CS/CNS/EE 253: Advanced Topics in Machine Learning
Topic: Running time analysis for Offline and Online Optimization **Lecturer:** Andreas Krause
Scribe: Chris Kennelly **Date:** Jan. 20, 2010

5.1 Online versus Offline SVMs

We start with a review of the Offline Support Vector Machine. Recall that we want to build a linear separator of a bunch of different points. We want to draw a plane to separate the two, but we don't want just *any* linear separator. We want one that maximizes the margin between the two regions.

We express this as an optimization problem where we maximize the margin M

$$\max_{v,M} M \tag{5.1.1}$$

over the normal vector v parametrizing the decision boundary such that $||v|| = 1$ and $y \cdot v^\top x_s \geq M$.

We can say that $w = v \cdot M$ and make the substitution:

$$\max_{w,M} M \tag{5.1.2}$$

such that $||w|| = \frac{1}{M}$ and $\forall s\colon y_s \cdot w^\top x_s \geq 1$

We can then transform this into:

$$\min_{w} ||w||^2 \tag{5.1.3}$$

such that $\forall s\colon y_s \cdot w^\top x_s \geq 1$

This approach works when the data is separable but breaks when it isn't. We introduce slack variables into each constraint: $\forall s\colon y_s \cdot w^\top x_s \geq 1 - \epsilon_s$. To avoid allowing slack variables to dominate the entire model, we add a penalty factor to our objective function:

$$\min_{w,\epsilon} ||w||^2 + C \sum_s \epsilon_s \tag{5.1.4}$$

such that $\forall s\colon y_s \cdot w^\top x_s \geq 1 - \epsilon_s$ and $\epsilon_s \geq 0$ and $\lambda = \frac{1}{C}$.

This is the offline SVM (primal). This can be formulated with a new objective function to minimize hinge loss:

$$f(w) = ||w||^2 + \frac{1}{T} \sum_s^T \max\{0, 1 - y_s w^\top x_s\} \tag{5.1.5}$$

This is the objective function that we discussed last lecture; we can use online convex programming to optimize it, and call the resulting procedure the Online SVM (see Homework 1).

5.1.1 Running-time Analysis

How does this approach scale with dataset size? Typically, we take an algorithm, give it a dataset, and measure how long it takes. Generally, we would expect the running time to increase with data set size. In this lecture, we will, perhaps counterintuitively, see that in learning, running time can *decrease* with data set size.

In learning, we want to minimize the objective function. Most of the algorithms are iterative in nature and will–hopefully–converge to the optimal value. We can set a goal for a particular target error ϵ and stop early when we achieve it – thus, we may not have to process the entire data set.

If I fix a bound ϵ on the error, what is my running time to achieve this error? If I consider a fresh data point at every iteration, I should eventually reach an error level ϵ after a number of iterations and not see significant improvement *even if my dataset increases in size.*

How many iterations do I need to get a small training error?

In an offline setting, the entire training set can be examined to measure error expressed by

$$f(\tilde{w}) \leq \min_{w} f(w) + \epsilon_{acc},$$

where ϵ_{acc} is a bound on the approximation accuracy (that we allow ourselves when running the optimization procedure). There are a number of techniques to perform this optimization.

- Sequential minimum optimization (SMO)–a commonly technique–requires $\log \frac{1}{\epsilon_{acc}}$ iterations at a cost of m^2 per iteration (where m is the size of the training set).
- Interior point methods require $\log \log \frac{1}{\epsilon_{acc}}$ iterations at a cost of $m^{3.5}$ per iteration.

For online settings,

- The algorithm discussed in class gives an algorithm requiring $\frac{1}{\sqrt{\epsilon^2}}$ iterations, with constant cost per iteration
- The version in the Homework (called PEGASOS) improves that to $\frac{1}{\lambda\epsilon}$ iterations, again with constant cost per iteration

The online techniques have substantially lower costs per iteration at the expense of slower convergence rates. Armed with a million data points, these offline techniques become very computationally expensive. When is the break-even point between the algorithms?

The key idea is to not look at the *training* set error, but the *generalization* error (i.e., expected error on test set).

5.2 Generalization

Consider

$$f(w) = f_\lambda(w) = \lambda||w||^2 + \frac{1}{m}\sum_{i=1}^{m} l(w_i(x_i, y_i))$$

The training set is sampled i.i.d. from the data set. Suppose $(x_i, y_i) \sim P$. So the expected error of w is $l(w) = \mathbb{E}_{(x,y)\sim D} l(w_i(x, y))$. Ideally, we'd like to find $w^* = \arg\min l(w)$. One approach would be to use *empirical risk minimization*

$$w_E = \arg\min_W \hat{l}(w),$$

where $\hat{l}(w) = \frac{1}{m}\sum_{i=1}^{m} l(w_i(x_i, y_i))$.

Because of the law of large numbers, as we obtain more and more examples, $\hat{l}$ converges to l. Unfortunately, for small m, the variance of this estimator will be very large, and we will overfit.

Instead, we use *Regularized risk minimization*:

$$w_\lambda = \arg\min_w f_\lambda(w)$$

λ controls the trade-off between "goodness of fit" and model complexity.

Define

$$w_\lambda^* = \arg\min_w \mathbb{E}\left[f_\lambda(w)\right]$$

As we obtain more data, $l(w_\lambda^*)$ is fixed. $l(w_\lambda)$ will be worse than $l(w_\lambda^*)$ as it is based only on the data we have, but will converge to $l(w_\lambda^*)$ as more data is received.

This gives three sources of error:

- Approximation error: $l(w_\lambda^*)$, due to regularization ($\lambda > 0$, i.e., we're not minimizing the true loss l but f_λ).
- Estimation error: $l(w_\lambda) - l(w_\lambda^*)$, due to our limitations in being able to estimate error from limited data
- Training error: $l(\tilde{w}) - l(w_\lambda)$, due to running our optimization algorithm for a finite number of iterations (i.e., $\epsilon_{acc} > 0$).

We want to ensure our generalization error is at most ϵ. We can vary our model λ and ϵ_{acc} to reach that goal. As we get more and more data, we can be "sloppy" as our estimation error is sufficiently low. Even given unbounded running time, we may have too little data to obtain a desired ϵ; this is called the "data bounded regime."

Contrast the data bounded regime with the hypothetical scenerio of *infinite* data. Suppose $\exists w_0$, a hyperplane with large margin $M = \frac{1}{||w_0||}$ and low loss $l(w_0)$. This hyperplane will be optimal and we will attempt to approximate it by $\tilde{w}$:

$$l(\tilde{w}) = l(w_0) + \lambda\left(||w_0||^2 - ||\tilde{w}||^2\right) + \mathbb{E}\left[f(\tilde{w}) - f(w_0)\right]$$

Since this will be used to adjust how far we go in our optimization, we want to rewrite this equation in terms of otpimization error.

Theorem 5.2.1 *With probability* $\geq 1 - \delta$*:*

$$l(\tilde{w}) \leq l(w_0) + \lambda||w_0||^2 + \overbrace{2\left(\hat{f}(\tilde{w}) - f(w_0)\right)}^{\leq \epsilon_{acc}} + \frac{\log\frac{1}{\delta}}{\lambda m}$$

We want $|f(\tilde{w}) - f(w_0)| < \epsilon$. We can choose $\lambda, \epsilon_{acc}, m$ so that each component of error is bounded by $\frac{1}{3}\epsilon$. This gives the relations

$$\begin{aligned} \lambda &= O\left(\frac{\epsilon}{||w_0||^2}\right) \\ \epsilon_{acc} &= O\left(\epsilon\right) \\ m &= \Omega\left(\frac{||w_0^*||^2}{\epsilon^2}\right) \end{aligned}$$

If we have larger margins, we need fewer data points. If we want lower error, we need more data points.

5.2.1 Finite Data

In practice, we don't have infinite data. m is ultimately bounded. This constrains our choices for the other constants.

For an online SVM (PEGASOS):

$$l(\tilde{w}) \leq l(w_0) + O(\frac{1}{\lambda T}) + \lambda||w_0||^2 + O(\frac{1}{\lambda m})$$

Minimizing for λ, we pick $\lambda = \theta\left(\frac{\sqrt{1/T}+\sqrt{1/m}}{||w_0||}\right)$.

This gives running time as a function of m and ϵ:

$$T(m;\epsilon) = \theta\left(\frac{1}{\frac{\epsilon}{||w_0||} - O(\frac{1}{\sqrt{m}})^2}\right)$$

Further analysis shows that there is some minimal running time due to our error. Additionally, if there is too little data, we can run our algorithm for as long as we like and never obtain our desired generalization error. If we have more data, we can get our algorithm to unexpectedly run *faster*.