CS 536 : Optimization

16:198:536

At the core of most machine learning algorithms is some notion of optimization - how do you find the 'best' model? How do you find the 'best' parameters? Optimization is a complicated and deep subject in its own right, and the machine learning aspect of it frequently is concerned with questions like "what does it mean for a model to be best"? If you can answer that question for whatever problem you are faced with, a variety of algorithms exist to throw at your problem and generate a good model.

While optimization is distinct from machine learning, as a field, it is worth taking a brief overview of the subject and some approaches, especially if you are not familiar with the subject.

The fundamental optimization question is this: given a function $F(\underline{x})$ and some set of allowed $\underline{x}$ values C, solve the following problem:

$$\min_{\underline{x} \in C} F(\underline{x}). \tag{1}$$

Simple on its face - but much depends on the structure of F and even more on the structure of C. It is reasonable to claim that there is no general purpose optimization algorithm that is always effective. Indeed, for any optimization algorithm, it's relatively straightforward to generate either optimization instances where that algorithm won't apply at all, or optimization problems where that algorithm behaves exceptionally poorly. Typically, it is important to tailor your specific approach to the problem itself.

1 Unconstrained Optimization

We want to solve

$$\min_{\underline{x} \in \mathbb{R}^d} F(\underline{x}). \tag{2}$$

Typical optimization algorithms are based on a principle of hill climbing - essentially, at some proposed $\underline{x}$, can you find a 'nearby' or similar $\underline{x}'$ that works even better and produces a better value of the objective function? Especially in continuous cases, it's useful to use as much local information about F as possible to determine where better $\underline{x}'$ might be - in particular, we look at the derivative. One of the first results you learn in calculus is that for continuously differentiable functions, optima will occur when the derivative is zero, or in the high dimensional case when the gradient is zero:

$$\nabla_{\underline{x}} F(\underline{x}) = 0. \tag{3}$$

This essentially indicates that moving slightly in any direction from $\underline{x}$ does not change the value of F at all. This provides one of the 'classic' approaches to optimization in the unconstrained case set up the vector equation $\nabla F = 0$, a system of d-many equations, and solve for the values of $\underline{x}$ that satisfy this equation. The solution may not be unique, and you may have to test and compare the value of F at various solution values to determine the true minimum, but if it is solvable this gives you an easy mechanism for identifying the optima. *Note: We could also generalize the second-derivative test for identifying optima as maxima or minima or saddle points, but this would require computing on the Hessian of F. Reasonable for some F but frequently annoying.*

Consider trying to identify the point that minimizes the distance to some point $\underline{a}$ as well as some point $\underline{b}$. We could express this as

$$\min_{\underline{x} \in \mathbb{R}^d} \; ||\underline{x} - \underline{a}||^2 + ||\underline{x} - \underline{b}||^2. \tag{4}$$

In this case, the gradient of the objective function is given by

$$\nabla F = 2(\underline{x} - \underline{a}) + 2(\underline{x} - \underline{b}). \tag{5}$$

Setting this equal to zero and solving for $\underline{x}$, we get a unique optimum of $\underline{x}^* = (1/2)(\underline{a} + \underline{b})$ - geometrically the midpoint between $\underline{a}$ and $\underline{b}$, exactly our intuition for the solution in this case.

One of the advantages of this approach is that it trades a high dimensional optimization problem for the algebraic problem of solving a system of equations. Equation solving is well understood and well studied, and many established methods can be applied here. Nevertheless, especially of F is highly non-linear in terms of the values of $\underline{x}$, solving this system may be quite difficult.

In order to get around this computational difficulty, most optimization algorithms rely on a scheme of iterative updating - getting back to the hill climbing idea we started with. In particular, given a 'guess' at the solution of $\underline{x}^t$, how can you construct an improved guess $\underline{x}^{t+1}$? One way is to consider a slight modification of $\underline{x}^t$, i.e.,

$$\underline{x}^{t+1} = \underline{x}^t + \underline{d}^t, \tag{6}$$

where $\underline{d}^t$ represents a step in some direction, with the goal that $F(\underline{x}^t) < F(\underline{x}^t + \underline{d}^t)$. The goal then is that $\underline{x}^t \to \underline{x}^*$, a minimum of F. But how to choose this step? Two decisions need to be made here, the *direction* of the step, and the *size* of the step. One simple approach is to simply standardize the choice of direction, and then worry about the size of the step in that direction. If the direction is given, determining the stepsize is effectively reducing a d-dimensional optimization problem to a 1-dimensional optimization problem - finding how far to move in that direction. Two traditional choices for direction give us classical optimization algorithms: **gradient descent** and **coordinate descent**.

1.1 Gradient Descent

In Gradient Descent, we effectively take the 'direction of maximum decrease' - F is decreasing fastest in the opposite direction of the gradient, so we change $\underline{x}^t$ by moving in that direction:

$$\underline{x}^{t+1} = \underline{x}^t - \alpha_t \nabla_{\underline{x}} F(\underline{x}^t). \tag{7}$$

In this case, α_t represents the stepsize or *learning rate* - how much we should change $\underline{x}^t$ by moving in the specified direction. For sufficiently small α_t, this will always produce a decrease in F. If α_t is too large, we change $\underline{x}^t$ too much and potentially overshoot the goal - convergence to a minimum $\underline{x}^*$ will be slow, if at all. If α_t is too small, we don't change $\underline{x}^t$ very much in any one step, and again convergence to $\underline{x}^*$ will be slow. Choosing good values of the stepsize is a large problem - various schemes exist for trying to adaptively choose good rates and optimize convergence. However, a traditional and computationally simple choice is to just take $\alpha_t = \alpha > 0$, i.e., fix the step size as a constant.

Consider the previous example again, where $F(\underline{x}) = ||\underline{x} - \underline{a}||^2 + ||\underline{x} - \underline{b}||^2$. In this case, gradient descent would take the form of choosing some $\underline{x}^0$ at random, and then iterating the update

$$\begin{aligned} \underline{x}^{t+1} &= \underline{x}^t - \alpha \nabla_{\underline{x}} F(\underline{x}^t) \\ &= \underline{x}^t - \alpha(2\underline{x}^t - 2\underline{a} + 2\underline{x}^t - 2\underline{b}) \\ &= (1 - 4\alpha)\underline{x}^t + 2\alpha(\underline{a} + \underline{b}). \end{aligned} \tag{8}$$

If we choose α such that $0 < \alpha < 1/4$, we see that this update is effectively shrinking $\underline{x}^t$ by a constant factor, and adding a constant term. This action of repeatedly shrinking, then re-adding, amounts to a contraction mapping, and will have a unique fixed point $\underline{x}^*$ given by

$$\underline{x}^* = (1 - 4\alpha)\underline{x}^* + 2\alpha(\underline{a} + \underline{b}), \tag{9}$$

which solves to

$$\underline{x}^* = (1/2)(\underline{a} + \underline{b}), \tag{10}$$

exactly as expected.

Gradient descent is one of the 'standard' optimization algorithms, frequently used in training for a variety of ML models: one of the reasons we structure our neural networks the way we do is that it makes the gradient very easy to compute, and thus gradient descent easy to implement for determining the optimal weights and parameters. As noted, there are adaptive schemes for trying to pick better step sizes (for instance, solving the 1-d optimization problem of choosing the best stepsize along that direction), but constant stepsize can be highly effective, even if it requires some tuning.

Some final thoughts on gradient descent:

- Gradient descent is effectively a 'first order' method, utilizing only information from the gradient. It amounts to approximating a function with a tangent line/plane, and moving along that tangent towards the optimum of the function. Second order methods might pull in the second derivatives or the Hessian matrix, effectively approximating the function F with a quadratic approximation, and trying to optimize theses successive quadratic approximations. These are more effective and have better convergence rates - but at the added computational cost associated with computing and utilizing the Hessian matrix.
- Gradient descent is a surprisingly *robust* algorithm - even if you don't compute the exact gradient, repeatedly computing something approximately equal to the gradient and descending in these directions still demonstrates good convergence to the minimum. This is frequently utilized when F is in the form of a large sum (for instance summing errors on training data points) - rather than compute the full gradient of F, we can approximate the gradient of the sum by computing and adding the gradients of only some of the terms of the sum. This is the idea behind **stochastic gradient descent**.
- Other methods for improving gradient descent include **momentum methods**. The idea behind the method is this: if moving in the direction $\underline{x}^t - \underline{x}^{t-1}$ was a good idea previously, it seems like a bad idea to completely throw this away during the next step updating $\underline{x}^t$ to $\underline{x}^{t+1}$. To correct for this, we add a 'momentum' term, effectively carrying the iterate a little bit further in the same direction of motion:

  $$\underline{x}^{t+1} = \underline{x}^t - \alpha_t \nabla_{\underline{x}} F(\underline{x}^t) + \beta_t \left(\underline{x}^t - \underline{x}^{t-1}\right). \tag{11}$$

  Again this frequently demonstrates improved convergence properties (and is the basis of a lot of the standard best gradient descent algorithms) but does require some additional effort to tune and manage.

- **Termination:** One aspect to implementing the algorithm remains - when to terminate? When in the business of numerically approximating things, it is rare that you will be able to compute the *exact* solution, but when is your approximation good enough? How do you know when you are sufficiently close to the optimum? A complete answer to this question would depend on the specific problem to be solved. But as a starting point, we know that at the optimum we should have that $\nabla F = 0$. Which means that when we are close to the optimum (subject to some smoothness conditions on the derivatives) we should have that ∇F is *close* to 0. This provides a very nice termination condition: terminate the algorithm when $||\nabla_{\underline{x}} F(\underline{x}^t)||^2$ falls below some specified threshold ϵ.

1.2 Coordinate Descent

While gradient descent is in many ways 'standard', it is important to remember that the choice of $-\nabla F$ as the direction of descent was effectively arbitrary. *Any* choice of descent direction will do in many ways. If we do not want to go to the trouble of computing ∇F each time, a good choice for descent direction is along a coodinate axis. Let $\underline{e}_i = (0, 0, \ldots, 1, \ldots, 0)$ (the all-zero vector with a 1 in the i coordinate). At time t, we could take the direction of descent to be $\underline{e}_i$ for some choice of i. This update amounts to 'freezing' everything but the i-th coordinate of $\underline{x}^t$ and determining what a better choice of that coordinate value would be for $\underline{x}^{t+1}$. Again, we see that the d-dimensional optimization problem is reduced in this way to a one dimensional optimization problem. Every iteration of the algorithm, we select a coordinate of $\underline{x}^t$ (either cycling through the coordinates or via random selection) and update only that coordinate to generate $\underline{x}^{t+1}$. Termination occurs when no more coordinates can be updated - notice that if no more coordinates can be updated, $\partial F/\partial x_i(\underline{x}) = 0$ for each component i, which means we have reached a point where $\nabla F = 0$.

Consider again optimizing $F(\underline{x}) = ||\underline{x} - \underline{a}||^2 + ||\underline{x} - \underline{b}||^2$, but under coordinate descent. If we expand F, isolating the terms that depend only on a given x_i, we can express F as

$$F(\underline{x}) = (x_i - a_i)^2 + (x_i - b_i)^2 + F', \tag{12}$$

where F' contains the remaining terms. Given an iterate $\underline{x}^t$, if we tried to improve/optimize on the value of x_i^t, the above suggests taking $x_i^{t+1} = (1/2)(a_i + b_i)$. In this way, coordinate descent can be fairly easily optimized at every step, and will converge in d steps to the exact solution $\underline{x}^{d+1} = (1/2)(\underline{a} + \underline{b})$.

1.3 An Example: Least Squares, Ridge, Lasso

Consider two ML examples of the above algorithms, applied to least squares regression.

1.3.1 Least Squares and Ridge Regression

In the case of Least Squares/Ridge Regression, we want to optimize

$$\min_{\underline{w}} \ \frac{1}{2}||\underline{y} - X\underline{w}||^2 + \frac{1}{2}\lambda||\underline{w}||_2^2. \tag{13}$$

This is the Ridge Regression problem, but we recover Least Squares Classic simply taking $\lambda = 0$.

We can solve this explicitly using the stationarity condition $\nabla F = 0$, but gradient descent as potential advantages. Note for instance that

$$\nabla_{\underline{w}} F(\underline{w}) = -X^{\mathrm{T}}\underline{y} + X^{\mathrm{T}}X\underline{w} + \lambda\underline{w}. \tag{14}$$

Utilizing this for a gradient descent update, we have for an initial random selection $\underline{w}^0$,

$$\begin{aligned}\underline{w}^{t+1} &= \underline{w}^t - \alpha\left[-X^{\mathrm{T}}\underline{y} + X^{\mathrm{T}}X\underline{w}^t + \lambda\underline{w}^t\right] \\ &= \left(I - \alpha X^{\mathrm{T}}X - \alpha\lambda I\right)\underline{w}^t - \alpha\left[-X^{\mathrm{T}}\underline{y}\right]\end{aligned} \tag{15}$$

Note, unlike solving explicitly for $\underline{w}^*$, this update can be computed without having to explicitly invert $[X^{\mathrm{T}}X + \lambda I]$. If α is chosen sufficiently small such that $(I - \alpha X^{\mathrm{T}}X - \alpha\lambda I)$ represents a contraction on $\underline{w}^t$, the above will converge to a unique fixed point given by

$$\underline{w}^* = \left(I - \alpha X^{\mathrm{T}}X - \alpha\lambda I\right)\underline{w}^* - \alpha\left[-X^{\mathrm{T}}\underline{y}\right], \tag{16}$$

or

$$\underline{w}^* = \left[X^{\mathrm{T}}X + \lambda I\right]^{-1}\left[X^{\mathrm{T}}\underline{y}\right], \tag{17}$$

the usual regression solution.

1.3.2 Lasso Regression

In this case, we want to solve

$$\min_{\underline{w}} \ ||\underline{y} - X\underline{w}||^2 + \lambda||\underline{w}||_1. \tag{18}$$

Note that in this case, we are somewhat stuck in that the objective function above is not continuously differentiable - the l_1 norm does not have a derivative at 0. This represents some immediate technical complications to the applying previous discussion on gradient descent.

However, as an iterative update / hill climbing scheme, coordinate descent is still applicable here. Consider expressing the above objective function with the dependence on w_i made explicit (we have m training data points, each of dimension $k+1$ with the zero-th coordinate assumed to be 1, so w_0 corresponds to the bias - we will only Lasso on the non-bias components):

$$\begin{aligned}F(\underline{w}) &= \sum_{j=1}^{m}(y^j - \underline{w}.\underline{x}^j)^2 + \lambda\sum_{j=1}^{k}|w_j| \\ &= \sum_{j=1}^{m}(-x_i^j w_i + C_i^j)^2 + \lambda\sum_{j=1}^{k}|w_j|,\end{aligned} \tag{19}$$

where $C_i^j = y^j - \underline{w}_{-i}.\underline{x}^j_{-i}$ (the remaining terms of the square error, discounting the i-th component.

- **Updating the Bias Term:** Given $\underline{w}$, we want to choose w_0 to optimize the above. Because there is no l_1 term on w_0, we can simply optimize in the usual way (noting $x_0^j = 1$):

$$\frac{\partial F}{\partial w_0} = \sum_{j=1}^{m} 2(-x_0^j)(-x_0^j w_0 + C_0^j) = -2\sum_{j=1}^{m}(-w_0 + C_0^j). \tag{20}$$

  The best choice of w_0 is when the above is 0, which gives an update

$$w_0' = \frac{1}{m}\sum_{j=1}^{m} C_0^j = w_0 + \frac{1}{m}\sum_{j=1}^{m}\left(y^j - \underline{w}.\underline{x}^j\right). \tag{21}$$

- **Updating Non-Bias Terms:** To update a non-bias term, we need to choose a value of w_i to optimize

$$F(\underline{w}) = \left[\sum_{j=1}^{m}(x_i^j)^2\right] w_i^2 - 2\left[\sum_{j=1}^{m} x_i^j C_i^j\right] w_i + \lambda|w_i| + \left[\sum_{j=1}^{m}(C_i^j)^2 + \lambda\sum_{j\neq i}^{k}|w_j|\right], \tag{22}$$

Note, we can discard any terms that don't depend on w_i when trying to find the best w_i. For a general function of the form

$$Ax^2 - Bx + \lambda|x|, \tag{23}$$

what is the best choice of x? Note that A in this case is positive.

For a standard problem like this, we would like to take the derivative and set it equal to zero - the problem is that the derivative of $|x|$ does not exist at zero. We can generalize this idea of looking at the derivative (and looking for zeros of the gradient) to looking at zeros of the **sub**gradient. The subgradient generalizes the gradient when the derivative does not exist; the subgradient is the set of all tangent slopes at a point that would bound the function from below. In one dimension, the sub-derivative of $|x|$ is given by $\text{sign}(x)$ - we have $\text{sign}(x) = \{1\}$ if $x > 0$, because $|x| = x$ in this regime. Similarly we have $\text{sign}(x) = \{-1\}$ for $x < 0$, since $|x| = -x$ in this regime. At $x = 0$ though, we have $\text{sign}(0) = [-1, 1]$, because any slope in this range will give a tangent line at $x = 0$ that bounds $|x|$ from completely from below. For functions where the derivative does not exist but the subgradient does, instead of $\nabla F = 0$, optima occur when $0 \in \text{subgradient}F$. Applying this here, the optimum (and it must be a minimum - why?) will occur when

$$0 \in 2Ax - B + \lambda\text{sign}(x), \tag{24}$$

or

$$-\frac{1}{\lambda}(2Ax - B) \in \text{sign}(x). \tag{25}$$

Working this by cases gives the following solution:

$$x^* = \begin{cases} \frac{B/\lambda - 1}{2A/\lambda} & \text{if } 1 < B/\lambda \\ \frac{B/\lambda + 1}{2A/\lambda} & \text{if } B/\lambda < -1 \\ 0 & \text{if } -1 \leq B/\lambda \leq 1, \end{cases} \tag{26}$$

or

$$x^* = \begin{cases} \frac{B-\lambda}{2A} & \text{if } \lambda < B \\ \frac{B+\lambda}{2A} & \text{if } B < -\lambda \\ 0 & \text{if } B \in [-\lambda, \lambda], \end{cases} \tag{27}$$

Applying this to our case, we have

$$\begin{aligned} A &= \left[\sum_{j=1}^{m}(x_i^j)^2\right] = (X_i)^{\mathrm{T}}.X_i \\ B &= 2\left[\sum_{j=1}^{m} x_i^j C_i^j\right] \\ &= 2\left[\sum_{j=1}^{m} x_i^j(y^j - \underline{w}.\underline{x}^j + w_i x_i^j)\right] \\ &= 2\left[\sum_{j=1}^{m} x_i^j(y^j - \underline{w}.\underline{x}^j) + w_i\sum_{j=1}^{m}(x_i^j)^2\right] \\ &= 2(X_i)^{\mathrm{T}}(\underline{y} - X\underline{w}) + 2w_i(X_i)^{\mathrm{T}}.X_i, \end{aligned} \tag{28}$$

where X_i is the i-th feature column of the data matrix X. This gives us our update formula for the i-th coordinate (simplifying):

$$w_i' = \begin{cases} w_i + \frac{(X_i)^{\mathrm{T}}(\underline{y} - X\underline{w}) - \lambda/2}{(X_i)^{\mathrm{T}}.X_i} & \text{if } \frac{-(X_i)^{\mathrm{T}}(\underline{y} - X\underline{w}) + \lambda/2}{(X_i)^{\mathrm{T}}.X_i} < w_i \\ w_i + \frac{(X_i)^{\mathrm{T}}(\underline{y} - X\underline{w}) + \lambda/2}{(X_i)^{\mathrm{T}}.X_i} & \text{if } w_i < \frac{-(X_i)^{\mathrm{T}}(\underline{y} - X\underline{w}) - \lambda/2}{(X_i)^{\mathrm{T}}.X_i} \\ 0 & \text{if } w_i \in \left[\frac{-(X_i)^{\mathrm{T}}(\underline{y} - X\underline{w}) - \lambda/2}{(X_i)^{\mathrm{T}}.X_i}, \frac{-(X_i)^{\mathrm{T}}(\underline{y} - X\underline{w}) + \lambda/2}{(X_i)^{\mathrm{T}}.X_i} \right], \end{cases} \tag{29}$$

With both these update equations for the bias and non-bias weights, we are now in a position to do coordinate-descent to compute the solution to Lasso: starting with some random choice for $\underline{w}^0$, update $\underline{w}^t$ to $\underline{w}^{t+1}$ by choosing some coordinate in $i \in \{0, \ldots, k\}$ and updating that coordinate according to the correct update formula.

A Small Numerical Example: For $m = 10$, $k = 4$, a data set was generated taking $X_i \sim N(0, 1), i > 0$:

$$X = \begin{pmatrix} 1 & 2.53954 & 0.286795 & 0.168501 & -0.199211 \\ 1 & -1.08862 & -0.0249975 & -2.09938 & 1.00711 \\ 1 & 1.63615 & 0.574396 & -0.981347 & 0.114764 \\ 1 & 0.437649 & -1.48439 & -1.11575 & -0.832625 \\ 1 & -0.632709 & 0.191735 & -0.9027 & -0.0650515 \\ 1 & -1.39928 & -2.10481 & 0.100416 & 2.26596 \\ 1 & -1.03209 & 0.917898 & -0.781086 & 0.137913 \\ 1 & 0.0179114 & 1.35139 & -0.664107 & -1.28817 \\ 1 & -2.02876 & 3.02379 & 1.26562 & -0.188491 \\ 1 & -0.10032 & 1.3035 & -0.65843 & 0.314657 \end{pmatrix} \tag{30}$$

A model was evaluated on this data set using $Y = 3.0X_0 + 0.9X_1 - 0.4X_2 + 0.1X_3 + N(0, 0.01)$ - in this case, X_4 is an irrelevant feature. This yielded output values of

$$Y^{\mathrm{T}} = \begin{pmatrix} 5.25802 & 1.94495 & 4.02179 & 3.84626 & 2.18401 & 2.58642 & 1.49117 & 2.47472 & 0.164915 & 2.38225 \end{pmatrix} \tag{31}$$

Doing straight linear regression on this yields a model of

$$\hat{Y} = 2.99695 + 0.89846X_1 - 0.384931X_2 + 0.104557X_3 + 0.0109784X_4, \tag{32}$$

which is in fairly good agreement with the data.

However, starting with an estimate $\underline{w}^0 = 0$ and $\lambda = 0.25$, the first few rounds of this Lasso update scheme (cycling through each component) yields the following progression - given every 5 iterations of the method:

$$\begin{aligned} \hat{Y}_0 &= 0 \\ \hat{Y}_5 &= 2.63545 + 0.922573X_1 - 0.296005X_2 - 0.0952267X_3 + 0.0996947X_4 \\ \hat{Y}_{10} &= 2.84056 + 0.909523X_1 - 0.302463X_2 + 0.0669618X_4 \\ \hat{Y}_{15} &= 2.89914 + 0.908594X_1 - 0.332574X_2 + 0.034032X_3 + 0.0424004X_4 \\ \hat{Y}_{20} &= 2.93354 + 0.901017X_1 - 0.350475X_2 + 0.0539721X_3 + 0.0240272X_4 \\ \hat{Y}_{25} &= 2.95314 + 0.895438X_1 - 0.36207X_2 + 0.0652344X_3 + 0.0120103X_4 \\ \hat{Y}_{30} &= 2.96481 + 0.891638X_1 - 0.369264X_2 + 0.07189X_3 + 0.00441556X_4 \\ \hat{Y}_{35} &= 2.97182 + 0.889212X_1 - 0.373696X_2 + 0.0758773X_3 \\ \hat{Y}_{40} &= 2.97602 + 0.887772X_1 - 0.376316X_2 + 0.078273X_3 \end{aligned} \tag{33}$$

At this point, X_4 has effectively been pruned as irrelevant, and stays out of the active feature set. Repeating this, the iterates converge to the following model:

$$\hat{Y} = 2.9801 + 0.887594X_1 - 0.377653X_2 + 0.0807256X_3, \tag{34}$$

which is both in good agreement with the 'true' model, as well as the result found by Mathematica's built in numerical optimizer:

$$\hat{Y}_{\text{mathematica}} = 2.98009 + 0.887591X_1 - 0.377654X_2 + 0.0807303X_3 - 1.0446522692972705 * 10^{-10}X_4. \tag{35}$$

2 Constrained Optimization

In this section, we consider the problem of

$$\min_{\underline{x} \in C} \; F(\underline{x}), \tag{36}$$

where C is some subset of $\mathbb{R}^d$. Constrained optimization is typically much more difficult than unconstrained optimization - the constraint set C restricts how we can move about in search of the solution. In terms of hill climbing, it is no longer sufficient to move to a nearby point with a better value of the objective function, we must ensure that we remain within the constraint set as well. The condition $\nabla F = 0$ is no longer sufficient to identify a local optimum - the constrained optimum might lie on the boundary of C where F could still be increased ($||\nabla F|| > 0$) if only we could step outside of C.

As could well be imagined - much depends on the structure of C and how it is specified. For instance if C is fractal, jagged, or disconnected, optimizing over C can be quite difficult. Ideally, C represents a convex set like a sphere or a plane or a polytope, in which case movement through the set in the manner of hill climbing type algorithms is relatively easy.

Constrained problems are typically solved with one of three approaches: in **projection methods**, an iterative algorithm like gradient descent is augmented with an extra step that brings the iterates back into the constraint set C if they every step out; in **penalty or barrier methods**, the constrained problem is turned into an unconstrained problem, by adding a cost to the objective function that reflects whether the constraints are satisfied and how much they are violated; in **Lagrangian or dual methods**, the original constrained optimization problem is replaced with a much more weakly (if at all) constrained problem in terms of the Lagrange multipliers of the constraints. In these second two kinds of approaches, the constrained problem is exchanged for an unconstrained or sequence of unconstrained problems for which the previous unconstrained solutions may be applied.

2.1 Projection Methods

The problem with things like gradient descent generally is that even if $\underline{x}^t$ is 'feasible' and lies within the constraint set, the update

$$\underline{x}^{t+1} = \underline{x}^t - \alpha_t \nabla_{\underline{x}} F(\underline{x}^t) \tag{37}$$

may carry the iterate outside the constraint set so $\underline{x}^{t+1}$ is infeasible.

A simple way to correct this might be to 'drag' $\underline{x}^{t+1}$ back into the constraint set. To this end, we define the projection operator:

$$P_C(\underline{x}) = \text{argmin}_{\underline{x}' \in C} \; ||\underline{x} - \underline{x}'||^2. \tag{38}$$

Thus $P_C(\underline{x})$ is the point in C that is nearest to $\underline{x}$. If $\underline{x}$ is in C, we get immediately that $P_C(\underline{x}) = \underline{x}$, otherwise P_C will return the point on the boundary of C that is closest to $\underline{x}$.

This suggests modifying gradient descent to **projected gradient descent**, taking

$$\underline{x}^{t+1} = P_C(\underline{x}^t - \alpha_t \nabla_{\underline{x}} F(\underline{x}^t)). \tag{39}$$

It can be shown, by way of justifying this algorithm, that in the constrained case, under suitably nice F and C, any local minimum $\underline{x}^*$ must satisfy for all sufficiently small $\alpha > 0$

$$\underline{x}^* = P_C(x^* - \alpha \nabla_{\underline{x}} F(\underline{x}^*)). \tag{40}$$

This generalizes the $\nabla F = 0$ condition to the constrained case. Note, if $\underline{x}^*$ is in the interior of C, this condition is equivalent to $\nabla F = 0$. If $\underline{x}^*$ is on the boundary of C, this condition indicates that the direction of greatest decrease carries you immediately out of C.

In terms of implementing this algorithm, however, note that it requires being able to compute the projection. The projection is a constrained optimization problem itself and thus many of the previous complaints apply. However, in this case we have a very simple objective function (quadratic, positive definite), and if C is similarly nice, the projection is easy to calculate. For instance, if C is the plane defined by $\underline{u}.\underline{x} = 0$ (taking $\underline{u}$ as a unit vector), you can show that

$$P_C(\underline{x}) = \underline{x} - (\underline{u}.\underline{x})\underline{u}. \tag{41}$$

Similarly, if C is a sphere of radius r, centered on the point $\underline{v}$, then the projection is simply

$$P_C(\underline{x}) = \underline{v} + (\underline{x} - \underline{v}) \min\left[\frac{r}{||\underline{x} - \underline{v}||}, 1\right]. \tag{42}$$

In each case, we could easily implement projected gradient descent for constraint sets like these or similar. Note, when the projection is easily computed, the problem of generating **feasible** solutions is easily settled - picking any point $\underline{x}$, $P_C(\underline{x})$ is immediately feasible for this constraint set.

2.2 Penalty and Barrier Methods

Much depends on how the constraint set C is specified. Frequently, C will be defined in terms of a set of equality or inequality constraints. In general, we will be interested in C of the form

$$C = \{\underline{x} \in \mathbb{R}^d : \forall i = 1, \ldots, k : h_i(\underline{x}) = 0 \text{ and } \forall j = 1, \ldots, l : g_j(\underline{x}) \leq 0\}. \tag{43}$$

The following methods can be mixed and matched as appropriate, but it is convenient to separate them for discussion:

2.2.1 Penalty Methods and Equality Constraints

Consider C defined by the set of equality constraints $\{h_i(\underline{x}) = 0\}_{i=1,\ldots,k}$. Construct a new optimization function, defined by

$$F_\beta(\underline{x}) = F(\underline{x}) + \beta \left(\sum_{i=1}^{k} h_i(\underline{x})^2\right). \tag{44}$$

The additional terms above function as a cost or penalty paid by deviation away from $h_i(\underline{x}) = 0$. If β is taken to be quite large, the above should be minimized in an unconstrained way by taking $\underline{x}$ not only close to the minimum of F, but also near the region of space where each $h_i(\underline{x})$ is close to or exactly zero.

However, for any given value of β the minimum of $F_\beta(\underline{x})$ may or may not correspond to the exact constrained minimum of F. But the larger the value of β, the closer these should be. **Penalty Methods** proceed by constructing

a sequence of penalized unconstrained optimization problems, with successively large values of β: Given a sequence $\beta_t \to \infty$, define the optimization problem instance

$$O_t := \min_{\underline{x}} F(\underline{x}) + \beta_t ||\underline{h}(\underline{x})||^2. \tag{45}$$

Define $\underline{x}_t^*$ to be the solution to O_t. Under certain smoothness assumptions on F and $\underline{h}$ and the steady increase of β_t, it can be shown that $\underline{x}_t^* \to \underline{x}^*$, a solution to the constrained optimization problem. Any unconstrained optimization algorithm may be applied to these sequential penalty problems to compute $\underline{x}_t^*$.

However it is worth asking - why not just start with a really exceptionally large value of β and call it a day? Why solve a sequence of problems rather than one highly penalized problem? In fact, the solution to one of these problems can greatly facilitate the solution to the next, through the notion of the **warm start**. Any one optimization problem might take some large number of iterations to solve. But if the solution to O_t is 'close to' the solution to O_{t+1}, then the solution to O_t can be used as the starting point for something like gradient descent in the solution to O_{t+1}. This more 'informed' beginning, or warm start to the algorithm, can facilitate convergence to the solution to O_{t+1} sooner.

A Penalized Method of Solution: Given F and $\underline{h}(\underline{x})$, and $\beta_t \to \infty$, define $F_t(\underline{x}) = F(\underline{x}) + \beta_t ||\underline{h}(\underline{x})||^2$.

- Select some initial $\underline{x}_0^0$ at random, and compute a minimizer of $F_0(\underline{x})$ to some degree of accuracy - call it $\underline{x}_0^*$.
- Set $\underline{x}_1^0 = \underline{x}_0^*$, and compute a minimizer of $F_1(\underline{x})$ to some degree of accuracy - call it $\underline{x}_1^*$.
- Set $\underline{x}_2^0 = \underline{x}_1^*$, and compute a minimizer of $F_2(\underline{x})$ to some degree of accuracy - call it $\underline{x}_2^*$.
- Repeat this, taking $\underline{x}_{t+1}^0 = \underline{x}_t^*$, and computing a minimizer $\underline{x}_{t+1}^*$ of $F_{t+1}(\underline{x})$.
- Repeat until convergence of the $\{\underline{x}_t^*\}$ to a suitable threshold.
- Return the final $\underline{x}^*$.

To see the validity of this approach, consider the case of a single equality constraint $h(\underline{x}) = 0$. If $\underline{x}_t^*$ solves the unconstrained problem at step t and $h(\underline{x}_t^*) \to 0$, then we must have that

$$\nabla F(\underline{x}_t^*) + 2\beta_t h(\underline{x}_t^*) \nabla h(\underline{x}_t^*) = 0. \tag{46}$$

Note that if $\underline{x}_t^* \to \underline{x}^*$ and F, h are sufficiently smooth, then we would have $\nabla F(\underline{x}_t^*) \to \nabla F(\underline{x}^*)$ and $\nabla h(\underline{x}_t^*) \to \nabla h(\underline{x}^*)$ - but what of the $\beta_t h$ scalar term?

Note that we can take this vector equation and effectively solve for $\beta_t h(\underline{x}^* t)$ by dotting each side of the equation with ∇h - yielding (if the gradient of h is non-zero)

$$2\beta_t h(\underline{x}_t^*) = -\frac{[\nabla h(\underline{x}_t^*)]^{\mathrm{T}} \nabla F(\underline{x}_t^*)}{||\nabla h(\underline{x}_t^*)||^2}. \tag{47}$$

Based on the previous comment about the convergence of ∇F and ∇h, the above tells us that we should have

$$2\beta_t h(\underline{x}_t^*) \to -\frac{[\nabla h(\underline{x}^*)]^{\mathrm{T}} \nabla F(\underline{x}^*)}{||\nabla h(\underline{x}^*)||^2}. \tag{48}$$

If we call this constant λ, then the original gradient equation (in the limit as $t \to \infty$) will converge to

$$\nabla F(\underline{x}^*) + \lambda \nabla h(\underline{x}^*) = 0, \tag{49}$$

exactly the classical 'Lagrange multiplier' optimality condition which along with $h(\underline{x}^*) = 0$ identifies $\underline{x}^*$ as a constrained optimum. Note, this discussion generalizes to multiple equality constraints with multiple Lagrange multipliers.

Penalty methods illustrate a commonly useful approach to solving constrained optimization problems - turning them into a sequence of unconstrained optimization problems that converge in some sense to the constrained problem. We can also generalize these methods to inequality constraints as well: imagine that C were defined in terms of a set of inequalities we wanted satisfied, $\{g_i(\underline{x}) \leq 0\}_{i=1,\ldots,l}$. In this case, one problem we face is that because the constraint is one sided, it doesn't matter 'how negative' or how far from 0 in the negative direction g_i is. To model this with a penalty function, we would want to penalize g_i, but only in one direction - we don't have the same kind of symmetry we would with the h_i as before.

One approach potentially would be to introduce a penalty like the following:

$$F_\beta(\underline{x}) = F(\underline{x}) + \beta \left(\sum_{i=1}^{l} \max(0, g_i(\underline{x})) \right), \tag{50}$$

which captures this asymmetry and penalizes positive g_i. However, this would introduce a non-differentiable element into the objective function. We might be able to engineer a coordinate-descent solution in this case, but gradient descent in the usual way would not apply because the maximum function would not be differentiable at 0.

An alternative approach would be to use a penalty like the following:

$$F_\beta(\underline{x}) = F(\underline{x}) + \sum_{i=1}^{l} e^{\beta g_i(\underline{x})}. \tag{51}$$

In this case, as g_i increases, the penalty increases, but as β increases the penalty is increasingly flat for all $g_i(\underline{x}) \leq 0$ - essentially approximating the effect of the maximum function above but in a differentiable way.

An alternate approach to penalty methods for inequality constraints that better captures the one-sidedness of the constraint is the **Barrier Method**.

2.2.2 Barrier Methods and Inequality Constraints

Penalty methods assess a cost for violating the constraints to try to drive the iterates of the algorithm into a region where the constraint is satisfied. Barrier methods, or interior point methods, start from within the constraint set and try to erect a barrier to keep the iterates from ever exiting the constraint set. Let the constraint set be defined by $C = \{\underline{x} : \forall i = 1, \ldots n : g_i(\underline{x}) \leq 0\}$. We would like to ensure that the g_i are negative, and enforce a cost for allowing the g_i to approach 0, with *infinite* cost at $g_i = 0$.

Consider augmenting the objective function in the following way:

$$F_\epsilon(\underline{x}) = F(\underline{x}) + \epsilon \left[-\sum_{i=1}^{l} \ln(-g_i(\underline{x})) \right]. \tag{52}$$

In this case, if all the g_i are fairly negative, this adds a small amount (modulated by ϵ) to the overall cost, but the closer any g_i gets to 0, the added cost grows, diverging to ∞ as $g_i \to 0$ from below. Thinking in terms of gradient descent, attempting to minimize this augmented objective function will move the iterates of gradient descent away from the boundaries. The above is the 'logarithmic barrier', but any positive barrier function $B(g_i(\underline{x}))$ which diverges to ∞ as $g_i \to 0$ from below will do.

However, if the constrained optimum is actually at the boundary of C with $g_i = 0$ for some g_i, this barrier will keep you from every reaching it. Thus, similar to the penalty methods, we transform the constrained problem into

a sequence of unconstrained problems, with increasingly weak barriers, allowing the iterates closer and closer to the boundary of C. This leads to the following method, similar to the penalized method with warm starts:

A Barrier Method of Solution: Given F, $\{g_i\}$, a barrier function B and $\epsilon_t \to 0$, define $F_t(\underline{x}) = F(\underline{x}) + \epsilon_t \left[\sum_{i=1}^{l} B(g_i(\underline{x}))\right]$.

- Select some initial $\underline{x}_0^0$ at random *within the constraint set* C, and compute a minimizer of $F_0(\underline{x})$ to some degree of accuracy - call it $\underline{x}_0^*$.
- Set $\underline{x}_1^0 = \underline{x}_0^*$, and compute a minimizer of $F_1(\underline{x})$ to some degree of accuracy - call it $\underline{x}_1^*$.
- Set $\underline{x}_2^0 = \underline{x}_1^*$, and compute a minimizer of $F_2(\underline{x})$ to some degree of accuracy - call it $\underline{x}_2^*$.
- Repeat this, taking $\underline{x}_{t+1}^0 = \underline{x}_t^*$, and computing a minimizer $\underline{x}_{t+1}^*$ of $F_{t+1}(\underline{x})$.
- Repeat until convergence of the $\{\underline{x}_t^*\}$ to a suitable threshold.
- Return the final $\underline{x}^*$.

A Note On Implementation: It is important when updating the iterates that you modulate the stepsize as necessary to keep from stepping outside the constraint set. The gradient of the barrier should typically force the iterates into the interior, but overshoot can happen.

To see the validity of the approach, consider a single inequality constraint $g(\underline{x}) \leq 0$. Suppose that the $\underline{x}_t^*$ converge to some $\underline{x}^*$. If C is a closed set, then $\underline{x}^*$ is in C and is therefore a feasible solution, and $F(\underline{x}_t^*) \to F(\underline{x}^*)$ by the continuity of F. But we also get that

$$F(\underline{x}_t^*) + \epsilon_t B(\underline{x}_t^*) \geq F(\underline{x}_t^*) + \epsilon_{t+1} B(\underline{x}_t^*) \geq F(\underline{x}_{t+1}^*) + \epsilon_{t+1} B(\underline{x}_{t+1}^*). \tag{53}$$

Since $\epsilon_t B(\underline{x}_t^*)$ will always be non-negative, taking the limit of this as $t \to \infty$ yields

$$F(\underline{x}_t^*) + \epsilon_t B(\underline{x}_t^*) \geq F(\underline{x}^*). \tag{54}$$

However, observe the following: for *any* $\underline{x}$, we have that (since $\underline{x}_t^*$ is the minimizer)

$$F(\underline{x}) + \epsilon_t B(\underline{x}) \geq F(\underline{x}_t^*) + \epsilon_t B(\underline{x}_t^*) \geq F(\underline{x}^*). \tag{55}$$

Taking the limit as $t \to 0$, the left side of this inequality converges, and we get

$$F(\underline{x}) \geq F(\underline{x}^*). \tag{56}$$

This tells us that not only is $\underline{x}^*$, the limit of the iterates $\underline{x}_t^*$ a constrained minimum of F in C, it is actually a **global** minimizer over the constraint set. This is a much more powerful guarantee than in the previous methods.

3 Duality

We are concerned again with constrained optimization, in particular solving

$$\begin{aligned} &\min_{\underline{x} \in \mathbb{R}^d} F(\underline{x}) \\ &\text{(s.t.) } \forall i = 1, \ldots l : h_i(\underline{x}) = 0 \\ &\qquad \forall j = 1, \ldots, l : g_j(\underline{x}) \leq 0. \end{aligned} \tag{57}$$

Duality is largely concerned with unifying the objective function and the constraint functions through the use of the **Lagragian function:**

$$L(\underline{x}, \underline{\lambda}, \underline{\mu}) = F(\underline{x}) + \sum_{i=1}^{k} \lambda_i h_i(\underline{x}) + \sum_{j=1}^{l} \mu_j g_j(\underline{x}). \tag{58}$$

Note that if $\underline{x}^*$ is a solution to the constrained optimization problem, then for any $\underline{\lambda}$ and $\underline{\mu} \geq 0$, we have that

$$F(\underline{x}^*) \geq L(\underline{x}^*, \underline{\lambda}, \underline{\mu}) \tag{59}$$

This suggests that maximizing this lower bound could provide some useful information about the optimal constrained value $F(\underline{x}^*)$. The problem of course is that we don't know what $\underline{x}^*$ is. We can however relax this bound even further, considering a lower bound on a lower bound. For any $\underline{\lambda}$ and $\underline{\mu} \geq 0$, we would have

$$F(\underline{x}^*) \geq L(\underline{x}^*, \underline{\lambda}, \underline{\mu}) \geq \min_{\underline{x} \in \mathbb{R}^d} L(\underline{x}, \underline{\lambda}, \underline{\mu}). \tag{60}$$

We can take this one step further: if the above holds for *all* $\underline{\lambda}$ and $\underline{\mu} \geq 0$, we can try to choose them to make the lower bound as close to the upper bound as possible, that is

$$F(\underline{x}^*) \geq \max_{\underline{\lambda}, \underline{\mu} \geq 0} \min_{\underline{x} \in \mathbb{R}^d} L(\underline{x}, \underline{\lambda}, \underline{\mu}). \tag{61}$$

Define the dual function $Q(\underline{\lambda}, \underline{\mu}) = \min_{\underline{x} \in \mathbb{R}^d} L(\underline{x}, \underline{\lambda}, \underline{\mu})$ - note, this is an *unconstrained* optimization problem with respect to $\underline{x}$.

Referring to the original problem as the **Primal Problem**, we can connect it to the **Dual Problem**, $\max_{\underline{\lambda}, \underline{\mu} \geq 0} Q(\underline{\lambda}, \underline{\mu})$. The duality theorem (and you'll note the very few assumptions made on the objective function or the constraints) says that if optimizers exist for either problem, then we have

$$F(\underline{x}^*) \geq Q(\underline{\lambda}^*, \underline{\mu}^*). \tag{62}$$

If $\underline{x}$ satisfies the constraints, its *primal feasible*, and if $\mu_j \geq 0$ for all j, then $\underline{\mu}$ is dual feasible.

> An Important Question: What in the above analysis depended on $\underline{x}$ being a vector of real values?

If Q can be easily computed, this suggests a nice exchange of a complex constrained optimization problem to a potentially much nicer one. However, the fact that the above is in inequality is concerning - the optimal value of Q may tell us *something* about the optimal value of F, but not the actual value and not the optimizer $\underline{x}^*$ we need. This is referred to as the duality gap - if there is a duality gap, then the dual problem provides only limited information about the primal problem. *Note: limited information is still some information, and the dual problem can be useful in cases where bounding the primal problem has some use.*

Under some situations, however, there is no duality gap, and we can make strong connections between the primal problem in the dual problem. For instance, we have the following result:

> If F is convex over $\mathbb{R}^d$, and the constraints are linear functions of $\underline{x}$, then if there is a constrained optimum for the primal, there is no duality gap and there is some dual feasible solution such that $F(\underline{x}^*) = Q(\underline{\lambda}^*, \underline{\mu}^*)$.

Other conditions exist for ensuring the lack of a duality gap and the existence of dual feasible solutions, but this will be sufficient for our purposes.

It is worth considering what primal solutions and dual solutions 'look like': Suppose that $\underline{x}^*$ was a solution to the primal problem, and $g_j(\underline{x}^*) < 0$ (strict inequality). In that case, any dual solution would need to satisfy $\mu_j^* = 0$

- if it did not, the value of Lagrangian could be increased by decreasing that value of μ_j^*. Hence we see that not only should $\mu_i^* \geq 0$ always, we should also have that if $g_j(\underline{x}^*) < 0$, then $\mu_j^* = 0$. We say that if $g_j(\underline{x}^*)$ is **active** if $g_j(\underline{x}^*) = 0$, and **inactive** if $g_j(\underline{x}^*) < 0$; μ_j^* that are non-zero identify active constraints.

This is formalized somewhat in the Karush-Kuhn-Tucker conditions, which describe necessary conditions for constrained optimizers:

KKT Necessary Conditions for Local Optima: Under some regularity conditions on F and $\{h_i, g_j\}$, if $\underline{x}^*$ is a solution to the primal, then there exist $\underline{\lambda}, \underline{\mu}$ such that

- $\nabla F(\underline{x}^*) + \sum_{i=1}^k \lambda_i \nabla h_i(\underline{x}^*) + \sum_{j=1}^l \mu_j g_j(\underline{x}^*) = 0$
- $h_i(\underline{x}^*) = 0$ for all $i = 1, \ldots, k$
- $g_j(\underline{x}^*) \leq 0$ for all $j = 1, \ldots, l$
- $\mu_j \geq 0$ for all $j = 1, \ldots, l$
- $\mu_j g_j(\underline{x}^*) = 0$ for all $j = 1, \ldots, l$

This generalizes the 'classical' Lagrange multiplier conditions for constrained optimization.

3.1 Deriving the Dual SVM

Consider the primal problem for the SVM:

$$\begin{aligned} &\min_{\underline{w}, b} \frac{1}{2}||\underline{w}||^2 \\ &\text{(s.t.) } \forall i = 1, \ldots m : y^i(\underline{w}.\underline{x}^i + b) \geq 1. \end{aligned} \tag{63}$$

The Lagrangian in this case, observing no equality constraints, would be given as

$$\begin{aligned} L(\underline{w}, b, \underline{\mu}) &= \frac{1}{2}||\underline{w}||^2 + \sum_{i=1}^m \mu_i(1 - y^i(\underline{w}.\underline{x}^i + b)) \\ &= \frac{1}{2}||\underline{w}||^2 + \sum_{i=1}^m \mu_i - \underline{w}.\left(\sum_{i=1}^m \mu_i y^i \underline{x}^i\right) - b\sum_{i=1}^m \mu_i y^i. \end{aligned} \tag{64}$$

The dual function is defined to be $Q(\underline{\mu}) = \min_{\underline{w}, b} L(\underline{w}, b, \underline{\mu})$. To compute the dual function, we want to minimize the above for $\underline{w}, b$ in an unconstrained way. In particular, we want the derivatives with respect to all these values to be zero:

$$\begin{aligned} \nabla_{\underline{w}} L &= \underline{w} - \sum_{i=1}^m \mu_i y^i \underline{x}^i \\ \partial L/\partial b &= -\sum_{i=1}^m \mu_i y^i. \end{aligned} \tag{65}$$

Hence, in order for a minimum to exist, we want

$$\begin{aligned} \underline{w}^* &= \sum_{i=1}^m \mu_i y^i \underline{x}^i \\ 0 &= \sum_{i=1}^m \mu_i y^i. \end{aligned} \tag{66}$$

The importance of this second equation is somewhat odd as it doesn't actually specify b at all. However, if this condition is not true, then the derivative of L with respect to b is *never* zero - as a result, L could always be reduced by choosing smaller or large values of b (depending on the sign) and the minimum would not exist, or rather would be $-\infty$. This condition ensures that a minimum with respect to b exists, and informs the choice of feasible $\{\mu_i\}$ we can take.

Substituting in this value of $\underline{w}^*$ into the Lagrangian, we get

$$\begin{aligned} L(\underline{w}^*, b, \underline{\mu}) &= \frac{1}{2}||\sum_{i=1}^m \mu_i y^i \underline{x}^i||^2 + \sum_{i=1}^m \mu_i - \left(\sum_{i=1}^m \mu_i y^i \underline{x}^i\right) \cdot \left(\sum_{i=1}^m \mu_i y^i \underline{x}^i\right) - b\left(\sum_{i=1}^m \mu_i y^i\right) \\ &= \sum_{i=1}^m \mu_i - \frac{1}{2}||\sum_{i=1}^m \mu_i y^i \underline{x}^i||^2 - b\left(\sum_{i=1}^m \mu_i y^i\right). \end{aligned} \tag{67}$$

With this, we can then express the dual function succinctly:

$$Q(\underline{\mu}) = \begin{cases} -\infty & \text{if } \sum_{i=1}^m \mu_i y^i \neq 0 \\ \sum_{i=1}^m \mu_i - \frac{1}{2}||\sum_{i=1}^m \mu_i y^i \underline{x}^i||^2 & \text{if } \sum_{i=1}^m \mu_i y^i = 0. \end{cases} \tag{68}$$

We see that when maximizing Q, we can discard any $\underline{\mu}$ such that $\sum_i \mu_i y^i \neq 0$. We can now express the dual problem, expanding out that norm:

The Dual SVM: Given a data set $\{(\underline{x}^i, y^i)\}_{i=1,\ldots,m}$, solve the following optimization problem

$$\begin{aligned} \max_{\underline{\mu} \in \mathbb{R}^m} \; & \sum_{i=1}^m \mu_i - \frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \mu_i y^i (\underline{x}^i . \underline{x}^j) y^j \mu_j \\ \text{(s.t.)} \; & \sum_{i=1}^m \mu_i y^i = 0 \\ & \forall i = 1, \ldots, m : \mu_i \geq 0. \end{aligned} \tag{69}$$

The previous result on duality gaps for convex functions with linear constraints (as is the case here) tells us that the optimal value of the dual SVM problem is equivalent to the optimal value of the primal SVM problem: $F(\underline{w}^*, b^*) = Q(\underline{\mu}^*)$.

3.2 Reconstructing the Primal SVM

Delaying solving the dual SVM for the $\underline{\mu}^*$, we know that $Q(\underline{\mu}^*) = F(\underline{w}^*, b^*)$ for optimal primal values $\underline{w}^*, b^*$. If we solve the dual SVM problem, this tells us the optimal value of the primal SVM problem - but what are the optimal values of $\underline{w}^*$ and b^*?

Based on the derivation of the dual from the primal, the 'link' between the two problems occurred when

$$\underline{w}^* = \sum_{i=1}^m \mu_i^* y^i \underline{x}^i. \tag{70}$$

This gives us a method for reconstructing the optimal weight vector. But what about the optimal bias value b^*? The problem is that the dependence on b was effectively eliminated, through $\sum_i \mu_i y^i = 0$ condition - the b term falls out of the Lagrangian under this condition. In some sense, any value of b will do.

Except that b needs to be **feasible** - it needs to satisfy the conditions of the primal problem, i.e., for all $i = 1, \ldots, m$:

$$1 - y^i(\underline{w}^* . \underline{x}^i + b) \leq 0. \tag{71}$$

Looking back at the previous discussion of active vs inactive constraints, for any $\mu_i^* > 0$, we have that the corresponding constraint is satisfied *with equality*, i.e.,

$$1 - y^i(\underline{w}^* . \underline{x}^i + b) = 0. \tag{72}$$

This corresponds to training data points that lie 'on the margin', as close to the separator as possible. These are the support vectors. The discussion of active versus inactive constraints translates directly in the SVM case to the following result:

If $\mu_i^* > 0$, then $(\underline{x}^j, y^j)$ is a support vector that lies exactly on the margin of the separator.

We can use this to identify the optimal bias value b^* - having identified a support vector using the $\mu_i^* > 0$ condition, we can solve for b:

$$b^* = y^i - \underline{w}^* . \underline{x}^i. \tag{73}$$

We may therefore reconstruct the solution to the primal in the following way:

Given a solution $\underline{\mu}^*$ to the Dual SVM problem, the solution to the primal problem is given by

$$\underline{w}^* = \sum_{i=1}^{m} \mu_i^* y^i \underline{x}^i, \tag{74}$$

and, for any support vector $(\underline{x}^i, y^i)$ for which $\mu_i^* > 0$,

$$b^* = y^i - \underline{w}^* . \underline{x}^i. \tag{75}$$

3.3 Solving the Dual SVM: SMO

But how to solve the Dual SVM and get the optimal $\underline{\mu}^*$ values (if solutions exist)?

$$\begin{aligned} \max_{\underline{\mu} \in \mathbb{R}^m} \; & \sum_{i=1}^{m} \mu_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \mu_i y^i (\underline{x}^i . \underline{x}^j) y^j \mu_j \\ \text{(s.t.)} \; & \sum_{i=1}^{m} \mu_i y^i = 0 \\ & \forall i = 1, \ldots, m : \mu_i \geq 0. \end{aligned} \tag{76}$$

We cannot apply techniques like gradient descent here, because gradient descent on a feasible iterate may carry you outside the constraint set - violating either the positivity, or the linear constraint. Projected gradient descent can be difficult as well - while projecting onto the linear constraint is reasonable (as discussed previously), the positivity constraints additionally complicate the issue.

Coordinate descent is also problematic here: while coordinate descent makes it easier to control and satisfy the positivity constraints, selecting and modifying a single coordinate would violate the linear constraint.

Penalty and barrier methods could be applied here, but a classical approach is to modify coordinate descent: this is the **SMO Algorithm**. For a proposed feasible value $\underline{\mu}$, instead of selecting and modifying a single coordinate,

consider modifying *pairs* of coordinates. Modifying two coordinates at a time, any decrease in one can be made up for with an increase (or decrease) in the other that maintains the equality constraint.

Suppose that $\underline{\mu}$ is a proposed feasible solution - all coordinates are positive, the linear constraint is satisfied. We would like to find new values for μ_i, μ_j that increase the value of the objective function while maintaining the equality constraint. If i is a positive class point and j is a negative class point, any increase or decrease in μ_i must be matched with a corresponding increase or decrease in μ_j to balance it out. If i is a positive class point and j is also a positive class point (or both are negative class points), any increase or decrease in μ_i must be matched with a corresponding decrease or increase in μ_j.

To summarize, if we modify μ_i to $\mu_i + \epsilon$, we need to modify μ_j to $\mu_j - y^i y^j \epsilon$. In addition however, we need to worry about the positivity constraint, so we only accept ϵ such that

$$\begin{aligned} \mu_i + \epsilon &\geq 0 \\ \mu_j - y^i y^j \epsilon &\geq 0. \end{aligned} \tag{77}$$

To find the 'optimal' value of ϵ to modify these two coordinates, we thus want to solve (introducing the notation $K_{i,j} = y^i(\underline{x}^i.\underline{x}^j)y^j$)

$$\begin{aligned} \max_{\epsilon \in \mathbb{R}} & \left[(\mu_i + \epsilon) + (\mu_j - y^i y^j \epsilon) + \sum_{i' \neq i,j} \mu_{i'} \right] \\ & - \frac{1}{2}\left[(\mu_i + \epsilon)^2 K_{i,i} + (\mu_j - y^i y^j \epsilon)^2 K_{j,j} + 2(\mu_i + \epsilon) K_{i,j} (\mu_j - y^i y^j \epsilon) \right] \\ & - \frac{1}{2}\left[2(\mu_i + \epsilon) \sum_{k \neq i,j} K_{i,k}\mu_k + 2(\mu_j - y^i y^j \epsilon) \sum_{k \neq i,j} K_{j,k}\mu_k + \sum_{k \neq i,j} \sum_{h \neq i,j} \mu_k K_{k,h} \mu_h \right] \\ \text{(s.t.)} \ & \mu_i + \epsilon \geq 0 \\ & \mu_j - y^i y^j \epsilon \geq 0. \end{aligned} \tag{78}$$

Simplifying, we can eliminate any terms from the objective function that don't have to do with ϵ:

$$\begin{aligned} \max_{\epsilon \in \mathbb{R}} & \ \epsilon(1 - y^i y^j) - \frac{1}{2}\epsilon^2 \left[K_{i,i} + 2y^i y^j K_{i,j} + K_{j,j} \right] - \epsilon \left[\sum_{k=1}^{m} K_{i,k}\mu_k - (y^i y^j) \sum_{k=1}^{m} K_{j,k}\mu_k \right] \\ \text{(s.t.)} \ & \mu_i + \epsilon \geq 0 \\ & \mu_j - y^i y^j \epsilon \geq 0. \end{aligned} \tag{79}$$

Let $\epsilon_{i,j}(\underline{\mu})$ be the solution to the above optimization problem - quadratic objective, one variable, two linear constraints, this is relatively easy to solve. SMO then takes the following iteration:

- Propose a feasible $\underline{\mu}^0$.
- Pick two coordinates i, j, compute the solution $\epsilon_t = \epsilon_{i,j}(\underline{\mu}^t)$.
- Update $\underline{\mu}^{t+1}$ with

$$\mu_k^{t+1} = \begin{cases} \mu_i^t + \epsilon_t & \text{if } k = i \\ \mu_j^t - y^i y^j \epsilon_t & \text{if } k = j \\ \mu_k^t & \text{else.} \end{cases} \tag{80}$$

- Iterate until convergence.