Mixed-Design ANOVA

In contrast to the prior models described, mixed-design ANOVA is distinguished by having one of the factor variables be analyzed across subjects and the other factor be a within-subjects variable.

Multivariate ANOVA (MANOVA)

This one is similar to one-way and two-way ANOVA, except it is particularly used to analyze multivariate sample means, or when there are two or more explanatory variables in a given data set.

Having addressed the various ANOVA models, the next section talks about the method by which we evaluate the results: the F-statistic.

F-Statistic and F-Distribution

Named after Ronald Fisher, the F-statistic is the ratio of two statistical variances. F-statistics are based upon the F-distribution, a continuous probability distribution (see Figure 8-1). We denote this distribution as the null distribution of a given test statistic for the F-test. Let's assume we have variables A and B such that they both have chi-square distributions with n and d degrees of freedom respectively such that

$$X = \frac{\frac{A}{n}}{\frac{V}{d}},$$

$$f(x) = \left(\frac{\Gamma\left(\frac{n}{2} + \frac{d}{2}\right)}{\Gamma\left(\frac{n}{2}\right)\Gamma\left(\frac{d}{2}\right)} \right) \left(\frac{n}{d}\right) \left(\frac{\left[\left(\frac{n}{d}\right)x\right]^{\frac{n}{2}-1}}{\left[1+\left(\frac{n}{d}\right)x\right]^{\frac{n}{2}+\frac{d}{2}}} \right), \quad x \in (0, \infty)$$

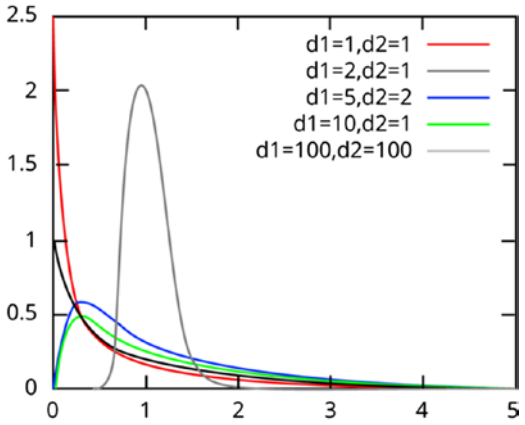

Figure 8-1. *PDF for F-distribution*

Say, for example, we're considering a one-way ANOVA and assume that the means of a set of populations are equal and normally distributed. We define the F-statistic as

$$F = \frac{\frac{SSE}{k}}{\frac{SSR}{n-k-1}} = \left(\frac{\frac{\sum \left(\hat{Y}_i - \bar{Y} \right)^2}{k}}{\left(\frac{\sum \left(Y_i - \hat{Y}_i \right)^2}{n-k-1} \right)} \right),$$

where k is degrees of freedom and n is the number of n response variables. The null hypothesis states that a model created using only an x-intercept and a model created by the user yield indistinguishable results (within a given confidence interval). The alternative hypothesis states that the model the reader creates is significantly better than a model featuring only the x intercept. Just as when testing any other measure of statistical significance, this is determined based on the threshold we want to set. (90% level of confidence, 95% level of confidence, and so on).

Let's now use a toy example to apply and explain the concepts we just have addressed. For this example, we will be using the iris data set:

```
#Loading Data
data("iris")

#Simple ANOVA
#Toy Example Using Iris Data as Y
y <- iris[, 1]
x <- seq(1, length(y), 1)
plot(y)
```

The data set will be utilized to create response and/or explanatory variables in the following experiments. In the first toy example, we take the first column of the iris data set (representing the sepal length of each observation) and make this explanatory variable. However, before we perform a one-way ANOVA, let's validate the assumptions necessary to fit data to a linear model. We'll begin by visually inspecting our data, as shown in Figure 8-2.

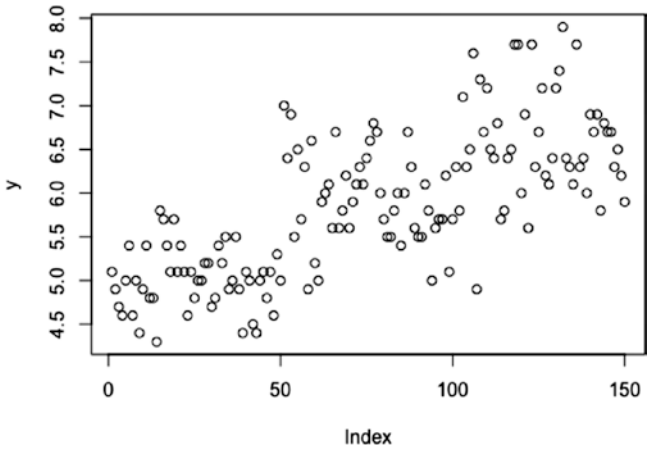

Figure 8-2. *Visualization of data*

Immediately, we notice that the data is fairly linear in its orientation, featuring a positive slope. This is a good first indicator, but we should dig deeper to ensure that the rest of our assumptions are satisfied. In this instance, we'll focus on plotting the residuals of a fitted model. By *residuals*, I mean the quantity left over from the remainder of the actual value minus the value predicted by the model. You should heavily utilize residual analysis when working with linear models, but also in general, because they provide great visual insight into how well a particular model works as well as the orientation of the data. In Figure 8-3, we see the following plots created from fitting a linear model for x and y:

```
plot(glm(y~x))
```

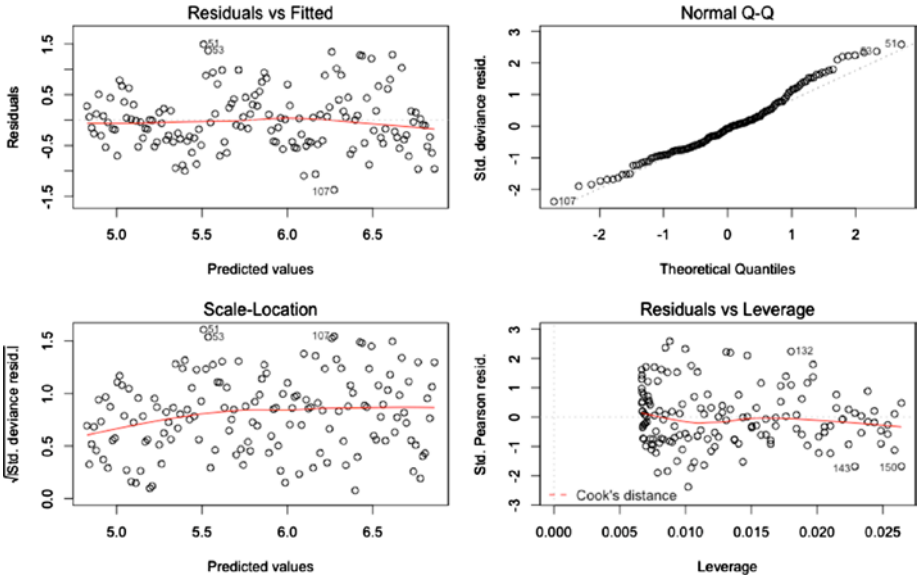

Figure 8-3. *Residual plot*

Note the graph in the top righthand corner of the four displayed in Figure 8-3. This is a *quantile plot*, which effectively displays how much the distribution of the residuals is normal. When closely inspecting the plot, we can see that a considerable amount of the data lies on the dotted 45-degree angle line, which is the marker of normality in the data. However—and as is the case usually—we notice that the tail ends tend to lift slightly above this line. It's useful to note that almost always, the data we will qualify as being normally distributed will exhibit similar patterns. In the real world, most data tends to be close to normally distributed when we have enough of it, but it's unlikely that it will be perfectly normally distributed. As such, we accept here that the data is normally distributed and move onto validating the remaining assumptions. When the data is normally distributed, it can be fit to a linear model and therefore we can reasonably estimate the values within the range of the x variable.

Because we also require that errors exhibit constant variance, let's turn our attention to the plot in the top lefthand corner. Note here a plot with an x-axis that denotes the value that the regression outputted and a y-axis detailing the value of the residual. The horizontal line through the center of the plot represents the region where the fitted value is equal to the actual value, or where the residual for an observation is zero. When referring specifically to our data, we can see that generally speaking, the shape of the residuals plotted seems to be consistent from the left to the right side of the plot. As such, we would state that the residuals in fact do exhibit constant variance. If not, we would notice that there would distinct patterns in the shape of the scatter plot that would either become more exaggerated or less exaggerated from the left to right side of the plot.

Most importantly, pay attention to the plot on the bottom righthand side of the figure. It addresses an important concept for understanding how certain data points can alter the fitted line of the regression model. *Leverage* is described as a relative measure to how large the difference in value a particular observation is from the rest of the data set. Observations that specifically have high leverage are denoted in R by placing the index adjacent to the data point. We define leverage with

$$h_i = \frac{X_i - \bar{X}}{\sum_{j=1}^{n}\left(X_j - \bar{X}\right)} + \frac{1}{n}$$

where n = number of observations, X_i = ith observation of X, $\bar{X}$ = mean of all the observations within X, and i = 1,2,…,n.

Highly related to leverage is the concept of *Cook's distance*, which directly estimates the influence a specific observation has on this regression model. We define Cook's distance as

$$D_i = \frac{e_i^2}{s^2 p}\left[\frac{h_i}{\left(1-h_i\right)^2}\right],$$

where e_i^2 = squared residuals of a given observation, s^2= mean squared error of the model, p = the number of parameters in a model, h_i= ith diagonal of the H matrix where $\mathrm{H} = X\left(X^T X\right)^{-1} X^T y$, i = 1,2,…,n, and n = number of observations.

Typically, we consider an observation as being particularly influential if its Cook's distance value is greater than 1 or if its distance value is greater than 4/n. Which threshold to use is ultimately up to you, but it's obvious that this will depend on the case, and it's worth inspecting on an experimental basis which provides a data set with more or fewer outliers, and how that would affect your end goal. If, for example, the purpose of an experiment is anomaly detection, it might be foolish to reduce the threshold such that more noise in the data set is qualified as a signal. When referring back to our specific plot, we can see that a considerable amount of data points are being flagged as being influential. We will keep this in mind as we move forward with our model choice.

When assessing all the plots in the data set, we can confidently say that although there are outliers, and our assumptions aren't met perfectly, the robustness of OLS regression allows these slight deviations to be overcome. As such, it's reasonable to choose OLS regression as a model for this task, and therefore ANOVA will yield statistically significant results. When executing the code, we observe the following:

```
simpleAOV <- aov(y ~ x)
summary(simpleAOV)

             Df Sum Sq Mean Sq F value Pr(>F)
x             1  52.48   52.48   156.3 <2e-16 ***
Residuals   148  49.69    0.34
```

Just as when we use the `summary()` function on a glm object, we are given a measure of its statistical significance. Instead of a Z-score, though, we're given an F-score from the concept addressed prior to this example and its relative p score. In this instance, we can say with greater than 99% significance that the results we reject the null hypothesis. As such, this model is a significantly better fit than an intercept-only model, and therefore we can be more confident in its results. However, let's say we'd like to compare more than one fitted model. As such, let's inspect what happens when we include more than one variable, but study the interaction between the two of them as well.

As we can see in the following code, we use the second and third columns as explanatory variables in this model. When fitting our model, we multiply both explanatory variables together. When executing the code, we observe the following results:

```
#Mixed Design Anova
x1 <- iris[,2]
x2 <- iris[,3]
mixedAOV <- aov(y ~ x1*x2)
summary(mixedAOV)

             Df Sum Sq Mean Sq F value   Pr(>F)
x1            1   1.41    1.41    12.9 0.000447 ***
x2            1  84.43   84.43   771.4  < 2e-16 ***
x1:x2         1   0.35    0.35     3.2 0.075712 .
Residuals   146  15.98    0.11
```

Our residuals as significantly smaller, and all of variables are statistically significant within at least a 90% confidence interval. Let's execute the following code and visually compare the two models in Figure 8-4:

```
par(mfrow = c(2,2))
plot(glm(y ~ x1*x2))
dev.off()
```

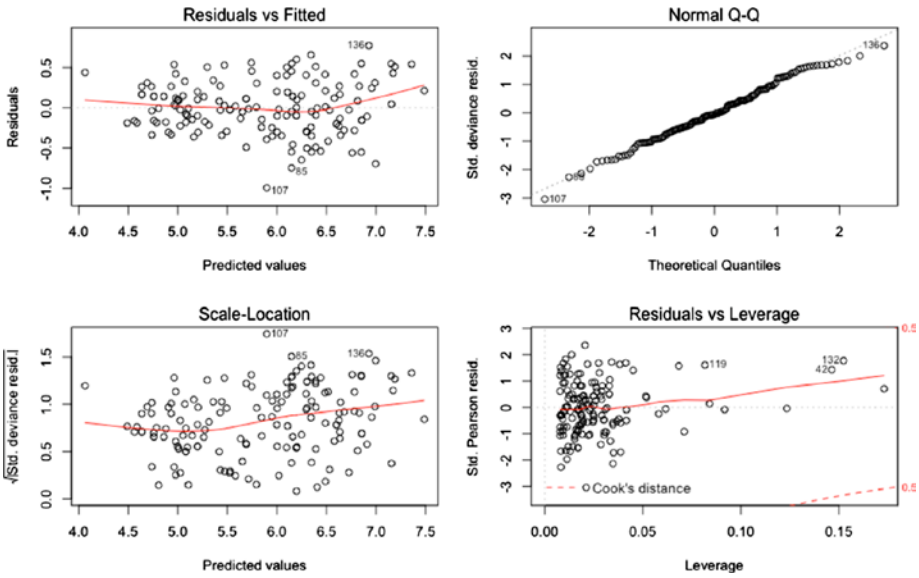

Figure 8-4. *Mixed design ANOVA plot*

We can see that all of the assumptions we need to fulfill are done so significantly better. Virtually all the residuals are normally distributed as displayed in the normal Q-Q plot, the residuals exhibit constant variance, and a considerably smaller amount have leverage. As such, when choosing between the two models we've defined, it's reasonable for us to choose the second of the two in comparison to the first.

This is a brief example of how we can use ANOVA in the course of model selection. In Chapters 10 and 11, you will learn to effectively perform these same analyses with respect to comparing deep learning and machine learning algorithms.

Let's now discuss in greater detail how to structure our experiments, with the guidance of Fisher's principles.

Fisher's Principles

Ronald Fisher, one of the most distinguished statisticians of all time, gave an explanation of principles for experimental design. The following are descriptions of his principles, as well as general advice with respect to how you might want to implement them:

1. *Statement of Experiment*: You should explicitly state the scenario that inspired the experiment, very explicitly giving an outline of the steps that will be taken place in the experiment on a very high level. It is generally accepted that the introduction should include a high-level overview of the topic, and each section should describe a different component in greater detail, logically progressing from beginning of the experiment to the end.

2. *Interpretation and its Reasoned Basis*: From the beginning, it's reasonable to give what you might expect to be the reasonable outcomes. You should state the outcomes that you feel must be considered, but realize that providing an endless list of outcomes for those to whom you report is not likely to be very helpful. Moreover, when discussing all the possible outcomes, do so in a manner that provides actionable insights for those reading your research. Research that does give actionable insights self-evidently leaves more room for misapplication.

3. *The Test of Significance*: In the context of evaluating machine learning and deep learning solutions, a simple suggestion is to bootstrap the test statistics used to evaluate a given model. It's reasonable to assume that if you draw enough test statistics over a long enough time, the data will be normally distributed. From this point, a Z-test can be performed to determine the reasonable level of statistical confidence one has in the model.

4. *The Null Hypothesis*: This hypothesis should state that the results shown have no significance, and any deviation between testing populations is due to some extraneous error such as improper sampling or deviations from proper experimental practices. This must be a component of all statistical testing.

5. *Randomization: The Physical Basis of the Validity of the Test*: When performing a test, the results reached should be performed in a manner such that this outcome was not biased. In some cases, this may require randomized observations of data to remove any inherent biases present in the modeling of the experiment that would lead to some results.

6. *Statistical Replication*: The results reached from a test should and must be replicable. Results reached that are unreasonable given the constraints inherent to the data set and environment in which we expect to observe such an occurrence are not as valuable as results that are replicable.

7. *Blocking*: The process by which different experimental groups are compartmentalized such that different variations and biases are reduced or prevented entirely from affecting the results of an experiment.

Plackett-Burman Designs

Created in the 1940s by Robin Plackett and J. P. Burman, Plackett-Burman designs are a method of finding a quantifiable dependence of explanatory variables, which we call *factors* in this case, of which each factor has L levels. The overall objective is to minimize the variance of the estimates of dependencies using a limited amount of experiments. To fulfill this goal, an experimental design is chosen such that each combination for any given pair of factors appears an equal number of times throughout each experimental "run."

The Plackett-Burman design requires a small number of experiments, specifically a multiple of 4 up to 36, and that the design have N samples that can study up to k parameters, where k = N – 1. In the case that L = 2, an orthogonal matrix in which each element is either –1 or 1 is used. This matrix is also known as a *Hadamard matrix*. This method is useful for identifying the main effects of different factors on the response variable, such that we can eliminate factors that seem to have little to no effect. Plackett and Burman themselves give specific designs for L equaling 3, 4 , 5, and 7.

Have a look at the matrix in Figure 8-5, which visually depicts a Plackett-Burman design. When performing this *design of experiments* (DOE), you must write the appropriate row as the first row of the design table. In this instance, we begin with a +, -, +, -, +, +. This is a permutation of the sequence that appears in every row, which represents a treatment combination. You can think of a treatment combination as a unique combination of a feature set. The second row is then created by shifting sequence in the prior row to the right by one column. This process is repeated for each of the remaining rows. The final row features all negative elements. It's important to recognize, though, that Plackett-Burman designs can't describe whether the effect on a given factor will result in the effect of another, and it similarly can't know the effects themselves given a small enough design. This design is considered to be a preparatory step to data analysis, and it's suggested that alternative preparatory steps be juxtaposed alongside it in addition to other steps taken subsequently afterwards.

Plackett-Burman 8-Run Matrix

Treatment Combinations	Factors						
	A	B	C	D	E	F	G
1	+	-	-	+	-	+	+
2	+	+	-	-	+	-	+
3	+	+	+	-	-	+	-
4	-	+	+	+	-	-	+
5	+	-	+	+	+	-	-
6	-	+	-	+	+	+	-
7	-	-	+	-	+	+	+
8	-	-	-	-	-	-	-

Figure 8-5. *Plackett-Burman matrix*

Space Filling

These methods don't require discrete parameters, and the sample size is chosen independently from the total number of parameters. These are recommended for instances in which the reader would like to create response surfaces, but it should be noted that it becomes difficult to determine the main effects and interactions of a given or set of parameters respectively.

Full Factorial

Full factorial is one of the most popular methods of experimental design, in which N = 2^K, where k is equal to the number of factors. As an example, let's have k factors where L = 2. In this model, we don't distinguish between nuisance and primary factors prior to the experiment taking place. Given that L = 2, we will denote them as a high, "h", or low, "l", level. High-level factors receive a value of 1, and low-level factors receive a value of –1. We determine the interaction of the variables as the product of the individual factors. From any experiment that is possible given the factorial constraint, the samples in which the factors are changed one at a time are still a part of the sample space. This allows for the effect of each factor over the response variable. Let's now also define M as the main intersection of a variable X. This is the difference between the average response variable at the high-level samples and the average response at the low-level samples. If we have three factors with two levels per factor, M for X_1 would be defined as the following:

$$M_{X_1} = \frac{y_{h,h,h} + y_{h,h,l} + y_{h,l,h} + y_{h,l,l}}{4} - \frac{y_{l,l,l} + y_{l,l,h} + y_{l,h,l} + y_{l,h,h}}{4}$$

If we wanted to see the interaction between two or more factors, the equation would be the same, except the interaction of the variables would be represented by the product of the variables, rather than the individual values a factor possesses at a given state. Both of the main effects and the intersection effect statistics give an effective method of determining the degree to which individual, or combinations of, factors affect the response variable. Full factorial designs do not complicate the data in any such manner and present a transparent method of examining variable effects. If there are more than two levels, an adjustment must be made to take the average effect of all the levels on a given response variable, where the denominator is N, such that

$$\bar{y} = \frac{\sum_{i}^{L_1} \sum_{j}^{L_2} \sum_{l}^{L_3} \sum_{m}^{L_4} y_{i,j,l,m}}{N}$$

Halton, Faure, and Sobol Sequences

Within the umbrella of space filling techniques, many of these are motivated by pseudo-random number generators. *Pseudo-random numbers* are series-generating sets that pass randomness tests. We denote a pseudo-random number generator as the following function:

$$\phi:[0,1)\rightarrow[0,1),\quad \gamma_{\mathrm{k}}=\phi(\gamma_{k-1}),\quad k=1,2,\ldots$$

We must choose a value of ϕ that gives a uniform distribution of γ_{k}. A popular method to achieve this is the Van der Corput sequence, where we have a base, b, ≥ 2 and successive integer numbers n are expressed in their b-adic expansion form such that the following is true

$$n=\sum_{j=1}^{T}a_j b^{j-1},$$

$$\varphi_b:\mathrm{N}_0\rightarrow[0,1),$$

$$\varphi_b(n)=\sum_{j=1}^{T}\frac{a_j}{b^j}$$

where a represents the coefficients of the expansion.

Halton sequences use base-two, base-three, and base-five for Van der Corput sequences in first, second, and third dimensions respectively. This pattern continues such that prime numbers are used for the base in every successive dimension. With this being said, multidimensional clustering causes high correlations between dimensions, effectively defeating the purpose of experimental design in and of itself. In an effort to combat this problem, Faure and Sobol sequences use only one base for all dimensions and a different permutation of the vector elements for each dimension.

A/B Testing

When designing applications, websites, and/or dashboard applications, it's useful to determine the effect changes in certain functionality have on the product. We can imagine, for example, that an engineer is trying to determine with some statistical certainty whether the implementation of a new feature has had an effect on acquiring new users. For such situations, it's recommended that the person use something known as A/B testing. Broadly, *A/B testing* refers to the statistical hypothesis testing methods used to compare two data sets, a control group and a test group, which are A and B respectively. We can also modify the test such that we can test A and multiple additional control tests.

The motivation for A/B testing is simple in that the development of different products, regardless of whether they have machine learning or deep learning capabilities, allows us to determine with statistical confidence whether we have made improvements from the original iteration to the next. That said, we can use these processes as a series of experimentations to iteratively move from one generation of software to the next to observe the improvements in efficiency. Commonly, the beta-binomial hierarchical model is one of the most popular methods by which we can A/B test a control group over a multiple test groups. As such, we will review this model. First, however, let's review a simple two-sample A/B test.

Simple Two-Sample A/B Test

Assume that here we're comparing one control group against one test group and that we're trying to see whether our new website generates more clicks due to feature changes. We will firmly show that although this test is stable for two examples, you should avoid using this for more than two samples. Let's say that we have two data sets representing the different attributes of the various websites and we want to test within a 95% confidence level. For this, we would use a t-test. Now let's also assume after the t-test is performed, we observe that the difference in the means is significantly different and that x2 is significantly improved from the prior model. Now let's assume that we keep on making different versions of our web page and continuously try to use this model. After nine different tests, x2 still is proving to be the most superior model. But when we run x2, we actually see no difference in improvement from clicks from x2 to the other websites. This common problem with two-sample A/B testing is due to false positives.

Next, I'll show the probability of ten individual hypothesis tests showing correct results via the binomial distribution. Let event A = x2 let's say better than nine other counterparts at 90% confidence interval, B = x2 is better than nine other counterparts at 95% confidence interval, and C = x2 better than nine other counterparts at 99% confidence interval:

$$P(\mathrm{A}) = .90^{10} = 34.87\%, \quad P(B) = .95^{10} = 59.87\%, \quad P(C) = .99^{10} = 90.44\%$$

Stated simply, under events A, B, and C we could expect that our experiments yield 6.5, 4.013, and .95 false positives. Although the 99% confidence interval performs the best in this example, we can see under the other confidence intervals why this methodology would become a problem. As such, for testing multiple groups, it is recommended that we use the beta-binomial distribution.

Beta-Binomial Hierarchical Model for A/B Testing

Bayesian statistics is a school of thought on the concept of probability. Here, it becomes the theoretical underpinning for this model and also can be used to provide modified hierarchical models with respect to the distribution. In Bayesian statistics, we often refer to the prior and posterior distributions. The *prior* distribution refers to the probability distribution with respect to some parameter (data that we have already acquired), whereas the *posterior* refers to the probability distribution of some parameter with respect to the data (data which we want to acquire). The prior distribution and the

posterior distribution form a *conjugate* distribution. For ease of analysis, we typically seek to use distributions within the same family for the respect prior and posterior distributions, and that's why in this hierarchical model we are using the beta and binomial distributions.

The *beta* distribution is a probability distribution bound within the interval [0,1] with parameters α and β that ultimately control the shape of the distribution. Typically, we use the beta distribution to statistically model random variables. As stated earlier, within the same family as the beta distribution is the *binomial* distribution. This is often used to model probability distributions that feature independent binary outcomes, such as coin flips. We define the probability density function for both the beta and binomial distributions (see Figures 8-6 and 8-7) respectively as

$$\frac{x^{\alpha-1}(1-x)^{\beta-1}}{\frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(a+\beta)}},$$

$$\binom{n}{k} p^k (1-p)^{n-k}$$

where n = the number of successes, k = total number of trials, and p = the probability of success.

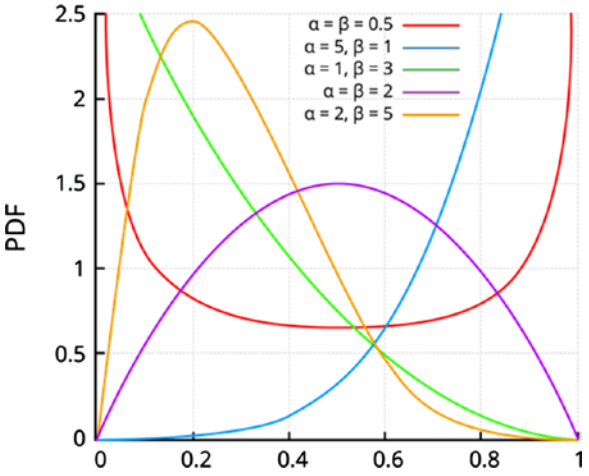

Figure 8-6. *Beta distribution*

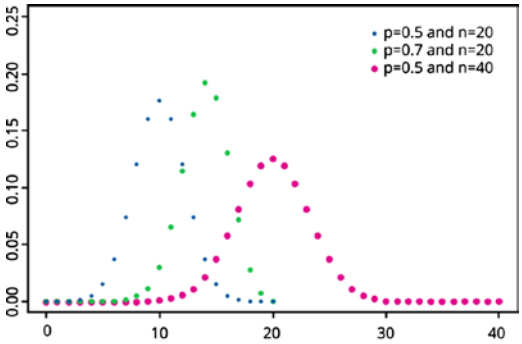

Figure 8-7. *Binomial distribution*

We then model our posterior expectations from the beta distribution and the prior distribution as the binomial, whereupon we compare the difference in means between the prior and the posterior distributions to compare website performance.

Feature/Variable Selection Techniques

Now that we've discussed several experimental design models, let's talk about steps you would want to take after you have more of an understanding of the factors in a given data set. Variable selection seems to be directly related to experimental design, but this section will discuss more specific algorithms to be used for the purpose of reducing dimensionality and less exploratory methods for analyzing variables and their interactions with the response variable. This is important for a multitude of reasons, but can often be a major element of optimizing machine learning algorithms when deploying them. Feature selection is a much less tedious process than parameter tuning, particularly in deep learning models. As such, it can be a quick way to creating models that train quicker and produce more accurate outputs. As with many techniques described earlier, caution must be taken because too much feature selection can result in creating overfitted models.

Backwards and Forward Selection

Backward selection is one of the simplest variable selection methods and is particularly common when using simple or multiple linear regression. Preliminarily, you should take the data set with all explanatory variables and regress them against the response variable. After this step, choose a statistical significance level appropriate for the given situation (85%, 90%, 95%, and so on). One variable at a time, we remove the variables with the lowest statistical significance from the data set (such as the statistical significance yielded from the summary() function when using a glm() model). We the regress the new subset of the original data set and continue until all the variables in the data set are statistically significant. In forward selection, the process is the same as the prior method, except the distinction is that you start with a model with no variables, add variables, and check their

statistical significance. If they're at or above the threshold, they should be added. If not, they should be removed. Considerations to keep in mind when using these methods are to reduce statistical noise considerably but to avoid a model that's overfitted to the test data, particularly if out-of-sample prediction is the end goal to the model being built. You should also be careful to not remove too many variables as to reduce performance.

For deep learning, some models have feature selection embedded into them. Specifically, certain layers within CNNs arguably exist for the purpose of eliminating noise such that the data left is rich with information. Specifically, pooling layers can be thought of as doing this. By reducing the input size, we ease the computational load from input to output while also assisting the algorithm in more accurately tuning the weights between these layers and ultimately classifying an image.

Beyond using P-values, you can choose other statistical criteria to determine which variables to retain/remove. Among those most common are Akaike information criterion (AIC) and Bayes information criterion (BIC):

$$AIC = 2k - 2\ln(\hat{L}),$$

$$BIC = \ln(n)k - 2\ln\left(\hat{L}\right),$$

$$\hat{L} = p\left(x \mid \hat{\theta}, M\right)$$

where L is the max likelihood function for the model, $\hat{\theta}$ are the parameters, and k is the number of parameters.

AIC and BIC are very closely related. AIC is based within the field of information theory, and the goal is to choose a model with the minimum possible AIC value. By definition of the function, the greater the magnitude of the log-likelihood, the smaller the AIC value. Henceforth, models that are more closely fit to the data will ultimately have lower AIC values. BIC is ultimately motivated by Bayesian statistics and is similar to AIC. BIC scores specifically are used to evaluate the performance of a model on a training set, where we choose the model that yields the smallest BIC. In particular, BIC penalizes models that have more parameters rather than less. Because of this, BIC inherently prefers models that do not overfit to the data set, hence making a criterion by which you're encouraged to choose a model that generalizes to the data you're analyzing. Note that the BIC can't handle complex collections of models, and it should only be assumed to be valid in instances of when n is substantially greater than k. With respect to considerations for AIC, the AIC values computed must be across the same data. Specifically, it's not an objective measure such as the coefficient of determination.

Principal Component Analysis (PCA)

Principal component analysis (PCA) is one of the most commonly used variable selection techniques that can exclusively be used for numerical data. Mentioned earlier in several examples, PCA is a statistical method used to reduce dimensionality of data sets. Simply stated, we transform the data into new variables called *principal components* and

eliminate the principal components that explain negligible amounts of the variance exhibited within the data set. The benefit of this technique is that we preserve the variance of the data set while being able to perform visual and exploratory analysis much easier than prior to the transformation.

Our goal is to find the linear function of random variables from the x vector with the vector of constants from the alpha vector with the maximum variance. This linear function produces our principal components. Be that as it may, each principal component must be in order of decreasing variance, and each principal component must be uncorrelated with each other. Our objective is the following:

$$\text{Maximize } Var\left(\alpha_k' x\right) = \alpha_k^i \sum alpha_k$$

We seek to use constrained optimization, because without a constraint the value of a_k could be infinitely large. As such, we'll choose the following normalization constraint, where $\alpha_k' a_k = 1$.

The Lagrange multiplier method is a tool for constrained optimization of differentiable functions. In particular, it's helpful for finding local maxima and minima of a respective function subject to a given constraint. Within the context of the experiment, the Lagrange multipliers are applied as follows

$$\alpha_k' \sum a_k - \lambda\left(\alpha_k' a_k - 1\right),$$

$$\frac{d\left(\alpha_k' \sum a_k - \lambda\left(\alpha_k' a_k - 1\right)\right)}{d\alpha_k} = 0,$$

$$\sum \alpha_k - \lambda\alpha_k = 0,$$

$$\sum \alpha_k = \lambda_k \alpha_k$$

with the final step of the equation yielding the eigenvector α_k and its corresponding eigenvalue λ_k. Our objective is to maximize λ_k, and with the eigenvectors defined in decreasing order. If λ_1 is the largest eigenvector, then the first principal component is defined as $\sum \alpha_1 = \lambda\alpha_1$. In general, we define a given eigenvector as the k-th principal component of x and that the variance of a given eigenvector is denoted by its corresponding eigenvalue. I'll now demonstrate this process when k = 1 and when k > 2. The second principal component maximizes the variance subject to being uncorrelated with the first principal component with the non-correlation constraint being as follows:

$$cov\left(\alpha_1' x \alpha_2' x\right) = \alpha_1' \sum \alpha_2 = \alpha_2' \sum \alpha_1 = \alpha_2' \lambda_1 \alpha_1' = \lambda_1 \alpha_2' \alpha_2 = 0,$$

$$\alpha_2' \sum \alpha_2 - \lambda_2\left(\alpha_2' \alpha_2 - 1\right) - \phi\alpha_2' \alpha_1$$

$$\frac{d\left(\alpha_2' \sum \alpha_2 - \lambda_2\left(\alpha_2'\alpha_2 - 1\right) - \phi\alpha_2'\alpha_1\right)}{d\alpha_2} = \sum \alpha_2 - \lambda_2\alpha_2 - \phi\alpha_1 = 0,$$

$$\alpha_1' \sum \alpha_2 - \alpha_1'\lambda_2\alpha_2 - \alpha_1'\phi\alpha_1 = 0,$$

$$0 - 0 - \phi 1 = 0,$$

$$\phi = 0,$$

$$\sum \alpha_2 - \lambda_2\alpha_2 = 0$$

This process can be repeated up to k = p, yielding principal components for each of the p random variables. Limitations associated with PCA are numerous, though, and must be considered for the problem type. Foremost, PCA assumes that there are linear correlations across features. Obviously, this is not necessarily always the case in a practical context and therefore renders the results yielded by PCA questionable. Secondly, PCA only can be used on numerical data sets and the downfalls of numerically encoding categorical data (discussed later in this chapter) can add implicit biases that render the results of this technique useless. Moreover, PCA explicitly assumes variance is the most important statistic with regard to analyzing a data set. Although variance is often an important statistic, in some problem cases it might not necessarily be.

An example of how PCA can be applied to deep learning is through the process of PCA whitening. When we refer to *whitening*, we mean the process of making the input data less homogenous, in an effort to make the data less homogenous from one observation to another. In the instance of a CNN, this can be of great use for image classification. Specifically, in image data many pixels adjacent to one another often have similar, if not the same, values within a large region.

An example of this would be to look at the MNIST data set and see which patches of the image are black versus which are white. PCA whitening instead yields an eigendecomposition of the matrix such that this homogeneity is removed. As such, the features of each individual are significantly less similar than in their original form, but the variance within the data is preserved, as is a benefit when performing an eigendecomposition on a matrix.

Factor Analysis

Factors are unobservable variables that are highly correlated with one another and that influence a given explanatory variable. Unlike the ultimate purpose of PCA, dimensionality reduction, *factor analysis* seeks to locate independent variables. Moreover, we would like to determine what influence the factors have on the surface attributes. It's built from the assumption that observed variables can be reduced to a subset which exhibit similar variance. In factor analysis, we require that the data must be normally distributed and that there are virtually no outliers within the data set. We also

should seek to analyze data that is numerous in it's observations, and the correlations, while not nearly linear as to avoid multicollinearity, must be moderate to high across the data set. The typical factor analysis model is given by

$$X_j = a_{j1}F_1 + a_{j2}F_2 + \ldots + a_{jm}F_m + e_j, \; j = 1, \ldots, p$$

Where e_j = the unique and specific factor to a given explanatory variable, j = factor loadings, X_j = an explanatory variable, and m = the underlying factors

Factor loadings can be thought of as weights, where they denote the degree to which they influence a given factor with respect to an individual variable. Surface attributes are denoted as the individual explanatory variables. Typically, a factor analysis model will yield factors such that there are no correlations between the individual variables, so we have independent variables, similar to principal components. It should be noted that factors are not created but are revealed based on correlations between surface attributes. Factors, which are unseen, can be intangible yet conceivable. For example, we could image factors within a given experiment being an individual's reading or writing ability when compared to one individual. These attributes aren't objective with respect to how we measure them, but when assessing a standardized test with a reading and writing section, for example, obviously affect a given person's score.

Limitations of Factor Analysis

Factor analysis can find a method of obtaining patterns in data generated even from random numbers. As such, one should keep in mind that if structure can be found in random data, than the patterns they appear to observe in their structured data could also be misconceived. Moreover, the structure found in the data ultimately is a derivative of the variables/data set inputted into the factor analysis. Simply stated, there are not objective patterns in data sets that make themselves apparent, and ultimately restructuring of data sets/variables can cause significant divergence in the results yielded by a factor analysis. As such, how one interprets the results of a factor analysis ultimately is far more subjective than it may seem. That said, it is recommended that factor analysis be used alongside statistical methods and/or the data be structured such that it conforms to assumptions known to be true within the domain of the problem being handled.

Handling Categorical Data

Among all of the difficulties that you might come across, one of the greatest challenges comes with handling and analyzing categorical data, or data that is numeric. Typically, we often encounter categorical data as a factor variable with different levels. This section talks about some common problems that will be encountered along with possible solutions, with considerations to keep in mind.

Encoding Factor Levels

For example, let's say we have a data set where we are analyzing one variable, which is all of the streets in a given neighborhood. This is particularly interesting example because the streets could all be names, (such as "Maple Street," "Spruce Street," "Redwood Street," and so on), or they could all be numbers (1st Street, 2nd Street, 3rd Street, and so forth). If the streets are names, we can take the approach, to encode the streets by number. This is an easy way to give each variable a unique identifier, but it has limitations. Machine learning algorithms will interpret the levels as an indication of value rather than a unique identifier, which in essence gives no descriptive data about the "quality" of the observation. To be specific, if we label "Maple Street" as 1 and "Spruce Street" as 2, many algorithms might interpret Spruce Street to be of higher importance than Maple Street, when there is no evidence to determine this. When considering the case of the numbers, this same problem is present, but it's just implicit and not induced by label encoding. Another limitation of this technique is that if the encoded variable is highly correlated with other variables, multicollinearity might be introduced to the data set where it otherwise would not have existed.

Categorical Label Problems: Too Numerous Levels

In keeping with the example of using street names, we can imagine many cities where this would cause us to have a factor with hundreds or even thousands of individual streets. Although a variable with variation yields better results than a variable with absolutely none, this can also cause difficulties when performing model evaluation. As such, in these instances it can be a good idea to encode the variables and use a classification/regression tree or random forest model. Also, a suggested method is to encode the variables and use K-means clustering to get the cluster number, whereupon we replace the levels with this variable. Although this still in many ways has the bias of the encoded variable we discussed before baked into the clustering observation, it's nonetheless a method of reducing the levels effectively and should be explored when necessary.

Canonical Correlation Analysis (CCA)

Very closely related to PCA is canonical correlation analysis (CCA), a method of finding linear combinations of two variables such that they have the maximum possible covariance with each other. Typically, this is a data preprocessing technique and is appropriate in the same instances where multivariate linear regression would be used, but specifically when there are two sets of multivariable data sets that we want to examine the relationship between:

Given two vectors $X, Y \in \mathbb{R}^{m x n}$ and directions $\alpha \in \mathbb{R}^m$ and $\beta \in \mathbb{R}^n$:

$$\alpha, \beta = \text{argmax } cov(X\alpha, Y\beta)$$
$$\|X\alpha\|_2 = \|Y\beta\|_2 = 1$$

Wrappers, Filters, and Embedded (WFE) Algorithms

When assessing some of the more advanced variable selection techniques, we approach WFE algorithms. *Wrapper* algorithms are distinguished by running each feature subset possible over the data and evaluating the model performance, leading to the selection of a subset that performs the best with a given model. *Embedded* algorithms are explicitly written into the process of a model (L1 regularization with LASSO). *Filter* methods attempt to assess the merits of the feature by looking at the data itself rather than evaluating its performance on the methods alone.

Relief Algorithm

Designed by Aha, Kibler, and Albert in 1991, the *relief* algorithm is a feature-based weight algorithm inspired by instance-based learning. Each feature is assigned a weight denoting its relevance of the feature to the target. This algorithm is randomized and the updates of relevance values depend on the difference between the selected instance and the two nearest instances.

Algorithm

1) Given $\{(x_n, y_n)\}_{n=1}^{N}$, set $w^0 = \frac{1}{I}$, $T =$ number of iterations, $\sigma =$ kernel width, $\theta =$ stopping criterion.

2) For t = 1 : T

   a. Calculate pairwise distances w.r.t. w^{t-1}.

   b. Calculate P_m, P_h, and P_o.

   c. Update weights.

   d. If $\left\|w^t - w^{t-1}\right\| < \theta$, break.

Other Local Search Methods

Many of the algorithms addressed in the latter parts of this text will draw inspiration from, if not be directly related to, this subfield of optimization, typically used for computationally intensive optimization problems. We consider all possible solutions as being in a set we denote as the *feature space* or *search space*. The target is the global optimum that satisfies the optimization problem we seek to solve. Local search algorithms are initiated with a random element from the feature space and over each iteration chooses a new solution based on information yielded from the current neighborhood. After this stage, the algorithm will move to a given neighborhood in the nearest vicinity, but depending on the problem the search algorithm may choose more than one neighborhood.

Hill Climbing Search Methods

Prior to the development of machine learning that occurred in the 1980s and 1990s, *hill climbing* tended to be one of the more popular search methods. Hill climbing forms the motivation for many newer search methods described in this chapter and is still a useful technique with respect to parameter tuning. As with other search methods, hill climbing seeks to optimize an objective function within the locality of the current point. Hill climbing works best for functions that have one maximum or one minimum, so as to allow the algorithm to find the solution of the problem with relative ease. However, it faces many problems for functions with an abundance of local minima. To combat this, many different heuristics and methods, like random restarts to avoid local minima and stochastic neighborhood selection for the search trajectory, have been added to the basic hill climbing algorithm.

Genetic Algorithms (GAs)

Genetic algorithms are considered a direct outgrowth of the field of artificial intelligence, as they directly mock the process of evolution. In this algorithm, several subsets of the total feature space "evolve" so that the next subset is statistically better than the last iteration. The evolution process stops when a better subset can't be created, and the best of the subsets is chosen as the answer. The advantage of this algorithm over others is that genetic algorithms can accumulate information about a given feature space over many iterations, the process is inherently parallel so there is less probability of being stuck in local minima, and the algorithm in and of itself is relatively easy to understand. Among GAs' limitations are the fact that if there is an abundance of local optima, the GA doesn't always converge upon the global optimum. Also, this algorithm is likely not an optimal choice for deployment, because it has difficulty scaling, since the feature space size increases exponentially with the number of possible subsets.

Algorithm

- Choose an initial random population of solutions to choose from.
- Evaluate the solution based on some statistical criterion, such as MSE.
- Select the best individuals to be used.
- Generate new individuals by "mutating" the prior selected solutions.
- Evaluate the fitness of the new solutions.
- Stop when some criterion has been reached, such a loss tolerance.

Simulated Annealing (SA)

Among the heuristic techniques we will cover, one of the few probabilistic models assessed is SA. Inspired in name from *annealing* in metallurgy, SA imitates the effect of slowly cooling as slowly decreasing the probability of accepting worse solutions. We consider each solution as a state and that the neighborhood in which the algorithm can search progressively gets smaller. The algorithm converges upon a solution either after the feature space has been entirely searched, or another stopping criterion has been reached. 1

Algorithm

- T = Temperature = hot, Frozen = Stopping Criterion.
- While (Temperature != Frozen), move to a random point in the feature space and compute Δ Engery.
- If ΔEnergy < 0 or loss tolerance, accept new state with probability $e^{-\frac{\Delta E}{T}}$ while system in thermal equilibrium at current T.
- If (E decreasing over last few iterations), $T = T(itertion + 1)$, Else T = Frozen.

The greatest difficulty with SA is the amount of parameter tuning required, which becomes time consuming as the amount of feature (and corresponding feature space) increase in size. Furthermore, there isn't a general baseline or rule of thumb for any of these parameters, further increasing the difficulty of this technique with heavily changing data sets. It should likely be considered more of a research technique than one you would deploy in an algorithm.

Ant Colony Optimization (ACO)

Ant colony algorithms (ACOs) are a set of optimization algorithms first introduced in the 1990s. Most useful for combinatorics problems, ACOs been used for tasks such as vehicle routing, computer vision, feature subset selection, quantitative finance, and other fields. The intuition is based on the activities of swarms of ants, and the ultimate goal is typically finding the best options given set of randomized options from a feature space. We can imagine an ant colony in this context to be a graph with nodes connected by edges, where each node represents one of the k features in the data set. The ant travels along the edges, "dropping pheromones" to attract more ants along subsequent iterations. The pheromones by design decay over time, but ants who travel along the shortest possible edges from point x to point y deposit more pheromones along a given path. Because ants are attracted to paths with more pheromones, this acts as the method by which an optimal solution is found. Each "ant" moves from one given state with a probability given by

$$p_{x,y}^{k} = \frac{\tau_{xy}^{\alpha}\left(\eta_{xy}^{\beta}\right)}{\sum_{j \in J_i^k} \tau_{xj}^{\alpha}\left(\eta_{xj}^{\beta}\right)}$$

where τ_{xy}^{α} = pheremone deposited on a given path, η_{xy}^{β} = the proportion of the distance from x : y to the sum of all paths'distances, β = tuning parameter, J_i^K = neighbor nodes that have not been visited

With pheromones updated as

$$\tau_{xy}(t+1) \rightarrow (1-\rho)\tau_{xy} + \sum_k \Delta\tau_{xy}^k$$

where τ_{xy} = pheremone deposited, and ρ = pheremone evaporation rate.

We denote $\Delta\tau_{xy}$ as the amount of pheromone dropped on a given path by an individual ant given by

$$\Delta\tau_{xy} = \begin{cases} Q/L_k, & \textit{if kth ant travels along xy} \\ 0, & \textit{elsewhere} \end{cases}$$

where Q = some constant, and L_k = a loss function defined by the user.

Although ACO problems are successful for instances in which there aren't very large numbers of features, and it typically performs better than simulated annealing and genetic algorithms, the problems become exponentially more difficult to solve with the addition of more nodes. In addition to this, although convergence is guaranteed, it is uncertain as to when convergence actually will occur.

Algorithm

- Initialize by creating full solution space.
- While stopping criterion not reached, position each ant at a given starting node.
- For each ant, choose next node via state transition rule.
- Apply pheromone update until every ant has reached a given solution.
- Evaluate each solution based on the selection criterion.
- Update best solution and apply pheromone update on this path.
- Repeat until convergence upon global optimum.

Variable Neighborhood Search (VNS)

VNS is a family of feature subset selection algorithms that are meant to deal with combinatorics challenges and henceforth provide guaranteed convergence. Developed in the late 1990s, VNS was inspired by the desire to find solutions for discrete and continuous optimization problems (linear and nonlinear programming problems are an example). The assumptions within VNS are that a local minimum with respect to a given neighborhood is in theory perhaps not the local minimum in another neighborhood, that

local minima are relatively close to each other between one or more neighborhoods, and that a global minimum are local minima for all neighborhoods within the solution space. Among the algorithms available for VNS with respect to local search methods, there are related extensions that are more specified for given tasks. For feature-based selection, we will look at the filter-based algorithm for VNS.

Algorithm

- Find an initial solution S.
- Select the set of neighborhoods N_k for $k = 1, \ldots, j$ where j = # of neighborhoods and a stopping criterion.
- Set k = 1 and generate a random point S' from the k^{th} neighborhood of $S\left(S' \in N_k(S)\right)$.
- Apply a search method such that the stopping criterion, if based on an objective function, is closer to being reached.
- If this solution is better than the prior solution, update the solution to the current one. Else, set k = k + 1 and retain the current solution.
- Continue until convergence upon global optimum or stopping criterion is reached.

Typically, we choose an information quotient or linear correlation as an evaluation function within these algorithms, but this is ultimately a parameter that can be altered. If you feel more advanced, feel free to implement your own deep learning and/or machine learning algorithms where instead of traditional gradient descent, you use one of the aforementioned search methods for parameter optimization. Although this can be difficult, it will provide you with an excellent exercise to get familiar with specific algorithms, while also helping you understand how performance is affected by specific operations within a given algorithm. That brings us to a similar topic with respect to refining existing machine learning algorithms: reactive search optimization.

Reactive Search Optimization (RSO)

RSO is a relatively new innovation in the field of optimization. It produces interesting implications that are worth mentioning for more advanced readers. The purpose of RSO lends itself to being of particular use to those who intend on creating machine learning platforms and tools that are intended for users who aren't as technically adept as the typical machine learning engineer. *Intelligent optimization* refers to a more specific area of research within RSO, but is nonetheless relevant. In this paradigm, we evaluate the effectiveness of different learning schemes. There are broadly three, which we will refer to as online, offline, and a combination of the two with varying proportions. This is the idea of implementing algorithms in different environments such that they have different search histories, which ultimately affect the action of the epoch that is currently in session.

Reactive Prohibitions

Prohibition-based techniques and *intelligent schemes*, in contrast to basic heuristics such as local search, are what provide the intellectual motivation for *tabu search*. Tabu search methods mainly gained their initial traction in the 1980s, and it has proved a large area of research given the fertile ground it occupies. Tabu search (TS) is particularly noteworthy when comparing it against local search methods because of the use of prior information gleaned from the data set, and how that influences the new iterations' outcomes. Assume that we have a feasible search space that is composed of binary strings with a length $L: \mathcal{X} = \{0,1\}^L$, X is the current configuration, and N(X) is the previous neighborhood. The following equation is related to tabu search that is prohibition-based

$$X^{t+1} = \text{BestNeighbor}\left(N_A\left(X^t\right)\right),$$

$$N_A\left(X^{t+1}\right) = \text{ALLOW}\left(N\left(X^{t+1}\right), X^0, \ldots, X^{t+1}\right)$$

where the ALLOW function selects a subset of $\left(N\left(X^{t+1}\right)\right)$ such that it is dependent on the entire search trajectory $X^0, \ldots, X^{t+1}$.

Tabu search algorithms are classified in many ways, but the initial distinguishing factor I'll elaborate on is deterministic versus stochastic systems within TS algorithms. The most basic form of tabu search is denoted as *strict tabu search*. In this algorithm, we observe N(X) to have the following value:

$$N_A\left(X^{t+1}\right) = \left\{X \in N\left(X^{t+1}\right) \; s.t. \; X \notin \left\{X^0, \ldots, X^{t+1}\right\}\right\}$$

When introducing a prohibition parameter, T, that determines how long a move will remain prohibited after the execution of its inverse, we can obtain two algorithms that are different from strict tabu search. A neighbor is allowed if and only if it is obtained from the current point by applying a direction to the search such that its inverse has not been used during the last T iterations, such that

$$N_A\left(X^{t+1}\right) = \left\{X = \mu \circ X^t \; s.t. \text{LastUsed}\left(\mu^{-1}\right) < \left(t - T\right)\right\},$$

where LastUsed() is the last usage time of move μ. If T changes with the iteration counter, the general dynamical system that generates the search trajectory comprises an additional evolution equation for T such that

$$T^t = \text{React}(T^{t-1}, X^0, \ldots, X^t,$$