Data Science Toolkit

Matt Goodwin

November 29, 2018

Contents

1 Statistical Modeling

1.1 Overview and Theory

When discussing modeling it is important to keep in mind that “all models are wrong but some are useful” [1]. The world is extremely complex and it can be impossible to create a model that perfectly approximates the underlying mechanisms that make our world turn.

There are different approaches to modeling depending on the discipline you come from, but personally I like the idea of the function approximation approach suggested by applied math and statistics. Taking this approach allows us to use probability theory combined with decision theory and to be able to visualize these concepts in a euclidean geometric space.

Bishop, from his book Pattern Recognition, has a really nice overview of some of these concepts. The starting point I think for modeling starts with independent variable or covariate X (which could be a vector - see notation section) and dependent variable Y (also could be a vector). We want to know:

1. The nature of the relationship between the variables (inference).

2. Given an independent variable, determine the dependent variable (prediction).

Using probability we can completely summarize the relationship and the uncertainty between the two variables with the joint distribution $P(X,Y)$. We use probability because for many problems we are interested in, we generally cannot come to a completely deterministic relationship between the independent and dependent variables. This is partly because of measurement error, but also because the number of independent variables needed to perfectly determine the dependent variable is potentially infinite.

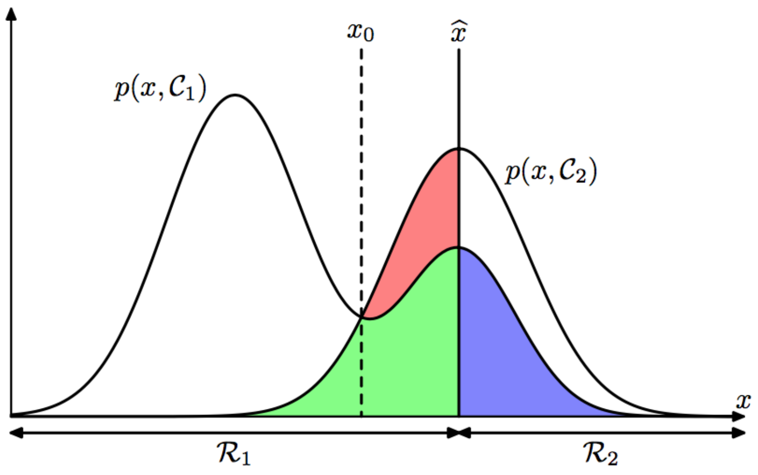
For example, imagine we wanted to predict the number of ice cream cones we will sell on a particular day. Some variables such as the time of year or location of the ice cream store may provide us enough information to make a pretty good prediction or to understand the relationship between the independent and dependent variables fairly well. But to perfectly predict the number of ice cream cones we would need to know everything from the state of the road conditions, to whether or not a family from out-of-state decided to take a vacation. Since this is impossible, we acknowledge variability and error in our estimates using probability.

I think the key to understanding this is to remember that the moment we use only a subset of all the possible features we would need for a perfectly deterministic relationship, then we must introduce uncertainty. We cannot say for certain that only knowing today is July 1 will lead to high ice-cream sales, but we can say the probability is higher than January 1st. When I have a training sample $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$, I treat this as the truth (which it is) but I need to remember that these are draws coming from a distribution. I guess in that sense $P(X,Y)$ is a model itself, something we are forced to use because we don’t know all the features needed for a deterministic relationship.

One side note to make here is, as ESLII mentions, sometimes the relationship IS deterministic but the randomness comes from the fact that we have limited data. If we had a different training data set then we could get different results but the underlying relationship is deterministic. These problems

[1] attributed to George Box

Figure 1.1: Plot from Bishop showing visually the optimal decision boundary



can be handled by techniques appropriate for the error-based models described previously (see pg 28 of ESLII).

1.1.1 Decision Theory

As mentioned, we may want to perform inference, or in other words understand what $P(X, Y)$ looks like using information from a sample. This can give us an understanding of how the variables are related. In many practical applications however, we want to be able to predict Y given X. This is where decision theory comes into play. Decision theory is designed to help us make the optimal decision given inputs. Bishop gives a nice overview that I try and summarize in my own words below.

Lets approach this by treating the dependent variable Y as a categorical variable taking on values 0 or 1. For simplicity assume X is a single continuous variable. We then have for $P(X, Y)$ a three-dimensional distribution where $P(Y|X)$ is a probability mass function. When making a decision called the *decision step* we formulate some rule that divides the input space into *decision regions*. If an instance falls into a certain decision region (based on X) it is predicted to be a 0 or 1. We want to minimize our mistakes as much as possible so we aren't assigning an instance to 0 when it should really be 1. The probability of a mistake can be written as:

$$P(mistake) = P(X \in R_1, 0) + P(X \in R_0, 1) \tag{1.1}$$

where R_1 is the region where an instance is assigned a 1 and R_0 is the region where an instance is assigned a 0.

Back to our example. Instead of ice cream sales, treat Y as a categorical variable where 1 is a "good" ice cream sales day and 0 is "bad". If x_1 = "July 1st" is in R_1 we decide to assign it a 1, based on our decision rule. However, even though our model $P(X, Y)$ says that the probability of a high-selling day ($Y = 1$) is high in this region, there is a still a chance that it is a low-selling day because again, we are using a probability distribution for a model since we don't have all of the features we need for a deterministic model. The probability of it being a low-selling day for all X in R_1 is $P(X \in R_1, 0)$, which is a mistake.

We want to minimize our mistakes as much as possible so we choose regions where $P(X \in R_1, 0) + P(X \in R_0, 1)$ is as small as possible. To me it is easier to see this by thinking of the probability of being correct instead of the probability of being incorrect. This changes the problem from one of minimization to one of maximization. The optimal decision boundary therefore is the location that creates R_1 and R_0 such that $P(X \in R_1, 1) > P(X \in R_1, 0)$ everywhere in R_1 and $P(X \in R_0, 0) > P(X \in R_0, 1)$ everywhere in R_0. If the decision boundary were shifted either way then we would loose out on area under the distribution of being correct.

To visualize this better refer to figure 1.1.1 from Bishop. If our decision boundary were at x_0 then the probability of being correct would be the two humped distribution completely colored in. This is the largest the probability of being correct can be. If we went with $\hat{x}$ however, then we loose out on the red region for being correct, which is suboptimal. (I like to think of a three dimensional distribution here whereas Bishop has an image with two different distributions which would need to be normalized appropriately but the concept is the same).

We can use the product rule to write:

$$\begin{aligned} P(X \in R_1, 1) > P(X \in R_1, 0) &\implies P(1|X \in R_1)P(X \in R_1) > P(0|X \in R_1)P(X \in R_1) \\ &\implies P(1|X \in R_1) > P(0|X \in R_1) \end{aligned} \tag{1.2}$$

So the maximization problem is equivalent to choosing the higher conditional probability for each region. This rule is known as the *Bayes classifier* and the error rate of the Bayes classifier is known as the *Bayes rate*. The Bayes classifier is used as a benchmark in classification as it is the optimal solution to classification if the probability distributions are known.

1.1.2 Expected Loss Function

Imbedded in the above discussion describing how to find the optimal decision rule is a concept called the *loss function*. This is a function that takes as input the true class and predicted class (resulting from the chosen decision rule) and outputs a value encoding the error of the prediction. In the above examples we assume that the loss function is outputting a 0 for each class predicted correctly and a 1 for each class predicted incorrectly, so in other words all classes are weighted the same in terms of misclassification (this is also known as the 0-1 loss function). In some applications however, such as medical diagnosis, we want to weight some classes higher than others when calculating misclassification. For example, when diagnosing cancer it is much better to predict someone who is healthy as having cancer than the other way around.

For classification we can think of this function as a matrix known as the *loss matrix*, but in general we can think of it as taking in two variables - the true class and the predicted class:

$$L(G, \hat{G}(X)). \tag{1.3}$$

One issue with using this measure however is that we don't know the true class G. We can choose some decision rule to get us $\hat{G}$, but since we are dealing with probability distributions we won't know for sure whether the true class is a high-sales ice cream day or a low-sales ice cream day for example. Instead of finding $\hat{G}$ that minimizes the loss function, we can instead minimize the expectation of the loss function or in other words minimize the average loss function weighted by the probabilities for G and X.

$$\begin{aligned} E[L(G,\hat{G}(X))] &= \iint_{G,X} L(G,\hat{G}(X))P(G,X)dGdX \\ &= \int_X \sum_{k=1}^{K} L(G_k,\hat{G}(X))P(G_k|X)P(X)dX \\ &= E_X \sum_{k=1}^{K} L(G_k,\hat{G}(X))P(G_k|X) \end{aligned} \tag{1.4}$$

where $k = 1,...K$ are all of the classes, in our example either 0 or 1. We want to find a classifier $\hat{G}(X)$ such that the expected loss is minimal. To do this we can minimize the inner quantity pointwise since this corresponds to the minimum of the entire quantity (the minimum of an average is the minimum of the separate quantities in the average). This leads us to write:

$$\hat{G(x)} = \text{argmin}_{g\epsilon G} \sum_{k=1}^{K} L(G_k, g)P(G_k|X) \tag{1.5}$$

If we are using the 0-1 loss function then we can simplify this to:

$$\hat{G(x)} = \text{argmin}_{g\epsilon G} \left[1 - P(g|X=x)\right]. \tag{1.6}$$

This took some thought for me to understand why we could simplify down to this. I think the best way to see it is to remember that this is a function of g. If we have $K = 3$ for example then we can write out for each possible value of G_k:

$$\begin{aligned} g = G_1 &\implies P(G_2|X=x) + P(G_3|X=x) \implies 1 - P(G_1|X=x) \\ g = G_2 &\implies P(G_1|X=x) + P(G_3|X=x) \implies 1 - P(G_2|X=x) \\ g = G_3 &\implies P(G_1|X=x) + P(G_2|X=x) \implies 1 - P(G_3|X=x) \end{aligned} \tag{1.7}$$

since our loss function is 0 when it is a true classification and 1 when it is a misclassification. Since we are minimizing, the best choice for g is the one where $P(g|X = x)$ is the largest (for each x) which corresponds to the Bayes classifier. Thus, we have proven that under the 0-1 loss function, the optimal decision is the Bayes classifier as we found in our previous discussion. Note that this is optimal when we know the distribution which most times we don't.

When we look more closely at the expected loss function $E[L(G,\hat{G}(X))]$, implied in here is some decision that is made that is represented by $\hat{G}(X)$. Since this decision is typically made based off of some sample (training data) and that sample is considered random, then this implies that we are omitting or hiding a important random variable - the random variable representing the sampling process. To see this better we can write the expectation above in the form of two expectations. The first is known as the *test error*, the *generalization error*, or *prediction error*, all according to ESLII:

$$E[L(G,\hat{G}(X))|T]. \tag{1.8}$$

The variable T represents a training set that we use to make a decision rule $\hat{G}(X)$. The training set T is fixed in this expectation and the expectation is over the entire distribution of X and G. Essentially, this is measuring the expected loss on potential new data fed into a model that was built on a given training set T. It is measuring how well our model generalizes to the entire population based off a model built on T.

If we take an expectation over all training sets and everything that is random then we have the original expected loss talked about earlier:

$$\begin{aligned}
E_T[E[L(G,G(\hat{X}))|T]] &= E_T[E_{G,X}[L(G,G(\hat{X}))|T]] \\
&= E_T\left[\iint_{G,X} L(G,\hat{G}(X))P(G,X|T)dGdX\right] \\
&= \iiint_{G,X,T} L(G,\hat{G}(X))P(G,X|T)P(T)dGdXdT \\
&= \iiint_{G,X,T} L(G,\hat{G}(X))P(G,X,T)dGdXdT \\
&= E_{G,X,T}[L(G,\hat{G}(X))] \\
&= E[L(G,\hat{G}(X))]
\end{aligned} \tag{1.9}$$

ELSII calls this the *expected test error* or *expected prediction error*. This measures how well our average model generalizes to the entire population, where average model is referring to the average model over all training sets T.

Really our goal at this point would be to find the prediction error for a given training set T. This is the goal because we are only given one training set and we want to know what the generalization error is over the entire distribution of Y, X for the model derived from that training set. It appears that this is harder to do in practice and most methods actually estimate the *expected* prediction error better. Therefore we will focus on estimating the expected prediction error. See ESLII pg. 220 for more discussion.

The discussion of the loss functions above uses random variables and expectations and is referring to when we know the distributions involved. In practice however, we are only given a sample from that distribution and so when finding a model we use whats known as the *cost function* which adds up the loss function for each data point being used:

$$J(\theta) = \sum_{i=1}^{N} L(g_i, \hat{G}_\theta(x_i)) \tag{1.10}$$

TODO: Show graphs of general loss functions - see Stanford machine learning cheat sheet
TODO: I think the expected loss is also known as the Risk function
TODO: Show the regression loss function formulation. Note that the conditional mean is the optimal decision under the least squares loss just like the Bayes classifier is under the 0-1 loss. These loss functions are nice theoretically but may not be realistic in practice
TODO: Explore the idea that the conditional mean is optimal IF we know the distribution just like the Bayes Classifier is optimal IF we know the distribution.
TODO: Explore the dimensionality issue

1.1.3 Bias-Variance Tradeoff

The bias-variance tradeoff refers to two sources of error when evaluating models - the bias and the variance. There is also a third source of error which we call the "irreducible error".

As explained in this article, there is a slight confusion in data science between decomposing the error for an estimator, and decomposing the error for a model or a predictor. The decomposition is really about the same but there are some key insights to be aware of. The decomposition below is for a predictor. The decomposition for an estimator can be found in various books and other resources such as Casella/Berger.

First of all the bias of a model is defined as:

$$\text{Bias}\left(\hat{f}(X)\right) = E\left[\hat{f}(X) - f(X)\right] \tag{1.11}$$

and variance of a model is:

$$\text{Var}\left(\hat{f}(X)\right) = E\left[\hat{f}(X)^2\right] - E\left[\hat{f}(X)\right]^2. \tag{1.12}$$

Knowing these definitions we can then take the expected loss function and perform the following decomposition (assuming squared error loss):

$$\begin{aligned}
E[L(Y,\hat{f}(X))] &= E_T[E_{Y,X}[L(Y,f(\hat{X}))|T]] \\
&= E_{X,Y|T}[E_T[L(Y,f(\hat{X}))]] \quad \text{(Drop } |T \text{ at this point since inner expectation is over } T\text{)} \\
&= E_{X,Y}[E_T[(Y-\hat{f}(X))^2]] \\
&= E_{X,Y}[E_T[(Y^2-2Y\hat{f}(X)+\hat{f}(X)^2)]] \\
&= E_{X,Y}[E_T[Y^2]-E_T[2Y\hat{f}(X)]+E_T[\hat{f}(X)^2]+E_T[\hat{f}(X)]^2-E_T[\hat{f}(X)]^2] \\
&= E_{X,Y}[Y^2-2YE_T[\hat{f}(X)]+E_T[\hat{f}(X)^2]+\text{Var}_T(\hat{f}(X))] \\
&= E_{X,Y}[(Y-E_T[\hat{f}(X)])^2+\text{Var}_T(\hat{f}(X))] \\
&= E_{X,Y}[\text{Bias}_T(\hat{f}(X))^2+\text{Var}_T(\hat{f}(X))]
\end{aligned} \tag{1.13}$$

This shows that for the squared error loss we can decompose the expected loss function into a bias term and a variance term. There is typically an irreducible error term in most decompositions but those decompositions make additional assumptions (such as constant variance) and I wanted to stay more general. In reality the irreducible error term is rolled up in the expectation over X, Y. One another note to make here is that I think we can drop the condition on T since the only quantity that depends on T is the model $\hat{f}(x)$ and since the model is wrapped up in an expectation over T then we don't need to worry about the conditional.

What this decomposition reveals are different sources for error. The variance term reveals how much a model varies over training sets. The bias term reveals how far off our model is averaged over all training sets. Different models perform differently in regards to these two terms. Linear regression for example has high bias (meaning if we average over all training sets, the model is relatively wrong), but low variance (the model won't change drastically with a new training dataset). Decision trees are the opposite - they have low bias (over all training sets the average tree is relatively not too far off the truth), but high variance (decision trees can look completely different depending on the given training dataset).

It appears that ensembles can decrease both sources of error by averaging low bias models. When we take an average the variance decreases (see discussion on Bagging and Random Forest).

One issue that isn't as satisfying to me here is that this neat decomposition is for squared-error loss only. Most textbooks leave the bias-variance decomposition at this point. What bugs me is there is really no discussion about this decomposition for a *general* loss function. One paper I found that addresses this issue (sort-of) is found here by Pedro Dominguez. I'd like to explore this a little more. Also see here for a python package that gives you the decomposition and refers to the Dominguez paper.

1.1.4 Generative vs. Discriminative Models

The previous discussions about probability distributions that explain the relationship between dependent and independent variables sets us up nicely for understanding what generative vs. discriminative models are. **Generative models** are models that attempt to find or approximate the original distribution $P(X,Y)$. They are called generative because once we've found a generative model we can *generate* synthetic data from the model, inputs AND outputs. We can also use generative models to make predictions by using Bayes rule to find the posterior distribution $P(Y|X)$ and then use decision theory to make the prediction (essentially assign the instance to the class with the highest probability distribution, if we are using a traditional loss function). **Discriminative models** attempt to model the posterior density $P(Y|X)$ directly and then use decision theory to make predictions using that posterior density.

Both of these approaches first do whats called the *inference stage* (finding the distribution) and then use the posterior probabilities in the *decision stage*. A third option exists where we directly find a function $f(X)$ that maps inputs to outputs. The function $f(X)$ is known as a *discriminant function*.

There are pros and cons to each approach as Bishop mentions which I summarize here.

Generative model pros:

- Allows us to find the marginal density $P(X)$ which tells us the likelihood of given inputs and helps us identify inputs that may not be common and therefore less accurate. This is a form of outlier detection.
- Allows us to generate synthetic data
- TODO: Read Andrew Ngs' and Michael Jordan's paper

Generative model cons:

- Could be considered a waste of effort if only goal is prediction.
- Since we are attempting to find the entire density $P(X,Y)$ we may need more data in order to find accurate posterior distributions.

Discriminative model pros:

- Once we've found this model and our loss function changes, then we only need to change the loss function - we don't need to retrain the entire model compared to a discriminate function.
- Reject option - Bishop likes this concept where we can determine areas we aren't as confident the model can do a good job with and instead ask a human to make the classification.
- We can deal better with class imbalance - TODO: Bishop has a good synopsis that I might write in later
- Combine models - TODO: Gives an example of the naive Bayes model

TODO: I think I've made a connection here between the discriminative model and discriminative function. For least squares the optimal decision is the condition mean and that is what f(x) is below.

The discriminative model $P(Y|X)$ allows for completely summarizing the way Y depends on X. When we use another model like the additive error model, we make a further assumption that the errors are independent of X and that they have a constant variance. So the additive error model puts further constraints on the discriminative model. We can still think of the additive error model as some conditional distribution, but a distribution that is simplified.

We can also go the other direction by thinking of the discriminative model $P(Y|X)$ as the equation $y_i = f(x_i) + \epsilon_i$ but not putting constraints of any kind on ϵ_i. The idea is that there is some "true function" out there and then there are some errors off of that true function that gives us our dependent variable.

In either case the next step is to then think of other modeling assumption for $f(x_i)$ such as the linear model.

1.2 Generalized Linear Models

There are three main components that make up the Generalized Linear Model (GLM):

1. Random component - assume the response variable comes from a probability distribution

$$Y_i \sim f(\mu_i) \tag{1.14}$$

   where $\mu_i = E(Y_i)$ and f is a probability distribution.

2. Link component - connects the random component to the systematic component

$$g(\mu_i) = \eta_i \tag{1.15}$$

3. Systematic component - this is the linear part

$$\eta_i = x_i'\beta \tag{1.16}$$

1.2.1 Logistic Regression

Using the component concepts outlined above, for logistic regression we have:

1. Random component: $Y_i \sim f(\pi_i)$ where f is the Bernoulli distribution (since Y will be a binary variable when using logistic regression). $E(Y_i) = \pi_i$ which is the probability that Y_i is 1.

2. Link component: this is the logit function which is defined as:

$$\text{logit}(\pi_i) = \log\left(\frac{\pi_i}{1-\pi_i}\right) = \eta_i \tag{1.17}$$

3. Systematic component: tying this all together we have:

$$\log\left(\frac{\pi_i}{1-\pi_i}\right) = x_i'\beta \tag{1.18}$$

Assumptions:

1. Linearity in log-odds

2. Independence of $Y_i|x_i$ and $Y_j|x_j$

3. Bernoulli response variable

For interpretation we can say that with a one unit increase in X_1 for example, then the log odds of a success goes up by β_1. We can also use the multiplicative odds where a one unit increase in X_1 leads to a e^{β_1} multiplicative change in odds on average.

The probability can be calculated by:

$$\begin{aligned}
\log\left(\frac{\pi_i}{1-\pi_1}\right) &= \beta_0 + x_i\beta_1 \\
\frac{\pi_i}{1-\pi_i} &= e^{\beta_0+x_i\beta_1} \\
\pi_i &= e^{\beta_0+x_i\beta_1} - e^{\beta_0+x_i\beta_1}\pi_i \\
\pi_i &= \frac{e^{\beta_0+x_i\beta_1}}{1+e^{\beta_0+x_i\beta_1}}
\end{aligned} \tag{1.19}$$

1.3 Ensemble: Bagging

Bagging (bootstrap aggregating) is based on the concept that the variance of averaged random variables is less than the variance of the random variables individually. To see this say we have a random sample $X_1, ..., X_N$ where the mean of X_i is μ and the variance is σ^2 . The expected value of the average of this random sample is $\frac{1}{N}\sum_{i=1}^{N} E[X_i]$ and the variance can be decomposed as:

$$\begin{aligned}
Var\left(\frac{1}{N}\sum_{i=1}^{N}X_i\right) &= E\left[\left(\frac{1}{N}\sum_{i=1}^{N}X_i\right)^2\right] - E\left[\frac{1}{N}\sum_{i=1}^{N}X_i\right]^2 \\
&= E\left[\left(\frac{1}{N}\right)^2\left(\sum_{i=1}^{N}X_i\right)^2\right] - \left(\frac{1}{N}\right)^2 E\left[\sum_{i=1}^{N}X_i\right]^2 \\
&= \left(\frac{1}{N}\right)^2\left(E\left[\left(\sum_{i=1}^{N}X_i\right)^2\right] - E\left[\sum_{i=1}^{N}X_i\right]^2\right) \\
&= \left(\frac{1}{N}\right)^2 Var\left(\sum_{i=1}^{N}X_i\right) \\
&= \left(\frac{1}{N}\right)^2 N\sigma^2 \\
&= \frac{\sigma^2}{N}
\end{aligned} \tag{1.20}$$

The second to last step is possible because the X_i are independent and the variance of a sum of independent variables is the sum of the variances. Since X_i are identically distributed then the variance will be σ^2 for all X_i.

This same idea applies to predictors or models. To apply this concept to models, B bootstrapped samples are derived and a model $\hat{f}^b(x)$ is built on each sample. These samples are then averaged together:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x). \tag{1.21}$$

TODO: Expound on bagging a little more especially the connection to a posterior Bayes mean, squared error vs. 0-1, and the error breakdown between bagging vs. single model.

TODO: The method of bagging mentioned previously is an attempt to reduce the variance of models by averaging them together. This seems to work particularly well for noisy, high-variance models such as trees. When we average these trees together the bias does not decrease (related to the idea that the expected value of a random variable is the same as the expected value of the average of i.d. random variables), but the variance does decrease. The amount it decreases depends on whether or not the random variables are i.i.d. or i.d. If i.i.d then variance decreases by a factor of N

1.4 Ensemble: Random Forrest

Bagging averages models together to reduce error, in particular the variance component to the error. As mentioned, this is analogous to averaging i.i.d. random variables together with variance σ^2 leading to an overall variance of $\frac{\sigma^2}{B}$ where B are the number of random variables in the average.

The issue with this approach is when the models (or random variables in our analogy) are correlated.

1.5 Ensemble: Bumping

This is where we train models on various bootstrapped samples and choose the best bootstrapped sample and the model on that sample. Essentially we are expanding the space of possible models.

1.6 Ensemble: Boosting

The concept of boosting has lead to some of the most powerful algorithms in machine learning. Boosting falls under a general class of algorithms known as ensembles (bagging would be another example of ensemble algorithms where we run separate models and then aggregate at the end by averaging). The general concept of boosting is that we use a *weak learner* (a model that does only slightly better than random guessing) to model the original data, calculate the errors, run a new weak learner model on the errors, combine the results with the first weak learner, and repeat until some stopping criteria (that avoids overfitting). Thus boosting algorithms stack multiple learners on top of each other instead of modeling separately and then combining in some way at the end like bagging.

At its heart boosting is really a simple basis function expansion or an additive model. This type of model attempts to approximate a function by treating it as a linear combination of other functions, usually more simple functions:

$$f(x) = \sum_{m=1}^{M} \beta_m b(x;\gamma_m) \tag{1.22}$$

where β_m are the basis function coefficients and $b(x;\gamma_m)$ are the basis functions with parameters γ_m. The ideal way of fitting this model would be to minimize some loss function by finding the optimal parameters γ_m and coefficients β_m (both for all m) all at once, but in practice this can be computationally intensive.

An alternative to this approach which approximates the optimal solution to 1.22 is *forward stage-wise additive modeling*. Instead of optimizing over all basis functions at once we instead optimize over one basis function at a time while keeping all previously found basis functions fixed. To be more clear we first fit a weak learner $f_0(x)$ to the data and then in a loop we find models for $m = 1$ to M minimizing the cost function over the training data:

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)) \tag{1.23}$$

$$f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m) \tag{1.24}$$

This last equation is the recursive relationship that is often written out to describe boosting but more generally with different notation might be written out as:

$$F_m(x) = F_{m-1}(x) + f_m(x) \tag{1.25}$$

Any loss function can be used, but its interesting to look at the scenario when the loss function is the squared error loss. In this case we would have:

$$
\begin{aligned}
(\beta_m, \gamma_m) &= \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)) \\
&= \arg\min_{\beta,\gamma} \sum_{i=1}^{N} (y_i - f_{m-1}(x_i) - \beta b(x_i;\gamma))^2 \\
&= \arg\min_{\beta,\gamma} \sum_{i=1}^{N} (r_{im} - \beta b(x_i;\gamma))^2
\end{aligned}
\tag{1.26}
$$

where r_{im} is the residual between the previous model's prediction and the observed target value y_i. This implies that when using the squared loss function we are training the m^{th} model on the error of the previous model, which in multiple dimensions is a vector giving us direction and magnitude (this is key when thinking of how this relates to the gradient). The resulting m^{th} model will then be an approximation (not perfect, no models ever are) to the error which is then added to the $(m-1)^{th}$ model to help correct that model. From this perspective we can think of each subsequent model as attempting to approximate the error of the previous model and account for that error in the overall model.

We can use different loss functions that lead to different insights. For example, if we use the absolute value error loss we will end up training our weak learners on the $sign$ function. If we use the exponential loss function then we get Adaboost which is discussed in section 1.6.2.

1.6.1 Gradient Boosting

The discussion in the previous discussion is fairly straightforward and intuitive, especially when considering the squared loss function. However, we can better generalize and relate the concepts above to some well established mathematical techniques to give better clarity, namely the gradient descent algorithm.

The gradient or steepest descent algorithm is an iterative optimization technique to find the minimum of a function. There is no guarantee that we will be able to find the global minimum, instead we settle for a local minimum that is hopefully near the global minimum. There are different related techniques to gradient descent but gradient descent is the classic. A good overview of gradient descent methods is found here, which TODO: I briefly summarize below. TODO: put this in an optimization section?

The general gradient descent algorithm is typically written as:

$$
x_t = x_{t-1} + \eta\left(-\nabla f(x_{t-1})\right) \tag{1.27}
$$

Notice that this is a recurrence relationship, very similar to the one mentioned in 1.25. Here the goal is to find the location x_t that minimizes the function f. For gradient boosting however, we can plug in $F_m(x)$ for x_t. The function $f(x_{t-1})$ can be substituted by the loss function L. This gives us:

$$
F_m(x) = F_{m-1}(x) + \eta\left(-\nabla L(y, F_{m-1}(x))\right) \tag{1.28}
$$

If we were to write out the vectors in this equation for $(x_i, y_i), i = 1, ..., N$ we would have:

$$\begin{bmatrix} F_m(x_1) \\ \dots \\ F_m(x_N) \end{bmatrix} = \begin{bmatrix} F_{m-1}(x_1) \\ \dots \\ F_{m-1}(x_N) \end{bmatrix} + \eta \left(- \begin{bmatrix} \frac{\partial L(y_1, F_{m-1}(x_1)}{\partial F_{m-1}(x_1)} \\ \dots \\ \frac{\partial L(y_N, F_{m-1}(x_N)}{\partial F_{m-1}(x_N)} \end{bmatrix} \right) \tag{1.29}$$

If the loss function is the squared error loss then $\nabla L(y, F_{m-1}(x))$ will become $-2(y - F_{m-1}(x))$ which is the residual, or the error between the $m-1^{th}$ model and the true dependent variable. This implies then that when we train each subsequent model on the residual we are in reality performing gradient descent in prediction space and are iteratively approaching the minimum of the loss function which implies we are getting closer to the target values. We can summarize the general gradient boosting algorithm as follows:

1. Train a weak learner $F_0(x)$ on the original training data $(x, y)_T$.
2. Train a weak learner on $-\nabla L(y, F_0(x))$.
3. Add this weak learner to $F_0(x)$ to get $F_1(x)$.
4. Repeat for $m = 1$ to M until some stopping criteria.

TODO: Expand this out for decision trees and for XGBoost

1.6.2 Adaboost

2 Terms and Notation

2.1 Variable notiation

Below explains notation used commonly when setting-up machine learning models. Note that all vectors are assumed to be column vectors. To help understand the notation I use the example of predicting the sales of ice cream cones.

- X - represents an input variable. Even though input variable implies a single variable this could also be a vector. If we wanted to access a single variable from the input vector then we use notation X_j. So for example X could include variables that describe the temperature (X_j), time or year (X_{j+1}), etc.
- Y - represents a *quantitative* output variable. This could be the sales of ice cream cones in dollars.
- G - represents a *qualitative* output variable. This could be if we sale over 50 ice cream cones for example (yes or no).
- x_i - represents an observed value of the variable X. Again this could be a vector. So to get the observed scalar value of the temperature for example we would write x_{ij}.
- $\boldsymbol{X}$ - matrix typically with dimensions Nxp.
- $\boldsymbol{x_j}$ - in general vectors are not bold unless the distinction is being made that this is the vector of all observation on X_j. So $\boldsymbol{x_j}$ is of length N and x_i is of length p.

3 Basic Statistical Concepts

3.1 Inference

Inference is referring to using data to figure out the underlying properties of a population (which in turn allows us to understand the relationship between variables). I've been thinking of inference as referring to the process to understand the relationship between variables in a linear regression, but I think this is too narrow of a view. For example, if we look at using a t-test to compare two samples what we are really doing is using the data to estimate what the two distributions are that the data comes from and then determine if that is reasonable or not. TODO: How do the various techniques in statistics fit in with this idea of finding the parameters of the underlying data?

Glossary

dummy variable A vector where each element is either 0 or 1 and is used to represent a specific class. For example, if we have K classes then a dummy variable would be of length K and if we wanted to represent class 1, we would have a "1" in the first position in the vector and everywhere else would be 0.. 1

estimator A point estimator as defined by Cassella/Berger is any function $W(X_1, X_2, ..., X_n)$ of a sample. Any statistic is an estimator.. 1, 2

test A categorical variable that has ordering such as low, medium, and high, but no notion of a metric.. 1