solid line) is represented by $\mathbf{w}^T\mathbf{x} + b = 0$, and the black vector shows the direction and magnitude of $\mathbf{w}$. The margin is represented by the blue and green dotted lines, which correspond to $\mathbf{w}^T\mathbf{x} + b = +1$ and $\mathbf{w}^T\mathbf{x} + b = -1$ respectively. Here, 1 is chosen because it's a convenient number. The support vectors (circled in red) are the points that lie exactly on the margin.

Note that the distance between two hyperplanes of the form $\mathbf{w}^T\mathbf{x} + b = c_1$ and $\mathbf{w}^T\mathbf{x} + b = c_2$ is given by $D = |c_1 - c_2|/\|w\|$. Therefore, we can see that in order to correctly classify our points and maximize our margin, we need to solve the following constrained optimization problem:

$$\begin{aligned} &\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad &\mathbf{w}^T\mathbf{x}_i + b \geq +1, \quad \text{if } y_i = +1 \\ &\mathbf{w}^T\mathbf{x}_i + b \leq -1, \quad \text{if } y_i = -1 \end{aligned}$$

In the case where our data is not linearly separable, we may want to allow our algorithm to make some mistakes during training time. That is, we want to allow $\mathbf{w}^T\mathbf{x}_i + b$ to be a little less than $+1$ for some of the positive training points or a little greater than -1 for some of the negative ones. However, we want to

minimize the total error of our algorithm during training. This results in the following problem formulation:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i \tag{1}$$

$$\text{s.t.} \quad \mathbf{w}^T\mathbf{x}_i + b \geq +1 - \xi_i, \quad \text{if } y_i = +1 \tag{2}$$

$$\mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i, \quad \text{if } y_i = -1 \tag{3}$$

$$\xi_i \geq 0, \quad \forall i \tag{4}$$

Here, the ξ_i's represent the error we make on a given sample. C represents the trade off between making fewer mistakes and creating a larger margin.

In class, you have learned that the following problem:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{s.t.} \quad h_i(\mathbf{x}) = 0, \quad i = 1, \ldots, l \end{aligned}$$

can be solved by first setting up the Lagrangian

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^{l}\lambda_i h_i(\mathbf{x}),$$

then taking partial derivatives, and finally setting them to zero. What you are actually doing here is maximizing the Lagrangian with respect to λ, and then minimizing with respect to $\mathbf{x}$. That is,

$$\min_{\mathbf{x}} \max_{\boldsymbol{\lambda}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}).$$

It turns out that when you have inequality constraints:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{s.t.} \quad g_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, l \end{aligned}$$

the generalized Lagrangian takes the form

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}) = f(\mathbf{x}) + \sum_{i=1}^{l}\alpha_i g_i(\mathbf{x})$$

and the optimization problem becomes

$$\begin{aligned} &\min_{\mathbf{x}} \max_{\boldsymbol{\alpha}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}) \\ &\text{s.t.} \quad \alpha_i \geq 0 \end{aligned}$$

Note the extra constraint on the Lagrange multipliers, α_i. We can write this more compactly like so: $\min_{\mathbf{x}} \max_{\boldsymbol{\alpha}:\alpha_i \geq 0} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha})$

Furthermore, if you have multiple inequality constraints,

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{s.t.} \quad g_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, l \\ &\qquad h_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, m \end{aligned}$$

you can simply tack them onto the Lagrangian:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) + \sum_{i=1}^{l}\alpha_i g_i(\mathbf{x}) + \sum_{i=1}^{m}\beta_i h_i(\mathbf{x})$$

and the new optimization problem is

$$\min_{\mathbf{x}} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$
$$\text{s.t.} \quad \alpha_i \geq 0$$
$$\beta_i \geq 0$$

(a) Convert the inequality constraints 2 and 3 found in the SVM optimization problem into an inequality constraint of the form $g_i\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T\right) \leq 0$ that does not involve "if" conditions. (Hint: use the value of the class labels, y_i)

(b) Write down the Lagrangian for our optimization problem $\mathcal{L}\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T, \boldsymbol{\alpha}, \boldsymbol{\beta}\right)$.

(c) Recall from above that solving the SVM formulation directly as described above is equivalent to solving the following optimization problem:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\geq 0,\beta_i\geq 0} \mathcal{L}\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T, \boldsymbol{\alpha}, \boldsymbol{\beta}\right).$$

This is called the primal form of the SVM. The dual optimization problem is as follows

$$\max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\geq 0,\beta_i\geq 0} \min_{\mathbf{w},b,\boldsymbol{\xi}} \mathcal{L}\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T, \boldsymbol{\alpha}, \boldsymbol{\beta}\right)$$

and solving it turns out to be equivalent to solving the primal.[1]

We want to solve the inner minimization term analytically, and then maximize over the new function numerically. To minimize the Lagrangian, set the partial derivatives with respect to $\mathbf{w}$, b and ξ_i to zero. One of these equations will give you an expression for $\mathbf{w}$. Plug that into the Lagrangian and simplify. The other two equations will help cancel things out. You should end up with a new Lagrangian, $\mathcal{L}^*$ in terms of only α_i's, y_i's and $\mathbf{x}_i$'s:

$$\mathcal{L}^*(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$

and the constraints

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$
$$0 \leq \alpha_i \leq C, \quad \forall i$$

Show your work.

(d) You will now implement your own SVM solver. Recall that we are trying to solve

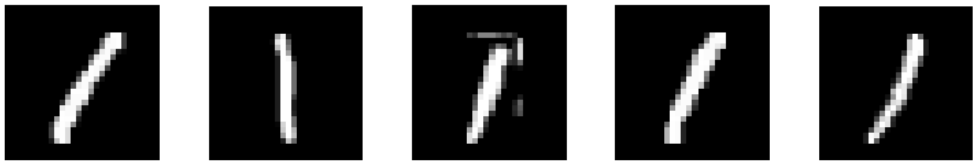
$$\begin{aligned}\max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\geq 0,\beta_i\geq 0} \min_{\mathbf{w},b,\boldsymbol{\xi}} \mathcal{L}\left(\begin{bmatrix}\mathbf{w} & b & \boldsymbol{\xi}\end{bmatrix}^T, \boldsymbol{\alpha}, \boldsymbol{\beta}\right) &= \max_{\boldsymbol{\alpha}:0\leq\alpha_i\leq C} \mathcal{L}^*(\boldsymbol{\alpha}) \\ &= \max_{\boldsymbol{\alpha}:0\leq\alpha_i\leq C} \left(\sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j\right)\end{aligned}$$

subject to $\sum_i \alpha_i y_i = 0$. This is a quadratic programming problem and you can solve it using the MATLAB `quadprog` function. Before you do this, you need to get our optimization problem into the right form:

Therefore, find $\mathbf{H}$ and $\mathbf{f}$ such that $\mathcal{L}^*(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{H}\boldsymbol{\alpha} + \mathbf{f}^T\boldsymbol{\alpha}$.

[1] The two problems are not always equivalent, but they are equivalent in our case because the cost functions and the constraints are convex. You don't need to worry about why that is for this problem.

(a) Sample zeros



(b) Sample ones

Figure 2: Sample images from handwriting dataset

(e) Now implement your own SVM solver. It should take n-D data, corresponding class labels, and C as input. It should output $\mathbf{w}$ and b.

Some hints:

- In part (c), you derived an expression for $\mathbf{w}$ given $\boldsymbol{\alpha}$. The expression for b is a little trickier. Recall that the support vectors are those points that lie exactly on the boundary. It turns out that the support vectors are exactly those for which the corresponding α is greater than 0 and less than C.[2] Therefore, for a support vector,

$$y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) = 1$$
$$\implies b = \frac{1}{y_i} - \mathbf{w}^T\mathbf{x}_i$$

  In practice, it is better to average over all support vectors. Also, in your program, use $0 + \varepsilon < \alpha < C - \varepsilon$ where ε is some small number like 10^{-5}. Machine precision makes it so that zero is never really zero...
- Our current formulation is a maximization problem, but the MATLAB `quadprog` function solves a minimization problem. You need to slightly modify our problem to convert it into a minimization problem.
- Don't forget to pass the constraints on α into the solver.
- If you are using the MATLAB `quadprog` function, you need to specify the right algorithm. Call `quadprog` with this syntax:
  `quadprog(H,f,A,b,Aeq,beq,lb,ub,[], optimset('Algorithm', 'interior-point-convex', 'Display', 'off'))`.)

(i) 2d data is provided in two separate text files - one each for the positive (`2d_positive_class.txt`) and negative classes (`2d_negative_class.txt`). Implement your SVM solver using $C = 0.01, 0.1, 1$. Plot the data (using different symbols for the positive and negative classes), as well as your decision boundary $\{\mathbf{x} : \mathbf{w}^T\mathbf{x} + b = 0\}$, the positive margin $\{\mathbf{x} : \mathbf{w}^T\mathbf{x} + b = +1\}$ and the negative margin $\{\mathbf{x} : \mathbf{w}^T\mathbf{x} + b = -1\}$ for each value of C.

(ii) What happens to the margin as C decreases? What do you expect to happen as $C \to 0$? Use Eq. 1 and/or look at the dual cost function carefully to help with your reasoning.

[2] This is due to the "complementary slackness" condition from the set of Karush-Kuhn-Tucker conditions (generalized version of the first-order necessary conditions you learned in class).

(iii) We have supplied you with a dataset consisting of 28x28 pixel images of handwritten digits that are converted into vectors of intensity by concatenating the columns of the images, resulting in a 784 dimensional vector. The data is split into two separate files, one for the ones (`handwriting_positive_class.txt`) and one for the zeros (`handwriting_negative_class.txt`). Each row in each file corresponds to a single image. Figure 2 shows a few example images from the dataset. We have supplied you with a MATLAB function (`displayImage.m`) that takes as input one of the image vectors and displays it for you. This data was taken from the MNIST dataset [3].

Now use your SVM solver on the handwriting dataset. You algorithm will learn to classify between the zeros and ones. Split your data into 10 parts (MATLAB function `crossvalind` is helpful for this). For each part, train on the other nine and use the tenth for testing. This is called 10-fold cross validation. Use $C = 1$.[4] Report your average accuracy.

(iv) Visualize one of the $\mathbf{w}$'s you learned in part (iii) as an image (you can use the provided `displayImage.m` function). Comment on what your algorithm learned, and what it thinks is important.

[3] Data taken from `http://yann.lecun.com/exdb/mnist/`. Data converted into a more accessible format with help from `http://ufldl.stanford.edu/wiki/index.php/Using_the_MNIST_Dataset`

[4] In practice, you should optimize over C by using cross-validation.