CAMB 698 Final Paper

Vincent Wu

2018-12-10

Contents

Chapter 1

Preword

This book is written and compiled using the `bookdown` package for the R statistical language. Sections of this book will feature code and graphs produced using R as well. As will be discussed in the next chapter, the purpose of this paper is to introduce and broadly cover common machine learning techniques that are used in the biological sciences. Any included R code is for generating graphs and results to provide additional content in conveying the different techniques. As such, I opted to include the R code as a starting point for any readers who want to explore the techniques.

The following code in this chapter are needed to load in additional software packages as well as to load in the data that will be used.

```r
library(tidyverse)
library(usedist)
library(qiimer)
library(reshape2)
library(ggplot2)

library(ape)
library(tree)
library(Rtsne)
library(class)
library(randomForest)
library(e1071)
library(cluster)
```

```r
load("data/poop_across_penn1.Rdata")
```

```r
# Create vendor/mice dataframe
s_vendor_all <- s %>%
  filter(grepl("ARC Vendor Experiment", Experiment)) %>%
  rename(Vendor = Mouse_Source_Vendor) %>%
  mutate(SubjectID = factor(paste("Mouse", Mouse_Number))) %>%
  mutate(SampleType = trimws(as.character(SampleType))) %>%
  arrange(SampleType, Vendor, SubjectID)

# Identify and remove suspicious samples
suspect_SampleIDs <- c("Tac.33.CE.Day1", "Env.13.Stool.Day0")

# Set final dataframe
s_vendor <- s_vendor_all %>%
  droplevels() %>%
  filter(!(SampleID %in% suspect_SampleIDs))
```

Chapter 2

Background

Data processing and analysis are fundamental aspects of conducting scientific research, from the initial raw data to visualizing the results. With the recent advances of high throughput technologies (i.e. the many "-omics", multiparametric flow cytometry, etc.), the accompanying data is often high-dimensional and difficult for manual efforts to analyze. The development of machine learning methods aims to help with this problem, thus helping to make sense of the data to draw meaningful and accurate conclusions.

The purpose of this paper is to introduce and explore some of the different machine learning methods that are commonly used in the biological sciences. To help demonstrate the methods and to maintain context across all of the methods, a single dataset (will be referred to as the PAP dataset) was provided by the PennCHOP Microbiome Program (courtesy of Dr. Kyle Bittinger). Mice were purchased from vendors with the purpose of assessing whether the mice have different phenotypes from different vendors. The fecal microbiota was sequenced from the mice, resulting in a distance matrix (how distant each mouse's microbiome was from the other mice sampled) as well as the relative abundance of bacterial species for each mouse. Additionally, metabolites as well as different immune phenotypes were assessed for each mouse. This high-dimensional dataset is representative of datasets that are seen with microbiome research.

Machine learning methods can fall under two main branches – supervised versus unsupervised learning. The former branch consists of methods where the concept of the output is already known. Techniques like regression and classification methods strive to produce an output (which vendor the mice are from) from an input (some or all of the variables such as the distances, the relative abundances, etc.). The latter branch contains methods where the output is not exactly known. Different clustering models attempt to use the input to find if the data can be clustered into unique groups.

While this paper is primarily focused on machine learning techniques, a concept from statistical analysis is critical for understanding the benefits and drawbacks of certain techniques. This concept, known as the *bias-variance tradeoff*, captures the relationship between approximation bias, estimation variance, and the ability to accurately predict a value (Shalizi, 2018). Approximation bias can be loosely defined as the inherent bias in using a particular prediction function to estimate a value. The estimation variance can be defined as how spread out are the estimates. The ability to accurately predict a value is negatively affected by both the approximation bias and the estimation variance (Shalizi, 2018).

As such, the bias-variance tradeoff explores this relationship between approximation bias and estimation variance. Since reducing the approximation bias will increase the estimation variance, it may appear to be a futile attempt to maximize the ability to predict a value. The relationship, as Shalizi writes, is not "one-for-one" (Shalizi, 2018) – whereupon it is possible to reduce the error of predictions by introducing a little bias to reduce the variance. This concept will be a common motif in many of the methods in this paper.

Chapter 3

Dimensionality reduction

3.1 Introduction to dimensionality reduction

Dimensionality reduction is a necessary tool for working with high-dimensional data, which can be seen from this dataset. Each of the diversity distances against each of the mice is a single variable, as is each of the relative abundances for different species. A variable is a dimension in this scenario – creating a dataset with high dimensionality. While rich and informative, this dataset would be difficult to visualize beyond two dimensions. Without going too far into the mathematics behind these methods, dimensionality reduction creates a means to observe the data from a different perspective and assists in reducing the computational burden for the machine learning techniques that will be discussed.

3.2 PCA and PCoA

Principal components analysis (PCA) and principal coordinates analysis (PCoA) are common techniques used both in and outside of the biological sciences. PCA makes new variables that are linear combinations of the original variables (Shalizi, 2018). PCoA is similar in concept but takes in a distance matrix (such as the one used for our dataset) to transform into new coordinates where the axes of this coordinate system are not correlated with each other. The power of PCA and PCoA is that all new variables have no correlation with each other and can explain all the covariance from the original data. The data points in PCA or PCoA space can be easily visualized as seen in the below graph. It is common to display the variance explained by each of the principal component axes as a means to show how well the principal components can explain the variance in the original data.

3.2.1 PCoA example

```r
## this code was modified from Kyle Bittinger's code

# get unweighted unifrac distances
uu <- dist_subset(uu, s_vendor$SampleID)

# run pcoa
pc <- pcoa(uu)

# create dataframe for ggplot2
pc_df_uu <- cbind(s_vendor, pc$vectors[s_vendor$SampleID, 1:3])

# calculate variance coverage by axis
pc_pct <- round(pc$values$Relative_eig * 100)

# finish setting up dataframe
pc_df_uu <- pc_df_uu %>%
  mutate(Label = ifelse(SampleID %in% suspect_SampleIDs, SampleID, ""))

# make fig
ggplot(pc_df_uu, aes(x = Axis.1, y = Axis.2)) +
  geom_point(aes(color = Vendor, shape = SampleType)) +
  geom_text(aes(label = Label)) +
  labs(
    title = "PCoA plot of Unweighted Unifrac Distances across Mice",
    x = paste0("PCoA Axis 1 (", pc_pct[1], "%)"),
    y = paste0("PCoA Axis 2 (", pc_pct[2], "%)")) +
  theme_classic()
```

In Figure 3.1, the PCoA plot was created from the microbiota distance matrix for each mouse (two points per mouse, since the microbiota was sampled in the cecum as well as in the stool). As shown on the axes, the first PCoA Axis explains 31% of the variance in the original distance matrix. Creating a PCoA plot allows for a quick and relatively simple way to visualize high-dimensional data to inform future decisions on machine learning. From this plot, it is interesting to note the separation of the different mice microbiota samples based on where the mice were bought.

3.3 t-SNE

Another commonly used dimensionality reduction technique is t-Distributed Stochastic Neighbor Embedding (t-SNE) (Maaten and Hinton, 2008), which is a type of manifold learning. t-SNE starts by calculating pairwise distances in the high-dimensional space and

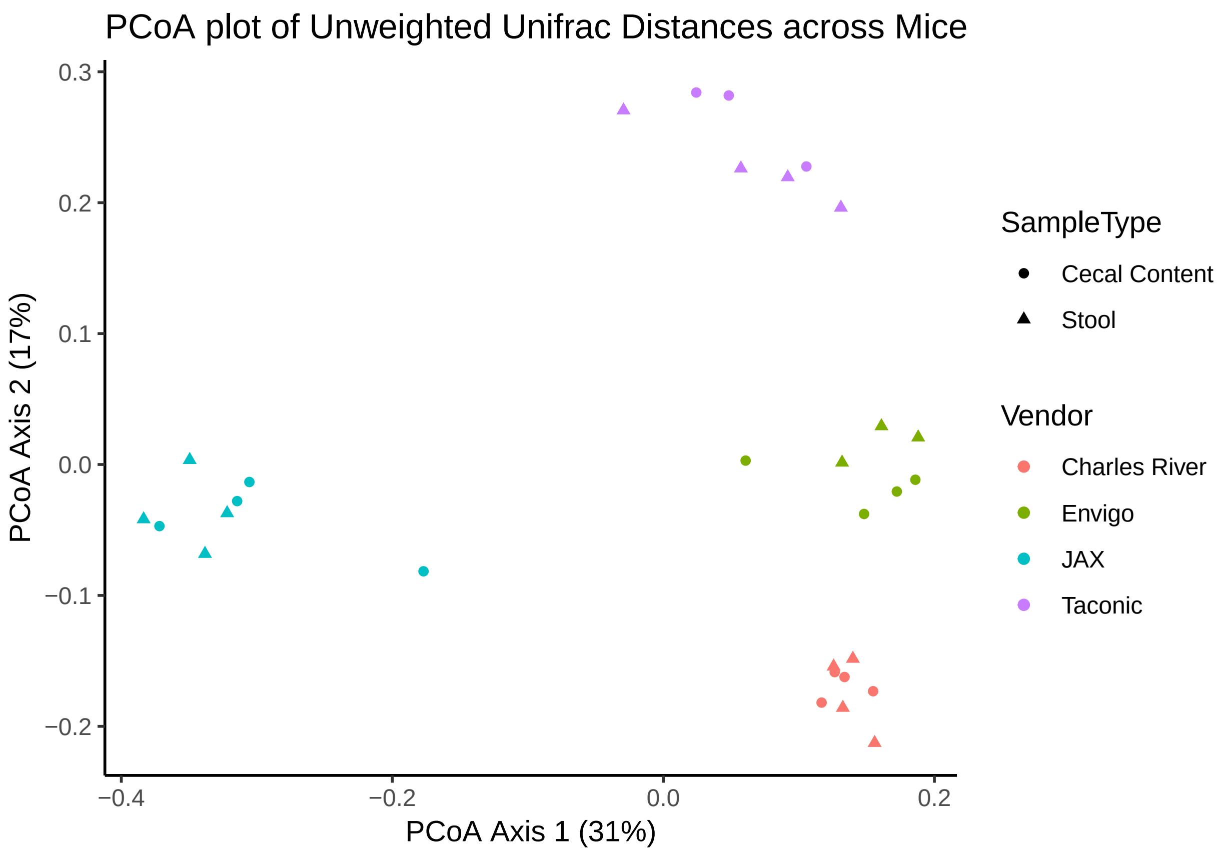

Figure 3.1: PCoA plot of microbiota distances between mice

using that information to calculate probabilities of a point being next to each other (Strayer, 2018). The method then randomly maps the points onto a two-dimensional space and attempts to move the points – so that the probabilities of being next to the other points is similar to the original probabilities in the high-dimensional space (Strayer, 2018). While a powerful technique, there are important caveats when compared to techniques such as PCA. The usage of random placing and probability-based calculations mean that when each time t-SNE is run, the result is slightly different (unlike in PCA or PCoA where each run is guaranteed to be the same). Furthermore, different settings in determining the calculation of conditional probabilities can impact the final outcome of the two-dimensional mapping.

3.3.1 t-SNE example

```r
# create tsne model
do_tsne <- function(dist_matrix, perp) {
  set.seed(9)
  tsne_model <- Rtsne(as.matrix(dist_matrix),
                      check_duplicates = FALSE,
                      is_distance = TRUE,
                      pca = FALSE,
                      perplexity = perp,
                      theta = 0.5,
                      dims = 2)

  return(as.data.frame(tsne_model$Y))
}

df_tsne <- do_tsne(uu, 1) %>%
  mutate(perp = 1) %>%
  cbind(s_vendor)

for (i in c(2:9)) {
  df <- do_tsne(uu, i) %>%
    mutate(perp = i) %>%
    cbind(s_vendor)

  df_tsne <- rbind(df_tsne, df)
}

ggplot(df_tsne, aes(x = V1, y = V2, color = Vendor)) +
  geom_point() +
  labs(title = "t-SNE Plot on Unweighted UniFrac Distances across Mice",
       subtitle = "With varying perplexity settings",
       x = "t-SNE Axis 1",
```

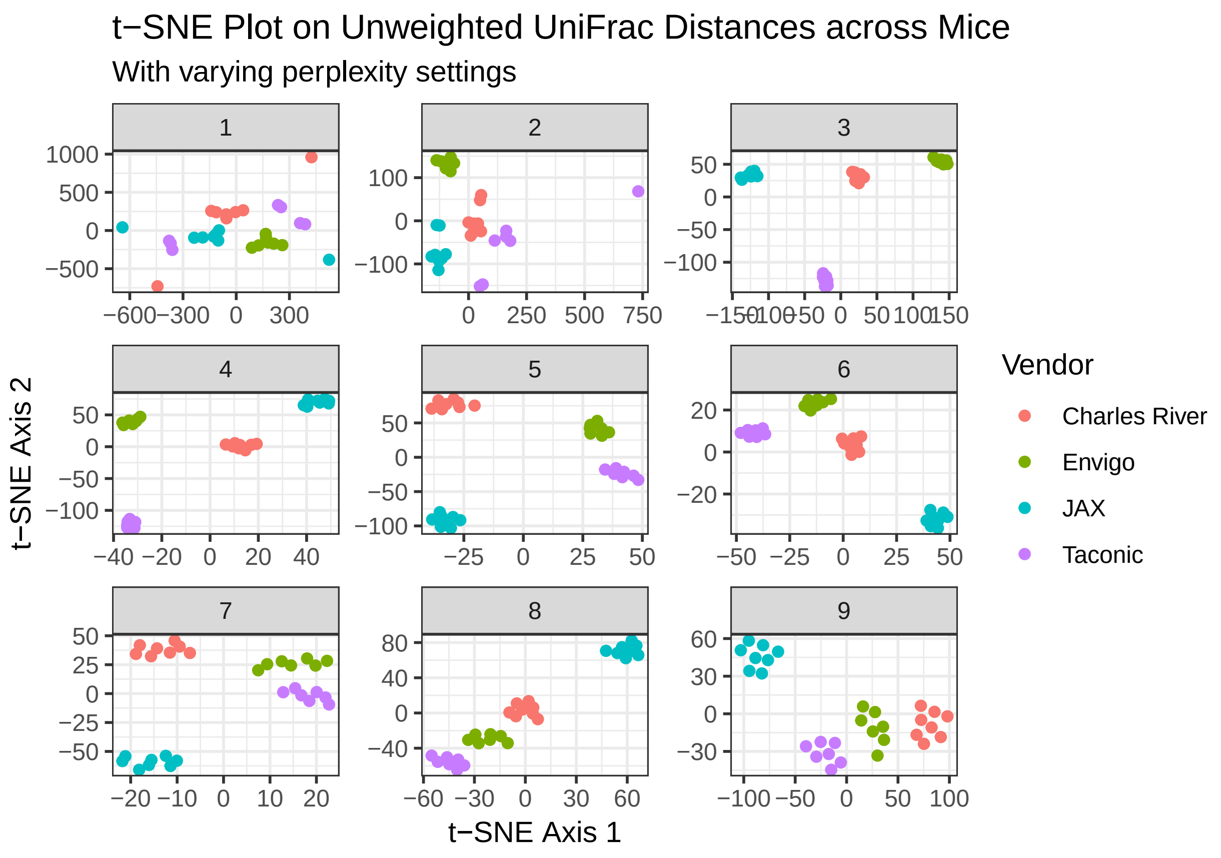

Figure 3.2: t-SNE plot of microbiota distances between mice

```
       y = "t-SNE Axis 2") +
  theme_bw() +
  facet_wrap(~ perp, scales = "free")
```

In Figure 3.2, t-SNE analysis was done on the same distance matrix as in Figure 3.1 to transform into a two-dimensional space. As with PCoA, one can see the differential clustering by vendor. As mentioned in the previous paragraph, t-SNE results can change based on the different settings. Figure 3.2 demonstrates how changing the `perplexity` can impact the final result.

3.4 Other dimensionality reduction techniques

PCA, PCoA, and t-SNE are common techniques used for dimensionality reduction, but there are other methods that may perform better in different contexts. For instance, a recent paper describes a new technique similar to t-SNE for single-cell RNA sequencing called uniform approximation and projection (UMAP) (Becht et al., 2018). This method, like t-SNE, is considered a nonlinear dimensionality reduction technique, as opposed to PCA where the usage of linear combinations makes it a linear dimensionality reduction technique. While there are multiple techniques available, the usage of PCA, PCoA, and t-SNE are sufficient to demonstrate the machine learning techniques in the next section.

Chapter 4

Supervised Learning

4.1 Introduction to supervised learning

This section covers supervised learning techniques – where the output variables are already known. While there are many techniques available, this paper will cover (1) regressions (multiple linear vs logistic vs multinomial), (2) K-nearest neighbors, (3) decision trees, (4) random forests, (5) deep learning, (6) linear discriminant analysis, and (7) support vector machines. All of these methods can be found in biomedical research papers, especially those involving microbiome and next-generation sequencing data. Some of these methods will include the code implementation, whereas others will have only a description due to either data or paper restraints.

4.2 Regressions

Regression is one of the simple (but still powerful) forms of supervised learning. In brief, regression is an attempt to understand the relationship between the input and output (Shalizi, 2018). A linear regression (commonly seen in papers with a plot, a regression line, a r^2 value, and a p-value) establishes a line with the lowest mean squared error (distance between the data points and the line). The r^2 value, also known as the coefficient of determination, describes how much of the variation in the output (typically the y-axis variable) is explained by the input (typically the x-axis variable). As mentioned in the background, the bias-variance tradeoff plays a role here as the ability to predict an output is determined by the mean squared error (or an equivalent measure) (Shalizi, 2018).

4.2.1 Linear regression example

```r
aa <- readRDS("data/aa.rds")
mice <- readRDS("data/mice.rds")
```

```r
lr_df <- aa %>%
  filter(Tissue == "Stool") %>%
  select(MouseID, Tissue, Metabolite, ConcentrationNZ)

lr_his_df <- lr_df %>%
  filter(Metabolite == "Glycine") %>%
  mutate(Glycine = ConcentrationNZ) %>%
  select(MouseID, Tissue, Glycine) %>%
  mutate(y = NA,
         meta_comp = NA)

lr_g_df <- lr_his_df

lr_df <- lr_df %>%
  filter(Metabolite != "Glycine")

uniq_meta <- unique(lr_df$Metabolite)
for (m in uniq_meta) {
  df <- lr_df %>%
    filter(Metabolite == m) %>%
    mutate(Glycine = lr_his_df$Glycine,
           y = ConcentrationNZ,
           meta_comp = Metabolite) %>%
    select(., -ConcentrationNZ, -Metabolite)

  lr_g_df <- rbind(lr_g_df, df)
}

lr_g_df <- lr_g_df %>% filter(!is.na(meta_comp))

ggplot(lr_g_df, aes(x = Glycine, y = y)) +
  geom_point(alpha = 0.75) +
  geom_smooth(method = "lm", se = FALSE) +
  theme_bw() +
  labs(title = "Amino acid comparisons in stool",
    subtitle = "Glycine concentration vs other amino acids",
    x = "Glycine concentration",
    y = "Other amino acid concentration") +
  facet_wrap(~ meta_comp, ncol = 3, scales = "free")
```

In Figure 4.1, metabolomic data was used due to the restraints inherent to regression functions. Amino acid concentrations were measured from stool samples. Glycine concentrations were then compared with the other amino acids and a linear regression line was drawn. From looking at how well the lines can represent the data points, this graph is meant to demon-

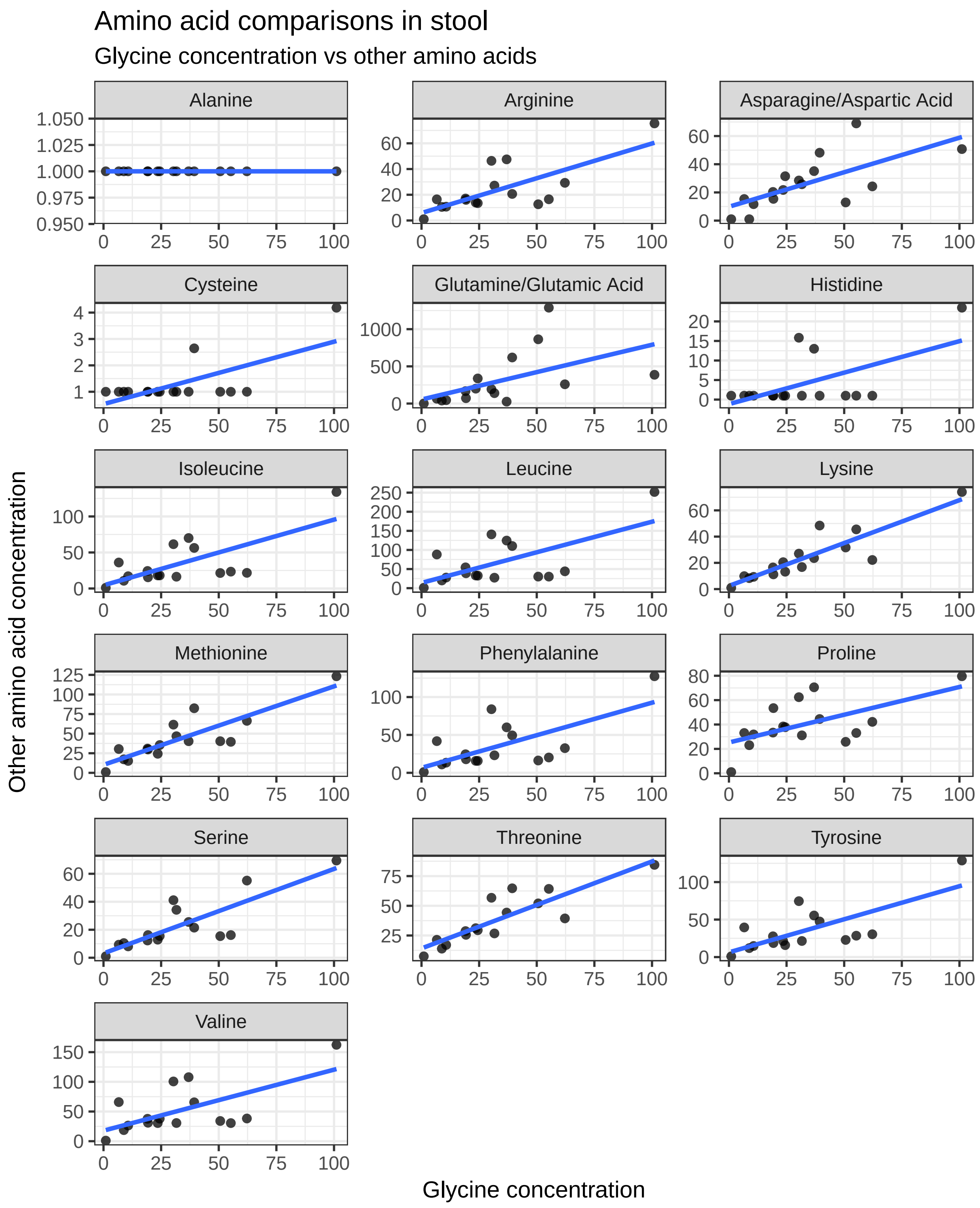

Figure 4.1: Linear regression example

strate that some data, but not all, may work well with a linear regression model. Even though linear regression may be simple, it can be useful (i.e. for the relationship between glycine and lycine) or not informative (i.e. for the relationship between glycine and histidine).

There are many types of regressions that are extensions of linear regression[1]. For instance, a multilinear regression factors in more than one input variable to predict an output variable. Logistic regression is common in the biological sciences when trying to analyze a dependent variable that has a binary output (i.e. $y = 0$ or $y = 1$) (Shalizi, 2018). This type of regression calculates the probability of $y = 1$, as a function of the independent variables. As such, a probability of greater or equal to 0.5 will result in a prediction that $y = 1$, otherwise it will predict $y = 0$. Multinomial regression (also known as polytomous regression) builds on the same principles as logistic regression. The main difference is that multinomial regression works for a dependent variable that has more than two categories (i.e. $y = 0$, $y = 1$, or $y = 2$).

4.3 K-nearest neighbors (KNN)

KNN is a type of regression which takes into the account the distance between points, hence its name (Shalizi, 2018). This algorithm calculates the distance between data points and a new data point of interest (will be referred to as the query) to predict its classification (Harrison, 2018). Of the k (a positive integer that is less than the number of original data points) data points that are closest to the query, the algorithm will then return the mode or mean of the classifications of those data points (which is already known) (Harrison, 2018). The thought is that the samples with the closest distances to the query are likely to have the same classification, which would be the predicted output for the query (Harrison, 2018). KNN falls under the category of a linear smoother in creating a regression curve that is smooth and close to the actual mean, when k is increased (Shalizi, 2018). The choosing of k for optimal KNN performance is important due to the bias-variance tradeoff but will not be discussed in this paper due to paper length restraints (Shalizi, 2018).

4.3.1 KNN example

```r
knn_df <- pc_df_uu %>%
  select(Vendor, Axis.1, Axis.2)

run_knn <- function(k) {
  res <- c()

  for (i in c(1:100)) {
```

[1] A useful resource for understanding regressions and other statistical topics is Pennsylvania State University's online documentation for their statistic courses. For example, STAT 504 covers multinomial logistic regression and can be found at this link (https://onlinecourses.science.psu.edu/stat504/node/172/).

```
    df <- knn_df %>%
      mutate(trial = sample(2, n(), replace = TRUE, prob = c(0.67, 0.33)))

    knn_train <- df %>%
      filter(trial == 1)

    knn_test <- df %>%
      filter(trial == 2)

    knn_prd <- knn(train = knn_train %>% select(., -Vendor),
                   test = knn_test %>% select(., -Vendor),
                   cl = knn_train$Vendor,
                   k = k)

    prop <- sum((knn_test$Vendor == knn_prd) == TRUE,
                na.rm = TRUE) / length(knn_prd)
    res <- c(res, prop)
  }

  return(res)
}

res_df <- data.frame(k = c(rep(3, 100), rep(4, 100), rep(5, 100)),
                     accuracy = c(run_knn(3),
                                  run_knn(4),
                                  run_knn(5)))

ggplot(res_df, aes(x = accuracy, fill = as.factor(k), color = as.factor(k))) +
  geom_density(alpha = 0.6) +
  labs(title = "KNN Regression with Varied k values",
       color = "k value",
       fill = "k value",
       x = "Prediction accuracy",
       y = "Density") +
  theme_bw()
```

In Figure 4.2, the KNN method was applied to the PCoA transformed data points (as seen in Figure 3.1). The data was split into a training and a testing dataset in order to see how well the KNN could predict vendor with different values of k. The graph demonstrates how different values of k can impact prediction accuracy.

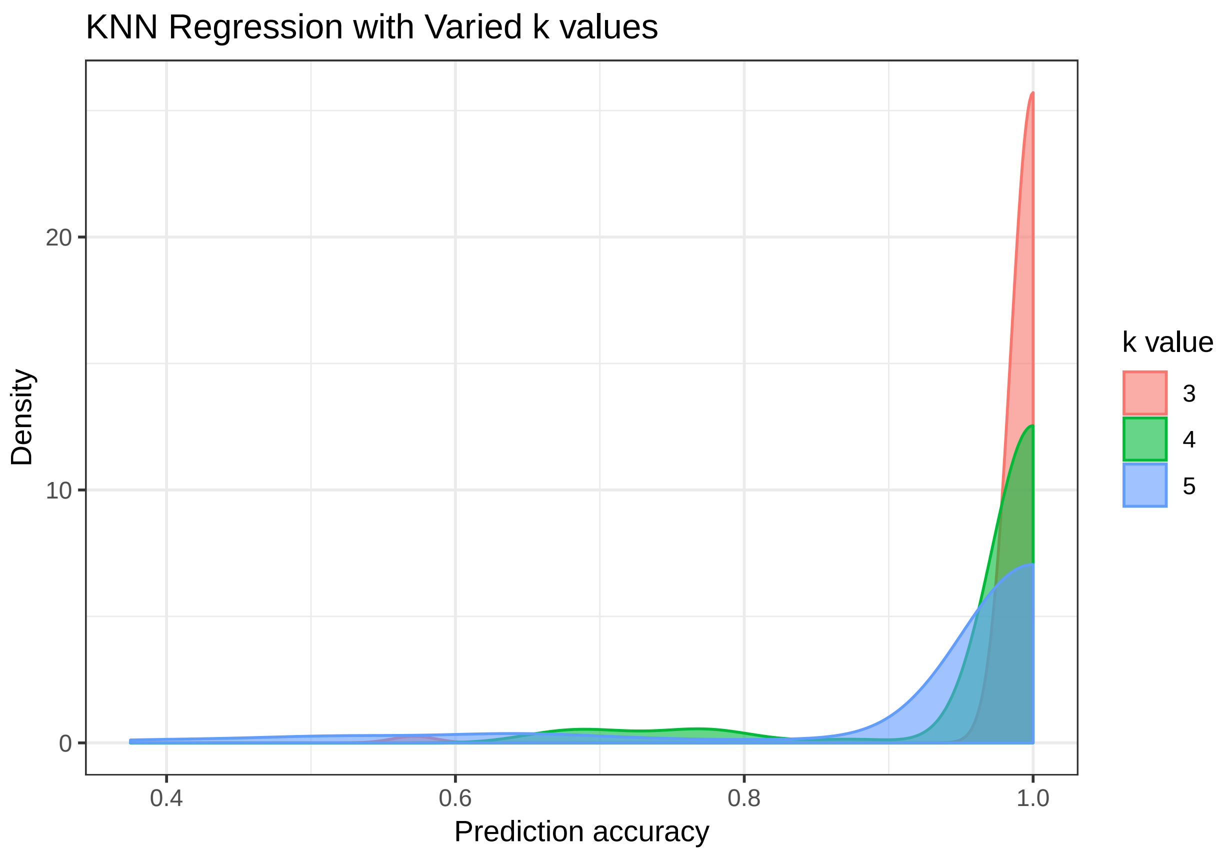

Figure 4.2: KNN example

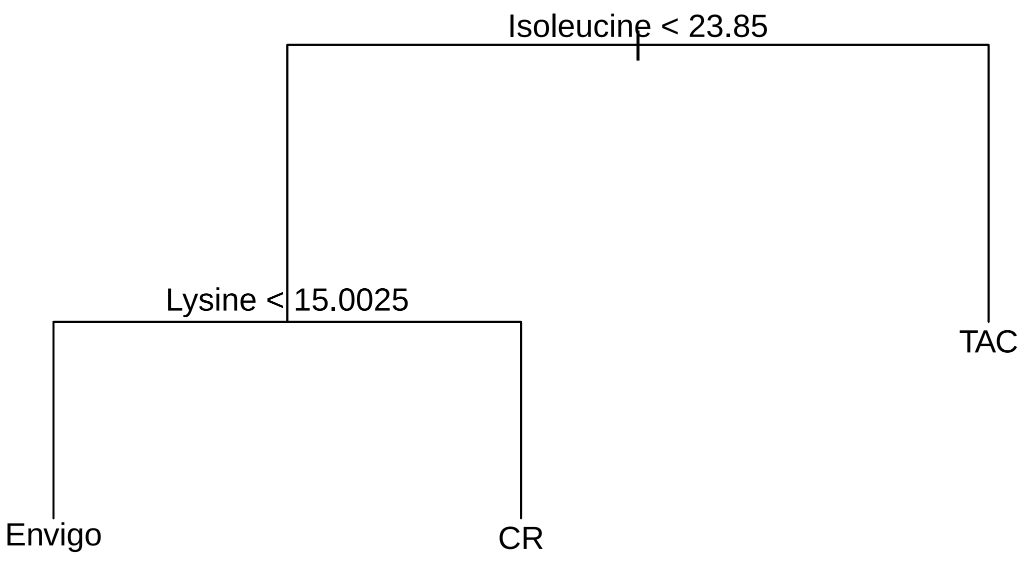

Figure 4.3: Decision tree example using amino acid concentrations to predict vendor

4.4 Decision trees

Decision trees fall into the category of nonparametric supervised learning methods, meaning that it does not depend on the distribution of the variables. In brief, decision trees separate the data based on different variables and construct a tree for predicting the final classification. The algorithm aims to create the smallest tree (requires the least number of variables) that increases information gain (selecting variables that can split the data into two subsets that are individually homogenous).

```
aa_dt_df <- lr_df %>%
  left_join(select(mice, MouseID, Vendor), by = "MouseID") %>%
  mutate(Metabolite = gsub("[/ ]", "_", Metabolite)) %>%
  spread(Metabolite, ConcentrationNZ) %>%
  mutate(Vendor = as.factor(Vendor))

aa_lbls <- colnames(aa_dt_df)[5:length(aa_dt_df)]

tree_fm <- as.formula(paste("Vendor ~ ", paste(aa_lbls, collapse = " + "), sep = ""))

tree_res <- tree(tree_fm, data = aa_dt_df)

plot(tree_res)
text(tree_res)
```

In Figure 4.3, the decision tree method was applied to the metabolite dataset (as seen in Figure 4.1). All the amino acids detected were used to try to classify the samples by vendor. As seen in the result, isoleucine levels ≥ 23.85 can predict samples from TAC. Proceeding down the decision tree can further separate the samples by using lysine levels.

4.5 Random forests

This method is an extension of decision trees with the same aim of predicting the final classification for a data point (Breiman, 2001; Liaw et al., 2002). The algorithm starts by randomly grabbing a chunk of data from the dataset (called the training data) and creating a single decision tree. It will then test the accuracy of that tree by using it on the data that was not initially used for constructing the tree. The algorithm then repeats this process by grabbing another random chunk of data and constructing another tree. By repeating this process multiple times, the algorithm will create a large number of decision trees, hence the term random forests. Once a random forest is established, the trees will take in new data and will each generate their own prediction of what the classification is. The algorithm will select the most common classification and return that as the predicted classification for the new data.

```
forest_res <- randomForest(tree_fm, data = aa_dt_df)

importance(forest_res)
```

```
##                            MeanDecreaseGini
## Arginine                         0.52426159
## Asparagine_Aspartic_Acid         0.48782994
## Cysteine                         0.08388865
## Glutamine_Glutamic_Acid          0.79407455
## Histidine                        0.10707186
## Isoleucine                       1.33005098
## Leucine                          0.90613227
## Lysine                           1.01425911
## Methionine                       1.16218851
## Phenylalanine                    0.97286201
## Proline                          0.54284661
## Serine                           0.63527573
## Threonine                        0.73175700
## Tyrosine                         1.23192239
## Valine                           0.75157881
```

The output of random forests includes its ability to accurately predict the samples as well as the relative importance of the independent variables used to construct the forest. Isoleucine is identified as an important separator, which supports the finding in Figure 4.3.

4.6 Neural networks (NN)

NN and the field of deep learning, like many of the methods mentioned here, are topics that can each be written about in their own paper (or book) (Goodfellow et al., 2016). This subsection only serves to briefly introduce this and include resources for further exploration.

NNs, in short, aim to mimic how neurons act[2]. The input of data (analogous to neurotransmitters) results in an output (analogous to whether or not the neuron fires). The beauty of neural networks lies within its name, where multiple layers of neurons are connected with each other. NNs have impressive predictive capabilities by learning from vast amounts of data. For instance, NNs can be trained to recognize objects in pictures, to recognize handwriting, and other tasks that strike at the core of deep learning (Nielsen, 2015; Goodfellow et al., 2016). NN are increasingly present in the biological sciences as research groups contribute towards large and accessible databases, which present prime opportunities for the development of neural networks to predict outputs such as treatment outcomes (Snow et al., 1994; Kappen and Neijt, 1993) and diagnoses (Ercal et al., 1994; Acharya et al., 2018; Esteva et al., 2017).

4.7 Support vector machines (SVM)

SVM is a popular method (Furey et al., 2000; Guyon et al., 2002) used for classification in the biological sciences because of its ability to deal with high dimensional data and to look at non-linear relationships. This method searches for a hyperplane (in 2D: a line; in 3D: a plane, etc.) that can separate the data points, which is determined by having the largest average distance from each of the data points to the hyperplane (Le, 2018). SVM employs a method called a kernel trick[3] to perceive data points in different dimensional space in order to find the best hyperplane (Le, 2018). After a hyperplane is identified, new data points can be classified by which side they fall on the hyperplane.

4.7.1 SVM example

```r
svm_df <- pc_df_uu %>%
  select(Vendor, Axis.1, Axis.2)

# get best model on cost
model_svms <- tune(svm,
                   as.factor(Vendor) ~ .,
                   data = svm_df,
                   kernel = "radial",
                   ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))

# get parameters for best model
best_svm <- model_svms$best.model
```

[2]The developers of a software package called TensorFlow (an open-source package for running neural networks) have also created an interactive simulation of neural networks at this link ((https://playground.tensorflow.org/)).

[3]This post ((https://stats.stackexchange.com/questions/152897/how-to-intuitively-explain-what-a-kernel-is)) offers an explanation about how kernel tricks work.

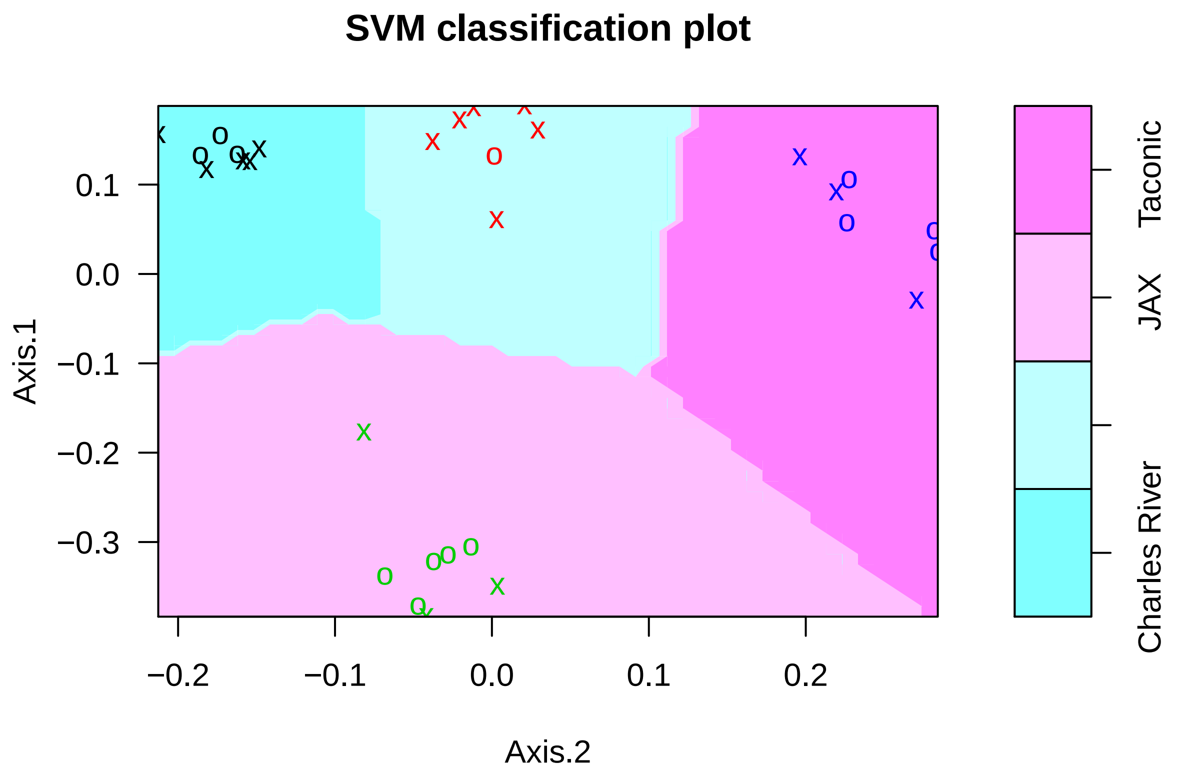

Figure 4.4: SVM classification of unweighted UniFrac distances

```
# predict based on SVM and plot
pred <- predict(best_svm, svm_df, decision.values = TRUE)
plot(best_svm, svm_df)
```

In Figure 4.4, SVM was applied to the PCoA transformed dataset (as seen in Figure 3.1). If the data points from Figure 3.1 are mirrored and then rotated, the points will be the same as seen here in Figure 4.4. Based on the shading, it is interesting to note how SVM can demarcate areas associated with different vendors in the PCoA space. These boundaries are effective, at least for this dataset, in classifying the samples by the correct vendor.

Chapter 5

Unsupervised Learning

5.1 Introduction to unsupervised learning

Unsupervised machine learning is arguably a more difficult task as the classifications are not known *a priori*. These methods use clustering to discover ways that the data can be grouped. Techniques mentioned above such as PCA, PCoA, and t-SNE can be useful in viewing the data in a lower dimension and can be a good starting point for unsupervised learning. This paper will cover 1) hierarchical clustering, 2) divisive clustering, and 3) hidden Markov models. As with the supervised learning section, some of these methods will be only be described while others will contain accompanying R code and graphics.

5.2 Hierarchical agglomerative clustering

This type of unsupervised learning can be seen from recent papers that utilize heatmaps and have dendrograms drawn on either or both axes. These dendrograms are the final product of hierarchical clustering. Distance measurements between each data point are used to cluster the data points, which can then be portrayed as a binary tree (each branch point separates into two children). In a hierarchical clustering dendrogram, the similarities between two points can be assessed by looking at the height of the first shared node (analogous to looking at the most recent common ancestor in a phylogeny). Hierarchical clustering can provide insight on any underlying structure and can be useful when accompanied with heatmaps to show clusters in a quantitative and qualitative manner.

5.3 Hierarchical divisive clustering

Divisive clustering is a type of hierarchical clustering that differs from agglomerative clustering by its approach in dividing the data. Agglomerative clustering, as its name suggests, starts with the data points as individual parts and starts grouping those together to create the tree (as if drawing the tree from the branches to the root). Divisive takes the opposite approach and starts with all the data points in one group and divides that group to create the tree (as if drawing the tree from root to the branches). The advantage of divisive clustering is that it takes into account how the data looks at a global perspective, whereas agglomerative clustering loses that perspective when starting with each data point as an individual.

Two of the methods to divide the first group for divisive clustering include k-means and partitioning across medoids (PAM). While there are significant differences between the two methods, the idea is similar in which clusters are created based on the proximity to the closest centroid (mean of a specific cluster of data points) or to the medoid (a data point that is selected as the center of a cluster). The difficulty for either method comes to choosing the value for k, which is the number of clusters that the scientist must assign *a priori*. There are multiple techniques for deciding k, such as the sum of squares method or the silhouette method.

```r
do_kmeans <- function(df, dist) {
  df_kmeans <- df
  sum_sq <- c()
  col_names <- c()
  sil_width <- c()
  iterations <- c(2:10)

  # run kmeans
  for (i in iterations) {
    kmeans_model <- kmeans(scale(df), i)

    col_name <- paste("cl_kmeans_", i, sep = "")
    col_names <- c(col_names, col_name)

    df_kmeans[, col_name] <- kmeans_model$cluster
    sum_sq <- c(sum_sq, kmeans_model$tot.withinss)

    sil <- silhouette(kmeans_model$cluster, dist)
    sil_width <- c(sil_width, summary(sil)$avg.width)
  }

  df_kmeans <- df_kmeans %>%
    gather_("k", "cluster", col_names) %>%
    mutate(k = gsub("^cl_kmeans_", "", k))

  # plot clustering
  p <- ggplot(df_kmeans, aes(x = V1, y = V2, color = as.factor(cluster))) +
    geom_point(alpha = 0.5) +
    labs(title = "k-means clustering",
         x = "Axis 1",
         y = "Axis 2",
         color = "Cluster #") +
    theme_bw() +
    # scale_color_viridis(discrete = TRUE) +
    facet_wrap( ~ as.numeric(k))

  # look at sum of squares
  p2 <- qplot(x = iterations, y = sum_sq) +
    geom_point() +
    labs(title = "Which k to pick?",
         subtitle = "Sum of squares method",
         x = "k",
         y = "Total within sum of squares") +
    theme_bw()

  return(list(p, p2))
}
```

```
rename_pc_df <- function(df) {
  df <- df %>%
    select(Axis.1, Axis.2) %>%
    rename(V1 = Axis.1,
           V2 = Axis.2)

  return(df)
}

graphs <- do_kmeans(rename_pc_df(pc_df_uu), uu)

graphs[[1]]
```

```
graphs[[2]]
```

In Figure 5.1, this clustering was done on the same PCoA transformed data for unweighted UniFrac distances between mice. The value of k (indicated by number in the gray bar) was varied and then graphed to show how the clusters appear (each cluster was assigned a number). In 5.2, it is possible to use the `total within sum of squares` measure to determine the best k value. The points dip and start to progress in a flat trajectory around $k = 4$, suggesting that $k = 4$ is the best k value for this specific data.

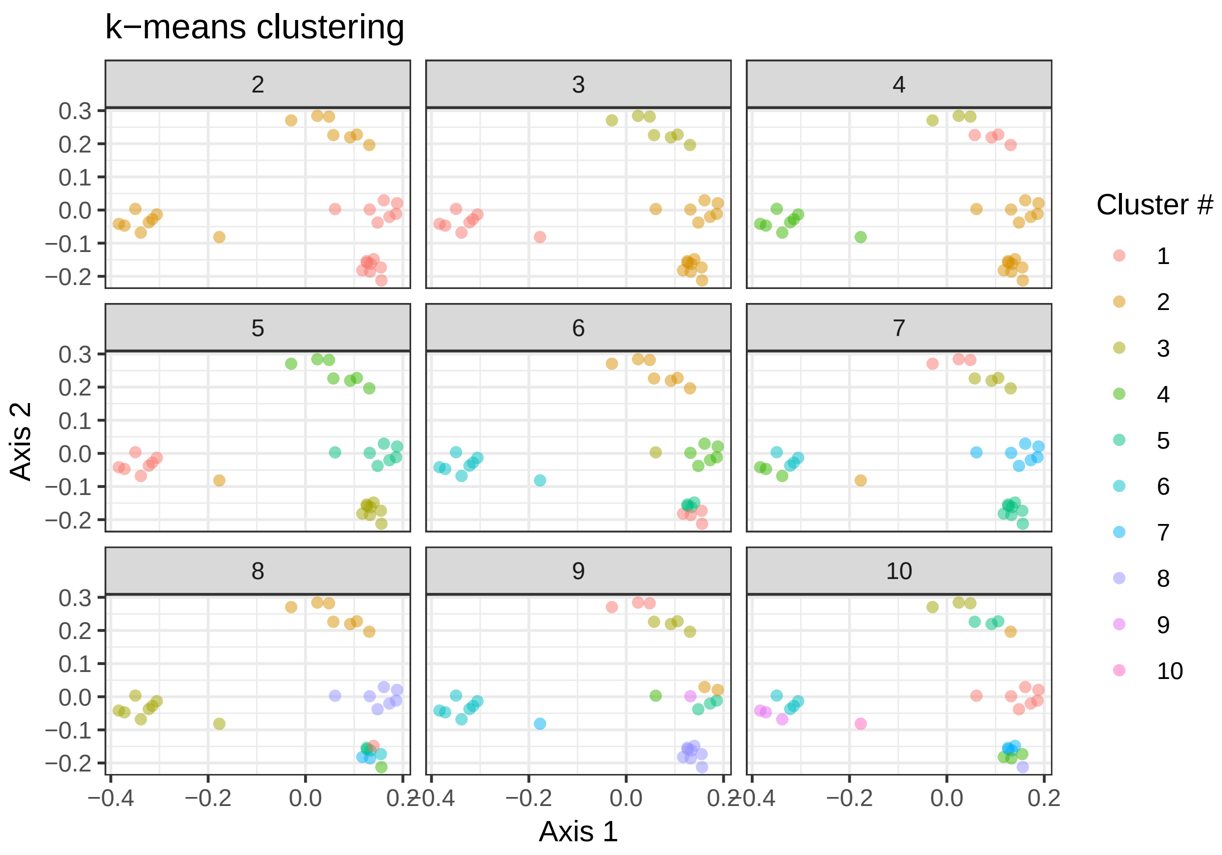

Figure 5.1: Kmeans clustering with different k values

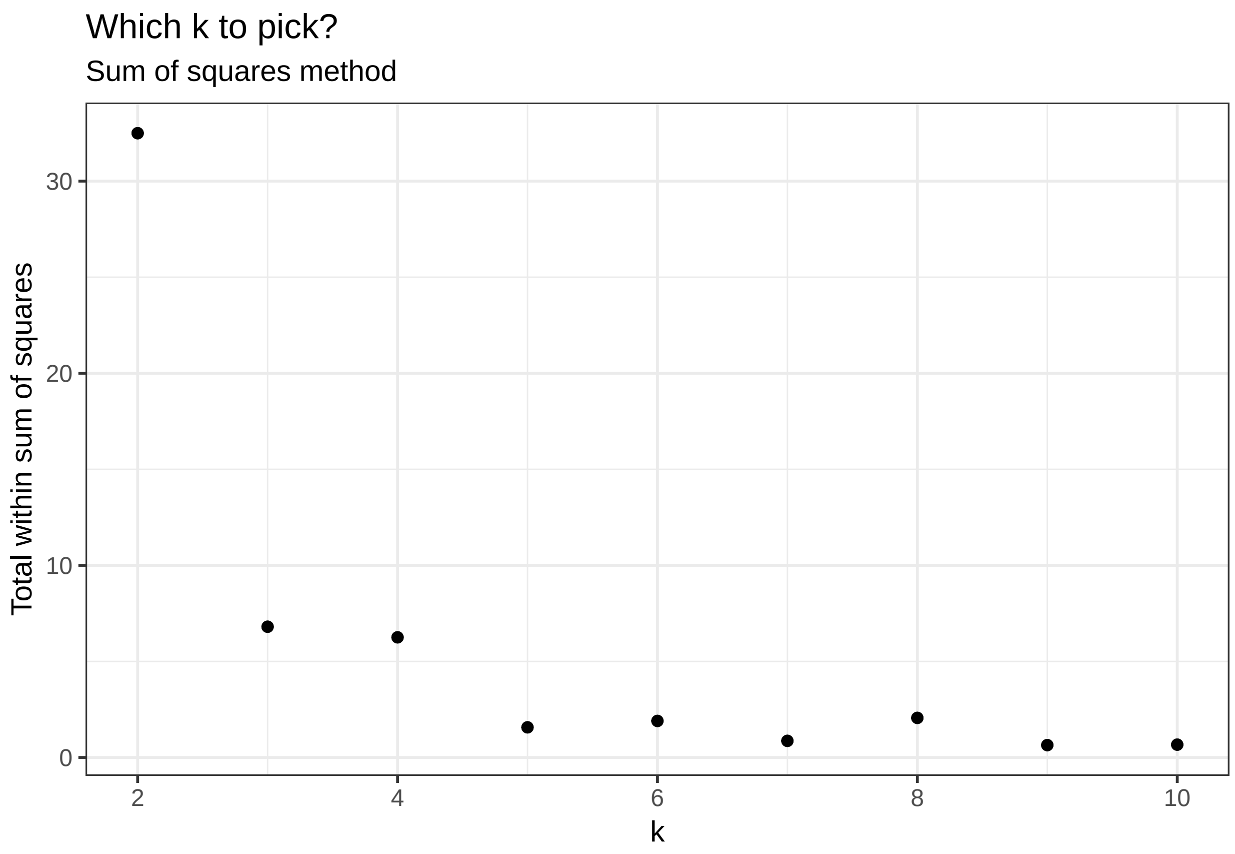

Figure 5.2: Deciding which k to use

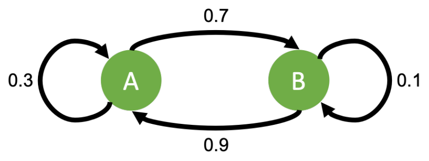

Figure 5.3: Simple HMM example

5.4 Hidden Markov models (HMM)

A Markov model is a model that describes a system where state transitions are random and are independent of past events. For example, the below figure shows a two state Markov model. From state A, there is a 30% chance of going back to state A or a 70% chance of going to state B regardless of any previous transitions. This simple model can be extended to multiple states with more transition weights. In HMMs, the overall idea is the same but there can also be hidden states and weights. HMMs are commonly used when studying nucleotide or amino acid sequences. Examples include sequence alignment tools as well as the HMMER program, which can take in an amino acid sequence and output proteins that are related to your sequence of interest. Furthermore, HMMs have been used for species-level and even strain-level identification for microbiome studies via the QIIME program, developed here at Penn.

Bibliography

Acharya, U. R., Oh, S. L., Hagiwara, Y., Tan, J. H., and Adeli, H. (2018). Deep convolutional neural network for the automated detection and diagnosis of seizure using eeg signals. *Computers in biology and medicine*, 100:270–278.

Becht, E., McInnes, L., Healy, J., Dutertre, C.-A., Kwok, I. W., Ng, L. G., Ginhoux, F., and Newell, E. W. (2018). Dimensionality reduction for visualizing single-cell data using umap. *Nature Biotechnology*.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Ercal, F., Chawla, A., Stoecker, W. V., Lee, H.-C., and Moss, R. H. (1994). Neural network diagnosis of malignant melanoma from color images. *IEEE Transactions on biomedical engineering*, 41(9):837–845.

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., and Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115.

Furey, T. S., Cristianini, N., Duffy, N., Bednarski, D. W., Schummer, M., and Haussler, D. (2000). Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422.

Harrison, O. (2018). Machine learning basics with the k-nearest neighbors algorithm. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

Kappen, H. and Neijt, J. (1993). Neural network analysis to predict treatment outcome. *Annals of oncology*, 4(suppl_4):S31–S34.

Le, J. (2018). Support vector machines in r. https://www.datacamp.com/community/tutorials/support-vector-machines-r.

Liaw, A., Wiener, M., et al. (2002). Classification and regression by randomforest. *R news*, 2(3):18–22.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press. http://neuralnetworksanddeeplearning.com/.

Shalizi, C. R. (2018). *Advanced Data Analysis from an Elementary Point of View*. Carnegie Mellon University, Pittsburgh, PA. http://www.stat.cmu.edu/~cshalizi/ADAfaEPoV/.

Snow, P. B., Smith, D. S., and Catalona, W. J. (1994). Artificial neural networks in the diagnosis and prognosis of prostate cancer: a pilot study. *The Journal of urology*, 152(5):1923–1926.

Strayer, N. (2018). t-sne explained in plain javascript. https://beta.observablehq.com/@nstrayer/t-sne-explained-in-plain-javascript.