1 Coordinate Descent

Recall that the LASSO objective function is convex but not differentiable. 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. An alternative optimization algorithm that can be used to solve LASSO is called **coordinate descent**.

While SGD focused 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. Specifically, the algorithm is as follows:

**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
```

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 to choose which feature to update is to just pick a random feature each iteration. After choosing the feature, we have to update $w_i \leftarrow \arg\min_{w_i} L(w)$. For the 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 do not. Where $x_j$ is the $j$-th column of $X$, we have

$$\|Xw - y\|_2^2 + \lambda \|w\|_1 = \| \sum_{j=1}^{d} w_j x_j - y \|_2^2 + \lambda |w_i| + \lambda \sum_{j \neq i} |w_j|$$

$$= \|w_i x_i + C^{(1)} \|_2^2 + \lambda |w_i| + C^{(2)}$$

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

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

Suppose the optimal $w^*_i$ is in $(0, \infty)$. Then the gradient of the objective at $w^*_i$ is

$$\lambda + \sum_{j=1}^{n} 2x_{j,i}(w^*_i x_{j,i} + C^{(1)}_j) = 0$$
Rearranging, we get

\[ w_i^* = \frac{-\lambda - \sum_{j=1}^{n} 2x_{j,i}C_j^{(1)}}{\sum_{j=1}^{n} 2x_{j,i}^2} \]

Writing \( a = -\sum_{j=1}^{n} 2x_{j,i}C_j^{(1)} \) 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 \((0, \infty)\). If \( w_i^* \in (-\infty, 0) \), then similar calculations will give us

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

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, when \( |a| \leq \lambda \), we set \( w_i^* = 0 \). Otherwise, if \( \frac{-\lambda + a}{b} > 0 \) we set \( w_i^* = \frac{-\lambda + a}{b} \), and if \( \frac{\lambda + a}{b} < 0 \) we set \( w_i^* = \frac{\lambda + a}{b} \). 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 \( \lambda \), we increase the threshold for \(|a|\) to be set to 0, and our solution becomes more sparse.

Coordinate descent is guaranteed to provide a solution for LASSO. 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 \( w_i \).

2 Sparse Least-Squares

Suppose we want to solve \( \|Xw - y\|^2 \), subject to a constraint that \( w \) is sparse. One way to approach this problem is from a greedy, bottom-up approach.

2.1 Matching Pursuit

In matching pursuit, we start off with a completely sparse solution, \( w = 0 \). At this point, the residual is \( y \). At each step of the algorithm, we pick the coordinate \( i \) such that \( x_i \) (the \( i \)-th column of \( X \)) is most correlated with the residual (that is, \( \langle x_i, w \rangle \) is maximized over all coordinates). We move \( w_i \) to an optimum and repeat. TODO: describe in more detail
2.2 Orthogonal Matching Pursuit

TODO: describe

3 Decision Trees

Consider the following method of classification: ask a series of “yes” or “no” questions until you have reached a point where you can make an educated guess about the class of a data point. For example, if we want to predict whether a person carries an umbrella today, we may have the following sequence of decisions:

IF person is in the Bay Area:
    IF it is between November and March:
        IF it is cloudy:
            THEN predict "carried umbrella"

Decision tree learning is the problem of learning a sequence of decision rules such as the above from data. Decision trees are unusual in that they involve no linear algebra and are based on human logic and reasoning, as opposed to mathematical optimization. Though they are outperformed in a number of domains by common classifiers such as KNN, SVM, and neural networks, they are still in use for their interpretability - a decision tree provides a rationale for its prediction, and the sequence of decisions can be manually inspected to find correlations between the class and different features. While decision trees suffer from overfitting and are not great classifiers by themselves, one can combine many different decision trees to improve their performance overall - this is called ensembling. Many data mining competition winners use XGBoost, which is a more sophisticated ensembling method for decision trees.

To fit a decision tree, we recursively perform the following procedure: pick the “best” feature and a threshold, split the data based on that threshold, and recurse on each sub-training set. If at any point our tree is “pure” in that all data in each training subset is of the same class, then we have achieved perfect training set accuracy and stop growing the tree. Clearly, this would be overfitting - for any set of distinct data points, we could come up with a long list of decision rules that would put each data point into its own bin and thus classify the training set correctly, even though this is not generalizable. We choose when to stop growing our tree using cross-validation.

We have not described how to determine the “best feature.” We desire the best feature to be the most informative - given the information we have, how much does information does adding this new feature give us? Thus we are led to the information-theoretic ideas of entropy and mutual information, to be described in the next lecture.