Lecture 76: Using an SVM continued

When the intersection of the line $\alpha_j = \alpha_j^* - s(\alpha_i - \alpha_i^*)$ with the box $0 \le \alpha_i, \alpha_j \le C$ is just a point, we cannot make progress:

* $s = -1$: $\begin{cases} \alpha_i^* = 0 \ \& \ \alpha_j^* = C \Rightarrow \alpha_j = \alpha_i + C \Rightarrow \alpha_i = 0 \ \& \ \alpha_j = C \\ \alpha_i^* = C \ \& \ \alpha_j^* = 0 \Rightarrow \alpha_j = \alpha_i - C \Rightarrow \alpha_i = C \ \& \ \alpha_j = 0 \end{cases}$

* $s = +1$: $\begin{cases} \alpha_i^* = 0 \ \& \ \alpha_j^* = 0 \Rightarrow \alpha_j = -\alpha_i \Rightarrow \alpha_i = 0 \ \& \ \alpha_j = 0 \\ \alpha_i^* = C \ \& \ \alpha_j^* = C \Rightarrow \alpha_j = -\alpha_i + 2C \Rightarrow \alpha_i = C \ \& \ \alpha_j = C \end{cases}$

These conditions can be summarized as $L = H$ (see figures on pages 95 & 96 on Notebook 2). Therefore, we should first compute L & H, and if they're equal, skip the optimization of i & j. Now we know how to jointly optimize α_i & α_j analytically. Suppose we've optimize some pairs of α's. How do we know when to stop? In other words, how do know when the algorithm has converged to the optimal solution (within some numerical tolerance)? Recall that the KKT conditions are sufficient & necessary conditions for an optimal point of a positive definite QP problem. Therefore, we can use the KKT conditions to check for convergence to the optimal point. The KKT conditions for SVM were worked out on pages 74 & 75 of Notebook 2:

* $\alpha_i = 0 \Rightarrow y^{(i)}(w \cdot x^{(i)} + b) \geq 1$ (no margin violation)
* $\alpha_i = C \Rightarrow y^{(i)}(w \cdot x^{(i)} + b) \leq 1$ (margin violation)
* $0 < \alpha_i < C \Rightarrow y^{(i)}(w \cdot x^{(i)} + b) = 1$ (support vectors)

At every step: $w \cdot x^{(i)} + b = Z_i = E_i + y^{(i)}$

$$\Rightarrow y^{(i)}(w \cdot x^{(i)} + b) = y^{(i)} E_i + 1$$

$$\Rightarrow y^{(i)}(w \cdot x^{(i)} + b) - 1 = y^{(i)} E_i$$

Therefore, we can rewrite the KKT conditions as follows:

* $\alpha_i = 0 \Rightarrow y^{(i)} E_i \geq 0$
* $\alpha_i = C \Rightarrow y^{(i)} E_i \leq 0$
* $0 < \alpha_i < C \Rightarrow y^{(i)} E_i = 0$

Let ϵ be a small positive number (typically $\epsilon \sim 0.001$). We will consider the KKT conditions satisfied if:

* $\alpha_i = 0 \Rightarrow y^{(i)} E_i \geq -\epsilon$
* $\alpha_i = C \Rightarrow y^{(i)} E_i \leq \epsilon$
* $0 < \alpha_i < C \Rightarrow -\epsilon \leq y^{(i)} E_i \leq \epsilon$.

The following will help us later: for a given example i, the KKT conditions are violated if: $(y^{(i)} E_i < -\epsilon$ and $\alpha_i < C)$ OR $(y^{(i)} E_i > \epsilon$ and $\alpha_i > 0)$.

After optimizing α_i & α_j, the threshold b can be chosen such that the KKT conditions are satisfied for examples i & j.

Suppose $0 < \alpha_i < C$ after optimization. Then:

$$y^{(i)}\left[\sum_{\substack{k=1 \\ k\neq i,j}}^{m} \alpha_k^* y^{(k)} K_{ki} + \alpha_i y^{(i)} K_{ii} + \alpha_j y^{(j)} K_{ji} + b\right] = 1$$

$$\Rightarrow Z_i - \alpha_i^* y^{(i)} K_{ii} - \alpha_j^* y^{(j)} K_{ji} - b^* + \alpha_i y^{(i)} K_{ii} + \alpha_j y^{(j)} K_{ji} + b = y^{(i)}$$

$$\Rightarrow b = b^* - E_i - y^{(i)} K_{ii} (\alpha_i - \alpha_i^*) - y^{(j)} K_{ji} (\alpha_j - \alpha_j^*)$$

Let's call this b_i; since obtained it by satisfying the KKT conditions for example i:

$$b_i = b^* - E_i - y^{(i)} K_{ii} (\alpha_i - \alpha_i^*) - y^{(j)} K_{ji} (\alpha_j - \alpha_j^*)$$

Similarly, if $0 < \alpha_j < C$ after optimization, b_j will ensure the j^{th} example satisfies the KKT condition:

$$b_j = b^* - E_j - y^{(j)} K_{jj} (\alpha_j - \alpha_j^*) - y^{(i)} K_{ij} (\alpha_i - \alpha_i^*)$$

Claim: if $0 < \alpha_i < C$ and $0 < \alpha_j < C$; $b_i = b_j$.

Proof: $b_j = b^* - E_j - y^{(i)} K_{ij} [-s(\alpha_i - \alpha_i^*)] - y^{(i)} K_{ij} (\alpha_i - \alpha_i^*)$

$$= b^* - E_j - y^{(i)} (\alpha_i - \alpha_i^*) (K_{ij} - K_{jj})$$

$$b_i = b^* - E_i - y^{(i)} K_{ii} (\alpha_i - \alpha_i^*) - y^{(j)} K_{ji} [-s(\alpha_i - \alpha_i^*)]$$

$$= b^* - E_i - y^{(i)} (\alpha_i - \alpha_i^*) (K_{ii} - K_{ji})$$

$$b_i - b_j = E_j - E_i - y^{(i)}(\alpha_i - \alpha_i^*)(K_{ii} + K_{jj} - 2K_{ij})$$
$$= E_j - E_i - \eta\, y^{(i)}(\alpha_i - \alpha_i^*)$$

Recall that $\alpha_i^{\text{unclipped}} = \alpha_i^* + \dfrac{y^{(i)}(E_j - E_i)}{\eta} \Rightarrow E_j - E_i = \eta\, y^{(i)}(\alpha_i^{\text{unclipped}} - \alpha_i^*)$

Finally: $$\boxed{b_i - b_j = \eta\, y^{(i)}(\alpha_i^{\text{unclipped}} - \alpha_i)}$$

Note that this result is always true, regardless of the values of α_i & α_j. When $0 < \alpha_i, \alpha_j < C$, $\alpha_i = \alpha_i^{\text{unclipped}} \Rightarrow b_i = b_j$.

claim: If $0 < \alpha_i < C$ & $\alpha_j = 0$, setting $b = b_i$ ensures both i & j satisfy the KKT conditions.

Proof: That the i^{th} example satisfies the KKT condition is true by the very definition of b_i. Since $\alpha_j = 0$, we need to prove that $y^{(j)}(w \cdot x^{(j)} + b_i) \geq 1$. Note that $w \cdot x^{(j)} = -b_j + y^{(j)}$

$$\Rightarrow y^{(j)}(w \cdot x^{(j)} + b_i) - 1 = y^{(j)}(-b_j + y^{(j)} + b_i) - 1 = y^{(j)}(b_i - b_j)$$
$$= y^{(j)}\eta\, y^{(i)}(\alpha_i^{\text{unclipped}} - \alpha_i)$$
$$= s\eta(\alpha_i^{\text{unclipped}} - \alpha_i)$$

$$\boxed{y^{(j)}(w \cdot x^{(j)} + b_i) - 1 = s\eta(\alpha_i^{\text{unclipped}} - \alpha_i)} \quad \textcircled{\#}$$

When $s = 1$ & $\alpha_j = 0$: $\alpha_i^{\text{unclipped}} - \alpha_i \geq 0$ (see page 96 of Notebook 2)

When $s = -1$ & $\alpha_j = 0$: $\alpha_i^{\text{unclipped}} - \alpha_i \leq 0$ (see page 95 of Notebook 2)

Therefore: $y^{(j)}(w \cdot x^{(j)} + b_i) - 1 \geq 0$, which proves our claim.

We can extend this result also to the case where $\alpha_j = C$:

When $\begin{cases} s=1 \text{ \& } \alpha_j = C: & \alpha_i^{\text{unclipped}} - \alpha_i \leq 0 \\ s=-1 \text{ \& } \alpha_j = C: & \alpha_i^{\text{unclipped}} - \alpha_i \geq 0 \end{cases}$

Therefore: $y^{(i)}(w \cdot x^{(i)} + b_i) - 1 \leq 0$, which proves our claim.

Summary so far: if $0 < \alpha_i < C$, setting $b = b_i$ ensures both i^{th} & j^{th} examples satisfy the KKT conditions. We can easily show that if $0 < \alpha_j < C$, setting $b = b_j$ ensures both i^{th} & j^{th} examples satisfy the KKT conditions:

$$\begin{aligned} -1 + y^{(i)}(w \cdot x^{(i)} + b_j) &= y^{(i)}(-b_i + y^{(i)} + b_j) - 1 \\ &= y^{(i)}(b_j - b_i) \\ &= -\eta(\alpha_i^{\text{unclipped}} - \alpha_i) \end{aligned}$$

$$y^{(i)}(w \cdot x^{(i)} + b_j) - 1 = \eta(\alpha_i - \alpha_i^{\text{unclipped}})$$

When $\begin{cases} \alpha_i = 0 \Rightarrow \alpha_i - \alpha_i^{\text{unclipped}} \geq 0 \Rightarrow y^{(i)}(w \cdot x^{(i)} + b_j) \geq 1 \checkmark \\ \alpha_i = C \Rightarrow \alpha_i - \alpha_i^{\text{unclipped}} \leq 0 \Rightarrow y^{(i)}(w \cdot x^{(i)} + b_j) \leq 1 \checkmark \end{cases}$

What happens if both α_i & α_j are at the bounds, i.e $\alpha_i \in \{0, C\}$ and $\alpha_j \in \{0, C\}$? We won't consider the cases where $L = H$, since in such cases we don't make any progress on α_i & α_j and simply skip the optimization of i & j.

Claim: If both α_i & α_j are at the bounds and $L \neq H$, setting b to any value between b_i & b_j ensures all KKT conditions are satisfied

proof: Let $b = t b_i + (1-t) b_j$ where $0 \leq t \leq 1$.

$$\begin{aligned} y^{(i)}(w \cdot x^{(i)} + b) - 1 &= y^{(i)}\left[-b_i + y^{(i)} + t b_i + (1-t) b_j\right] - 1 \\ &= y^{(i)}\left[b_i(t-1) + (1-t) b_j\right] \\ &= y^{(i)}(1-t)(b_j - b_i) \\ &= \eta(1-t)(\alpha_i - \alpha_i^{\text{unclipped}}) \end{aligned}$$

when $\begin{cases} \alpha_i = 0 \Rightarrow \alpha_i - \alpha_i^{\text{unclipped}} \geq 0 & \text{(assuming } L \neq H\text{)} \\ \alpha_i = C \Rightarrow \alpha_i - \alpha_i^{\text{unclipped}} \leq 0 & \text{(assuming } L \neq H\text{)} \end{cases}$

Therefore: $\begin{cases} \alpha_i = 0 \Rightarrow y^{(i)}(w \cdot x^{(i)} + b) \geq 1 \ \checkmark \\ \alpha_i = C \Rightarrow y^{(i)}(w \cdot x^{(i)} + b) \leq 1 \ \checkmark \end{cases}$

$$\begin{aligned} y^{(j)}(w \cdot x^{(j)} + b) - 1 &= y^{(j)}\left[-b_j + y^{(j)} + t b_i + (1-t) b_j\right] - 1 \\ &= y^{(j)}\left[-t b_j + t b_i\right] \\ &= y^{(j)} t (b_i - b_j) \\ &= \eta s t (\alpha_i^{\text{unclipped}} - \alpha_i) \end{aligned}$$

when $\begin{cases} \alpha_j = 0: \begin{cases} s = -1: \alpha_i^{\text{unclipped}} - \alpha_i \leq 0 & \text{(assuming } L \neq H, \text{ i.e } \alpha_i \neq C\text{)} \\ s = +1: \alpha_i^{\text{unclipped}} - \alpha_i \geq 0 & \text{(assuming } L \neq H, \text{ i.e } \alpha_i \neq 0\text{)} \end{cases} \\ \alpha_j = C: \begin{cases} s = -1: \alpha_i^{\text{unclipped}} - \alpha_i \geq 0 & \text{(assuming } L \neq H, \text{ i.e } \alpha_i \neq 0\text{)} \\ s = +1: \alpha_i^{\text{unclipped}} - \alpha_i \leq 0 & \text{(assuming } L \neq H, \text{ i.e } \alpha_i \neq C\text{)} \end{cases} \end{cases}$

Therefore: $\begin{cases} \alpha_j = 0 \Rightarrow y^{(j)}(w \cdot x^{(i)} + b) \geq 1 & \checkmark \\ \alpha_j = C \Rightarrow y^{(j)}(w \cdot x^{(i)} + b) \leq 1 & \checkmark \end{cases}$

Summary of determining the threshold b:

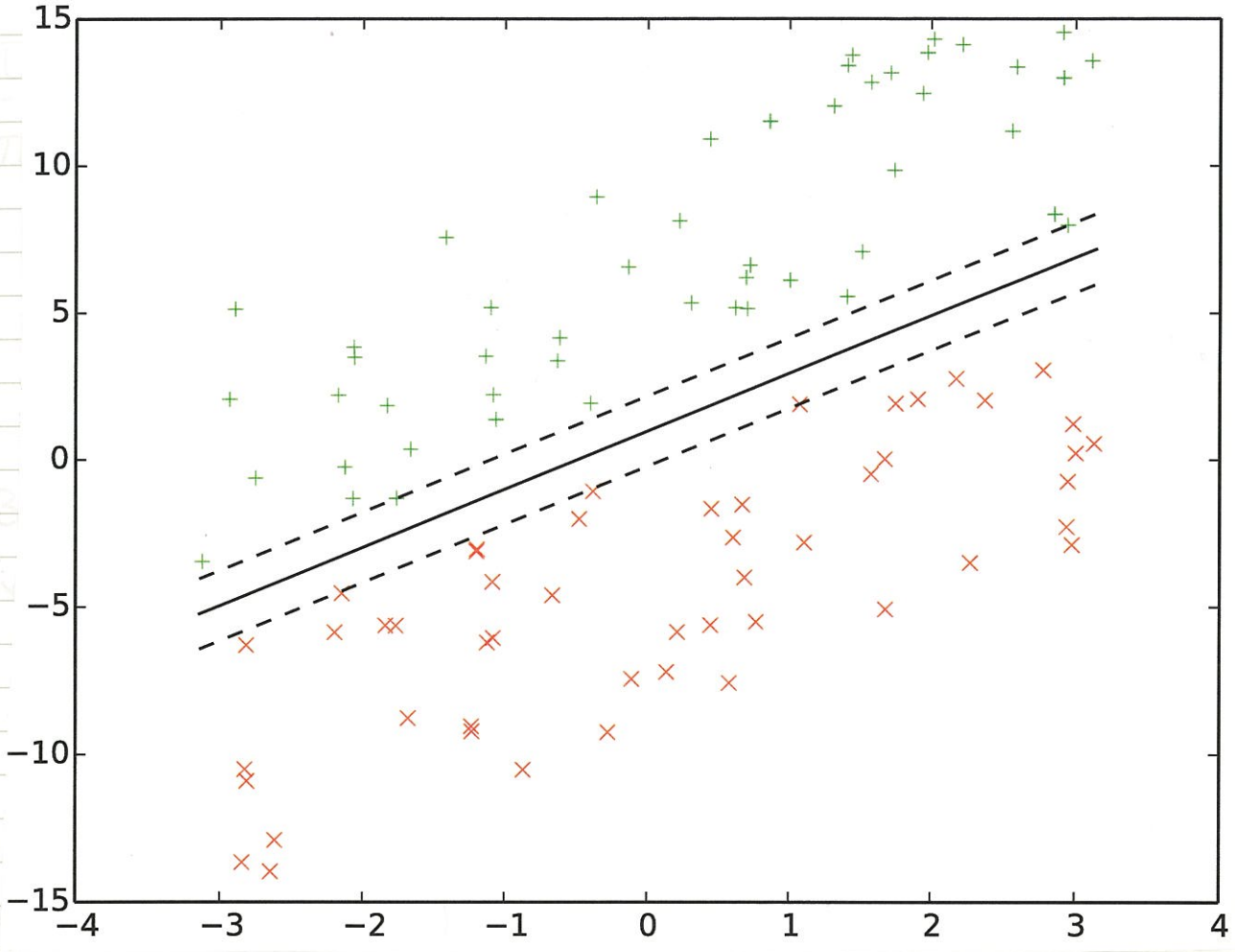
$$b = \begin{cases} b_i & \text{if } 0 < \alpha_i < C \\ b_j & \text{if } 0 < \alpha_j < C \\ \frac{b_i + b_j}{2} & \text{if } \alpha_i \& \alpha_j \text{ at bounds } \& \ L \neq H. \end{cases}$$

Note that in the last case we set b to the average of b_i & b_j, since it's in between b_i & b_j. We should make sure to check the condition $L == H$ before α_i's & b are determined, and continue to the next pair if the condition is true.

We now know how to optimize a pair of α's and pick b so that the KKT conditions are conditions for both examples. There's a theorem due to Osuna et al (1997) which guarantees convergence if at every step, at least one of the examples violates the KKT conditions prior to optimization. Much of the full SMO algorithm is dedicated to heuristics for choosing which α_i & α_j to optimize so as to maximize the objective function as much as possible. For large data sets, this is critical for the speed of the algorithm, since there are

$\binom{m}{2} = \frac{m(m-1)}{2}$ choices for α_i & α_j, and some will result in much less improvement than others. I will not go through all the heuristics in the full SMO algorithm, and instead pick a simple heuristic that works for toy examples: Iterate over $i = 1, \dots, m$, if α_i violates the KKT conditions, continue to pick α_j by choosing randomly from the remaining $m-1$ α's. Optimize α_i & α_j as discussed and repeat until all α_i examples satisfy the KKT conditions:

* Initialize $\alpha_i = 0 \ \forall i$ & $b = 0$ (Note that both $\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$ & $0 \le \alpha_i \le C$ are satisfied with this initialization)
* While True:
    - KKT_violations = 0
    - for $i = 1, \dots, m$:
        - Calculate E_i (see pgs 92 & 93 of Notebook #2)
        - $r = E_i y^{(i)}$
        - if ($r < -tol$ and $\alpha_i < C$) or ($r > tol$ and $\alpha_i > 0$):
            - KKT_violations += 1
            - select $j \neq i$ randomly
            - compute L & H (see page 96 of Notebook #2)
            - if L == H:
                continue to next i
            - Compute η (see page 93 of Notebook #2)
            - if $\eta <= 0$:
                continue to next i
            - compute new (clipped) α_i & α_j (see page 96 of Notebook #2)
            - compute b_i & b_j (see page 3) and then b (page 7)
    - if KKT_violations == 0:
        break



This is an example of applying the simplified SMO algorithm on the previous page using the linear kernel $K(x, z) = \langle x, z \rangle$ and $C = 10$. The number of examples is $m = 100$. This takes ~1-10 seconds to run. I tried increasing the # of examples to 500 & run time increased to about 2.5-3.5 minutes depending on the training set (which is randomly generated). As expected, size of the training set impacts SMO's run time a lot, which is why it's important

to use the full SMO algorithm in the real world.

Using SVM In Practice

There are two popular libraries for SVM:

(i) libsvm: Implements SMO in its full glory. The complexity is $O(m^2)$ or $O(m^3)$.

(ii) liblinear: Supports only linear SVM, but with complexity $O(m)$.

Because of their varying complexity, they should be used in different regimes: (As usual n = # of features m = # of training examples)

(i) n is large (relative to m), e.g. $n = 10{,}000$ & $m = 1000$: use linear SVM or logistic regression, otherwise we risk overfitting.

(ii) n is small, m is intermediate ($n = 1-1000$, $m = 10-10{,}000$):

* Use SVM with Gaussian Kernel. This is where the feature-space structure might be rich with non-linearities & where kernel SVMs shine.

* Apparently libsvm becomes painfully slow when $m \gtrsim 10{,}000$. This is not surprising, since we already know SMO is very sensitive to the size of the train set.

(iii) n is small, m is large ($n = 1-1000$, $m = 50{,}000+$):

* Create/add more features & use logistic regression or linear.

Apparently linear SVMs really shine when the # of features is large and comparable to the # of training example. One class of problems

which fall within this category is document classification, for instance mapping emails to spam/not spam. The # of features are usually $n \gtrsim 10,000$ (rough # of words under consideration) and training sizes of about $m \sim 50,000$ are not uncommon. Because the data is large and sparse, it's understandable there's not much gain from fitting non-linear models. I tried to understand why liblinear so much faster than libsvm. One thing I noticed is that the bias term b is ignored in liblinear. The claim is that the bias term doesn't matter much when there are many features & data is sparse. Remember that the linear constraint $\sum_{i=1}^{m} y^{(i)} \alpha_i = 0$ in the dual problem arises because of the bias term. If there's no bias term, this constraint is no longer present. When discussing SMO, we saw that the smallest # of α's that can be optimized together is 2, precisely because of this linear constraint. This in turn leads to many iterations because there are $\binom{m}{2}$ pairs to consider. There's also the fact that computing a non-linear Kernel $K(x, z)$ is probably more time consuming than just the dot product $x \cdot z$. A simple experiment in Python shows that the Gaussian kernel is at least 4 times slower to compute than the dot product. Actually, this doesn't explain why libsvm when used without a kernel is still slower than liblinear. Libsvm

doesn't use any special treatment for linear SVM. It's certainly conceivable that some tricks might cause speed up in the linear case. For instance, calculation of the SVM hypothesis (which we called Z in the derivation of SMO) is just a simple dot product & addition $(w \cdot x + b)$, but in the non-linear case it's a sum over all training set examples which involves Kernel computations. (constructing w from α's also doesn't involve Kernel computations).

Here's some more practical advice:

- Apply feature scaling before using the Gaussian Kernel, since we don't want to feed large numbers to an exponential.
- The Gaussian Kernel is the most popular non-linear Kernel.
- Multi-class classification: We can use one-vs-all method. The popular SVM libraries have support for this.