Data Science Toolkit

Matt Goodwin

June 24, 2019

Contents

1 Statistical Modeling Overview and Basic Theory

1.1 Overview

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. This is the approach that ESL takes. Taking this approach allows us to use probability theory combined with decision theory.

Bishop, from his book *Pattern Recognition and Machine Learning*, has a really nice overview of some of these concepts. The starting point I think for modeling, at least in a supervised setting, starts with the independent variable or covariate X and dependent variable Y (see notation section). We want to know:

1. The nature of the relationship between the variables (inference).
2. Given an independent variable, determine the dependent variable (prediction).

Bishop mentions that by 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 some of 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 ESL mentions, sometimes the relationship IS deterministic but the randomness comes from the fact that we have limited data. If we have a different training data set then we get different results but the underlying relationship is still the same since it is deterministic.

[1] attributed to George Box

These types of problems can be handled by similar techniques where the relationship between the variables is probabilistic (see pg 28 of ESL).

Along the above point, it may be tempting to think that the more features the better because we would be getting closer to a deterministic relationship where we could predict perfectly. The issue with this however is related to the problem just mentioned that when we build models we have only a sample from the distribution. We could then start to tune our model to the specific data set but not the true distribution. So I imagine a lot of features would be fine to have, but only if we have more and more data that approximate the true distribution. See the section on curse of dimensionality for more discussion along this point.

1.2 Decision Theory

As mentioned above, one key area of interest in understanding the relationship between X and Y is 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:

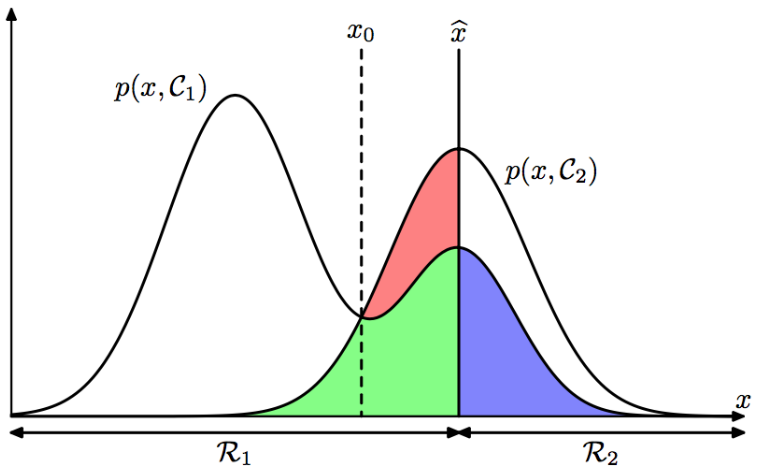
$$P(mistake) = P(X \in R_1, 0) + P(X \in R_0, 1) \tag{1.1}$$

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

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



area under the distribution of being correct.

To visualize this better refer to figure 1.2 from Bishop which is on pg 40 in his book. 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.

1.3 Expected Loss Function

Embedded 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. We can use this formulation to find the optimal decision rule by minimizing the function with respect to the decision rule.

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:

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

where G is the true class and $\hat{G}(X)$ is the predicted class - X representing the independent variables.

As Bishop points out, 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 the different classes, in our example either 0 or 1. We want to find a classifier $\hat{G}(X)$ such that the expected loss is minimal. As ESL illustrates, 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:

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

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

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

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. The point to make here is that in the presence of uncertainty, the Bayes classifier is really the best we can do under the 0-1 loss function.

The above discussion is more theoretical in nature than practical. In reality we will not know what the true distribution looks like, and instead only have a sample to work with. In order to make this minimization problem slightly more practical (but dealing with distributions still) we need to include the random variable that represents the sampling process. To see this better we write the expected loss in the form of two expectations. The first is known as the *test error*, the *generalization error*, or *prediction error*, all according to ESLII:

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

where the variable T represents the training set. Optimizing this expectation now should yield a different answer than before because of the dependence on T. We can also think of this quantity as the expected loss given training set 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 according to ESLII, would be to find (or approximate with a sample) 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 G,X for the model derived from that particular 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, as was mentioned before, 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:

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

1.4 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):

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

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 other note to make here is that I think we can drop the condition on T in the derivation 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 appears to be for squared-error loss only. Most textbooks leave the bias-variance decomposition discussion 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.

1.5 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. Bishop does a good job explaining the difference. **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

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 the discriminate function approach.
- 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

The discriminative model $P(Y|X)$ allows us, as Bishop says, to completely summarize 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.

1.6 Curse of Dimensionality

The *curse of dimensionality* [2] refers to the problem that models face in many dimensions. As both ESL and Bishop describe, our intuition break down in many dimensions. For example, Bishop gives the example of points in a unit sphere. In general the volume of a hypersphere can be written as:

$$V_n(R) = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} R^n. \tag{1.14}$$

where R is radius, n is the number of dimensions and Γ is the gamma function. So for example if we are in two dimensions ($n = 2$) we would get:

$$\frac{\pi}{\Gamma(2)} R^2 = \pi R^2. \tag{1.15}$$

What is interesting is if we consider a hyper sphere in D dimensions and then an inner hyper sphere of radius ϵ contained within the larger sphere. If we write out the volume between these two spheres, hold radius to 1, and consider this quantity as a proportion to the larger outer sphere we get:

$$\frac{\frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)} - \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)}(1-\epsilon)^D}{\frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)}} \tag{1.16}$$

[2] attributed to Richard Belllman

Simplifying we are left with:

$$1-(1-\epsilon)^D. \tag{1.17}$$

From this equation we can see that as we increase the dimension D, the quantity gets closer and closer to 1. Since the quantity is a proportion over the outer sphere this implies that most of the volume is in between the inner sphere and the outer sphere as the dimension increases. This is even true when the inner sphere is close to the same size as the outer sphere (when 1-ϵ is really small).

A better example roughly given by Bishop illustrates the issue when applied to modeling. Consider a naive model where for each new data point we assign a predicted label according to a majority rule within a uniform neighborhood (similar to nearest neighbors but splitting the space up into uniform cubes instead of a neighborhood around each data point). In one dimension say we look at the range between 0 and 4 and split the space up into 4 intervals of 1 unit each. If we are given a new point, 1.5 for example, then we assign that data point the majority label of all other points in the interval 1 to 2.

Say however, that we add a second dimension with the same range. The number of unit squares will now be 16. If we go to three dimensions we now have 64 squares, four dimensions 256, etc. This is an issue because in order to determine what each point should be in a given unit cube, we need to make sure we have data in each cube, implying that we need an exponentially increasing amount of data as we increase our dimensions. More specifically in one dimension we would need at least 4 data points to have a data point in each interval, but in four dimensions we would need at least 256 data points (however, in both scenarios there is no guarantee that we get a data point in each interval or cube, this just illustrates a general rule).

So what to do if we have limited data and many dimensions? There are a couple of ideas from Bishop to keep in mind that give us hope. The first idea is that in practice data tends to be more concentrated. This implies that the true dimensionality is potentially much lower. The other idea that Bishop mentions is that "real data will exhibit some smoothness properties" locally (see pg. 37). This I think implies that we can use models that rely on these assumptions. This last point is a little fuzzy to me still but I think ESL (on pg 32-33) sheds a little light by talking about the *complexity* of a model, in particular restraining the complexity of the model. As ESL mentions "this usually means some kind of regular behavior in small neighborhoods of the input space". Instead of using a nearest neighbor type approach for example, we can assume the data is linear in these local neighborhoods (which is a constraint on complexity) and use linear regression. The central tradeoff is that for models that do really well locally (nearest neighbors) face the curse of dimensionality and models that overcome the curse of dimensionality may not do well locally.

In summary, the curse of dimensionality is an important concept to keep in mind because it invalidates or severely handicaps naive models such as nearest neighbors in high dimensions and has forced the field of statistical modeling to develop more clever models. Many of these models include various assumptions (pg. 27 of ESL) to get at the true nature of the data, such as linear regression. When these assumptions are correct then the models have a chance at performing well and we have avoided the "exponential growth in complexity of functions" (ESL).

1.7 No Free Lunch Theorem

Bishop has a great summary of this concept from a podcast with Microsoft Research. To paraphrase, the No Free Lunch Theorem implies that all machine learning algorithms perform equally, averaging across all possible problems. In other words as Bishop says "there cannot be a single universal machine learning algorithm that will solve all problems" (see podcast transcript).

The caution here however is that this theorem is more of an abstract concept and really there may be algorithms that perform consistently well on the problems we care about or that we see in the real world (think deep learning for example). The larger point to take-away as Bishop mentions however, is that there are really two parts to a problem: the data and the assumptions (or in other words priors, constraints, etc.). Both are important and he makes the case for treating assumptions as "first-class citizens", not just the data. This makes a lot of sense to me because if our data is truly linear then a linear model would potentially do much better than a decision tree for example.

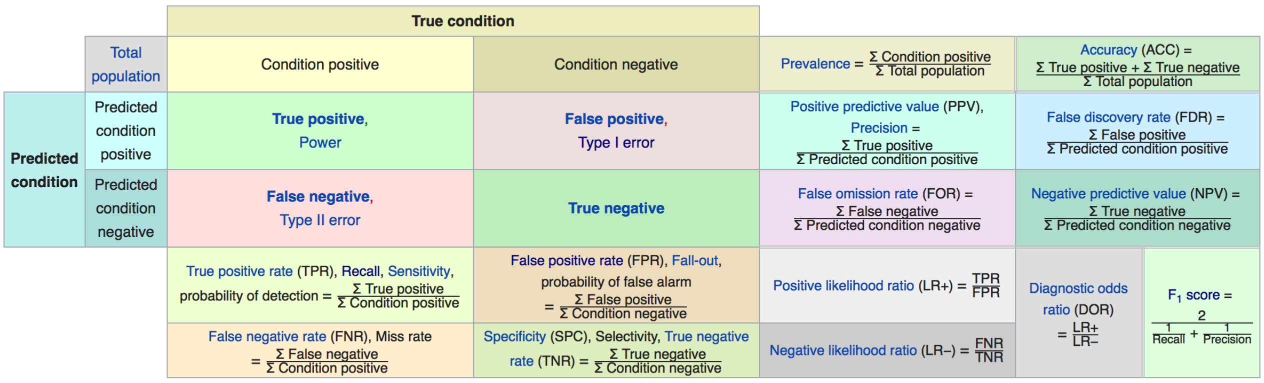
2 Assessment of Models

2.1 Classification Metrics

2.1.1 Confusion Matrix

Perhaps the place to start when evaluating classification models is the confusion matrix. Figure 2.1.1 shows an excellent figure from Wikipedia displaying what this matrix looks like.

Figure 2.1: Plot from Wiki showing confusion matrix



		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$	F_1 score = $\frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$		

The columns represent the true class (positive or negative) and the rows represent the predicted class (positive or negative). The cross-intersection of these predictions and truth create four areas: *true positives* (number of instances that are predicted positive that actually are positive), *false positives* (number of instances that are predicted positive that are actually negative), *false negatives* (number of instances that are predicted negative that are actually positive) and *true negatives* (number of instances that are predicted negative that actually are negative).

In the Wiki figure there are three other terms in the 2x2 confusion matrix namely power, Type 1 error, and Type 2 error. TODO

3 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

$$Y_i \sim f(\mu_i) \tag{3.1}$$

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

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

$$g(\mu_i) = \eta_i \tag{3.2}$$

3. Systematic component - this is the linear part

$$\eta_i = x_i'\beta \tag{3.3}$$

3.0.1 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:
$$\text{logit}(\pi_i) = \log\left(\frac{\pi_i}{1-\pi_i}\right) = \eta_i \tag{3.4}$$
3. Systematic component: tying this all together we have:
$$\log\left(\frac{\pi_i}{1-\pi_i}\right) = x_i'\beta \tag{3.5}$$

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:

$$\begin{aligned}
\log\left(\frac{\pi_i}{1-\pi_1}\right) &= \beta_0 + x_i\beta_1 \\
\frac{\pi_i}{1-\pi_i} &= e^{\beta_0 + x_i\beta_1} \\
\pi_i &= e^{\beta_0 + x_i\beta_1} - e^{\beta_0 + x_i\beta_1}\pi_i \\
\pi_i &= \frac{e^{\beta_0 + x_i\beta_1}}{1 + e^{\beta_0 + x_i\beta_1}}
\end{aligned} \tag{3.6}$$

4 Bayesian Statistics

4.1 Bayesian Hierarchical Models

The majority of this section comes from Bayesian Data Analysis (BDA) by Gelman et. al.

The need for hierarchical models arise when we have data that are dependent on parameters, which in turn are related to each other. The example that BDA gives is of the study of cardiac treatment effectiveness. It is reasonable to assume that the data y_{ij} collected over different hospitals come from different parameters, with a parameter for each hospital. We can then treat these parameters as coming from a higher distribution, and then estimate the parameters of the higher distribution based on the data. This allows us to control for the group level effect and helps us fit a better model.

5 Ensemble

5.1 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:

$$\begin{aligned}
Var\left(\frac{1}{N}\sum_{i=1}^{N} X_i\right) &= E\left[\left(\frac{1}{N}\sum_{i=1}^{N} X_i\right)^2\right] - E\left[\frac{1}{N}\sum_{i=1}^{N} X_i\right]^2 \\
&= E\left[\left(\frac{1}{N}\right)^2\left(\sum_{i=1}^{N} X_i\right)^2\right] - \left(\frac{1}{N}\right)^2 E\left[\sum_{i=1}^{N} X_i\right]^2 \\
&= \left(\frac{1}{N}\right)^2\left(E\left[\left(\sum_{i=1}^{N} X_i\right)^2\right] - E\left[\sum_{i=1}^{N} X_i\right]^2\right) \\
&= \left(\frac{1}{N}\right)^2 Var\left(\sum_{i=1}^{N} X_i\right) \\
&= \left(\frac{1}{N}\right)^2 N\sigma^2 \\
&= \frac{\sigma^2}{N}
\end{aligned} \tag{5.1}$$

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:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x). \tag{5.2}$$

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

5.2 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. This is the case when using decision trees for example. If use our random variable analogy then

5.3 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.

5.4 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:

$$f(x) = \sum_{m=1}^{M} \beta_m b(x;\gamma_m) \tag{5.3}$$

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 5.3 is *forward stagewise 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:

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)) \tag{5.4}$$

$$f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m) \tag{5.5}$$

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:

$$F_m(x) = F_{m-1}(x) + f_m(x) \tag{5.6}$$

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:

$$\begin{aligned}(\beta_m, \gamma_m) &= \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)) \\ &= \arg\min_{\beta,\gamma} \sum_{i=1}^{N} (y_i - f_{m-1}(x_i) - \beta b(x_i;\gamma))^2 \\ &= \arg\min_{\beta,\gamma} \sum_{i=1}^{N} (r_{im} - \beta b(x_i;\gamma))^2\end{aligned} \tag{5.7}$$

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 5.4.2.

5.4.1 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:

$$x_t = x_{t-1} + \eta\left(-\nabla f(x_{t-1})\right) \tag{5.8}$$

Notice that this is a recurrence relationship, very similar to the one mentioned in 5.6. 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:

$$F_m(x) = F_{m-1}(x) + \eta\left(-\nabla L(y, F_{m-1}(x))\right) \tag{5.9}$$

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

$$\begin{bmatrix} F_m(x_1) \\ ... \\ F_m(x_N) \end{bmatrix} = \begin{bmatrix} F_{m-1}(x_1) \\ ... \\ F_{m-1}(x_N) \end{bmatrix} + \eta \left(- \begin{bmatrix} \frac{\partial L(y_1, F_{m-1}(x_1))}{\partial F_{m-1}(x_1)} \\ ... \\ \frac{\partial L(y_N, F_{m-1}(x_N))}{\partial F_{m-1}(x_N)} \end{bmatrix} \right) \tag{5.10}$$

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

5.4.2 Adaboost

6 Neural Networks

6.1 Neural Networks

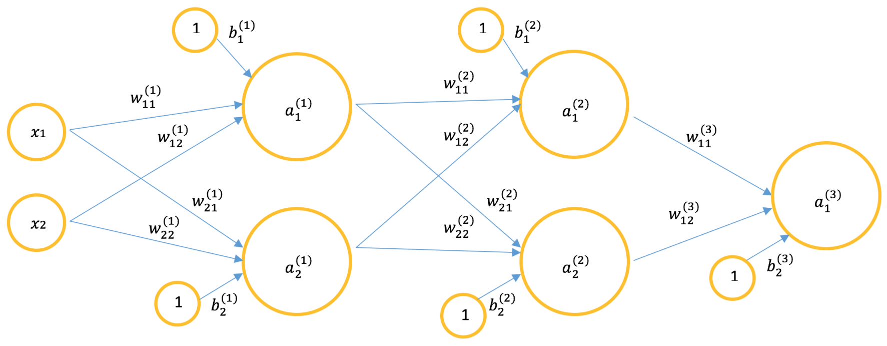
One of the best online resources I've discovered for understanding neural networks is from Michael Nielson. He's written an online textbook called "Neural Networks and Deep Learning" that can be found at `http://neuralnetworksanddeeplearning.com`. My notes and notation below are mainly based off of his chapter 2 for understanding the math behind neural nets. I've also included some notes from the Deep Learning book by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.

To help visualize these concepts, I've diagrammed a basic neural network in figure 6.1. The general idea of these networks is to pass in the original features x_1 and x_2 for example, as weighted linear combinations into the different nodes within the first layer. The first layer in this example has only two nodes. So for node 1 we pass in the linear combination:

$$x_1 w_{11}^{(1)} + x_2 w_{12}^{(1)} + (1) b_1^{(1)}. \tag{6.1}$$

The notation here is important to keep straight. For the weights the number in the superscript is referring to the current layer, the first number in the subscript is referring to which node the previous node output (or feature) is going to, and the second number in the subscript is referring to which node (or feature) the current weight is coming from. So for example, $w_{12}^{(1)}$ is a weight in the first layer (because of the superscript) and connects the second feature to the first node.

Figure 6.1: Example of a two layer neural network



This notation comes from Nielson's book and helps when we write everything out in matrix multiplication. To see what that looks like for a given layer:

$$\begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ \vdots \\ z_m^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & \dots & w_{1n}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & \dots & w_{2n}^{(1)} \\ \dots & \dots & \dots & \dots \\ w_{m1}^{(1)} & w_{m2}^{(1)} & \dots & w_{mn}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_m^{(1)} \end{bmatrix}$$

where n is the number of features, m is the number of nodes, and the z's are the weighted combinations. In shorthand we can write:

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}. \tag{6.2}$$

One other note to make here is that the bias ($b^{(1)}$) essentially just provides a constant in our linear combination since we are only multiplying it by 1.

Once this linear combination is fed into the node we pass it through what is known as an activation function. The purpose of this is to help our neural net model nonlinear behavior (see Deep Learning book for further discussion, pages 168-171). We represent the activation function with σ and write the output of the activation function as:

$$\mathbf{a}^{(1)} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) = \sigma(\mathbf{z}^{(1)}) \tag{6.3}$$

As Nielson points out we are treating the function σ here as a vectorized function. A more explicit way to write this out would be:

$$\begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_m^{(1)} \end{bmatrix} = \begin{bmatrix} \sigma(w_{11}^{(1)} x_1 & w_{12}^{(1)} x_2 & \dots & w_{1n}^{(1)} x_n + b_1^{(1)}) \\ \sigma(w_{21}^{(1)} x_1 & w_{22}^{(1)} x_2 & \dots & w_{2n}^{(1)} x_n + b_2^{(1)}) \\ \dots & \dots & \dots & \dots \\ \sigma(w_{m1}^{(1)} x_1 & w_{m2}^{(1)} x_2 & \dots & w_{mn}^{(1)} x_n + b_m^{(1)}) \end{bmatrix}$$

To continue this notation with each subsequent layer we really only need to change a few things. First of all, instead of the original features $x_1, x_2, ..., x_n$ we have $a^{(1)}, a^{(2)}, ..., a^{(m)}$ as the inputs into the next layer. This leads us to change our weight matrix as well so that we have a matrix that is pxm

where p is the number of nodes in the second layer and m is the number of outputs from layer 1 (before we had n representing the number of *features*). The other thing we need to change is the superscript for the variables such as $\mathbf{W}^{(1)}$ to $\mathbf{W}^{(2)}$ since we are in the second layer. For a generic layer from Nielson's book we can write:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}. \tag{6.4}$$

This leads us to also write:

$$\mathbf{a}^{(l)} = \sigma(\mathbf{z}^{(l)}). \tag{6.5}$$

Now that we have our notation straight we can talk about how the network is trained. Really at the heart of the backprop algorithm is gradient descent or some iterative optimization technique. To use gradient descent or similar techniques we need to calculate the derivatives of the cost function with respect to the parameters in the model or in the case of neural networks, the weights and biases. Because the neural network is made up of different layers or functions then what this amounts to is using the chain rule for each parameter.

Using our example and notation, lets find the derivative of the cost function with respect to the weight $w_{12}^{(1)}$ for example. In other words we want to find:

$$\frac{dC}{dw_{12}^{(1)}}. \tag{6.6}$$

A common loss function is the squared-error loss function or:

$$C = \frac{1}{2}(y_i - a_1^{(3)})^2 \tag{6.7}$$

where $a_1^{(3)}$ is the output from the last layer in our example. Note that we are doing this for one particular training example which we'll talk about later.

To get at the derivative for $w_{12}^{(1)}$ we rewrite the cost function above with all its nested functions:

$$\begin{aligned}
\frac{1}{2}(y_i - a_1^{(3)})^2 &= \frac{1}{2}(y_i - \sigma(w_{12}^{(3)}a_2^{(2)} + w_{11}^{(3)}a_1^{(2)} + b_2^{(3)}))^2 \\
&= \frac{1}{2}(y_i - \sigma(w_{12}^{(3)}\sigma(w_{22}^{(2)}a_2^{(1)} + w_{21}^{(2)}a_1^{(1)} + b_2^{(2)}) + w_{11}^{(3)}\sigma(w_{12}^{(2)}a_2^{(1)} + w_{11}^{(2)}a_1^{(1)} + b_1^{(2)}) + b_2^{(3)}))^2 \\
&= \frac{1}{2}(y_i - \sigma(w_{12}^{(3)}\sigma(w_{22}^{(2)}\sigma(w_{22}^{(1)}x_2 + w_{21}^{(1)}x_1 + b_2^{(1)}) + w_{21}^{(2)}\sigma(w_{12}^{(1)}x_2 + w_{11}^{(1)}x_1 + b_1^{(1)}) + b_2^{(2)}) \\
&+ w_{11}^{(3)}\sigma(w_{12}^{(2)}\sigma(w_{22}^{(1)}x_2 + w_{21}^{(1)}x_1 + b_2^{(1)}) + w_{11}^{(2)}\sigma(w_{12}^{(1)}x_2 + w_{11}^{(1)}x_1 + b_1^{(1)}) + b_1^{(2)}) + b_2^{(3)}))^2
\end{aligned} \tag{6.8}$$

We then use the chain rule to get:

$$\frac{dC}{dw_{12}^{(1)}} = (y_i - a_1^{(3)})(-\sigma'(z_1^{(3)}))[w_{12}^{(3)}\sigma'(z_2^{(2)})w_{21}^{(2)}\sigma'(z_1^{(1)})x_2 + w_{11}^{(3)}\sigma'(z_1^{(2)})w_{11}^{(2)}\sigma'(z_1^{(1)})x_2]. \tag{6.9}$$

Finding all partial derivatives with respect to our parameters, such as the weights and biases, gives us our gradient ∇C which is used in the gradient descent equation:

$$\theta_t = \theta_{t-1} + \eta(-\nabla C(\theta_{t-1})). \tag{6.10}$$

In this equation θ_t represents the parameters to the neural network at iteration t, C represents our cost function, and η is the learning rate. In the example above we used the squared-error loss for the cost function which in general form is given by:

$$C(\theta) = \frac{1}{2}(y_i - f_\theta(x_i))^2. \tag{6.11}$$

where f is the neural network, θ represents the parameters of the network (the weights and biases) and x_i, y_i represent the i^{th} training pair. It is key to remember that we are treating this loss function as a function of θ and keeping everything else fixed. The goal is to find the θ that minimizes the loss function.

6.2 Backprop

As we can see from the previous section, the algebra for this can get lengthy and messy.

$$\frac{dC}{dw_{12}^{(1)}} = a_2^0 \delta_1^{(1)} \tag{6.12}$$

The quantity $\delta_1^{(1)}$ is given by:

$$(w_{11}^{(2)}\delta_1^{(2)} + w_{21}^{(2)}\delta_2^{(2)})\sigma'(z_1^{(1)}) \tag{6.13}$$

We then recursively get the other δ's in the equation:

$$\begin{aligned} \delta_1^{(2)} &= w_{11}^{(3)}\delta_1^{(3)}\sigma'(z_1^{(2)}) \\ \delta_2^{(2)} &= w_{12}^{(3)}\delta_1^{(3)}\sigma'(z_2^{(2)}) \end{aligned} \tag{6.14}$$

For me, the best way to understand how a neural net works is to look at an actual network and learn by example. One of the classic examples is the XOR function. A discussion of this problem can be found in the Deep Learning book on pg. 167.

Figure 6.2 shows what the data looks like for this problem. We have two features, x_1 and x_2, that can take on two possible values, 0 or 1. We assign another binary variable y to each of the possible combinations of x_1, x_2 resulting in $(0,0) = 0, (0,1) = 1, (1,1) = 1, (1,0) = 0$.

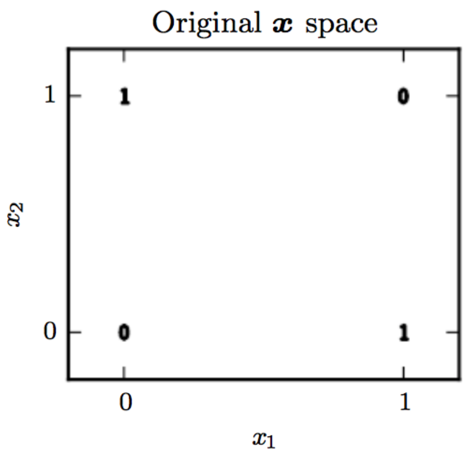
The goal here is to find a line that separates the response variable y between its two possible values, 0 or 1. As we can see from the picture this is not possible with the current setup.

7 Unsupervised

7.1 Principal Components Analysis

Principal Components Analysis (PCA) is one of the most common ways of reducing dimension of a dataset. The main algorithm is as follows:

Figure 6.2: XOR problem



1. Standardize dataset that we want to perform PCA on. This entails subtracting the mean and dividing by the standard deviation.
2. Get covariance matrix of dataset.
3. Do eigen value decomposition on covariance matrix.

If we have our data matrix X which is nxp (n number of instances, p number of features), then our correlation matrix is given by:

$$\frac{1}{n} X_s^T X_s \tag{7.1}$$

where X_s is the standardized matrix (subtract the mean and divide by standard deviation for each feature).

8 Basic Probability

8.1 Basic Probability Definitions

This majority of this section comes from the book "Probability: An Introduction" by Samuel Goldberg. I make a note otherwise.

The foundational definition of probability is the *sample space*. This is defined to be the set, S, of outcomes associated with an *experiment*, real or conceptual. Each element of this set must be an outcome of the experiment and the "performance of the experiment results in an outcome that corresponds to one and only one element of S" (Goldberg, pg. 46).

A basic example would be a coin toss. The sample space here would be $S = \{H, T\}$. With two coins one possible sample space could be $S = \{0, 1, 2\}$ counting the number of heads from flipping the two coins. This would be a poor choice of a sample space however because it cuts out information. A better choice would be $S = \{HH, HT, TH, TT\}$.

An *event* is a subset of the sample space. So if we wanted the event where the second coin is heads, the subset would be $\{HH, TH\}$.

Given a sample space $S = \{\sigma_1, ..., \sigma_n\}$, there are 2^n possible events (to understand why there are 2^n imagine each outcome σ_i as either “off" or “on". This implies that there are 2 possible scenarios for each outcome and with n different outcomes we have 2^n possible subsets). We can also consider each of these specific outcomes to be a simple event. The event $\{\sigma_1, \sigma_2\}$ is really made up of the simple events $\sigma_1 \cup \sigma_2$.

When assigning probabilities we start with assigning probabilities to each simple event. Probabilities of more complex events can then be derived from the probabilities of the simple events. Goldberg has a nice discussion on pg. 60 about the importance of clearly specifying the sample space and how the simple events are assigned probabilities. Sometimes it can be assumed how this is done, like dealing with a fair coin, but other times it needs to be clearly articulated.

The above discussion refers to single experiments like flipping a coin, etc. Many times however we want to know the probability of getting a certain outcome when we perform the same experiment multiple times. In this scenario we refer to the separate experiments as *trials* and the overall process as the experiment. If the sample space of one of these trials is S then the sample space of the experiment will be the cartesian product SXS if we are doing two trials. Since the trials are independent, the probability of a tuple will be the product of the probabilities of the individual simple events.

A *random variable* is simply a function whose “domain is a sample space and whose range is some set of real numbers" (pg. 159). The term random variable is a funny name because in reality it is neither random nor a variable. It is important to keep the true definition in mind. Note that the range of the random variable becomes a new sample space that is a set of the real numbers. Essentially we are just translating one sample space into another, where the new sample space are real numbers we can work with.

The *probability function* (pmf or pdf) of a random variable X is defined by:

$$f(x) = P(\{\sigma_k \in S | X(\sigma_k) = x\}) \tag{8.1}$$

which in words is a function that takes in the real numbers x and assigns probabilities to each real number - the probability that X has the value x. Another way to say how these probabilities are assigned is by looking at which simple events in the sample space when passed through the random variable give x and then taking the probability of all those simple events.

The *distribution function* (cdf) of a random variable X is defined by:

$$F(x) = P(\{\sigma_k \in S | X(\sigma_k) \leq x\}) \tag{8.2}$$

8.2 Random Variable Transformation

Sometimes we are interested in the probability distribution of a function of a random variable X, such as $Y = g(X)$. When doing these transformations it is important to keep track of the different domains and ranges of all functions involved, including the random variable X and the new random variable Y via the function g. The original sample space is sometimes denoted Ω and is the domain of X. The range of X is also a sample space, albeit in the real numbers universe. The random variable Y's domain is therefore the range of X. The question then becomes what is the probability distribution of this new range of Y, or this new sample space, based off the probabilities in the sample space of X (the range of X). It's possible that g maps two or more elements in X's sample space to one element in

Y's sample space and so when calculating probabilities for Y we need to take this into consideration.

A way to formally write this as found in Casella and Berger on page 48 is the following:

$$\begin{aligned} P(Y \in A) &= P(g(X) \in A) \\ &= P(x \in X : g(x) \in A) \\ &= P(X \in g^{-1}(A)) \end{aligned} \tag{8.3}$$

which is really just saying that the probability that Y is in A is the probability of all X that give $g(x) \in A$.

This implies then that if X is a discrete random variable, then Y is as well and the pmf of Y is given by:

$$f_Y(y) = P(Y = y) = \sum_{x \in g^{-1}(y)} P(X = x). \tag{8.4}$$

If X and Y are continuous variables however, then things are a little more complicated. As a first pass assume the function g is monotone. This implies that the relationship between X and Y is one-to-one and onto. We can write the cdf of Y as:

$$\begin{aligned} F_Y(y) &= P(Y \le y) \\ &= P(g(X) \le y) \\ &= P(\{x \in \chi : g(x) \le y\}) \\ &= \int_{\{x \in \chi : g(x) \le y\}} f_X(x)dx \end{aligned} \tag{8.5}$$

The last inequality occurs because we are taking the probability over the *set* of x that satisfies the condition $g(x) \le y$. We need to go one more step however so that we know which bounds to take the integral over. Since $g(x)$ is monotone we can write

$$F_Y(y) = \int_{\{x \in \chi : x \le g^{-1}(y)\}} f_X(x)dx = F_X(g^{-1}(y)) \tag{8.6}$$

However, if g is monotonically decreasing it becomes (think of the function g(x) = -x for example):

$$F_Y(y) = \int_{\{x \in \chi : x \ge g^{-1}(y)\}} f_X(x)dx = 1 - F_X(g^{-1}(y)) \tag{8.7}$$

To find $f_Y(y)$ then all we have to do is take the derivative of the cdf, split it up into two cases and use the chain rule. This gives us the famous formula:

$$f_Y(y) = f_X(g^{-1}(y)|\frac{d}{dy}g^{-1}(y)| \tag{8.8}$$

8.3 Probability Integral Transformation

Let $F_X(X)$ be any cdf of a continuous random variable X. Let $Y = F_X(X)$. Show that cdf of Y is a uniform cdf:

$$\begin{aligned} F_Y(Y) &= P(Y \le y) \\ &= P(F_X(X) \le y) \\ &= P(X \le F^{-1}(y)) \\ &= F_X(F^{-1}(y)) \\ &= y \end{aligned} \tag{8.9}$$

9 Classical Statistics

9.1 Statistical Tests

Perhaps the most important concept or theoretical underpinning of all classical statistics is the Central Limit Theorem. This theorem states that for a random sample $X_1, X_2, ..., X_n$, which is identically and independently distributed with mean μ and standard deviation σ, the sample mean is approximately distributed (converges in distribution) as such:

$$\bar{X} \sim N(\mu, \frac{\sigma^2}{n}) \tag{9.1}$$

where $\bar{X}$ is the sample average and as n goes to infinity.

Many of the classical significance tests are based off of this theorem. For example, say we want to understand if the true mean of a population is different than zero for some scenario. We would set up the test by first defining the null and alternative hypotheses:

$$\begin{aligned} H_0 &: \mu = 0 \\ H_A &: \mu \neq 0. \end{aligned} \tag{9.2}$$

Since we can't know the true population mean (we can't survey the entire population for example) we are limited to a random sample and the mean of that sample. However, we don't want to make a conclusion based solely off the sample mean alone because of the variability that is introduced by the random sample. For example, say the sample mean is 1. One conclusion (possibly false) we could make is that the population mean must be close to 1 as well and therefore we would reject the null hypothesis. It's possible however, that our sample happens to have a mean of 1 by chance whereas the population mean is really 0.

To account for this issue we use the central limit theorem to imagine a *sampling distribution* where we theoretically take many, many samples of size n from the original population and plot the means to form a distribution. This distribution, according to the CLT, will be distributed $N(\mu, \frac{\sigma^2}{n})$. Using our given sample or data, we temporarily make the assumption that the null hypothesis is true ($\mu = 0$) and figure out how many standard deviations away our sample mean $\bar{X}$ is from $\mu = 0$. This is the z-score.

Before doing this test we decide on a threshold we are comfortable with for determining if the z-score is too extreme to be due to just chance. One way to express this threshold is through critical values like 1.96 and -1.96 if doing a two-tailed test (more on those numbers in a second). If we find the z-score is greater than 1.96 or less than -1.96 in this scenario, then we can conclude that the sample mean was far enough away from 0 that it must not be due to chance alone. If we find the opposite however, then we conclude that we don't have enough evidence to reject the null hypothesis and assume our population mean is 0.

We know that z-scores in particular are distributed normally with mean zero and variance one. Using this distribution we can calculate what is known as the p-value, or the probability we get a z-score as extreme or more extreme than the one found, assuming the null hypothesis is true. We can then compare that p-value to the probability of a z-score in general being greater than or less than our critical values, such as 1.96 and -1.96. For these values the probability is 5% and can be written as $\alpha = 0.05$, which is a common threshold chosen in academia.

The interpretation of the p-value given above was defined in terms of the z-score, but in general it refers to the probability of getting any test statistic as extreme or more extreme than the one found, under the null hypothesis. P-values are the source of great confusion and it is important to remember the correct interpretation given above.

Table 9.1 below summarizes some of the basis statistical tests, followed by a more in depth discussion on each test.

One sample z-test:

The equation for the one sample z-test is given by:

$$z = \frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}} \tag{9.3}$$

Often times σ, the population standard deviation, is not known so we instead replace it with s, the sample standard deviation. As a rough rule, if $n > 30$, we say the sample standard deviation approximates the population standard deviation close enough and we feel more confident in treating the sampling distribution as normal. This is a rough rule so it seems when in doubt it is better to use the t-score.

The conditions for using the z-score are therefore:

- Random sample
- Independence condition (the individual observations in the sample are independent)

Statistical Test	When To Use	Statistic		
One sample z-test	n>30 or when we feel our sampling distribution is normal	$\frac{\bar{X}-\mu}{\frac{\sigma}{\sqrt{n}}}$		
One sample t-test	n<30			
Two sample z-test				
Two sample t-test				
Paired t-test				
ANOVA				
Chi-square				
Kolmogorov-Smirnov				

Table 9.1: Summary of basic statistical tests

- Normal condition (underlying population is normal or the sample size is large enough meaning $n > 30$). If $n < 30$ we need to look at the underlying data to see if the distribution is skewed or if there are outliers. If not then it may be safe to assume the sampling distribution will be normal. Note here as well that for proportions (when our original data is binary and we are doing a hypothesis test on the proportions) the test for normality is $np, n(1-p) > 10$.

One sample t-test: The equation for the one sample t-test is given by:

$$t = \frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}} \tag{9.4}$$

This is the same as the z-test but we use this test if we see that the population distribution is not normal and $n < 30$ (as a general rule of thumb). Again if we don't know the population standard deviation we replace σ with s. The other conditions (random sample and independence conditions described above) should be met as well.

Two sample z-test: Two sample tests are used to compare the means of two random and independent samples. The results are similar but instead of imagining a sampling distribution of $\bar{X}$, we instead imagine a sampling distribution of $\bar{X}_1 - \bar{X}_2$. This sampling distribution is found by first imagining the separate sampling distributions for $\bar{X}_1$ (mean μ_1 and standard deviation $\frac{\sigma_1}{\sqrt{n_1}}$) and $\bar{X}_2$ (mean μ_2 and standard deviation $\frac{\sigma_2}{\sqrt{n_2}}$). The mean therefore of $\bar{X}_1 - \bar{X}_2$ will be $\mu_1 - \mu_2$ and the standard deviation will be

$$\begin{aligned}
\sqrt{\text{Var}(\bar{X}_1 - \bar{X}_2)} &= \sqrt{\text{Var}(\bar{X}_1 + (-\bar{X}_2))} \\
&= \sqrt{\text{Var}(\bar{X}_1) + (-1)^2\text{Var}(\bar{X}_2) + (-1) * 2\text{Cov}(\bar{X}_1, \bar{X}_2)} \\
&= \sqrt{\text{Var}(\bar{X}_1) + \text{Var}(\bar{X}_2)} \quad \text{(since } \bar{X}_1 \text{ and } \bar{X}_2 \text{ are independent)} \\
&= \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}
\end{aligned} \tag{9.5}$$

Our null hypothesis will then typically be:

$$\begin{aligned}
H_0 &: \mu_1 - \mu_2 = 0 \\
H_A &: \mu_1 - \mu_2 \neq 0
\end{aligned} \tag{9.6}$$

If both samples are large enough ($n_1, n_2 > 30$) then we assume that the sampling distribution of $\bar{X}_1$ - $\bar{X}_2$ is normal and we can use the z-score:

$$z = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}. \tag{9.7}$$

Two sample t-test (variances are different):

If the one of the sample sizes is below 30 then it is best to use the t-test. The score below is used when the variances are different. This is known as Welch's t-test:

$$t = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}. \tag{9.8}$$

Two sample t-test (variances are same):

$$t = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}. \tag{9.9}$$

9.2 Analysis of Variance (ANOVA)

The previous test statistics are for when we want to compare a sample mean with some value or when we want to compare two sample means with each other. Analysis of Variance comes into play when we want to compare multiple samples (3 or more).

9.3 Power Analysis

Power is simply the probability that we get a test statistic more extreme than our critical value, assuming our alternative hypothesis is true - or maybe better said the probability we reject the null hypothesis given our alternative is true.

To better see what this looks like consider a simple one-sample t-test where our null hypothesis and alternative hypothesis are defined by:

$$\begin{aligned} H_0 &: \mu = 0 \\ H_A &: \mu > 0. \end{aligned} \tag{9.10}$$

Power is then defined as:

$$P(t > c|\mu = \theta) = \text{power}. \tag{9.11}$$

Involved in this equation are four parameters we are concerned with, namely the power, the effect size (represented as θ in this case since our null hypothesis is that $\mu = 0$), the critical value c (determined by our selection of α in our test), and the sample size (part of the test statistic t).

So for example, say we get a sample mean of $\bar{X}$ with a standard deviation s and sample size 10. Our t-statistic will be:

$$t = \frac{\bar{X} - 0}{\frac{s}{\sqrt{10}}} \tag{9.12}$$

Since we don't know the true standard deviation, and because our sample size is low, then we know this follows a t distribution with degrees of freedom 9. Therefore, if we choose $\alpha = 0.05$, the critical value will be around 1.833 from a standard t-table. The probability

$$P\left(\frac{\bar{X} - 0}{\frac{s}{\sqrt{10}}} > 1.833\right) \tag{9.13}$$

would give us our p-value, since this is assuming our null is true ($\mu = 0$). However, for power we assume that the alternative is true for some effect size. For example if 1 is the effect size then we have:

$$\begin{aligned} P\left(\frac{\bar{X}-0}{\frac{s}{\sqrt{10}}} > 1.833 | \mu = 1\right) &= P\left(\frac{\bar{X}-1+1}{\frac{s}{\sqrt{10}}} > 1.833 | \mu = 1\right) \\ &= P\left(\frac{\bar{X}-1}{\frac{s}{\sqrt{10}}} > 1.833 - \frac{1}{\frac{s}{\sqrt{10}}} | \mu = 1\right) \\ &= 1 - P\left(\frac{\bar{X}-1}{\frac{s}{\sqrt{10}}} < 1.833 - \frac{1}{\frac{s}{\sqrt{10}}} | \mu = 1\right) \end{aligned} \tag{9.14}$$

which is 1 minus the cdf of the standard t-distribution. What happened here is that as soon as we assumed the alternative was true ($\mu = 1$) then we needed to re-standardize the t-statistic to get back at the standard t-distribution.

This final line shows us the relationship between power, sample size, effect size, and the critical value. For example, if our effect size (1 in this case) were to be larger, then 1 minus the cdf would be larger and therefore the power would be larger indicating the fact that with a larger true effect size we are more confident in correctly rejecting the null. Another example would be if we increase the sample size (10 in this case) to a larger number. As this number increases, the same result occurs, and we are more confident in correctly rejecting the null.

This is an important issue to consider in the realm of big data because with lots of data we will almost surely reject the null because our power is so high, even though the effect size might be fairly miniscule. It's almost as if the sheer amount of data overwhelms the effect size in the formula. For example, imagine we take a small sample of 20 people and find that those who take some treatment have an extremely small positive improvement, but our statistical test comes back without enough evidence to reject the null. If we keep all else the same and only increase sample size and find the same effect size, we most likely will reject the null just because of the fact we have more samples which increases our power. The thing to keep in mind however, is that the effect size is the same in both cases and may not be meaningful.

10 Natural Language Processing

10.1 Tf-idf

Term frequency - inverse document frequency (tf-idf) is a metric that attempts to measure how important a word is to a document. It increases proportionally for a particular word as the number of times the word appears in a document increases, but decreases if that word appears in multiple documents within a corpus. This helps to account for words that appear regularly in language.

A basic implementation of tf-idf is given by the following. First find the term frequency function:

$$\mathrm{tf}(t, d) = f_{t,d} \tag{10.1}$$

where t is the term we are assessing, d is a particular document, and $f_{t,d}$ is the simple count of the number of times t appears in the document d. We can use other schemes in place of $f_{t,d}$; this is one of the more simple implementations.

The inverse document frequency part can be written as the following:

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{10.2}$$

where D is the corpus, N is the number of documents in the corpus, and $|\{d \in D : t \in d\}|$ is the number of documents the term t appears in. Again, there are other metrics than can be used in place of this function but the idea remains roughly the same.

To get the complete metric we multiply the tf part together with the idf part.

10.2 Skip-gram

The main idea in the Skip-gram approach is to use a simple neural network to find vector representations of words that encode context or a better way to say it is representations “that are useful for predicting the surrounding words in a sentence or document" (source Distributed Representations of Words and Phrases and their Compositionality).

The way this is done is to take all the words in a sentence or document and encode them as one-hot vectors. The length of these vectors will be equal to the number of words in the document. Training examples are then built by creating tuples of words that appear next to each other. So given the phrase “the quick brown fox", we generate training examples (“brown", “quick") and (“brown", “fox") or in their encoded vector form $([0,0,1,0],[0,1,0,0])$ and $([0,0,1,0],[0,0,0,1])$.

We then pass these training examples into a single layer neural network with no activation functions. Because we are passing in a one-hot vector this essentially slices a column of the weight matrix, and inputs into the single hidden layer. If, for example, we have 300 hundred hidden nodes and 10,000 words in our document, then our weight matrix will be $300x10,000$ and a slice of the weight matrix will be a vector of size 300. I'm assuming we still include bias in the hidden layer?

The second layers weight matrix will then be $10,000x300$, outputting a 10,000 length vector. The softmax function is used on the resulting vector which is defined as:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{10.3}$$

for $i = 1, ..., K$ and $\mathbf{z} = (z_1, ..., z_K)$.

In words, this function takes in a vector and outputs a vector of the same size, but where each element is exponentiated and divided by the sum of all other elements exponentiated. This new vector is therefore a discrete probability distribution. The name *softmax* comes from the concept that the function is attempting to approximate the arg max function. A better name would probably be the softargmax function, but softmax is commonly used.

Comparing the softmax output with the training label (the one-hot encoded word in the bigram associated with our input) and using gradient descent, will change the weights of the neural network in such a way that the softmax output will converge towards a vector with higher probability on the element associated with 1 in the training label's one-hot vector. The balance to this however, it that a particular training input will have multiple training labels because of the different bigrams, and so the probabilities will change for different elements, forcing the weights to account for the context of words and to encode each word. Another thought about this is that if we were to input the same word as we are trying to predict, then our output layer weight matrix will adjust the weights in the corresponding row in such a way that forces the softmax output to be as close to 1 as possible. However, because we are putting in different words, the corresponding row in the weight matrix will balance the different words to encode context.

In summary, the second layer's weight matrix will provide the 300 length vector representation of each of the 10,000 words. If we wanted to vector representation of the first word in the vocabulary, we would take the first row of the weight matrix. Its important to remember that the output of this whole process is not the model itself but the weight matrix giving us the vector representations. This means we can essentially have a pre-trained dictionary of vector representations, but only for words that were in our original model training. Implementations like FastText however, do something that encodes subwords and so we can actually get vector representations for words the model has never seen.

The paper "Distributed Representations of Words and Phrases and their Compositionality" does a couple of things that make the training easier such as removing common words like "the", putting common phrases together as one word, and doing a process called negative sampling where we only have say 6 output nodes where 5 of them are negative examples and one of them is the true example. This greatly reduces the scale of the problem.

10.3 Continuous Bag of Words (COBW)

This is almost the same approach as the Skip-gram model, but instead of passing in tuples to the model we pass in a sentence, or group of words (order doesn't matter hence the term "bag of words"), that surround the word of interest, and allow this word of interest to be the target value in the model. Each word in the group of words is one-hot encoded and then all one-hot encoded vectors are averaged together. At this point we proceed as before with the Skip-gram model, but with an input vector that is less sparse. This approach creates vectors that represent the target words, but that also encodes the context of the sentence or group of words that are passed in.

COBW is faster, but Skip-gram does better on less frequent words (see here). Also see this tutorial I used for a more intuitive understanding of word2vec.

10.4 Latent Dirichlet Allocation

The majority of this section comes from the original paper "Latent Dirichlet Allocation" by Andrew Ng, Michael Jordan, and David Blei and a lab from the applied math program at BYU found here.

Assume we have a vocabulary set of size V (meaning we consider V number of words), a corpus size of M, (meaning we have M different documents), and K different topics.

LDA is a generative model process where we assume a process that "generates" our documents and words. The BYU ACME lab does a good job of outlining this process:

1. Choose $\phi_k \sim \text{Dir}(\beta)$ for $1 \le k \le K$

For $1 \le m \le M$:

1. Choose $\theta_m \sim \text{Dir}(\alpha)$

2. Choose $z_{m,n} \sim \text{Cat}(\theta_m)$ for $1 \leq n \leq N_m$

3. Choose $w_{m,n} \sim \text{Cat}(\phi_{z_{m,n}})$ for $1 \leq n \leq N_m$

In words the first step is to draw probability vectors ϕ_k of length V for each topic k. Draws from a Dirichlet distribution return a vector of probabilities, essentially the multi-variate version of the beta distribution. This vector encodes the way each topic is represented by the vocabulary of V words.

The next step is to generate the M documents. This is done by again drawing from the Dirichlet distribution, but this time taking a vector of length K (θ_m) and thereby encoding the way the m^{th} document is represented by the K topics. In other words, this vector gives us the probabilities that the m^{th} document is assigned to topic k.

We then pass the θ_m vector into a categorical distribution and repeat N_m times, giving us N_m integers that represent each topic, where the probabilities of the categorical distribution are given by θ_m. The last step is to draw N_m words $w_{m,n}$ from another categorical distribution, which this time is parameterized by $\phi_{z_{m,n}}$, or the probability vector for the $z^{th}_{m,n}$ topic.

10.5 Problems

1.1 What is the random variable and associated pdf and cdf of the following experiment - “toss a coin until a head appears". Also prove that the cdf is a true cdf.

First of all we recognize that this is a geometric distribution which counts the number of failures until the first success. The geometric pmf is given by:

$$q^x p \tag{10.4}$$

where q is the probability of failure, p is the probability of success, and x is the number of failures. The support is on all nonnegative integers.

We note that the pmf gives the distribution of number of failures whereas we want to include the time the coin actually appears heads. Therefore we want $x+1$. This requires a transformation where $g(X) = X+1$.

Using equation 8.4 we have for $Y = g(X)$ and $g^{-1}(Y) = Y-1$:

$$f_Y(y) = \sum_{x \in g^{-1}(y)} P(X = x) = \sum_{x \in g^{-1}(y)} q^x p = q^{y-1} p \tag{10.5}$$

The last step occurs because g is a one-to-one function and the only x that maps to our y is $y-1$.

Now that we have our pmf, we want to find the cdf. This is done by doing the following

$$F_Y(y) = P(Y \leq y) = \sum_{i=1}^{y} (1-p)^{i-1} p = p\left[\frac{1-(1-p)^y}{1-(1-p)}\right] = 1-(1-p)^y \tag{10.6}$$

by the geometric series.

To prove this is a cdf we show that as y goes to $-\infty$ we get 0 (which is true because $F_Y(y)$ is defined to be 0 for $y < 0$, as it goes to ∞ we get 1 (which is true because $1-p$ is a number less than 1 and raised to infinity goes to 0), it is monotonically increasing, and lastly we show it is right continuous by $F_Y(y+\epsilon) = F_Y(y)$ as ϵ goes to 0.

1.2 Say we have a standard normal distribution that we take draws from $X_1, X_2, \ldots$ until we draw a value that is greater than some value p, in which case we stop sampling. What is the expected value of the draws?

The first step is to write down what we want. The way to think of this is that we want to know the expected value of X (where X is a normal random variable), but with the condition that $X < p$. This can be written as $E[X|X<p]$. Using the standard normal distribution we have:

$$\begin{aligned}
E[X|X<p] &= \int_{-\infty}^{\infty} x f_X(x|X<p)dx && \text{(by definition of expectation)}\\
&= \int_{-\infty}^{\infty} x \frac{f_X(x, X<p)}{P(X<p)} dx && \text{(by definition of conditional probability)}\\
&= \frac{1}{P(X<p)} \int_{-\infty}^{p} x \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} dx && \text{(limit integral bounds to match conditional prob.)}\\
&= \frac{1}{P(X<p)\sqrt{2\pi}} \int_{-\infty}^{p} x e^{\frac{-x^2}{2}} dx
\end{aligned}$$

The quantity $P(X<p)$ is given by the cdf of the standard normal distribution. The integral, using u-substitution, is found to be:

$$\begin{aligned}
\int_{-\infty}^{p} x e^{\frac{-x^2}{2}} dx &= \int_{-\infty}^{\frac{-p^2}{2}} -e^u du\\
&= -e^{\frac{-p^2}{2}} + e^{-\infty}\\
&= -e^{\frac{-p^2}{2}}
\end{aligned} \tag{10.7}$$

where $u = \frac{-x^2}{2}$.

All together we have $\frac{-e^{\frac{-p^2}{2}}}{P(X<p)\sqrt{2\pi}}$. If $p = 1$ for example then we have

$$\frac{-e^{\frac{-1}{2}}}{P(X<1)\sqrt{2\pi}} \approx \frac{-e^{\frac{-1}{2}}}{0.84 * \sqrt{2\pi}} \approx -0.288 \tag{10.8}$$

One question I had is if there is a difference between taking the expectation of a sequence from the distribution and stopping once we have a value that is greater than p vs. taking the expectation of a set sample with variables that are greater than p thrown out.

From a simulation perspective there is no difference if we think of the set sample as a bunch of sequences tied together, broken up by the values greater than p. In this light, the sequences are essentially defined by the realizations that are greater than p. So even though thinking about this process from a sequence point of view, I feel it is safe to conclude that we are just taking the normal random sample process approach, looking for the expectation of X conditioned on the event that $X < p$.

The code for this simulation is given below:

```python
from scipy.stats import norm
import numpy as np

mean_rvs = []
N = 10000
p = 1
for i in range(N):
    rvs = []
    less = True
    while less:
        rv = norm.rvs(size=1) # sample one normal, standardized random variable
        if rv < p: # if draw is less than p, add to sequence
            rvs.append(rv[0])
        else: # if draw is greater than p, stop sequence
            less=False
    if len(rvs)!=0: # take mean of sequence, as long as there is as least one random variable in sequence
        mean_rvs.append(np.mean(rvs))
```

where the mean of "mean_rvs" is around -0.288.

1.3 Find the probability that two points on a unit line have a distance less than 0.5.

$$\begin{aligned} P(|Y-X|<0.5) &= P(-0.5<Y-X<0.5) \\ &= P(Y-X<0.5)-P(Y-X<-0.5) \\ &= P(Y<X+0.5)-P(Y<X-0.5) \end{aligned} \tag{10.9}$$

We can think of Y, X having a joint uniform distribution on the unit square. This gives us every possible combination of these two variables. If we think of it in this context and relate the two variables together then we can deal with the probability terms above.

The first part of the last expression above is going to be the area (since this is the uniform distribution) below the line $Y = X + 0.5$ on the unit square. This ends up being $\frac{7}{8}$. The second part will be the area below the line $Y < X - 0.5$ which ends up being $\frac{1}{8}$. Therefore we have a difference of $\frac{6}{8}$ or 0.75.

11 Terms and Notation

11.1 Variable notation

Below explains notation used commonly when setting-up machine learning models and is taken from ESLII. 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.

12 Basic Statistical Concepts

12.1 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, 8

gamma The gamma function is defined as (n-1)!. 1, 11

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