Modeling Toolkit

Article 1: Introduction

Andrew Freeman

May 8, 2021

1 Introduction

Welcome to the Modeling Toolkit! This series of articles will look at a variety of models and techniques used in data science. I'm mainly writing this to help me to better understand the models and to have a reference for future use, but if anyone else finds it useful, that's just an extra perk.
These articles will touch on a lot of different concepts, but they will all follow more or less the same format (when applicable). First, I'll give an intuition of the model. What it is and why that might be a useful idea. Next, I'll delve into (some of) the math so that we can approach the model from a solid foundation. After that, I'll provide examples of this model in action. This will include a custom built script for some models and a library example for all of them. In the cases where I do build out a custom model, it won't be as efficient or accurate as the library model; it will be provided just to help explain the transition between theory and practice. Finally, I'll give some pros and cons for that model.

2 Notation

Just as a quick intro on mathematical notation, I will use non-bolded lower case letters (i.e. x) to represent scalars, bolded lower case letters ($\mathbf{x}$) to represent vectors, and bolded capital letters ($\boldsymbol{X}$) to represent matrices. Other notation will be introduced as needed.

3 Defining Models

To define a model, we first need to explain what it is that we want it do to. There are several reasons we would make a model: to make predictions, explain a phenomenon, or explore variable connections. In each of these cases, the underlying goal of the model is the same, however: we believe that there exists some function out there that links our variables. In these notes, we will take y as the variable we wish to predict and $\mathbf{x}$ as the vector of variables we are using to predict y. Therefore, we believe that there exists some function F such that

$$F(\mathbf{x}) = y$$

This simple line is the entire idea behind modeling. The problem is, it's not that simple. For instance, imagine we wish to predict how far a baseball will travel after a pitcher throws it. From basic physics, we know that if we know the initial velocity, height, and angle of the ball, we can calculate this value pretty accurately. Our guess will be accurate, but not exact. For instance, we would also need to consider the exact value of gravity in the location where the ball was thrown. The more siginificant figures we have for gravity, the more accurate we would be, but there would still be some error. We would also need the air resistance on the ball, which can be estimate but not exactly known. Further, what if a wind blows while the ball is in the air? This example may be contrived, but hopefully it gets the point across: no matter how many variables we measure, there will always be other variables that affect our results and cannot be measured. We usually denote these variables $\mathbf{z}$ and our function is more accurately represented as

$$F(\mathbf{x}, \mathbf{z}) = y$$

So if we can't (or simply don't want to) know $\mathbf{z}$ (possible because of the cost/time to measure it), what should we do? First, let's define a **loss function**. A loss function measures the difference between the true $\mathbf{y}$ that we observe and our estimates of $\mathbf{y}$ which we denote $\hat{\mathbf{y}}$. Any time we have a hat on a variable, it means that it is an estimate of that true variables. There are many loss functions that we will explore, but for now we will be generic and define a loss function as

$$L(\mathbf{y}, \hat{\mathbf{y}})$$

As a quick example, consider a game in which you flip a coin and you have to predict if it will land heads or tails. Then, we could let

$$L(y, \hat{y}) = I(y \neq \hat{y})$$

This notation introduces an indicator function, I. Indicators are very useful tools and are simple to understand: an indicator $I(X)$ takes value 1 if X is true and value 0 otherwise. So in this case, our loss is 1 if we predict a heads and then flip a tails, and 0 if we predict heads and flip a heads.
Now that we have a loss function, we can say what it is we want our model to do: minimize the loss. That is, we want our model to predict $\mathbf{y}$ as accurately as possible using the $\boldsymbol{X}$. We want this to hold for any values of $\mathbf{x}$ and y we get, so we minimize the expected value of this loss which we define as the **risk**:

$$R(F) = E_{\boldsymbol{X},\mathbf{y}}[L(\mathbf{y}, F(\boldsymbol{X}))]$$

So what F should we choose? We should choose

$$F^* = \underset{F}{\operatorname{argmin}}\ R(F)$$

We call F^* the target function. This function is the best function that takes as argument $\mathbf{x}$ and produces y for our given loss function. Note that this does not mean that F^* is any good; it just means that no function does better. As a trivial example, go back to the coin flipping problem. We can always predict heads for our flip and we will be right only half of the time. However, without adding in other variables, there's no way to do better than this. So don't think of the target function as the exact function that defines y. Think of it instead as the best we could ever hope to do with our limited resources.

Great, we know what we want to do. How do we get F^*? First of all, we often can't. It exists somewhere out there, but it may be very difficult to find for many reasons. The biggest reason is noise. Remember how we said that there were always $\mathbf{z}$ affecting our model? Well, we never know how much they impact our model. Because they are hidden, we can't tell if our model is using $\mathbf{x}$, $\mathbf{z}$, or some combination thereof. All we know is how well we fit the limited data we are given. Instead, we will try to estimate F^* to the best of our ability. But enough beating around the bush; let's talk about how we actually define a model.
There are lots of ways to describe a model. However, I am particular to the format that was taught in STATS 315B: any model has three components. These components are a function class, a score function, and an optimization search method. The explanation for these three as excerpted from my 315B notes are provided below, but the general idea is this: we first define all of the functions that we are considering for our estimate of the target function (the Function Class). Then, we define a function to tell us how well our estimate of the target function is doing (the Score Function). Finally, we define a way to minimize the Score Function over all functions in the Function Class (the Optimization Search Method).

3.1 I. Function Class

The function class is all of the possible functions that we consider for our model. We denote this class $\mathcal{F}$. By definition, it is guaranteed that our selected model is in this class, $\hat{F}(\boldsymbol{X}) \in \mathcal{F}$. It is not guaranteed that the target function is in this class. In fact, it is likely that it is not.
Examples:
Ordinary Least Squares: $\mathcal{F}$ = all linear functions = $a_0 + \sum_{j=1}^n a_j x_j$ for any $(a_0, \mathbf{a})$
Additive Modelling: $\mathcal{F}$ = all smooth additive functions = $\sum_{j=1}^n f_j(x_j)$ where f_j is smooth
Support Vector Machines: $\mathcal{F}$ = all polynomials up to a fixed order (determined by the kernel)
It's important to note that we need to impose some form of restriction on $\mathcal{F}$ - we can't just use the set of all possible functions. Why not? Well, there are an infinite number of functions that perfectly interpolate any data set, many of which do not look like our target function. Consider a line in two dimensions. We can have a sine-wave like function that perfectly hits every point and goes to $\pm\infty$ between points. This is clearly not a good fit. Imposing a requirement for smoothness tends to be sufficient in two-dimensions, but we rarely operate in two-dimensions, so more restrictions are typically required.

3.2 II. Score Function

The score function judges the (lack of) quality of the fitted model on the data. We define two loss functions: the population score function, which is what we ultimately would like to fit on the entire population, and the data score function, which we actually fit. If we had the distributions of $\boldsymbol{X}$ and $\mathbf{y}$, we could directly use the population score function and solve it statistically. This would produce the best possible fit. Unfortunately, this is often unreasonable, so we settle for a surrogate: the data score function.
Examples:
OLS: Population - $E_{\boldsymbol{X}\mathbf{y}}(\mathbf{y} - a_0 - \sum_{j=1}^n a_j \mathbf{x}_j)^2$
Data - $\frac{1}{N}\sum_{i=1}^N (y_i - a_0 - \sum_{j=1}^n a_j x_{ij})^2$
Ridge/Lasso: Data -

$$\frac{1}{N}\sum_{i=1}^N (y_i - a_0 - \sum_{j=1}^n a_j x_{ij})^2 + \begin{cases} \lambda \sum_{j=1}^n a_j^2 \\ \lambda \sum_{j=1}^n |a_j| \end{cases}$$

SVM: Population - $E_{\boldsymbol{X}\mathbf{y}} I(y_i \neq \text{sign}(\hat{F}(\mathbf{x}_i)))$ where $I(a)$ is an indicator function that is 1 iff it's argument is true, else 0.
Data - $\frac{1}{N}\sum_{i=1}^n [1 - y_i \hat{F}(\mathbf{x}_i)]_+ + \lambda \sum_{j=1}^n a_j^2$ (note that we take $y \in \{-1, 1\}$
Note that it is easier to optimize convex functions than non-convex functions, so we often substitute a convec score proxy for a non-convex actual target. Also, we note that by definition, ridge and lasso will not be optimal for the training data (since that is what OLS would find), but that they tend to do better on the *population* data.
This is the only part related to statistics, since the score is a function of random variables and hence is itself a random variable.

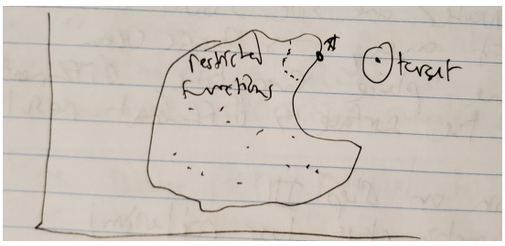
3.3 III. Optimization Search Method

Now that we have a class of functions and a score, we just have to minimize that score over that class:

$$\hat{\mathbf{a}} = underset\mathbf{a}\text{argmin } S(\{y_i, f(\mathbf{x}_i, \mathbf{a})\})$$

where $S(x)$ represents the score functions and $f(x) \in \mathcal{F}$.
Examples:
OLS: Dircet matrix algebra, $\hat{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\mathbf{y}$
Ridge Regression: Direct matrix algebra, $\hat{\beta} = (\boldsymbol{X}^T\boldsymbol{X} - \lambda I)^{-1}\boldsymbol{X}^T\mathbf{y}$
Lasso, SVM: Quadratic program (originally; have since been optimized to be faster)
For any convex problem, there exists a unique minimum solution on the provided data. For more flexible procedures (trees, neural nets, clustering), this is no longer the case; there may be several local minima, making the search harder. Further, we cannot directly solve the problem; we must use other methods like seepest/gradient descent. Examples of these algorithms include greedy search, steepest descent, an expectation-maximization (EM) algorithm, and numerical optimization. Worse, the solution depends on where you start solving. The example provided is as follows: pretend you are on the Rocky Mountains. If you spill a glass of water, the water will end up in the Gulf of Mexico. If you take a step to the other side

of the summit, the water now ends up in the Pacific Ocean. Extending this, even if we start in the same geographic location, each different data set is a different topology, which means we may not get the same results even if we define a universal starting place.
Note that we may not just want the global minimum; this could be overfitting.
Searching is further complicated by the bias-variance tradeoff. This is a common idea in learning, but here's a good way to think of it. The target function is in the restricted function space $\mathcal{F}$ with probability 0, which means that even the best function in $\mathcal{F}$ will be some "distance" from the target. This is the irreducible error, also known as the bias. Then, within $\mathcal{F}$, we get many different results depending on initial conditions or on the specific random sample of data we have and the noise in the data. The "distances" between the functions we could get with different conditions represents the variance. In the picture below, bias is the distance between the starred function and the target, and variance is determined by the size of the restricted function space.



3.4 Different Procedures

Any different methods for learning differ in at least one of I-III. Often in literature, people change III since it is easier to make small improvements to exisiting algorithms than it is to define new ones. Further, existing methods have use cases so improvements are useful; a new method may be very clever mathematically but if it doesn't help approximate real-world target functions, it's not very useful.
Throughout these notes, I will try to define all of the models we discuss in terms of this setup so that they are easy to compare.
Finally, I'll conclude this article with some notes on choosing and comparing models, taken from my 315B notes.

4 Model Selection and Comparison

4.1 Choosing Models

So how do we pick a model? In general, the answer is cross-validation (CV). There are a few things to keep in mind:

1. There is no "magic bullet". No method is universally better than any other (No Free Lunch Theorem)
2. Different models make different assumptions on the type of the target function (i.e. is it smooth? Linear? etc.)
3. Each method has a class of target functions $F^*(\boldsymbol{X})$, a sample size N, and a signal to noise ratio S for which it performs best (even if we may never want to estimate that class in practice because it isn't relevant)

    Note that we denote ϵ as noise

So to ultimately pick a model you should

1. Consider what is known about the problem and F^*
2. Try several different models and use the best or a committee (blend, ensemble)

    Note that ensembles do not always perform better than a single model by itself

4.2 Signal To Noise

We define the signal to noise ratio (SNR) as

$$\frac{Var(F^*(\boldsymbol{X}))}{Var(\epsilon)}$$

A small SNR indicates that F^* likely won't be very good, as most of y is defined by our $\mathbf{z}$. In this case, we favor simpler models because they are faster and easier to test (i.e. regression) and a complex model may not do much better because of the noise. For large SNR, we can use a more complicated model (i.e. neural networks) because they can learn more complicated functions and we know that F^* provides a good estimate of y (even if F^* is very complicated).

4.3 Model Comparison

A couple of notes on being careful when comparing models:
First, a good method in one situation may not be any good in others. Don't be too quick to generalize a model beyon the specific example you observe it in. Second, be aware of **selection bias**: in the literature, only the best papers are published. In these papers, only the best examples are provided, so that they have a chance of being published. So results are typically viewed on the best datasets and the best predictions for those sets. Further, predictions are random variables, which means that they vary. If you repeat an experiment 100 times, 5% of the time you will have significant results. These are what tend to be published. Moreover, if you work to make your results significant at the 95% level, they aren't significant at the 95% level since you deliberatly aimed to push them over this threshold.
Finally, be aware of **expert bias**. The level of success of a certain model depends on the analyst and their skill with the specific model. Authors have the most skill with their own model, so their model will obviously do well. Further, they are highly motivated to fine tune their model (to publish), and very unmotivated to tune any competitor models they are comparing against (to get better comparisons). In addition, they are tuning (or even creating) their model for a specific task; comparing on it may be cheating.
So when are comparisons most useful?

- When the whole point of a paper is to compare methods (and not to introduce a new model that beats old methods)
- When there are no vested interests
- When there is no skill differential amongst competitors (i.e. the author can use regression and NN equally well)

In the case that these states do not hold, it may be useful to compare the *other* models. The lead author doesn't care which model comes in second vs. last, as long as his model comes in first. Therefore, these models are more likely to be on even footing. Finally, competitions (i.e. Kaggle) can be useful since each entrant is submitting their best model (although it is still important to be aware of skill gaps).

5 Conclusion

Thanks for reading this article! Next, we will begin our delve into models with one of the earliest and simplest models: linear regression.