CS 189 Introduction to Machine Learning
Fall 2017

Note 21

1 Sparsity for SVMs

Recall the objective function of the soft-margin SVM problem:

$$\min_{w,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

Note that if a point x_i has a nonzero slack $\xi_i > 0$, by definition it must lie inside the margin. Due to the heavy penalty factor C for violating the margin there are relatively few such points, and thus the slack vector ξ is **sparse** — most of its entries are 0. We are interested in explaining why this phenomenon occurs in this specific optimization problem, and identifying the key properties that determine sparse solutions for arbitrary optimization problems.

To reason about the SVM case, let's see how changing some arbitrary slack variable ξ_i affects the loss. A unit decrease in ξ_i results in a "reward" of C, and is captured by the partial derivative $\frac{\partial L}{\partial \xi_i}$. Note that no matter what the current value of ξ_i is, the reward for decreasing ξ_i is constant. Of course, decreasing ξ_i may change the boundary and thus the cost attributed to the size of the margin $\|w\|^2$. The overall reward for decreasing ξ_i is either going to be worth the effort (greater than cost incurred from w) or not worth the effort (less than cost incurred from w). Intuitively, ξ_i will continue to decrease until it hits a lower-bound "equilibrium" — which is often just 0.

Now consider the following formulation (constraints omitted for brevity again):

$$\min_{w,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i^2$$

The reward for decreasing ξ_i is no longer constant — at any point, a unit decrease in ξ_i results in a "reward" of $2C\xi_i$. As ξ_i approaches 0, the rewards get smaller and smaller, reaching infinitesimal values. On the other hand, decreasing ξ_i causes a finite increase in the cost incurred by the $\|w\|^2$ — the same increase in cost as in the previous example. Intuitively, we can reason that there will be a threshold value ξ_i^* such that decreasing ξ_i further will no longer outweigh the cost incurred by the size of the margin, and that the ξ_i's will halt their descent before they hit zero.

There are many motivations for designing optimization problems with sparse solutions. One advantage of sparse solutions is that they speed up testing time. In the context of primal problems, if the weight vector w is sparse, then after we compute w in training, we can discard features/dimensions with 0 weight, as they will contribute nothing to the evaluation of the hypothesized regression values of test points. A similar reasoning applies to dual problems with dual weight vector v, allowing us to discard the training points corresponding to dual weight 0, ultimately allowing for faster evaluation of our hypothesis function on test points.

2 LASSO

Given the motivations for sparsity in SVMs, let's modify the ridge regression objective to achieve sparsity as well. The **least absolute shrinkage and selection operator (LASSO)** developed in 1996 by Robert Tibshirani, is one method that achieves this desired effect. LASSO is identical to the ridge regression objective, except that the L2-norm (squared) penalizing w is now changed to an L1-norm (with no squaring term):

$$\min_{w} \|Xw - y\|^2 + \lambda \|w\|_1$$

The **L1-norm** of w is a sum of absolute values of its entries:

$$\|z\|_1 = \sum_{i=1}^{d} |w_i|$$

Compare this to the **L2-norm** squared of w, the sum of squared values of its entries:

$$\|z\|_2 = \sum_{i=1}^{d} w_i^2$$

Just as in ridge regression, there is a statistical justification for using the L1-norm. Whereas ridge regression assumes a Gaussian prior over the weights w, LASSO assumes a **Laplace prior** (otherwise known as a **double exponential distribution**) over the weights w.

Let's understand why such a simple change from the L2 to L1-norm inherently leads to a sparse solution. For any particular component w_i of w, note that the corresponding loss in LASSO is the absolute value $|w_i|$, while the loss in ridge regression is the squared term w_i^2. In the case of LASSO the "reward" for decreasing w_i by a unit amount is a constant λ, while for ridge regression the equivalent "reward" is $2\lambda w_i$, which depends on the value of w_i. Thus for the same reasons as we presented for SVMs, LASSO achieves sparsity while ridge regression does not. There is a compelling geometric argument behind this reasoning as well.

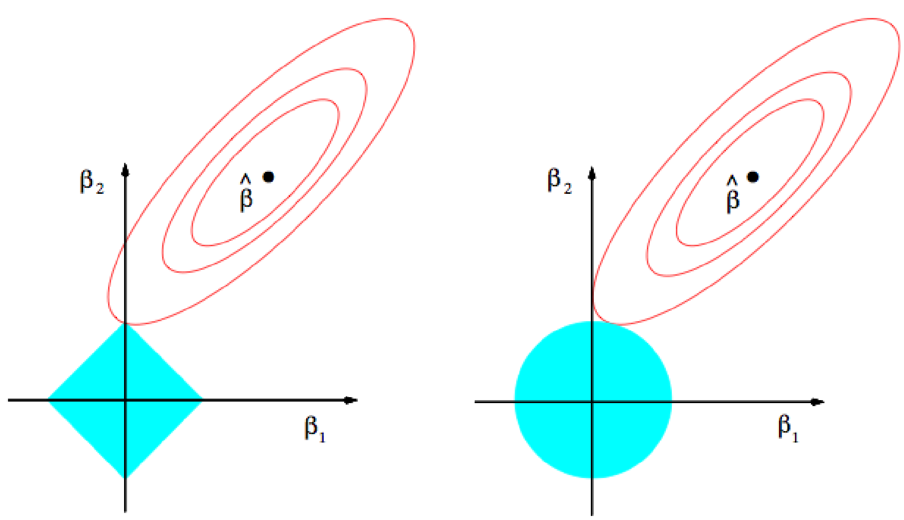

Figure 1: Comparing contour plots for LASSO (left) vs. ridge regression (right).

Suppose for simplicity that we are only working with 2-dimensional data points and are thus optimizing over two weight variables w_1 and w_2. In both figures above, the red ellipses represent isocontours in w-space of the squared loss $\|Xw-y\|^2$. In ridge regression, each isocontour of $\lambda\|w\|_2^2$ is represented by a circle, one of which is shown in the right figure. Note that the optimal w will only occur at points of tangency between the red ellipse and the blue circle. Otherwise we could always move along the isocontour of one of the functions (keeping its overall cost fixed) while improving the value of the the other function, thereby improving the overall value of the loss function. We can't really infer much about these points of tangency other than the fact that the blue circle centered at the origin draws the optimal point closer to the origin (ridge regression penalizes large weights).

Now, let's examine the LASSO case. The red ellipses represent the same objective $\|Xw-y\|^2$, but now the L1 regularization term $\lambda\|w\|_1$ is represented by diamond isocontours. As with ridge regression, note that the optimal point in w-space must occur at points of tangency between the ellipse and the diamond. Due to the "pointy" property of the diamonds, tangency is very likely to happen at the *corners* of the diamond because they are single points from which the rest of the diamond draws away from. And what are the corners of the diamond? Why, they are points at which one component of w is 0!

3 Coordinate Descent

Note that the LASSO objective function is convex but not differentiable, due to the "pointiness" of the L1-norm. This is a problem for gradient descent techniques — in particular, LASSO zeros out features, and once these weights are set to 0 the objective function becomes non-differentiable. **Coordinate descent** is an alternative optimization algorithm that can solve convex but non-differentiable problems such as LASSO.

While SGD focuses on iteratively optimizing the value of the objective $L(w)$ for each *sample* in the training set, coordinate descent iteratively optimizes the value of the objective for each *feature*.

Algorithm 1 Coordinate Descent

```
while w has not converged do
    pick a feature index i
    update w_i to argmin_{w_i} L(w)
end while
```

Coordinate descent is guaranteed to find the global minimum if L is **jointly convex**. No such guarantees can be made however if L is only **elementwise convex**, since it may have local minima. To understand why, let's start by understanding elementwise vs joint convexity. Suppose you are trying to minimize $f(x,y)$, a function of two scalar variables x and y. For simplicity, assume that f is twice differentiable, so we can take its **hessian**. $f(x,y)$ is element-wise convex in x if its hessian is psd when y is fixed:

$$\frac{\partial^2}{\partial x \partial x} f(x,y) \geq 0$$

Same goes for element-wise convexity in y.

$f(x,y)$ is jointly convex in x and y if its hessian $\nabla^2 f(x,y)$ is psd. Note that being element-wise convex in both x and y does not imply joint convexity in x and y (consider $f(x,y) = x^2 + y^2 - 4xy$ as an example). However, being joint convexity in x and y does imply being element-wise convex in both x and y.

Now, if $f(x,y)$ was jointly convex, then we could find the gradient wrt x and y individually, set them to 0, and be guaranteed that would be the global minimum. Can we do this if $f(x,y)$ is element-wise convex in both x and y? Actually not always. Even though it is true that $\min_{x,y} f(x,y) = \min_x \min_y f(x,y)$, we can't always just set gradients to 0 if $f(x,y)$ is not jointly convex. The reason for this is that while the inner optimization problem over y is convex, the outer optimization problem over x may no longer be convex. In the case when joint convexity is not reached, there is no clean strategy to find global minimum and we must analyze all of the critical points to find the minimum.

In the case of LASSO, the objective function is jointly convex, so we can use coordinate descent. There are a few details to be filled in, namely the choice of which feature to update and how w_i is updated. One simple way is to just pick a random feature i each iteration. After choosing the feature, we have to update $w_i \leftarrow \arg\min_{w_i} L(w)$. For LASSO, it turns out there is a closed-form solution (note that we are only minimizing with respect to one feature instead of all the features).

The LASSO objective is $\|Xw - y\|_2^2 + \lambda \|w\|_1$. It will be convenient to separate the terms that depend on w_i from those that don't. Denoting x_j as the j-th column of X, we have

$$\begin{aligned}
L(w) &= \|Xw - y\|_2^2 + \lambda \|w\|_1 \\
&= \|\sum_{j=1}^{d} w_j x_j - y\|_2^2 + \lambda |w_i| + \lambda \sum_{j \neq i} |w_j| \\
&= \|w_i x_i + C^{(1)}\|_2^2 + \lambda |w_i| + C^{(2)}
\end{aligned}$$

where $C^{(1)} = \sum_{j \neq i} w_j x_j - y$ and $C^{(2)} = \lambda \sum_{j \neq i} |w_j|$. The objective can in turn be written as

$$L(w) = \lambda |w_i| + C^{(2)} + \sum_{j=1}^{n} (w_i x_{j,i} + C_j^{(1)})^2$$

Suppose the optimal w_i is strictly positive: $w_i > 0$. Setting the partial derivative of the objective wrt w_i^* to 0, we obtain

$$\frac{\partial L}{\partial w_i} = \lambda + \sum_{j=1}^{n} 2x_{j,i}(w_i^* x_{j,i} + C_j^{(1)}) = 0 \implies w_i^* = \frac{-\lambda - \sum_{j=1}^{n} 2x_{j,i} C_j^{(1)}}{\sum_{j=1}^{n} 2x_{j,i}^2}$$

Denoting $a = -\sum_{j=1}^{n} 2x_{j,i} C_j^{(1)}$ and $b = \sum_{j=1}^{n} 2x_{j,i}^2$, we have

$$w_i^* = \frac{-\lambda + a}{b}$$

But this only holds if the right hand side, $\frac{-\lambda+a}{b}$, is actually positive. If it is negative or 0, then this means there is no optimum in $(0, \infty)$.

When $w_i^* < 0$, then similar calculations will lead to

$$w_i^* = \frac{\lambda + a}{b}$$

Again, this only holds if $\frac{\lambda+a}{b}$ is actually negative. If it is positive or 0, then there is no optimum in $(-\infty, 0)$.

If neither the conditions $\frac{-\lambda+a}{b} > 0$ or $\frac{\lambda+a}{b} < 0$ hold, then there is no optimum in $(-\infty, 0)$ or $(0, \infty)$. But the LASSO objective is convex in w_i and has an optimum somewhere, thus in this case $w_i^* = 0$. For this to happen, $\frac{-\lambda+a}{b} \leq 0$ and $\frac{\lambda+a}{b} \geq 0$. Rearranging, we can see this is equivalent to $|a| \leq \lambda$.

To summarize:

$$w_i^* = \begin{cases} 0 & \text{if } |a| \leq \lambda \\ \frac{-\lambda+a}{b} & \text{if } \frac{-\lambda+a}{b} > 0 \\ \frac{\lambda+a}{b} & \text{if } \frac{\lambda+a}{b} < 0 \end{cases}$$

where

$$a = -\sum_{j=1}^{n} 2x_{j,i}C_j^{(1)}, \quad b = \sum_{j=1}^{n} 2x_{j,i}^2$$

The term $\frac{a}{b}$ is the least squares solution (without regularization), so we can see that the regularization term tries to pull the least squares update towards 0. This is not a gradient-descent based update — we have a closed-form solution for the optimum w_i, given all the other weights are fixed constants. We can also see explicitly how the LASSO objective induces sparsity — a is some function of the data and the other weights, and when $|a| \leq \lambda$, we set $w_i = 0$ in this iteration of coordinate descent. By increasing λ, we increase the threshold for $|a|$ to be set to 0, and our solution becomes more sparse.

Note that during the optimization, weights can be set to 0, but they can also be "reactivated" after they have been set to 0 in a previous iteration, since a is affected by factors other than w_i.

4 Sparse Least-Squares

Suppose we want to solve the least squares objective, subject to a constraint that w is sparse. Mathematically this is expressed as

$$\begin{aligned} \min_{w} \quad & \|Xw - y\|_2^2 \\ \text{s.t.} \quad & \|w\|_0 \leq k \end{aligned}$$

Note that the **L0-norm** of w is simply the number of non-zero elements in w. This optimization problem aims to minimize the residual error between our prediction Xw and y while ensuring that the solution w is sparse. Solving this optimization problem is NP-hard, so we instead aim to find a computationally feasible alternative method that can approximate the optimal solution. **Matching pursuit** is a greedy algorithm that achieves this goal.

4.1 Matching Pursuit Algorithm

Recall that in ordinary least squares, we minimize the residual error $\|Xw - y\|_2^2$ by projecting y onto the subspace spanned by the columns of X, thereby obtaining a linear combination w^* of the columns of X that minimizes the residual error. Matching pursuit is a greedy algorithm that starts with with a completely sparse solution ($w = 0$) and iteratively "builds up" w until the the sparsity constraint $\|w\|_0 \leq k$ can no longer be met. The algorithm is as follows:

Algorithm 2 Matching Pursuit

initialize the residual $r = y$
initialize the weights $w = 0$
while $\|w\|_0 < k$ **do**
  find the index i for which the residual is minimized: $i = \arg\max_j \frac{|\langle r, x_j \rangle|}{\|x_j\|}$
  set the i'th entry of the weight vector to $w_i = \frac{\langle r, x_i \rangle}{\|x_i\|^2}$
  update the new residual value: $r \leftarrow r - w_i x_i$
end while

At each step of the algorithm, we pick the coordinate i such that x_i (the i-th column of X corresponding to feature i, *not* datapoint i) minimizes the residual r. This equates to finding the index i for which the length of the projection onto x_i is maximized:

$$i = \arg\max_j \frac{|\langle r, x_j \rangle|}{\|x_j\|}$$

Let's see why this is true. When we project the residual r on the vector x_j, the resulting projection has length $\frac{\langle r, x_j \rangle}{\|x_j\|}$. The length of the new residual follows from pythagoras theorem:

$$\|r_{old}\|^2 = \|r_{new}\|^2 + \left(\frac{\langle r_{old}, x_j \rangle}{\|x_j\|}\right)^2 \implies i = \arg\max_j \frac{|\langle r_{old}, x_j \rangle|}{\|x_j\|}$$

We move w_i to the optimum projection value and repeat greedily at each iteration. Note that once we change w_i we will never choose index i or change w_i again, because the new residual is (and all consequent residuals in future iterations are) orthogonal to x_i:

$$\langle r_{new}, x_i \rangle = \langle r_{old} - w_i x_i, x_i \rangle = \langle r_{old}, x_i \rangle - \langle w_i x_i, x_i \rangle = \langle r_{old}, x_i \rangle - \langle \frac{\langle r_{old}, x_i \rangle}{\|x_i\|^2} x_i, x_i \rangle = 0$$

This means that

$$i = \arg\min_j \frac{|\langle r_{new}, x_j \rangle|}{\|x_j\|}$$

and it will never satisfy the argmax again. While matching pursuit is not guaranteed to find the optimal w^*, in practice it works well for most applications. Setting the number of iterations is typically determined through cross-validation.