

Project 2 – Binary Classification with SVM

All characters and legal bodies appearing in this scenario are fictitious. Any resemblance to real persons, living or dead, or to existing corporations, is purely coincidental. (Having said that, I encourage attempts to draw parallels to existing scientific results)

1 Preamble

The portable X-ray scanner you helped to invent has made a name for itself in the security field, and now OptoCorp Inc. is looking to expand its market. One obvious application is to use it for field surgery, but acquiring license for medical uses is a long and painstaking process. As part of this process, the scanner has been sent to several field hospitals, and the field surgeons ranked the images produced by the scanner according to their quality and usefulness. Due to time constraints during the operations, they only ranked images that were either really bad (rank 0) or really good (rank 9), with handwritten digits. It is your task to design and implement an automatic classification system to separate those two classes.

2 More details

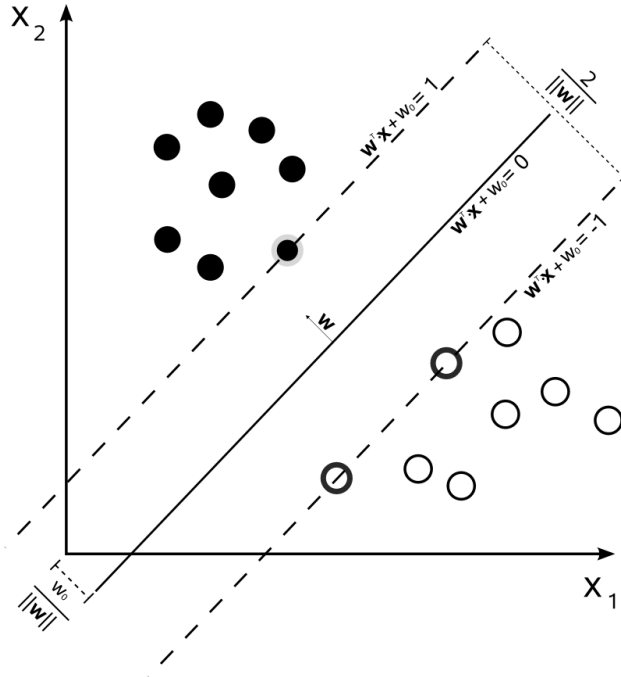
A **classification** task consist of deciding to which class (out of possibly many classes) an object belongs. When there are only two classes, it is known as **binary classification**. An object is represented by a feature vector $\mathbf{x} \in \mathbb{R}^n$. A **linear classifier** is a linear function of \mathbf{x} that separates the feature space into two classes:

$$c(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

- If $c(\mathbf{x}) < 0$, the object represented by \mathbf{x} belongs to the class w_1 labeled by -1 .
- If $c(\mathbf{x}) > 0$, the object represented by \mathbf{x} belongs to the class w_2 labeled by 1 .

The goal of linear classification is to find the parameters \mathbf{w} and w_0 of the separating hyperplane $c(\mathbf{x}) = 0$, such that the classification will be correct for any feature vector \mathbf{x} . Such a classifier is not unique, which allows us some freedom in building it. The true power of the classifier is only uncovered when we try to use it on never before seen data. Taking that into account, we would like to leave plenty of room for new data from both sides of the separating hyperplane. A plausible criterion for a good classifier is maximization of the margin between the separating hyperplane and the nearest data points from both sides (see Figure 1).

Figure 1: An example of a linear classifier in \mathbb{R}^2 . The two objects are represented by two features labeled by $\mathbf{x} = [x_1 \ x_2]$ (for example, suppose we want to decide if some animal is an elephant or a giraffe, we could use $x_1 = \text{Height}$ and $x_2 = \text{Weight}$). The figure shows the maximum margin classifier between the two classes in feature space. The points lying on the margin from both sides are the **support vectors**. A linear combination of them gives \mathbf{w} .



3 Problem statement

As an appetizer, we will assume that the two classes are separable (Figure 1). We start with a set of N feature vectors $\{x_i\}_{i=1}^N$ labeled by $\{y_i\}_{i=1}^N$, such that $y_i = -1$ if $x_i \in w_1$ and $y_i = 1$ if $x_i \in w_2$. Consider a hyperplane given by

$$c(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0 \quad (1)$$

✎ 1. Show that the distance from a point \mathbf{x} to the hyperplane defined by (1) is given by

$$d(\mathbf{x}) = \frac{|c(\mathbf{x})|}{\|\mathbf{w}\|}$$

(unless otherwise stated , $\|\cdot\|$ is the l_2 norm)

Instruction: define a point \mathbf{x}_0 on the hyperplane...

Since a hyperplane is defined up to the scale of \mathbf{w} , we can arbitrarily scale it such that $|c(\mathbf{x})| = 1$ at the nearest data points in both classes. Therefore, the margin is the sum of

the distances from the hyperplane to the nearest data points $\mathbf{x}_1 \in w_1, \mathbf{x}_2 \in w_2$ in each class :

$$d(\mathbf{x}_1) + d(\mathbf{x}_2) = \frac{1}{\|\mathbf{w}\|} + \frac{1}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|} \quad (2)$$

In order to maximize (2), we need to minimize the denominator $\|\mathbf{w}\|$, or rather minimize

$$f(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2$$

(the 1/2 factor is for convenience).

We also need to make sure that the given data points $\{x_i\}_{i=1}^N$ are classified correctly. We can write these constraints using the labels $\{y_i\}_{i=1}^N$ as follows:

$$y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1, \quad i = 1, 2, \dots, N$$

What we got is the following optimization problem:

3.1 Primal form SVM

$$\begin{aligned} \min_{\mathbf{w} \in \mathbb{R}^n, w_0 \in \mathbb{R}} \quad & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s.t} \end{aligned} \quad (3a)$$

$$y_i(\mathbf{w}^T x_i + w_0) \geq 1, \quad i = 1, 2, \dots, N \quad (3b)$$

✎ 2. Explain what makes problem (3) a convex optimization problem.

✎ 3. Show that the necessary and sufficient optimality conditions for problem (3) are given by:

$$y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1, \quad i = 1, 2, \dots, N \quad (4a)$$

$$\lambda_i \geq 0, \quad i = 1, 2, \dots, N \quad (4b)$$

$$\lambda_i(y_i(\mathbf{w}^T \mathbf{x}_i + w_0) - 1) = 0, \quad i = 1, 2, \dots, N \quad (4c)$$

$$\mathbf{w} = \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i \quad (4d)$$

$$\sum_{i=1}^N \lambda_i y_i = 0 \quad (4e)$$

By (4c) and (4d), \mathbf{w} is given as a linear combination of feature vectors \mathbf{x}_i that correspond to active constraints $c(\mathbf{x}_i) = \pm 1$ (why?). These vectors are known as **support vectors**, and they give this method its name: **Support Vector Machine (SVM)**.

3.2 Dual form SVM

The primal form SVM (3) has the annoying inequality constraints that makes its solution unpleasant. We will now develop the dual problem, and see that it has some nice appealing properties.

✎ 4. Show that the Lagrange dual of problem (3) is given by:

$$\max_{\boldsymbol{\lambda}} \quad h(\boldsymbol{\lambda}) \equiv \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \quad (5a)$$

s.t

$$\sum_{i=1}^N \lambda_i y_i = 0 \quad (5b)$$

$$\lambda_i \geq 0, \quad i = 1, 2, \dots, N \quad (5c)$$

Compared to the **linear inequality constraints** in problem (3), problem (5) has an **equality constraint** and a simple **bound constraint**, which are easier to handle. Another very interesting feature of the dual problem is that the feature vectors \mathbf{x}_i enter the problem only through **inner products**. This makes the problem independent of the dimensions of \mathbf{x} , requiring only that we have an efficient way to compute inner products. This property plays a key role in the construction of non-linear classifiers with SVM (more on that later).

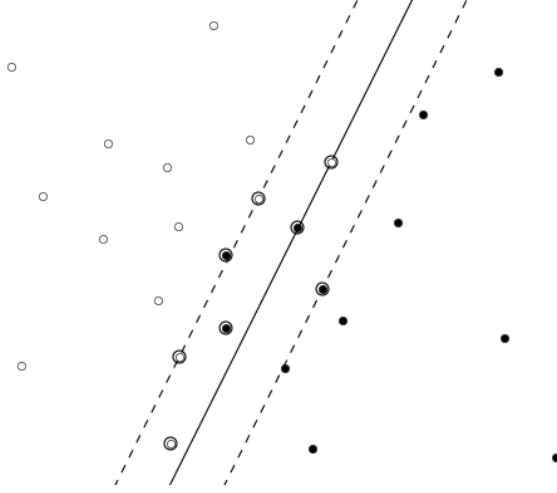
✎ 5. Suppose you have the solution $\boldsymbol{\lambda}^*$ of the dual problem. What is the connection between $h(\boldsymbol{\lambda}^*)$ and $f(\mathbf{w}^*)$? Find \mathbf{w}^*, w_0^* in terms of $\boldsymbol{\lambda}^*$

4 The non-separable case

The assumption that the two classes are separable is too restricting. What if there is no way to construct a separating hyperplane between the two classes (Figure 2)? Fortunately, it's possible to generalize problem (3) to handle the non-separable case, and still stay within the simple model of a linear classifier. We need to distinguish between three cases:

- Vectors that fall in the right class and satisfy the constraints: $y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1$
- Vectors that fall in the right class, but inside the band $0 \leq y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \leq 1$
- Vectors that fall in the wrong class $y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \leq 0$

Figure 2: The non separable case. Points can now be found also within the margin, and on the wrong side of the separating hyperplane. The goal is to make the number of these points as small as possible.



All three cases can be treated under a single type of constraint by introducing a new set of **slack variables** $\{\xi_i\}_{i=1}^N$:

$$\begin{aligned} y_i(\mathbf{w}^T \mathbf{x}_i + w_0) &\geq 1 - \xi_i \\ 0 &\leq \xi_i \leq 1 \end{aligned} \quad (6)$$

The goal is to keep the number of points with $\xi_i > 0$ minimal. Using the following **penalty function** :

$$I(\xi_i) = \begin{cases} 1 & \xi_i > 0 \\ 0 & \xi_i = 0 \end{cases} \quad (7)$$

this can be written as the following (discontinuous) optimization problem

$$\begin{aligned} \min_{\mathbf{w}, w_0, \xi} \quad & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N I(\xi_i) \\ \text{s.t} \quad & \end{aligned} \quad (8a)$$

$$y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1 - \xi_i \quad i = 1, 2, \dots, N \quad (8b)$$

$$\xi_i \geq 0 \quad i = 1, 2, \dots, N \quad (8c)$$

Instead of using the discontinuous penalty functions $I(\xi_i)$, we can approximate it with a **linear penalty function**:

$$\min_{\mathbf{w}, w_0, \xi} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i \quad (9a)$$

s.t

$$y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1 - \xi_i \quad i = 1, 2, \dots, N \quad (9b)$$

$$\xi_i \geq 0 \quad i = 1, 2, \dots, N \quad (9c)$$

Where C is a positive constant that controls the relative influence of the penalty. The fact that we chose to use a linear penalty function has a great effect on the dual of (9), as we shall see next:

✚ 6. Find the necessary optimality conditions for problem (9). Are they sufficient? Which points participate in the construction of \mathbf{w} now? Which points contribute the most?

✚ 7. Show that the dual problem of (9) is given by :

$$\max_{\boldsymbol{\lambda}} \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \quad (10a)$$

s.t

$$\sum_{i=1}^N \lambda_i y_i = 0 \quad (10b)$$

$$0 \leq \lambda_i \leq C, \quad i = 1, 2, \dots, N \quad (10c)$$

Notice how problem (10) resembles problem (5), with the only difference being the upper bound on $\boldsymbol{\lambda}$, which is due to our particular choice of penalty function. This formulation is also known as **soft margin SVM**.

✚ 8. Find the connection between $\boldsymbol{\lambda}^*$, the solution of the dual problem (10), and \mathbf{w}^*, w_0^* , the solution of the primal problem (9).

4.1 Unconstrained non-smooth formulation of SVM

The SVM problem can also be written as an unconstrained problem, using a non-smooth penalty function called **hinge loss**

$$l(z, y) = \max(0, 1 - y \cdot z) \quad (11)$$

with $z = \mathbf{w}^T \mathbf{x} + w_0$ and y being the label $\{-1, 1\}$. When z, y have the same sign, and $|z| \geq 1$, the penalty is zero. When they have opposite signs (misclassification) or $|z| < 1$ (points inside the margin), the penalty increases linearly.

9. Write (9) as an unconstrained problem using the hinge loss. Compute the sub-gradient of the objective you got.

In the next section, we will explore methods to solve the constrained version.

5 Optimization problems with equality constraints and simple bound constraints

We will explore a method to solve problem (10) based on the **method of augmented Lagrangian** (A.K.A the **method of multipliers**).

Reminder : For the general non-linear optimization problem (NLP) with equality constraints:

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad (12a)$$

s.t

$$h_i(\mathbf{x}) = 0, \quad i = 1, 2, \dots, M \quad (12b)$$

the augmented Lagrangian is given by:

$$\tilde{L}_\mu(\mathbf{x}, \boldsymbol{\lambda}) = L(\mathbf{x}, \boldsymbol{\lambda}) + \frac{\mu}{2} \sum_{i=1}^M \|h_i(\mathbf{x})\|^2$$

Where $L(\mathbf{x}, \boldsymbol{\lambda})$ is the Lagrangian for problem (12).

5.1 Partial elimination of constraints

The augmented Lagrangian technique is meant to handle equality constraints. It's possible to incorporate inequality constraints into the augmented Lagrangian as well with the aid of a barrier function or other tricks, but since the inequality constraints in problem (10) are simple bound constraints (also called box constraints), it's possible to handle them explicitly and elegantly through the mechanism of projection introduced next. For the rest of the section, we shall focus on optimization problems of the form:

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{s.t. } \mathbf{x} \in \mathbf{X} \quad (13)$$

Where \mathbf{X} is some convex set. For example, \mathbf{X} can be a box : $\{\mathbf{x} : \mathbf{a} \leq \mathbf{x} \leq \mathbf{b}\}$.

Definition 1 (Projection on a Convex Set). *Let $\mathbf{z} \in \mathbb{R}^n$. The projection of \mathbf{z} on the convex set \mathbf{X} is the unique minimizer of the following optimization problem:*

$$\begin{aligned} \mathcal{P}_{\mathbf{X}}(\mathbf{z}) = \operatorname{argmin}_{\mathbf{x}} \|\mathbf{z} - \mathbf{x}\|^2 \\ \text{s.t. } \mathbf{x} \in \mathbf{X} \end{aligned}$$

i.e., $\mathcal{P}_X(\mathbf{z})$ gives back the closest point to \mathbf{z} (in the Euclidean sense) that belong to the set X .

✎ **10.** Find the projection of $x \in \mathbb{R}$ on the convex set $[a, b]$. Divide into three cases :

- $x \leq a$
- $a < x < b$
- $x \geq b$

Using the *projection operator* we can modify the gradient descent algorithm to handle simple constraints of the form $\mathbf{x} \in X$:

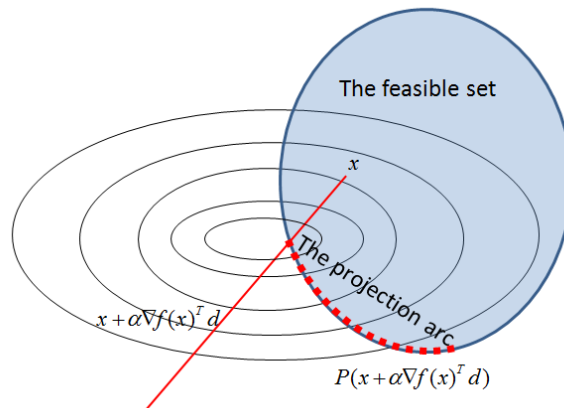
```


input : function  $f$ ; initial point  $\mathbf{x}_0$ 
output: (approximate) local minimizer  $\mathbf{x}^*$  of  $f$ 
Start with an initial guess  $\mathbf{x}_0$ 
for  $k = 1, 2, \dots$ , until convergence do
    Find descent direction  $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$ 
     $\alpha_k = \text{ArmijoRule}(@f, \mathbf{x}_k, f(\mathbf{x}_k), \nabla f(\mathbf{x}_k), \mathbf{d}_k, \sigma, \beta, \alpha_0, \text{Flag})$ 
    Update  $\mathbf{x}_{k+1} = \mathcal{P}_X(\mathbf{x}_k + \alpha_k \mathbf{d}_k)$ 
end
Return  $\mathbf{x}^* = \mathbf{x}^k$ 

```

Algorithm 1: Projected gradient descent algorithm

Figure 3: Armijo rule along the projection arc. If the step size was too large and took us out of the feasible region, the projection will take us back in. The backtracking is performed along the projection arc.



 **11. Armijo rule (backtracking)** - Implement inexact line search using the Armijo rule. Build a function `ArmijoRule` that accepts $@f, \mathbf{x}_k, \nabla f(\mathbf{x}_k), f(\mathbf{x}_k), \mathbf{d}_k, \sigma, \beta, \alpha_0, Flag$, (where $@f$ is the function handle, \mathbf{d}_k is the search direction at iteration number k , and σ, β are the parameters for the backtracking, $Flag$ defined below) and any other parameter you need, and performs backtracking until :

$$f(\mathbf{x}(\alpha)) - f(\mathbf{x}(0)) \leq \sigma \nabla f(\mathbf{x}(0))^T (\mathbf{x}(\alpha) - \mathbf{x}(0))$$

Define a flag $Flag$ to handle both cases:

- Backtracking along a straight line (for the unconstrained case):

$$\mathbf{x}(\alpha) = \mathbf{x}_k + \alpha \mathbf{d}_k$$

- Backtracking along the projection arc (for the constrained case. See Fig. 3):

$$\mathbf{x}(\alpha) = \mathcal{P}_{\mathbf{x}}(\mathbf{x}_k + \alpha \mathbf{d}_k)$$

(Notice that in this case, \mathbf{d}_k does not necessarily parallel to $\mathbf{x}(\alpha) - \mathbf{x}(0)$)

 **12. Stopping criterion** - For the following optimization problem with simple bound constraints :

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad s.t \quad \mathbf{a} \leq \mathbf{x} \leq \mathbf{b} \quad (14)$$

Show that the optimal point \mathbf{x}^* satisfies:

- $(\nabla f(\mathbf{x}^*))_i = 0$ if $a_i < x_i^* < b_i$
- $(\nabla f(\mathbf{x}^*))_i \leq 0$ if $x_i^* = b_i$
- $(\nabla f(\mathbf{x}^*))_i \geq 0$ if $x_i^* = a_i$

As in the unconstrained case, you may wonder if we can perform some preconditioning (i.e., scale the gradient, preferably with a non-diagonal PD matrix as in Newton's method) such that the convergence rate will improve. We'll see that without being careful, this is bound to fail.

 **13.** Consider the problem

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^2} f(\mathbf{x}) &= \min_{\mathbf{x}} \frac{1}{2} \left\| \mathbf{x} - \begin{bmatrix} -1 \\ 0.5 \end{bmatrix} \right\|^2 \\ s.t \quad & 0 \leq \mathbf{x} \leq 1 \end{aligned} \quad (15)$$

- Solve the problem analytically.

- Show that when starting at $\mathbf{x}_0 = [0 \ 0]^T$, and using $\mathbf{D} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \succ 0$

$$\mathbf{x}(\alpha) = \mathcal{P}_{\mathbf{X}}(\mathbf{x}_0 - \alpha \mathbf{D} \nabla f(\mathbf{x}_0)) = \mathbf{x}_0 \quad \forall \alpha > 0$$

i.e., naively scaling with a positive definite matrix may not produce a descent direction.

The key for mending this lies in the following observation:

If we knew the set of active constraints at \mathbf{x}^* , (variables that attain the minimum at the boundary of the constraint set), we could eliminate those variables from the problem, and optimize only on the remaining variables that attain the minimum in the interior of the feasible region. This practically renders the problem unconstrained for the latter variables, and Newton-like methods can be applied on the reduced problem. Obviously we cannot identify the active set before solving the problem, but it's easy to check if at the current iterate some of the variables belong to the active set or not according to the optimality conditions. Turns out that if we choose the scaling matrix in an appropriate way, the active set is found in a finite number of iterations.

Definition 2 (Diagonal matrix with respect to a subset of indices). *We say that a matrix D is diagonal with respect to a subset of indices $I \subset \{1, 2, \dots, n\}$ if*

$$d_{i,j} = 0 \quad \forall i \in I, \quad j = 1, 2, \dots, n \quad j \neq i$$

Example : if $I = \{r+1, \dots, n\}$ then the following matrix is diagonal w.r.t I

$$\mathbf{D} = \left[\begin{array}{c|ccc} \bar{\mathbf{D}} & & & \\ \hline & \mathbf{0} & & \\ & d_{r+1} & \dots & 0 \\ \mathbf{0} & \vdots & \ddots & \vdots \\ & 0 & \dots & d_n \end{array} \right]$$

Denoting the set of active constraints at the point x by

$$I^+(x) = \left\{ i \mid x_i = a_i, \frac{\partial(f(x))}{\partial x_i} > 0 \quad \text{or} \quad x_i = b_i, \frac{\partial(f(x))}{\partial x_i} < 0 \right\}$$

Then if \mathbf{D} is a positive definite symmetric matrix and diagonal w.r.t $I^+(\mathbf{x})$, it can be used to scale the gradient s.t

$$\exists \alpha > 0 \quad \text{s.t} \quad f(\mathbf{x}(\alpha)) < f(\mathbf{x}(0))$$

with

$$\mathbf{x}(\alpha) = \mathcal{P}_{\mathbf{X}}(\mathbf{x}_k - \alpha \mathbf{D} \nabla f(\mathbf{x}_k))$$

(you can try proving that on your own, or look at [1]). Specifically, if the Hessian of $f(\mathbf{x})$ at \mathbf{x}_k is PD, one can choose \mathbf{D} to be the inverse of the **reduced Hessian**:

$$\mathbf{H}_R(\mathbf{x}_k) = \left[\begin{array}{c|ccc} \bar{\mathbf{H}}(\mathbf{x}_k) & & & \mathbf{0} \\ \hline & 1 & \dots & 0 \\ & \vdots & \ddots & \vdots \\ \mathbf{0} & 0 & \dots & 1 \end{array} \right]$$

Where $\bar{\mathbf{H}}(\mathbf{x}_k)$ consists of second derivatives of $f(\mathbf{x})$ w.r.t to variables **not** in $I^+(\mathbf{x}_k)$ (i.e., it's a part of the full Hessian). This requires of course a re-indexing of the entries of \mathbf{x} such that all the variables with indices in $I^+(\mathbf{x}_k)$ will be in the bottom part of the vector \mathbf{x} at the current iterate. We can now state the **projected Newton** algorithm:

input : function f ; initial point \mathbf{x}_0
output: (approximate) local minimizer \mathbf{x}^* of f
 Start with an initial guess \mathbf{x}_0
for $k = 1, 2, \dots$, *until convergence* **do**
 Identify the set $I^+(\mathbf{x}_k)$ and re-index \mathbf{x}_k s.t $I^+(\mathbf{x}_k) = \{r+1, r+2, \dots, n\}$
 Find descent direction $\mathbf{d}_k = -\mathbf{H}_R^{-1}(\mathbf{x}_k) \nabla f(\mathbf{x}_k)$
 $\alpha_k = \text{ArmijoRule}(@f, \mathbf{x}_k, f(\mathbf{x}_k), \nabla f(\mathbf{x}_k), \mathbf{d}_k, \sigma, \beta, \alpha_0, \text{Flag})$
 Update $\mathbf{x}_{k+1} = \mathcal{P}_{\mathbf{x}}(\mathbf{x}_k + \alpha_k \mathbf{d}_k)$
end
 Return $\mathbf{x}^* = \mathbf{x}^k$

Algorithm 2: Projected Newton algorithm

✎ **14.** Write down the augmented Lagrangian optimization scheme for problem (10). The minimization step should take into account the bound constraints explicitly (i.e., don't try to incorporate them into the augmented Lagrangian, but rather treat them directly as we showed above). Include in your scheme an explicit formula for updating the Lagrange multiplier and the penalty parameter in each iteration. Write stopping criteria for both inner and outer iterations. Since the Hessian of the objective in problem (10) is only PSD (and therefore not necessarily invertible), regularize it by adding ϵ to the diagonal, i.e $\mathbf{H}_\epsilon = \mathbf{H} + \epsilon \mathbf{I}$.

6 Solving a real problem

Having developed some numerical schemes to solve problem (10), it is now time to test your ideas on real data. download from the website the following files:

- `xForTraining.mat` - A $[784 \times 1000]$ matrix containing images for the training phase of the SVM. The matrix contains 1000 $[28 \times 28]$ images of handwritten digits in column stack representation (i.e., `xForTraining(:,i)` is the column stack representation of the i^{th} image, and so on).
- `labelsForTraining.mat` - A vector of (non-binary) labels for the images in the training set. From these labels you should construct \mathbf{y} , the vector of binary labels, such that $y = -1$ for the digit 0, and $y = 1$ for the digit 9.
- `xForTest.mat` - A matrix containing images for the test phase of the SVM, in the same format.
- `labelsForTest.mat` - A vector of labels for the test images in the same format.
- `ExtractFeatures.m` - A function that extracts feature vectors from the raw data. The feature vectors consist of the first 50 principle components of the images in the training set (if you don't know what principle components are, think about it as a basis transformation, where in the new basis we need significantly less coefficients to represent the same data with little loss of information).
- `coeff.mat` - An argument required by `ExtractFeatures.m`

15. Implement the method we developed above for solving (10) (AL with Projected Gradient Descent with an option to use a scaling matrix) and construct the separating hyperplane. What are the stopping conditions you used? Introduce tolerances and a bound on the maximum number of iterations, as we did in the previous project. Compare convergence properties with and without the scaling matrix.

16. Test your classifier on the test images. What is the ratio of images that were classified correctly?

Submission instructions: make sure that your code outputs the following variables in the main workspace:

- \mathbf{w} - the normal to the separating hyperplane.
- w_0 - the intercept.
- λ - the vector of optimal dual variables.
- In `RunMe.mat`, provide a flag `Flag` to choose between projected gradient descent (`Flag = 'projGrad'`) and projected Newton (`Flag = 'projNewton'`).

7 Encore*

As we said earlier, the fact that the feature vectors enter the dual problem (10) only through inner products plays a key role in constructing **non-linear classifiers**. Instead of finding a linear classifier in the original feature space (a task that requires some compromise through the soft margin formulation we described above), one can transform the data non-linearly into a higher dimensional space, where it is believed to be more separable, and try to find a linear classifier there. Transforming back to the original lower dimensional feature space, the classifier will of course be non-linear. Practically, one doesn't have to explicitly carry out the transformation, but only to be able to compute inner-products in the higher dimensional feature space. For particular choices of non-linear transformations, it's possible to use a **kernel** - $K(\mathbf{x}_i, \mathbf{x}_j)$ to compute inner-products in the new space, using the feature vectors representation in the original space. Basically, you can use the same code to compute nonlinear classifiers, just replace $\mathbf{x}_i^T \mathbf{x}_j$ with $K(\mathbf{x}_i, \mathbf{x}_j)$. Some useful kernel functions can be found [here](#) and you can try constructing a non-linear classifier with one of them.

8 Tips

- You may find the following Matlab commands useful (for comparison purposes): SVMtrain, SVMclassify, quadprog, feval. For SVMtrain use the following options:

```
svmtrain(x,y,'METHOD','SMO','SMO_OPTS',opts,'BOXCONSTRAINT',C,'AUTOSCALE',  
false)  
With opts = svmsmoset('MaxIter',200000)
```

- In order to make sure your gradients are correct, you can compare with the finite difference approximation:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \frac{f(x_1, \dots, x_i + \Delta x, \dots, x_n) - f(x_1, \dots, x_i, \dots, x_n)}{\Delta x}$$

9 Acknowledgments

- Stanford's *Unsupervised Feature Learning and Deep Learning*.
- *MNISTS* dataset.

References

- [1] Bertsekas, D. P., *Nonlinear Programming*, Athena Scientific, 1999