Sequential Minimal Optimization

Seth Terashima

April 23, 2012

The problem

The story so far:

- We've had fun mathing our way to the dual, but...
- It would be nice if we could actually do something with it.

So let's take a look at Sequential Minimal Optimization.

The problem

We want to find λ that minimizes

$$\Psi(\lambda) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \langle x_i, x_j \rangle \lambda_i \lambda_j - \sum_{i=1}^{N} \lambda_i$$

subject to the constraints

$$0 \leq \lambda_i \leq C \text{ (for all } i) \qquad \text{and} \qquad \sum_{i=1}^{N} y_i \lambda_i = 0.$$

Each $y_i = \pm 1$ is the class of the training data x_i, each λ_i is the corresponding Lagrange multiplier, and C controls how "soft" we are willing to let the margin be.

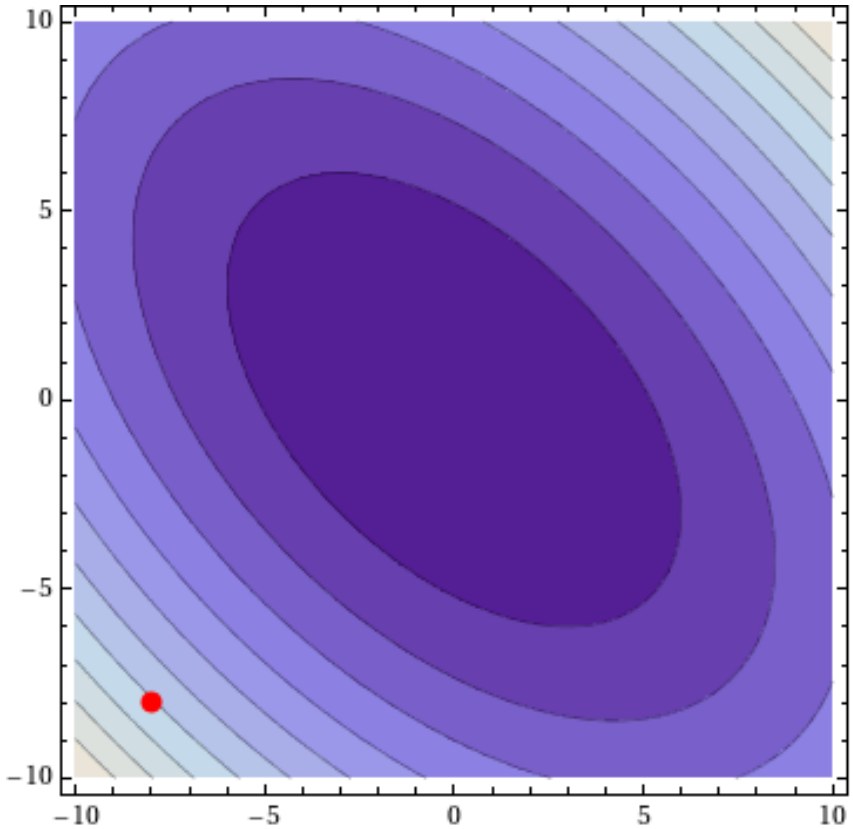
A solution for the constraint-free case

We can minimize $F(\lambda_1, \ldots, \lambda_n)$ one coordinate at a time. Starting with some point λ,

- Choose some coordinate $j \in \{1, 2, \ldots, n\}$
- View F as a single-variable function of λ_j by fixing the other $n - 1$ inputs
- Minimize F with respect to λ_j
- Update λ by setting λ_j to its optimal value, then repeat the process for other values of j

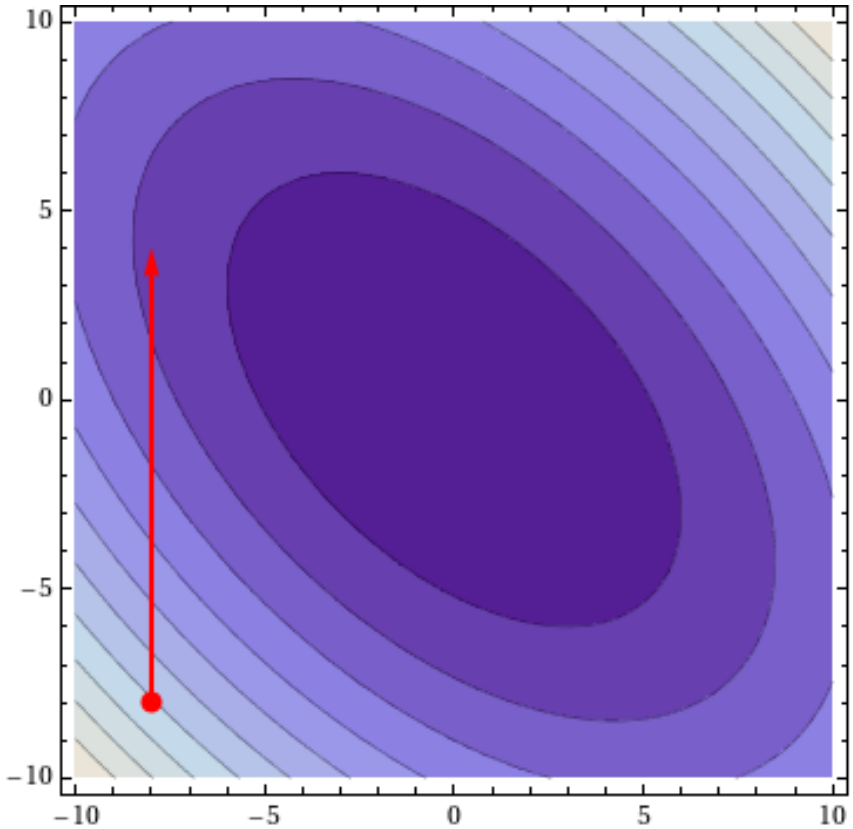
Example

$$F(x, y) = x^2 + xy + y^2$$



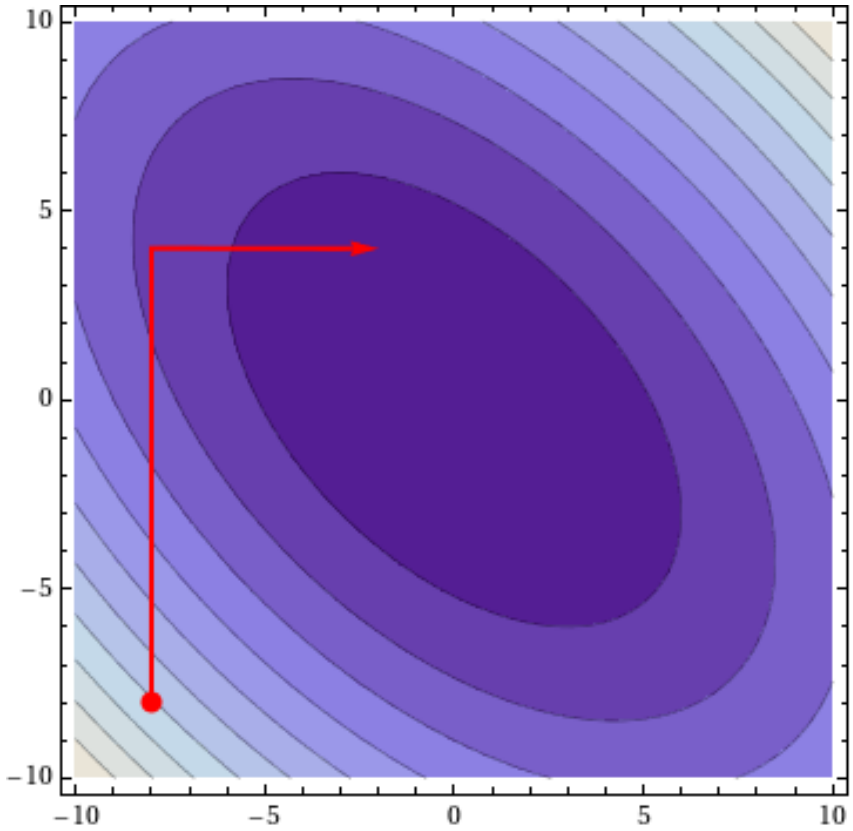
Example

$$F(x, y) = x^2 + xy + y^2$$



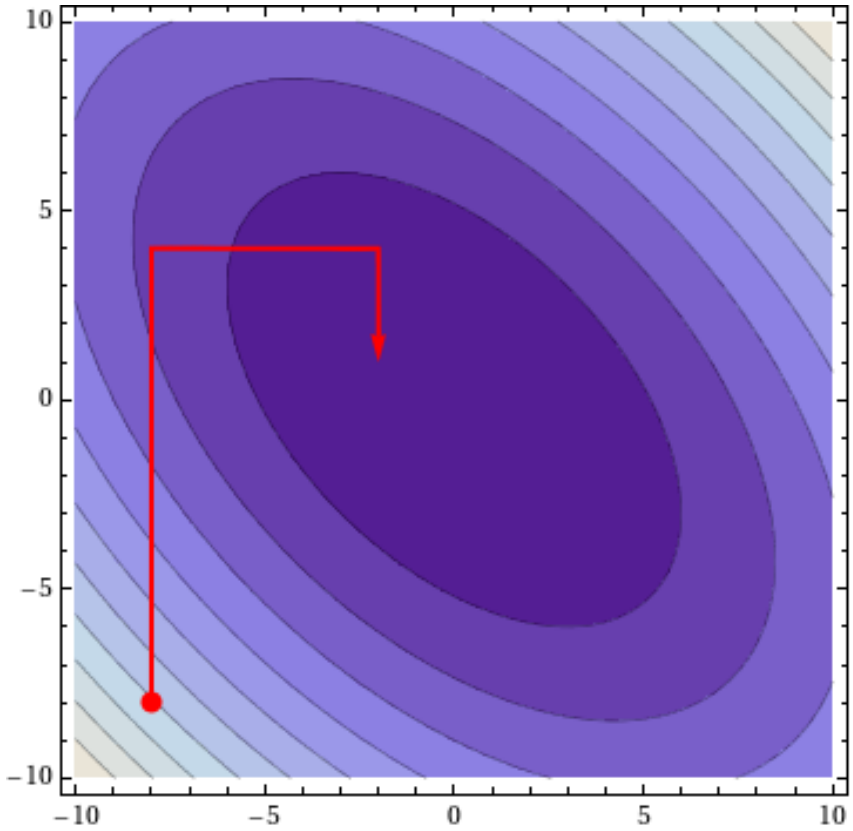
Example

$$F(x, y) = x^2 + xy + y^2$$



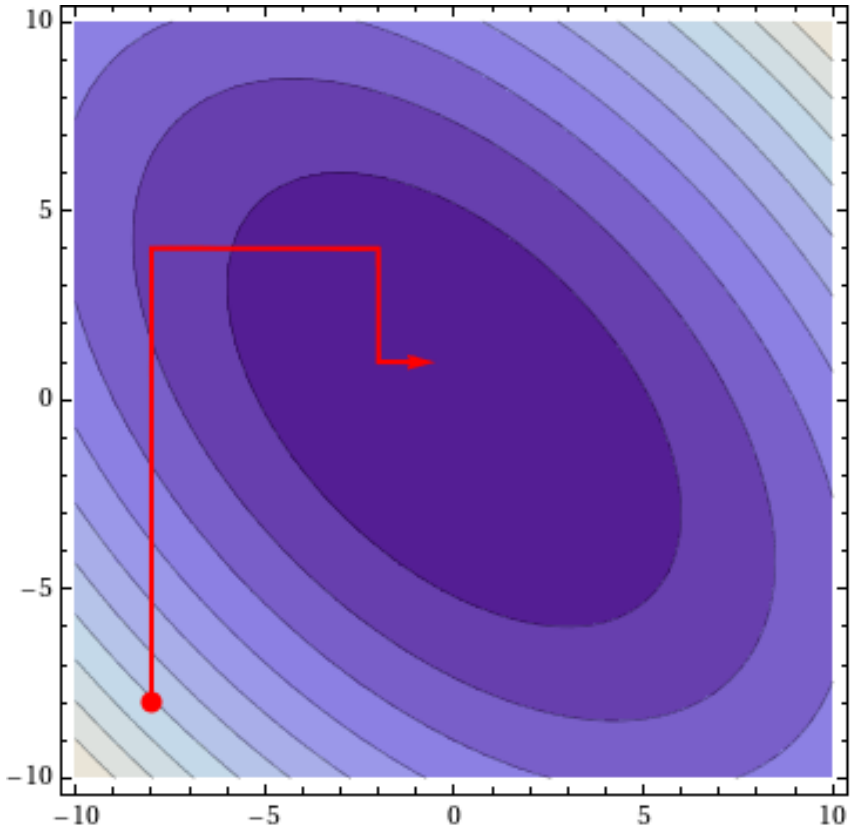
Example

$$F(x, y) = x^2 + xy + y^2$$



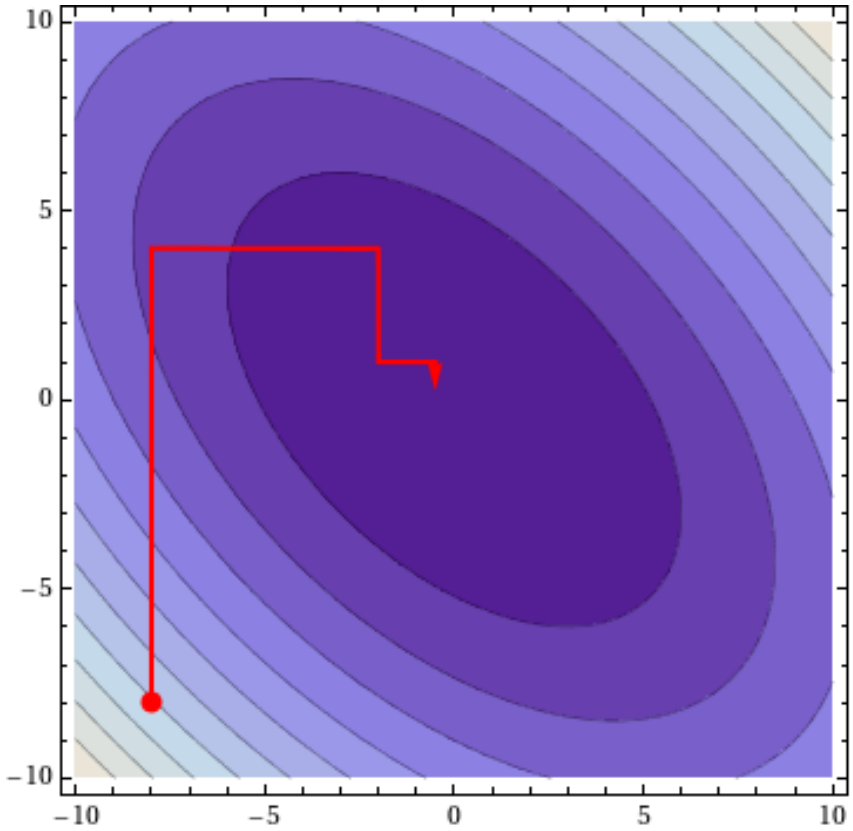
Example

$$F(x, y) = x^2 + xy + y^2$$



Example

$$F(x, y) = x^2 + xy + y^2$$



Great, but the solution doesn't meet the constraints.

First constraint

Our first constraint is $\sum_{i=1}^{N} y_i \lambda_i = 0$. The fix: Substitution.

1. Choose two coordinates, j and i.
2. Solve for λ_j in terms of λ_i (and the other multipliers):

$$\lambda_i = -\frac{1}{y_i} \sum_{k \neq i} y_k \lambda_k = -\frac{y_j}{y_i} \lambda_j + \text{garbage}$$

3. We are now back to optimizing a single-variable function.

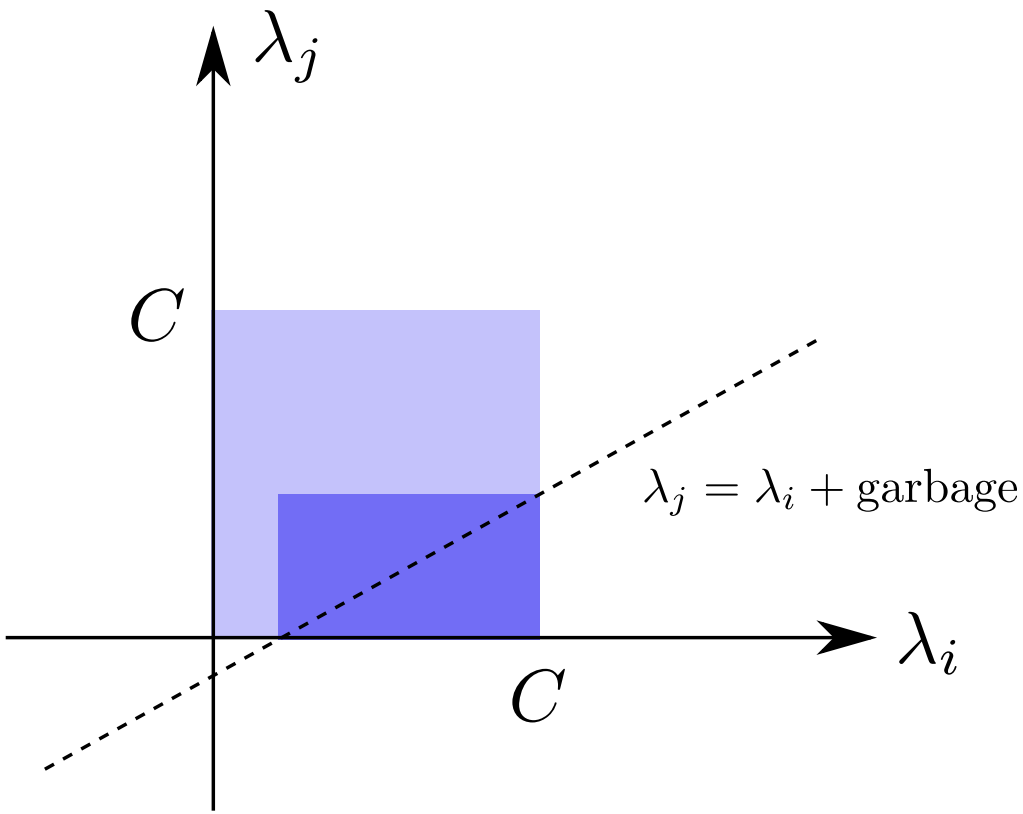
E.g., if $j = 1$, $i = 2$, and $y_1 = -y_2$, then

$$f(\lambda_1) = F(\lambda_1, \lambda_1 + \text{garbage}, \lambda_3, \ldots, \lambda_N)$$

meets the first constraint for all values of λ_1.

Second constraint

The second constraint says that for all i, $0 \leq \lambda_i \leq C$.



This is just a boundary condition. (Slope could be negative.)

To recap, we are trying to minimize

$$\Psi(\lambda) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \langle x_i, x_j \rangle \lambda_i \lambda_j - \sum_{i=1}^{N} \lambda_i$$

one coordinate at a time (but also changing a second coordinate to meet the linear constraint). When $j = 1$, $i = 2$, and $y_1 = -y_2$, we are minimizing

$$\begin{aligned} f(\lambda_1) &= \Psi(\lambda_1, \lambda_1 + \text{garbage}, \lambda_3, \ldots, \lambda_N) \\ &= c_2\lambda_1^2 + c_1\lambda_1 + c_0. \end{aligned}$$

We can do this analytically (read: quickly)!

Concavity given by second derivative:

$$f''(\lambda_1) = \langle x_1, x_1 \rangle + \langle x_2, x_2 \rangle - 2\langle x_1, x_2 \rangle$$

If this is positive, find global minimum

$$\lambda_2' = \lambda_2 + \frac{y_2(E_1 - E_2)}{f''(\lambda_1)}$$

(where $E_k = \hat{y}_k - y_k$), then use closest λ_1^{new} allowed by boundary conditions. Set

$$\lambda^{\text{new}} = (\lambda_1^{\text{new}}, \lambda_1^{\text{new}} + \text{garbage}, \lambda_3, \ldots, \lambda_N).$$

Choose new values for i, j, rinse, repeat.

So how do we choose j and i for each iteration?

- There is not a clear-cut solution
- We need some heuristics

And how do we decide when we're done?

- Knowing your destination is a good first step towards getting there.

Choosing j

Choosing j:

- A solution value for λ has the following properties (the KKT conditions):

$$\lambda_j = 0 \implies y_j \hat{y}_j \geq 1 \quad (1)$$

$$\lambda_j = C \implies y_j \hat{y}_j \leq 1 \quad (2)$$

$$0 < \lambda_j < C \implies y_j \hat{y}_j = 1 \quad (3)$$

- We just want to be "close enough" (within $\varepsilon \approx 0.001$) for all j.
- If there is some j that violates these, j is a candidate for optimization.
- Priority given to "unbound" multipliers (when $0 < \lambda_j < C$)
- Multipliers tend to become bound over time (why?)

Choosing i

- Recall that the global minimum of $f(\lambda_j)$ has value

$$\lambda_i' = \lambda_i + \frac{y_i(E_j - E_i)}{f''(\lambda_j)}.$$

- After choosing j, we choose i that maximizes $|E_j - E_i|$.
- Intuitively, this heurristic helps "move" λ_i by a large amount each iteration.

Recomputing the offset

- Our model is $\hat{y} = w \cdot x - b$
- Although b is not part of the dual (why not?), we need b to evaluate E_k and the KTT conditions
- After each iteration, we update b to be halfway between the values that would make x_i and x_j support vectors

Benchmarks

- Algorithms completed when all KKT conditions met within $\varepsilon = 0.001$
  The chunking algorithm used in the benchmark used a different convergence condition, but Platt was conservative.
- SMO showed better scaling than chunking, usually by a factor of N
- SMO time dominated by SVM evaluations — very fast with linear SVMs

SMO performed over a 1000 times faster than contemporary state-of-the-art alternatives on real-world data. Not bad.

Conclusion

- We needed an efficient way to minimize the dual
- SMO accomplishes this by changing two multipliers at a time until the KKT conditions are met
- SMO is reasonably simple and very fast compared to previous methods
- Heuristics might be a good place to look for improvements