19 Feature Selection

Feature selection in the high-dimensional setting (let $X \subset \mathbb{R}^{n \times d}$) has the goal of selecting the most useful subset of columns of X. The important question is "useful for what"?

We already saw the column sampling approach (called row sampling in our context there) to solving the column subset selection problem. This attempts to find the subset of columns that best capture the variance in the data, specifically the best rank-k subspace. While norm sampling (selecting proportional to the squared norm of the columns), the better answer is selecting according to the *leverage score*.

For this lecturing we will more naturally define "useful for what" in the context of regression. Now each row $x_i \in X$ is paired with a real label $y_i \in \mathbb{R}$. The *linear regression* problem seeks to find an $\alpha \in \mathbb{R}^d$ so that $\langle x_i, \alpha \rangle \approx y_i$. [Sometimes this is written with a learned constant offset, but this can be incorporated into this setting, but adding a 1 coordinate to each x_i.] The *least squared linear regression* specifically asks for the choice of α that minimizes

$$L(\alpha; X, y) = \sum_{i=1}^{n} (y_i - \langle x, \alpha \rangle)^2 = \|X\alpha - y\|_2.$$

And this has a closed form solution

$$\alpha = (X^T X)^{-1} X^T y. \tag{19.1}$$

Notably, this will almost always given a solution α that is non-zero in all coordinates. This means that each coordinate has some importance in this optimal least-squared prediction. So how do we choose which is best?

The answer is not so easy. But in this setting, at least we can provide some formalization to study it. We will provide:

- a formulation, called *Lasso*, that explains (under some assumptions) the best set of k coordinates
- an algorithm towards solving Lasso iteratively, Orthogonal Matching Pursuit.
- Discussion on "featurization" which, explains how to get to this formulation from non-Euclidean inputs

The hope is that these aspects together give insight into the way that this sort of approach can be solved more generally, and the troubling aspects of trying to draw too many conclusions from it.

Other elements to this story are left in the notes for historical reasons.

19.1 Regularization

By the Gauss-Markov Theorem, that (19.1) is the minimum variance (least squares) solution to the problem with P given that it is unbiased. *However*, it may be advantageous to bias towards a small slope solution.

This models the residual as only in the y direction, and thus implicitly assumes that the X coordinates have no error. Thus when noise happens, it happens in the y-coordinate, and we want to minimize the effect of this. To do so, we can "regress to the mean."

Example: *Consider a hard* TRUE-FALSE *test. Each student knows some fraction of the answers (say 50% of them) and guess on the rest. The expected score is 75%. Say this was the case for 100 students, and we took the 10 students who scored the best; say their average score was 80%. If we gave these same 10 students another similar test (still* TRUE-FALSE*, and they know half, guess half), then what is going to be their expected score: 75%. That is we expect them to regress towards the mean!*

So in linear regression, we expect that y-values will not be as wild in this observed data as it would be if we observed new data. So we want to give a prediction that made more extreme data have less affect. These solutions can have overall less variance, but do not have 0 bias.

Another view is that we have a prior (as in Bayesian statistics), say of weight $s/(s+n)$, that the mean value of the y-coordinates is correct. So we don't want to entirely use the raw data.

Tikhonov regularization. To this end, we can change the *loss* function, that which was measuring the error of our solution α. The Tikhonov regularization for a parameter $s \geq 0$ finds

$$\arg\min_{\alpha} \|X\alpha - y\|_2 + s\|\alpha\|_2. \tag{19.2}$$

This is also known as *ridge regression*. Magically, this can be solved just as easily as least squares by setting

$$\alpha_s = (X^T X + s^2)^{-1} X^T y.$$

19.1.1 Lasso

An alternative approach (the focus here) will be a bit more complicated to solve, but has a couple of other very nice properties. It is called the Lasso, or alternatively *basis pursuit*; for a parameter $s \geq 0$ it finds

$$\arg\min_{\alpha} \|X\alpha - y\|_2 + s\|\alpha\|_1. \tag{19.3}$$

Note that the only difference is that it has an L_1 norm on the $\|\alpha\|$ instead of the L_2 norm in ridge regression. This also prevents the simple matrix-inverse solution of ridge regression. However it will have two other very nice properties:

- In high dimensions, it will bias towards sparse solutions
- It forces one to consider multiple values of s, and hopefully choose a reasonable one.

A formal way to solve for Lasso solution (in fact all solutions) is called *Least Angle Regression*, and is documented below. But a simpler approach (with weaker guarantees), OMP, is more heavily used.

Hard versus soft constraint. The first insight is that instead of (19.3) it is equivalent to solve

$$\arg\min_{\alpha} \|X\alpha - y\|_2^2 \qquad \text{such that } \|\alpha\|_1 \leq t. \tag{19.4}$$

for some parameter t. Note that we have replaced the parameter s with another one t. For any value of s and solution α_s to (19.3), there is a value t that provides an identical solution α_t to (19.5). To see this, solve for α_s, and then set $t = \|\alpha_s\|_1$.

Moreover, for any choice of $k \in [0, d]$ number of desired non-zero coordinates, we can find values of s (or equivalently, of t), so that there are k non-zero coordinates. For instance, by binary search – although algorithms below will give more direct approaches.

Units. While non-regularized regression is does not care about units (since the choice of α cancels out the units), the regularized regression does! If $x_i =$ (weight in lbs, height in inches, age in years) then α_1 has units 1/lbs, α_2 has units 1/inches, and α_3 has units 1/years. Then the dot product is a unit-less calculation. However, then $\|\alpha\|_1$ adds up those 1/unit values, and so this operation does not make sense.

A common approach is then to first normalize the coordinates, so each coordinate does not have units, and thus the same for α. The two common approaches are:

- **standardize / normalize**: so each variable has mean 0 and variance 1. Mean 0 is accomplished by centering the data. Variance 1 is accomplished by then computing the variance (the $\sigma_i^2 = \|X_i\|^2$), and dividing all values by σ_i^2.
- **0-1 normalize**: so each variable is in range $[0, 1]$. First find smallest value, and subtract it from all values. Then find largest value, and divide all values by it.

These do not solve the issue of mixed units, but they do tend to help. Challenges are:

- If we observe more data, the way we normalize changes.
- Outliers might have a large effect on the scaling.
- If columns are co-linear (eg., height and weight are correlated), then they have accumulated their effect on a distance. Whereas independent values (e.g., income) will not be correlated, and could have.

Interpretation. So given a regression solution α, can we know which coordinates are the most important?
In general: NO!!
While this might work, it is dangerous, and may lead to false conclusions!
Why is this?

- The units in regular regression may affect which α_i coefficient is largest.
- If we normalize first, there may be co-linear coordinates that affect the scale. Only one may be "causal" and the other may be nuisance – how can we know which is which? In reality, probably both are nuisance, and correlated with something else causal – we measure what we can measure, not necessarily what drives the mechanism.
- If we consider the k non-zero Lasso coefficients, there may be a disjoint set of k coordinates with almost as much predictive power. In fact (by inspecting the Least Angle Regression algorithm), there may be two or more distinct sets of k coefficients which provide a Lasso solution for different values of s, t.
- If we use OMP (see below) we may have greedily picked the wrong first choice.

In general, I am not convinced (in general) much can be interpreted from coordinates of linear regression. There are methods like Shapely Values that attempt to isolate these effects. They do sometimes work, but are not always conclusive – there are other more subtle challenges.

19.1.2 Orthogonal Matching Pursuit

Orthogonal Matching Pursuit (OMP) is a greedy algorithm to solving Lasso (or other regression problems). Here it is sometimes called *forward subset selection*. This may be slightly easier to implement, but will not provide the optimal solution and allows one to cherry-pick a value s.

Algorithm 19.1.1 Orthogonal Matching Pursuit

Set $r = y$; and $\alpha_j = 0$ for all $j \in [d]$.
for $i = 1$ **to** t **do**
  Set $X_j = \arg\max_{X_{j'} \in X} |\langle r, X_{j'} \rangle|$.
  Set $\alpha_j = \arg\min_\gamma \|r - X_j\gamma\|^2 + s|\gamma|$.
  Set $r = r - X_j\alpha_j$.
Return α.

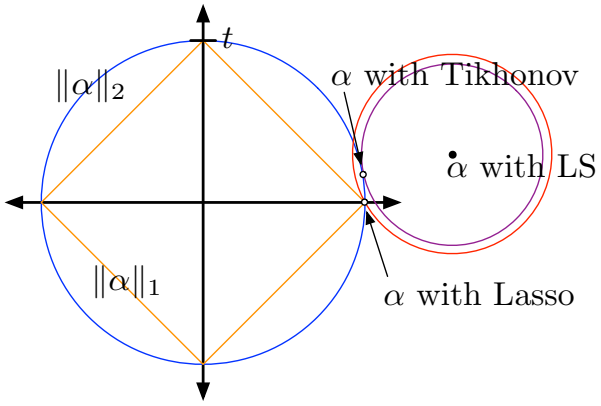

Figure 19.1: Radius t ball under L_1 and L_2, and the results of Lasso and Tikhonov regularization.

19.2 Least Angles Regression

The first insight is that instead of (19.3) it is equivalent to solve

$$\arg\min_\alpha \|X\alpha - y\|_2^2 \qquad \text{such that } \|\alpha\|_1 \leq t. \tag{19.5}$$

for some parameter t. Note that we have replaced the parameter s with another one t. For any value of s and solution α_s to (19.3), there is a value t that provides an identical solution α_t to (19.5). To see this, solve for α_s, and then set $t = \|\alpha_s\|_1$.

This same dual version exists for ridge regression as well. Why is this useful?

Sparsity. This biases solutions to have 0 along many of the coordinates α_j. This is illustrated in Figure 19.1 where the Lasso solution for α is restricted to lie within an L_1 ball of radius t, but otherwise be as close to y as possible. The least squares solution is the best possible fit for A. We can see the extra L_2 error around this solution which is minimized with respect lying in a radius t L_1 ball for Lasso or L_2 ball for Tikhonov regularization.

Note that the L_1 ball has "pointy" corners, and thus bias solutions for α towards these corners. The corners have some coordinate of α as 0. In higher dimensions, this L_1 ball has even more corners, and they play an even more prominent role in the solutions to these minimization problems.

As t becomes smaller, it is more and more likely that the solution to α is found on a higher-degree corner. But also the solution found becomes further and further away from the least squares solution. If we set $t = \infty$ then the Lasso (and Tikhonov) solution is the least squares solution. How do we balance these aspects?

Increasing t. As we increase t (our coefficient budget), then we allow some α_j to increase. If we start with $t = 0$, then all $a_j = 0$. Now using the (piecewise-) linear equation $t = \sum_{j=1}^{d} |\alpha_j|$ and

$$r(t) = y - \sum_{j=1}^{d} X_j a_j(t).$$

The goal is to minimize $\|r(t)\|$, and we note that changing some α_j have more effect on $r(t)$ than others.

First find, $j_1 = \arg\max_j |\langle X_j, r\rangle|$. This is the coordinate with maximum influence on $r(t)$. We vary this first and set $\alpha_{j_1}(t) = \alpha_j \cdot t$. We now increase t, while only varying α_j (as specified) until some other coordinate is worth increasing.

Next find j_2 such that $j_2 \neq j_1$ and has

$$|\langle X_{j_1}, r(t)\rangle| = |\langle X_{j_2}, r(t)\rangle|.$$

We can solve for the value t at which this will happen for each $j \neq j_1$ since the above is a linear equation in t. The index j_2 is selected in that it has the smallest value t_2 at which this equality happens. *This is the first time that (19.5) is minimized with* 2 *non-zero coefficients.* The next step is to reset the correlations (via the first derivatives) such that $|b_1| + |b_2| = 1$ as

$$\alpha_{j_1}(t) = \alpha_{j-1}(t_2) + (t - t_2)b_1 \qquad \text{for } t \geq 0$$
$$\alpha_{j_2}(t) = (t - t_2)b_2 \qquad \text{for } t \geq t_2.$$

This implies that as t increase, the optimal choice in α_j is linear in t with slopes defined by $b_1, b_2, \ldots$

We continue this, in each ith step finding a time t_i at which increasing some coefficient α_{j_i} will have $|\langle X_{j_i}, r(t_i)\rangle| = |\langle X_J, r(t_i)\rangle|$ where J is the set of indices we are tracking and X_J is the subset of columns of X corresponding to those indices. Then we add this j_i to J, update $r(t)$ for $t \geq t_i$, and recompute the derivatives $b_1, \ldots, b_i$, and find the next value t_{i+1} and so on. See Algorithm 19.2.1 for a more formal description of the algorithm.

Algorithm 19.2.1 Least Angle Regression

Set $\alpha_j = 0$, $b_j = 0$ for all $j \in [d]$.
Set $j_1 = \arg\max_j |\langle X_j, r(0)\rangle|$; $b_{j_1} = 1$; and $J = \{j_1\}$.
Set $r(t) = y - \sum_{j=1}^{d} X_j \alpha_j(t)$ where $\alpha_j(t) = b_j \cdot t$. *$\alpha_{j_1}(t) = t$, otherwise $\alpha_j(t) = 0$ for $j \neq j_1$*
for $i = 2$ **to** n **do**
  for all $j \notin J$ **do**
    Find $\tau_j > t$ such that $|\langle X_j, r(t)\rangle| = |\langle X_{j_1}, r(t)\rangle|$ *Note: all $j \in J$ have same $|\langle X_j, r(t)\rangle|$*
  Set $t_i = \min \tau_j$ and $j_i = \arg\min_j \tau_j$; $J = J \cup j_i$.
  Solve for b_j for all $j \in J$ such that $\sum_{j \in J} |b_j| = 1$. *Take derivatives of $|\langle X_j, r(t)\rangle|$ and normalize*
  For $t \geq t_i$ redefine $r(t) = y - \sum_{i=1}^{d} X_j a_j(t)$ where $\alpha_j(t) = \alpha_j(t_i) + (t - t_i)b_j$.
Return $\alpha(t)$ when its cross-validation score is smallest.

So in the process we have solved the optimal choice of α for each value t (and hence each value s). We can choose the best one with *cross-validation* where we leave out some data and evaluate how well the model works on that data (more later). We can maintain this estimate and solve for the minimum (since we have all linear equations) along the way.

There is a variant of this algorithm where we may want to snap values α_j to 0 even if $j \in J$. This happens since initially b_j may be positive, but as J increases, it may become negative. Then when α_j hits 0, we remove it from J (this can time can be treated as its τ_j value.). Then we can later re-add it to J. This is needed to get the optimal solution for any t. Is a little more work, but could require an exponential number of steps.

19.3 Featurization

To build a linear regression model, or most models in machine learning or data mining, one typically needs the data to have form $X \subset \mathbb{R}^d$. With this setting, even without the same units on different columns/variables, one can invoke distance metric learning tricks to convert to such a setting; but this either needs to start with $X \subset \mathbb{R}^d$ or already some distance $\mathbf{d}$ that is trusted. For many initial data sets, this is not the case.

For different types of fields, there are different mechanisms for transforming into Euclidean.

Text field. If there are individual words, one can use word vector embedding to get a multi-dimensional embeddings (e.g., Word2Vec, GloVe). Yes, these are high-dimensional, but we can try to reduce dimension later if needed.

If the text is longer, then mechanisms like `SBert`, can give better embeddings that recognize the entire context.

Ordinal data. Data is ordinal if it has discrete values, but there is a clear order. Examples include

- grades: $A > A- > B+ > B > B- > C+ > C...$
- education levels: elementary, middle school, high school, trade school, college degree, MS, PhD

While there are more complicated approaches, the simple and easy solution is just to assign them integer values that encode the ordering. E.g., A = 10, A-= 9, B+ = 8, ...

Categorical Data. This data are ones that have no clear order, like gender, or which high school someone attended, or a choice from the menu of a restaurant.

The simplest way to encode this is one-hot encoding where each object gets its own dimension. So one coordinate that has options for eye color: brown, blue, green. This is mapped to a 3-dimensional vector so brown is $(1, 0, 0)$, blue is $(0, 1, 0)$, and green is $(0, 0, 1)$. So the mapping puts a 1 in the new coordinate that corresponds with the trait, and 0 in all others.
This can be a bit strange to take the average of such coordinates, and it can turn low-dimensional data into very high-dimensional data (e.g., a menu may have many items on it!). But this is the most common approach here.

Another data-driven approach is to use other coordinates which have real values. Either choose a meaningful one, or (after adjusting for units' co-linearity in an appropriate way) take the top principal component. Either gives a one-dimensional setting; define it by a unit vector v. Now for all of the data points with a certain categorical value j, let μ_j be the average value all this data along v. Now sort categories j along direction v, and use this as a continuous coordinate in place of the category.
This has the advantage that it does not blow up the dimension, and may find a meaningful order. However, it may jumble up different categories to be almost the same – when they are not. And it is data dependent, so as data changes, so would this embedding.