Introduction - What is Machine Learning?

At the intersection between Applied Mathematics, Computer Science, and Biological Sciences is a subset of knowledge known as Bioinformatics. Some find Bioinformatics and its cousin Data Science difficult to define. However, the most ubiquitous pictograph of Bioinformatics and Data Science may say a thousand words even if we cannot. See Figure 1. What separates these two pursuits is the domain knowledge that each focus on. Where Data Science may focus on understanding buying habits of individuals, Bioinformatics tends to focus on the understanding of DNA and its relevance to disease. It is now common to find new fields emerging such as the study of chemistry using applied mathematics and computers which beget the field of Chemo-informatics.[1] [2] Today, since hospitals are warehouses of modern medical records and knowledge they make career paths such as Healthcare-Informatics possible.[3]

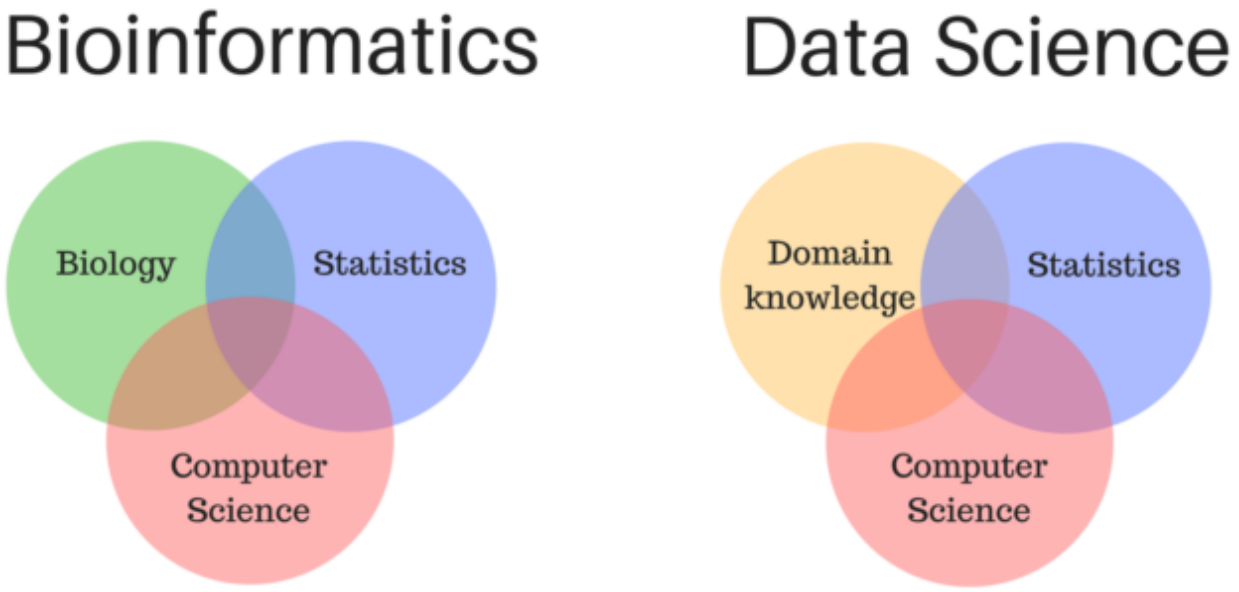


Figure 1: Venn Diagrams of Bioinformatics And Data Science

[4]

Generally speaking, Bioinformatics derives its knowledge from various items. To start our data may consist of numbers, text and even pictures as input. Numerical values may be from scientific sensors or instrumentation, such as Next Generation[5] DNA sequence data.[6] Text data is sourced from articles/books or even databases of scientific articles like PubMed which show the inter-connectivity of thought and research.[7] Graphics and pictures include graphical forms, such as relationship data that appears in genomic or proteomic or metabolic databases.[8]

However, Bioinformatics and Data Science imply a process as well. One piece of this process is Reproducible Research.[9] Reproducible Research and replication are linked but it goes beyond the ability of the work to

[1] https://www.acs.org/content/acs/en/careers/college-to-career/chemistry-careers/cheminformatics.html
[2] https://jcheminf.biomedcentral.com/
[3] https://www.usnews.com/education/best-graduate-schools/articles/2014/03/26/consider-pursuing-a-career-in-health-informatics
[4] http://omgenomics.com/what-is-bioinformatics/
[5] What is Next-Generation DNA Sequencing?, https://www.ebi.ac.uk/training/online/course/ebi-next-generation-sequencing-practical-course/what-you-will-learn/what-next-generation-dna-
[6] http://www.genome.ucsc.edu/
[7] https://arxiv.org/
[8] https://www.ebi.ac.uk/training/online/course/proteomics-introduction-ebi-resources/what-proteomics
[9] Roger Peng, The Real Reason Reproducible Research is Important, https://simplystatistics.org/2014/06/06/the-real-reason-reproducible-research-is-important/

be *independently verified*. The computer code and data must be provided. There locations explicitly spelled out such that other scientists may find and carry on the work and check all its calculations and methodology. Now that computers play such a large role procedurally the smallest error in a spreadsheet may plague the overall conclusion. Such is the case of the University of Mass. at Amherst found errors in a Harvard research paper.[10]

Machine Learning Is?

> "Machine learning is essentially a form of applied statistics with increased emphasis on the use of computers to statistically estimate complicated functions and a decreased emphasis on proving confidence intervals around these functions"
>
> — Ian Goodfellow, et al[11]

What is Predictive Modeling?

Although what this paper discusses is Predictive Modeling, it is under the umbrella of Machine Learning. The term 'Predictive Modeling' should bring to mind work in the computer science field, also called Machine Learning (ML), Artificial Intelligence (AI), Data Mining, Knowledge discovery in databases (KDD), and possibly even encompassing Big Data as well.

> "Indeed, these associations are appropriate, and the methods implied by these terms are an integral piece of the predictive modeling process. But predictive modeling encompasses much more than the tools and techniques for uncovering patterns within data. The practice of predictive modeling defines the process of developing a model in a way that we can understand and quantify the model's prediction accuracy on future, yet-to-be-seen data."
>
> Max Kuhn[12]

As an aside, I use `Predictive Modeling` and `Machine Learning` interchangeably in this document.

In the booklet entitled "The Elements of Data Analytic Style," [13] there is an useful checklist for the uninitiated into the realm of science report writing and, indeed, scientific thinking. A shorter, more succinct listing of the steps, which I prefer, and is described by Roger Peng in his book, The Art Of Data Science. The book lists what he describes as the "Epicycle of Analysis." [14]

The Epicycle of Analysis

1. Stating and refining the question,
2. Exploring the data,
3. Building formal statistical models,
4. Interpreting the results,
5. Communicating the results.

[10] https://www.chronicle.com/article/UMass-Graduate-Student-Talks/138763

[11] Ian Goodfellow, Yoshua Bengio, Aaron Courville, 'Deep Learning', MIT Press, 2016, http://www.deeplearningbook.org

[12] Max Kuhn, Kjell Johnson, Applied Predictive Modeling, Springer, ISBN:978-1-4614-6848-6, 2013

[13] Jeff Leek, The Elements of Data Analytic Style, A guide for people who want to analyze data., Leanpub Books, http://leanpub.com/datastyle, 2015

[14] Roger D. Peng and Elizabeth Matsui, The Art of Data Science, A Guide for Anyone Who Works with Data, Leanpub Books, http://leanpub.com/artofdatascience, 2015

Predictive Modeling

In general, there are three types of Predictive Modeling or Machine Learning approaches;

1. Supervised,
2. Unsupervised,
3. Reinforcement.

Due to the fact this paper only uses Supervised & Unsupervised learning and for the sake of this brevity, I discuss only the first two types of Predictive Models.

Supervised Learning

In supervised learning, data consists of observations x_i (where X may be a matrix of $\Re$ values) AND a corresponding label, y_i. The label y maybe anyone of C classes. In our case of a binary classifier, we have {'Is myoglobin', 'Is control'}.

Data set:

- $(X_1, y_1), (X_2, y_2),\ ...,\ (X_N, y_N)$;
- $y \in \{1, ...,\ C\}$, where C is the number of classes

A machine learning algorithm determines a pattern from the input information and groups this with its necessary title or classification.

One example might be that we require a machine that separates red widgets from blue widgets. One predictive algorithm is called a K-Nearest Neighbor (K-NN) algorithm. K-NN looks at an unknown object and then proceeds to calculate the distance (most commonly, the euclidean distance) to the K nearest neighbors. If we consider the figure below and choose $K = 3$, we would find a circumstance as shown. In the dark solid black on the K-Nearest-Neighbor figure, we find that the green widget is nearest to two red widgets and one blue widget. In the voting process, the K-NN algorithm (2 reds vs. 1 blue) means that the consignment of our unknown green object is red.

For the K-NN algorithm to function, the data optimally most be complete with a set of features and a label of each item. Without the corresponding label, a data scientist would need different criteria to track the widgets.

Five of the six algorithms that this report investigates are supervised. Logit, support vector machines, and the neural network that I have chosen require labels for the classification process.

[15]

What is a shallow learner?

Let us investigate the K-NN algorithm and figure 2.2 (K-Nearest-Neighbor) a little further. If we change our value of K to 5, then we see a different result. By using $K = 5$, we consider the out dashed-black line. This more considerable K value contains three blue widgets and two red widgets. If we ask to vote on our choice, we find that 3 blue beats the 2 red, and we assign the unknown a BLUE widget. This assignment is the opposite of the inner circle.

If a researcher were to use K-NN, then the algorithm would have to test many possible K values and compare the results, then choose the K with the highest accuracy. However, this is where K-NN falters. The K-NN algorithm needs to keep all data points used for its initial training (accuracy testing). Any new unknowns could be conceivably tested against any or all the previous data points. The K-NN does use a generalized

[15] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

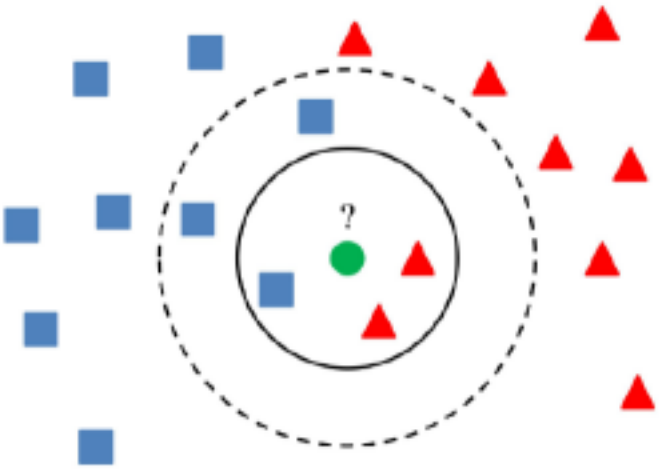


Figure 2: Example of K-Nearest-Neighbor

rule that would make future assignments quick on the contrary. It must memorize all the points for the algorithm to work. K-NN cannot delete the points until it is complete. It is true that the algorithm is simple but not efficient. Matter and fact, as the number of feature dimensions increases, this causes the complexity (also known as Big O) to rise. The complexity of K-NN is $O(\text{K-NN}) \propto nkd$.

Where n is the number of observations, k is the number of nearest neighbors it must check, and d is the number of dimensions.[16]

Given that K-NN tends to 'memorize' its data to complete its task, it is considered a lazy and shallow learner. Lazy indicates that the decision is left to the moment a new point is learned of predicted. If we were to use a more generalized rule, such as {Blue for $(x \leq 5)$} this would be a more dynamic and more in-depth approach by comparison.

Unsupervised Learning

In contrast to the supervised learning system, unsupervised learning does not require or use a **label** or **dependent variable**.

Data set:

- $(X_1), (X_2),\ ...,\ (X_N)$
- **No Y**

where X may represent a matrix of m observations by n features with $\Re$ values.

Principal Component Analysis is an example of unsupervised learning, which we discuss in more detail in chapter 3. The data, despite or without its labels, are transformed to provide maximization of the variances in the dataset. Yet another objective of Unsupervised learning is to discover "interesting structures" in the data.[17] There are several methods that show structure. These include clustering, knowledge discovery of latent variables, or discovering graph structure. In many instances and as a subheading to the aforementioned points, unsupervised learning can be used for dimension reduction or feature selection.

Among the simplest unsupervised learning algorithms is K-means. K-means does not rely on the class labels of the dataset at all. K-means may be used to determine any number of classes despite any predetermined

[16] Olga Veksler, Machine Learning in Computer Vision, http://www.csd.uwo.ca/courses/CS9840a/Lecture2_knn.pdf
[17] Kevin Murphy, Machine learning a probabilistic perspective, 2012, ISBN 978-0-262-01802-9

values. K-means can discover clusters later used in classification or hierarchical feature representation. K-means has several alternative methods but, in general, calculates the distance (or conversely the similarity) of observations to a mean value of the Kth grouping. The mean value is called the center of mass, the Physics term that provides an excellent analogy since the center of mass is a weighted average. By choosing a different number of groupings (values of K, much like the K-NN), then comparing the grouping by a measure of accuracy, one example being, mean square error.

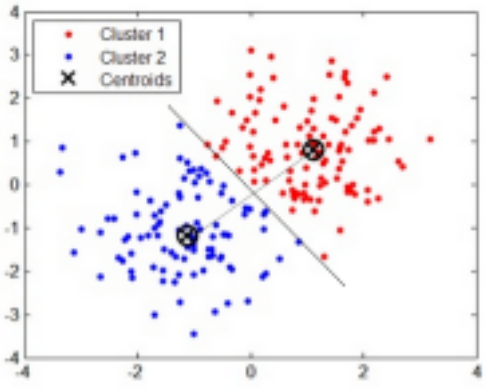


Figure 3: Example of K-Means

[18]

Five Challenges In Predictive Modeling

To many predictive modeling is a panacea for all sorts of issues. Although it does show promise, some hurdles need research. Martin Jaggi[19] has summarized four points that elucidate current problems in the field that need research. To this list, I have added one more point, which is commonly called the *Vaiance-Bias Tradeoff*.

Problem 1: The vast majority of information in the world is unlabeled, so it would be advantageous to have a good Unsupervised machine learning algorithms to use,

Problem 2: Algorithms are very specialized, too specific,

Problem 3: Transfer learning to new environments,

Problem 4: Scale, the scale of information is vast in reality, and we have computers that work in gigabytes, not the Exabytes that humans may have available to them. The scale of distributed Big Data,

The specific predictive models which are executed in this report are discussed in further detail in their own sections.

Problem 5: Bias-Variance Trade-Off.

The ability to generalize is a key idea in predictive modeling. This idea harkens back to freshman classes where one studied Student's t-test and analysis of variance.

$$E\left[\left(y_0 - \hat{f}(x_0)\right)^2\right] = Var(\hat{f}(x_0)) + \left[Bias(\hat{f}(x_0))\right]^2 + Var(\epsilon) \tag{1}$$

The bias-variance dilemma can be stated as follows.[20]

[18] https://www.slideshare.net/teofili/machine-learning-with-apache-hama/20-KMeans_clustering_20

[19] https://www.machinelearning.ai/machine-learning/4-big-challenges-in-machine-learning-ft-martin-jaggi-2/

[20] Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning; Data Mining, Inference, and Prediction, https://web.stanford.edu/~hastie/ElemStatLearn/, 2017

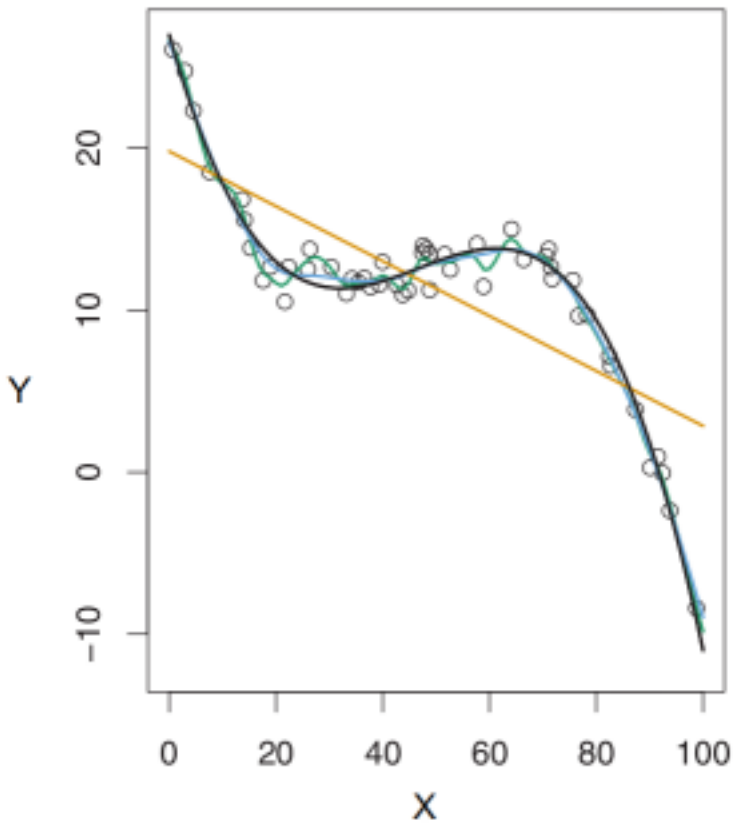


Figure 4: Bias-Variance Tradeoff

1. Models with too few parameters are inaccurate because of a large bias: they lack flexibility.
2. Models with too many parameters are inaccurate because of a large variance: they are too sensitive to the sample details (changes in the details will produce huge variations).
3. Identifying the best model requires controlling the "model complexity", i.e., the proper architecture and number of parameters, to reach an appropriate compromise between bias and variance.

One very good example is seen in figure 2.4, *Bias-Variance Tradeoff*. By considering the yellow-orange line we find a simple slope intercept model ($y \propto k \cdot x$) where the variance is high but the bias low and is not flexible enough. This is called underfitting. Looking at the green-blue line we see it follows the data set much more closely, e.g. ($y \propto k \cdot x^8$). Here the variance is very low but common sense tells us that the line is overfit and would not generalize well in a real world setting. Finally leaving us with the black line which does not appear to have too many parameters, ($y \propto k \cdot x^3$).

Research Description

Is there a correlation between the data points, which are outliers from principal component analysis (PCA), and six types of predictive modeling?

This experiment is interested in determining if PCA would provide information on the false-positives and false-negatives that were an inevitable part of model building and optimization. The six predictive models that have chosen for this work are Logistic Regression, Support Vector Machines (SVM) linear, polynomial kernel, and radial basis function kernel, and a Neural Network.

I have studied six different M.L. algorithms using protein amino acid percent composition data from two classes. Class number 1 is my positive control which is a set of Myoglobin proteins, while the second class is a control group of human proteins that do not have Fe binding centers.

Group	Class	Number of Class	Range of Groups
Controls	0 or (-)	1216	1, . . . , 1216
Myoglobin	1 or (+)	1124	1217, . . . , 2340

It is common for Data Scientists to test their data sets for feature importance and feature selection. One test that has interested this researcher is Principal component analysis. It can be a useful tool. PCA is an unsupervised machine learning technique which "reduces data by geometrically projecting them onto lower dimensions called principal components (PCs), with the goal of finding the best summary of the data using a limited number of PCs." [21] However, the results that it provides may not be immediately intuitive to the layperson.

How do the advantages and disadvantages of using PCA compare with other machine learning techniques? The advantages are numerable. They include dimensionality reduction and filtering out noise inherent in the data, and it may preserve the global structure of the data. Does the global and graphical structure of the data produced by the first two principal components provide any insights into how the predictive models of Logistic Regression, Neural Networks utilizing auto-encoders, Support Vector Machines, and Random Forest? In essence, is PCA sufficiently similar to any of the applied mathematics tools of more advanced approaches? Also, this work is to teach me machine learning or predictive modeling techniques.

The data for this study is from the Uniprot database. From the Uniprot database was queried for two protein groups. The first group was Myoglobin, and the second was a control group comprised of human proteins not related to Hemoglobin or Myoglobin. See Figure 1.5, *Percent Amino Acid Composition*. There have

[21] Jake Lever, Martin Krzywinski, Naomi Altman, Principal component analysis, Nature Methods, Vol.14 No.7, July 2017, 641-2

been a group of papers that are striving to classify types of proteins by their amino acid structure alone. The most straightforward classification procedures involve using the percent amino acid composition (AAC). The AAC is calculated by using the count of an amino acid over the total number in that protein.

- Percent Amino Acid Composition:

$$\%AAC_X = \frac{N_{Amino\ Acid\ X}}{Total\ N\ of\ AA} \tag{2}$$

The Exploratory Data Analysis determines if features were skewed and needed must be transformed. In a random system where amino acids were chosen at random, one would expect the percent amino acid composition to be close to 5%. However, this is far from the case for the Myoglobin proteins or the control protein samples. On top of this the differences between the myoglobin and control proteins can be as high as approximately 5% with the amino acid Lysine, K.

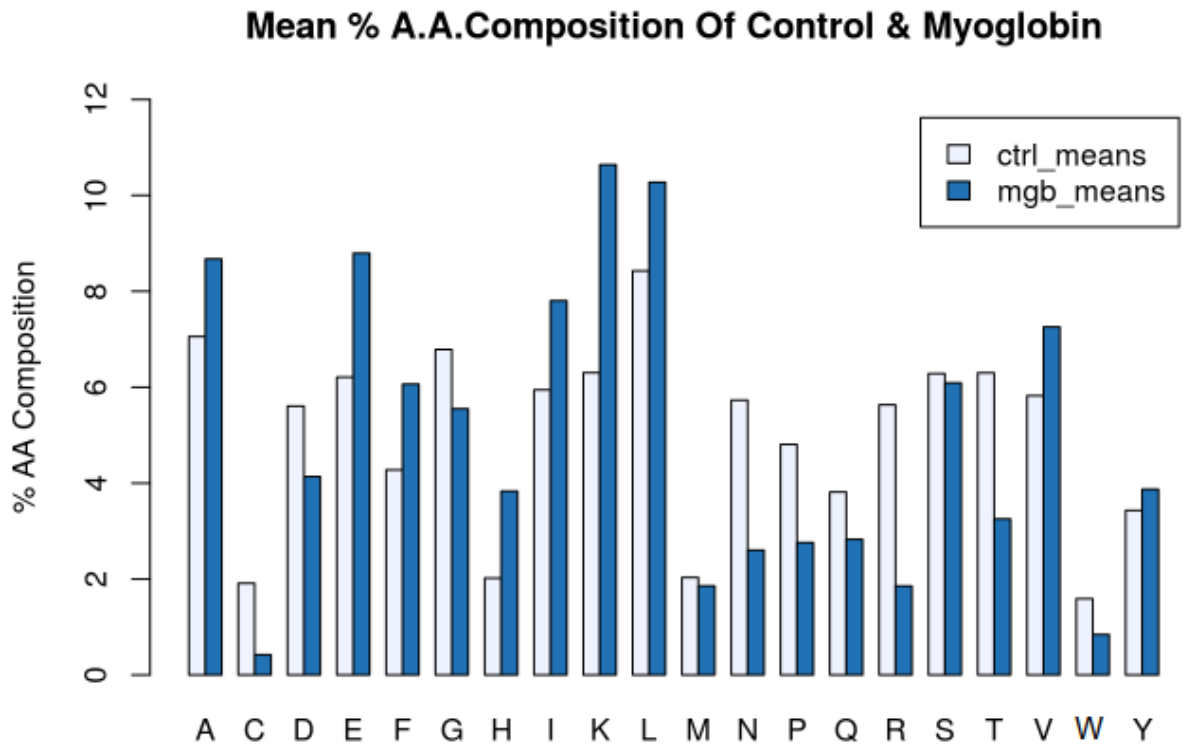


Figure 5: Mean Percent Amino Acid Compositions For Control And Myoglobin

Exploratory Data Analysis (EDA)

During EDA, the data is checked for irregularities, such as missing data, outliers among features, skewness, and visually for normality using QQ-plots. The only irregularity that posed a significant issue was the skewness of the amino acid features. Many of 20 amino acid features had a significant number of outliers, as seen by Boxplot analysis. However, only three features had skew, which might have presented a problem. Dealing with the skew of the AA was necessary since Principal Component Analysis was a significant aspect of this experiment.

Testing determined earlier that three amino acids (C, F, I) from the single amino acid percent composition needs transformation by using the square root function. The choice of transformations was natural log, log

base 10, squaring (x^2), and using the reciprocal ($1/x$) of the values. The square root transformation lowered the skewness to values of less than 1.0 from high points of greater than 2 in all three cases to {-0.102739 $\leq$ skew after transformation $\leq$ 0.3478132}.

Amino Acid	Initial skewness	Skew after square root transform
C, Cysteine	2.538162	0.347813248
F, Phenolalanine	2.128118	-0.102739748
I, Isoleucine	2.192145	0.293474879

Three transformations take place for this dataset.

`~/00-data/02-aac_dpc_values/c_m_TRANSFORMED.csv` and used throughout the rest of the analysis.

All work uses R[22], RStudio[23] and a machine learning library/framework `caret`.[24]

Caret library for R

The R/caret library is attractive to use for many reasons. It currently allows 238 machine learning models that use different options and data structures.[25] The utility of caret is that it organizes the input and output into a standard format making the need for learning only one grammar and syntax. Caret also harmonizes the use of hyper-parameters. Work becomes reproducible. Setting up the training section for caret, for this experiment, can be broken into three parts.

Tuning Hyper-parameters

The `tune.grid` command set allows a researcher to experiment by varying the hyper-parameters of the given model to investigate optimum values. Currently, there are no algorithms that allow for the quick and robust tuning of parameters. Instead of searching, a sizable experimental space test searches an n-dimensional grid in a full factorial design if desired.

Although some models have many parameters, the most common one is to search along a cost hyper-parameter.

> "The function we want to minimize or maximize is called the objective function or criterion. When we are minimizing it, we may also call it the cost function, loss function, or error function."[26]

The cost function (a term derived from business modeling, i.e., optimizing the cost) is an estimate as to how well models predicted value fits from the actual value. A typical cost function is the squared error function.

Example Cost Function: [27]

$$Cost \;=\; \Big(y_i - \hat{f}(x_i)\Big)^2 \tag{3}$$

It may be important to search the literature to determine if other researchers have used a specific range of optimum value, which may speed a search. For example, C.W. Hsu et al. suggest using a broad range of 20 orders of magnitude of powers of 2,

[22] https://cran.r-project.org/
[23] https://rstudio.com/
[24] http://topepo.github.io/caret/index.html
[25] http://topepo.github.io/caret/available-models.html
[26] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, http://www.deeplearningbook.org, 2016
[27] Roberto Battiti and Mauro Brunato, The LION way. Machine Learning-Intelligent Optimization, LIONlab, University of Trento, Italy", 2017", http://intelligent-optimization.org/LIONbook

R Code: Using TuneGrid command to test sequences of number such that the optimal cost function can be sought.

```
# tuneGrid = svmLinearGrid
svmLinearGrid <- expand.grid(C = c(2^(seq(-5, 15, 2))))
```

`expand.grid` will produce a sequence of numbers (in our case) from $2^{-5}, 2^{-3}$ *to* 2^{15} to be tested as values for the cost function.

Then switch to 4 or 5 orders of magnitude with 1/4 log steps,[28]

e.g. cost = `expand.grid(c(2^(seq(1, 5, 0.25))))`

for a finer search gird.

K-Fold Cross validation of results

Another valuable option that caret has is the ability to cross-validate results.

Cross-validation is a statistical method used to estimate the skill of machine learning models.[29]

> "The samples are randomly partitioned into k sets of roughly equal size. A model is fit using all samples except the first subset (called the first fold). The held-out samples are used for prediction by the recent model. The performance estimate measures the accuracy of the *out of bag* or *held out* samples. The first subset is returned to the training set, and the procedure repeats with the second subset held out, and so on. The k resampled estimates of performance are summarized (usually with the mean and standard error) and used to understand the relationship between the tuning parameter(s) and model utility."[30]

Cross-validation has the advantage of using the entire dataset for training and testing, increasing the opportunity that more training samples produce a better model.

R Code: 10 Fold cross-validation repeated 5 times

```
fitControl <- trainControl(method = "repeatedcv",     # Type of Cross-Validation
                           number = 10,               # Number of splits
                           repeats = 5,               # Produce 5 replicates
                           savePredictions = "final") # Save FP/FN predictions
```

Train command

The training command produces an object of the model. The first line should point out the "formula," which is modeled. The dependent variable is first. The ~ (Tilda sign) indicates a model is called. Then the desired features can be listed or abbreviated with the all (.) sign.

R Code: Train command

```
model_object <- train(Class ~ .,              # READ: Class is modeled by all features.
                      data = training_set,    # Data used
                      trControl = fitControl, # Cross Validation setup
                      method = "svmLinear",   # Use caret ML method
                      tune.Grid = Grid)       # Hyperparameter grid exploration
```

[28] Chih-Wei Hsu, et al., A Practical Guide to Support Vector Classification, 2016, http://www.csie.ntu.edu.tw/~cjlin
[29] https://machinelearningmastery.com/k-fold-cross-validation/
[30] Max Kuhn, Kjell Johnson, Applied Predictive Modeling, 2013, ISBN 978-1-4614-6848-6

Analysis of results

In binary classification, a two by two contingency table describes predicted versus actual value classifications. This table is also known as a confusion matrix for machine learning students. It is also known as a two by two contingency table in statistics.

2 x 2 Confusion Matrix	Actual = 0	Actual = 1
Predicted = 0	True-Negatives	False-Negatives
Predicted = 1	False-Positives	True-Positives

There are many ways to describe the results further using this confusion matrix. However, Accuracy is used for all comparisons.

$$Accuracy \ = \ \frac{TP + TN}{N_{Total}} \tag{4}$$

The second goal of this experiment is to produce the False Positives and False-Negatives and evaluating these by comparing them to the Principal Component Analysis Biplot of the first two Principal Components.

Exploratory Data Analysis

> "Exploratory data analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods." [1]

Introduction

The experiment described herein involves taking groups of proteins from the Uniprot.org database and comparing how well different machine learning techniques do at separating the positive from the negative control grouping. In this circumstance, proteins from the myoglobin family are analyzed against randomly chosen human proteins, which are not related to hemoglobin or myoglobin.

This work is to characterize the *anomalous points* derived from PCA and compare them to the false-positives and false-negatives generated from each of six machine learning approaches produces. For the sake of this paper *anomalous points* are defined as values greater than the absolute value of three times the standard deviation from of the first and second principal components.

$$Anomalous\ Points > |3\sigma| \quad where \quad \sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{1}$$

Therefore the M.L techniques will be:

1. Principal Component Analysis,
2. Logistic Regression,
3. SVM-Linear,
4. SVM-polynomial,
5. SVM-RBF,
6. Neural Network.

Four-Step Analysis

At this stage, data is inspected in a careful and structured way. Hence, I have chosen a four-step process:

1. Hypothesize,
2. Summarize,
3. Visualize,
4. Normalize.

Useful Guides for Exploratory Data Analysis

The summarization of the amino acid dataset is based on a hybrid set of guidelines;

1. NIST Handbook of Statistics,[2]
2. Exploratory Data Analysis With R by Roger Peng,[3]
3. Exploratory Data Analysis Using R by Ronald K. Pearson.[4]

[1] https://en.wikipedia.org/wiki/Exploratory_data_analysis
[2] https://www.itl.nist.gov/div898/handbook/
[3] Roger Peng, Exploratory Data Analysis with R, https://leanpub.com/exdata, 2016
[4] Ronald Pearson, Exploratory Data Analysis Using R, CRC Press, ISBN:9781138480605, 2018

Questions During EDA

Although exploratory data analysis does not always have a formal hypothesis testing portion, I do, however, pose several questions concerning the structure, quality, and types of data.

1. Do the independent variables of this study have large skewed distributions?
   1.1 If skews are greater than 2.0, then can a transformation be used for normalization?
   1.2 Determine what transformation to use?
2. Can Feature Selection be used, and which procedures are appropriate?
   2.1 Use the Random Forest technique known as Boruta[5] for feature importance or reduction?
   2.2 Will coefficients of correlation (R) find collinearity and reduce the number of features?
   2.3 Will principal component analysis (PCA) be useful in finding hidden structures of patterns?
   2.4 Can PCA be used successfully for Feature Selection?
3. What is the structure of the data?
   3.1 Is the data representative of the entire experimental space?
   3.2 Is missing data an issue?
   3.3 Does the data have certain biases, either known or unknown?
   3.4 What relationships do we expect from these variables?[6]

Analysis of RAW data

Raw data is considered: `./00-data/02-aac_dpc_values/c_m_RAW_AAC.csv`

```
# Import libraries, NO "doMC",
library(easypackages)
libraries("knitr", "readr", "RColorBrewer", "corrplot", "Boruta", "kableExtra")
```

```
# Import RAW data
c_m_RAW_AAC <- read_csv("./00-data/02-aac_dpc_values/c_m_RAW_AAC.csv")
Class <- as.factor(c_m_RAW_AAC$Class)
```

Visually inspect RAW data files

1. Use the command-line interface followed by the command `less`.
2. Check for binary instead of ASCII and bad Unicode.

Inspect RAW dataframe structure, `str()`

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 2340 obs. of  23 variables:
##  $ Class  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ TotalAA: num  226 221 624 1014 699 ...
##  $ PID    : chr  "C1" "C2" "C3" "C4" ...
##  $ A      : num  0.2655 0.2081 0.0433 0.0661 0.0644 ...
##  $ C      : num  0 0 0.00962 0.01381 0.03577 ...
```

[5] Miron Kursa, Witold Rudnicki, Feature Selection with the Boruta Package, DOI:10.18637/jss.v036.i11, 2010
[6] Ronald Pearson, Exploratory Data Analysis Using R, CRC Press, ISBN:9781138480605, 2018

```
##  $ D      : num  0.00442 0.00452 0.04647 0.06114 0.02861 ...
##  $ E      : num  0.031 0.0271 0.0833 0.074 0.0472 ...
##  $ F      : num  0.00442 0.00452 0.02564 0.02959 0.06295 ...
##  $ G      : num  0.0708 0.0769 0.0817 0.07 0.0443 ...
##  $ H      : num  0 0 0.0176 0.0187 0.0157 ...
##  $ I      : num  0.00885 0.0181 0.03045 0.04734 0.0701 ...
##  $ K      : num  0.28761 0.27602 0.00962 0.12426 0.05579 ...
##  $ L      : num  0.0442 0.0452 0.0577 0.0888 0.1359 ...
##  $ M      : num  0.00442 0.00452 0.01442 0.02465 0.02289 ...
##  $ N      : num  0.0177 0.0136 0.0641 0.0355 0.0558 ...
##  $ P      : num  0.0841 0.0995 0.0449 0.0434 0.0472 ...
##  $ Q      : num  0.00442 0.00905 0.04327 0.03353 0.02861 ...
##  $ R      : num  0.0133 0.0181 0.1202 0.0325 0.0415 ...
##  $ S      : num  0.0575 0.0724 0.1875 0.0838 0.0787 ...
##  $ T      : num  0.0531 0.0633 0.0625 0.0414 0.0744 ...
##  $ V      : num  0.0442 0.0543 0.0385 0.0671 0.0458 ...
##  $ W      : num  0 0 0.00481 0.01282 0.00715 ...
##  $ Y      : num  0.00442 0.00452 0.01442 0.03156 0.0372 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Class = col_double(),
##   ..   TotalAA = col_double(),
##   ..   PID = col_character(),
##   ..   A = col_double(),
##   ..   C = col_double(),
##   ..   D = col_double(),
##   ..   E = col_double(),
##   ..   F = col_double(),
##   ..   G = col_double(),
##   ..   H = col_double(),
##   ..   I = col_double(),
##   ..   K = col_double(),
##   ..   L = col_double(),
##   ..   M = col_double(),
##   ..   N = col_double(),
##   ..   P = col_double(),
##   ..   Q = col_double(),
##   ..   R = col_double(),
##   ..   S = col_double(),
##   ..   T = col_double(),
##   ..   V = col_double(),
##   ..   W = col_double(),
##   ..   Y = col_double()
##   .. )
```

Check RAW data `head` & `tail`

```
head(c_m_RAW_AAC, n = 2)
```

```
## # A tibble: 2 x 23
##   Class TotalAA PID       A     C       D      E       F      G     H       I
##   <dbl>   <dbl> <chr> <dbl> <dbl>   <dbl>  <dbl>   <dbl>  <dbl> <dbl>   <dbl>
```

```
## 1     0     226 C1    0.265     0 0.00442 0.0310 0.00442 0.0708     0 0.00885
## 2     0     221 C2    0.208     0 0.00452 0.0271 0.00452 0.0769     0 0.0181
## # ... with 12 more variables: K <dbl>, L <dbl>, M <dbl>, N <dbl>, P <dbl>,
## #   Q <dbl>, R <dbl>, S <dbl>, T <dbl>, V <dbl>, W <dbl>, Y <dbl>
```

```
tail(c_m_RAW_AAC, n = 2)
```

```
## # A tibble: 2 x 23
##   Class TotalAA PID        A       C      D      E      F      G      H      I
##   <dbl>   <dbl> <chr>  <dbl>   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1     1     335 M1123 0.0567 0.00299 0.0537 0.0716 0.0507 0.0507 0.0388 0.0776
## 2     1      43 M1124 0.0698 0       0.116  0.116  0.0930 0.0465 0      0.0233
## # ... with 12 more variables: K <dbl>, L <dbl>, M <dbl>, N <dbl>, P <dbl>,
## #   Q <dbl>, R <dbl>, S <dbl>, T <dbl>, V <dbl>, W <dbl>, Y <dbl>
```

Check RAW data types

```
is.data.frame(c_m_RAW_AAC)
```

```
## [1] TRUE
```

```
class(c_m_RAW_AAC$Class) # Col 1
```

```
## [1] "numeric"
```

```
class(c_m_RAW_AAC$TotalAA) # Col 2
```

```
## [1] "numeric"
```

```
class(c_m_RAW_AAC$PID) # Col 3
```

```
## [1] "character"
```

```
class(c_m_RAW_AAC$A) # Col 4
```

```
## [1] "numeric"
```

Check RAW dataframe dimensions

```
dim(c_m_RAW_AAC)
```

```
## [1] 2340   23
```

Check RAW for missing values

- **No missing values found.**

```r
apply(is.na(c_m_RAW_AAC), 2, which)
```

```
## integer(0)
```

```r
# sapply(c_m_RAW_AAC, function(x) sum(is.na(x))) # Sum up NA by columns
# c_m_RAW_AAC[rowSums(is.na(c_m_RAW_AAC)) != 0,] # Show rows where NA's is not zero
```

Number of polypeptides per Class:

- Class 0 = Control,
- Class 1 = Myoglobin

```
##
##    0    1
## 1216 1124
```

Numerical summary of RAW data

```
##      Class           TotalAA           PID                 A
##  Min.   :0.0000   Min.   :   2.0   Length:2340        Min.   :0.00000
##  1st Qu.:0.0000   1st Qu.: 109.8   Class :character   1st Qu.:0.05108
##  Median :0.0000   Median : 154.0   Mode  :character   Median :0.07364
##  Mean   :0.4803   Mean   : 353.8                      Mean   :0.07835
##  3rd Qu.:1.0000   3rd Qu.: 407.0                      3rd Qu.:0.10261
##  Max.   :1.0000   Max.   :4660.0                      Max.   :0.28000
##        C                 D                 E                 F
##  Min.   :0.000000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.000000   1st Qu.:0.03401   1st Qu.:0.05435   1st Qu.:0.03801
##  Median :0.007034   Median :0.05195   Median :0.07143   Median :0.04545
##  Mean   :0.011970   Mean   :0.04900   Mean   :0.07451   Mean   :0.05135
##  3rd Qu.:0.020408   3rd Qu.:0.06567   3rd Qu.:0.09091   3rd Qu.:0.05501
##  Max.   :0.159420   Max.   :0.17647   Max.   :0.50000   Max.   :0.37500
##        G                 H                 I                 K
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.04544   1st Qu.:0.01324   1st Qu.:0.04348   1st Qu.:0.05797
##  Median :0.06394   Median :0.02297   Median :0.05992   Median :0.08182
##  Mean   :0.06193   Mean   :0.02890   Mean   :0.06839   Mean   :0.08386
##  3rd Qu.:0.08625   3rd Qu.:0.04095   3rd Qu.:0.08216   3rd Qu.:0.12081
##  Max.   :0.36364   Max.   :0.13333   Max.   :0.50000   Max.   :0.28761
##        L                 M                 N                 P
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.07480   1st Qu.:0.01087   1st Qu.:0.01948   1st Qu.:0.02464
##  Median :0.09136   Median :0.01948   Median :0.04145   Median :0.03401
##  Mean   :0.09313   Mean   :0.01949   Mean   :0.04228   Mean   :0.03825
##  3rd Qu.:0.11688   3rd Qu.:0.02721   3rd Qu.:0.05788   3rd Qu.:0.04772
##  Max.   :0.25000   Max.   :0.11111   Max.   :0.12563   Max.   :0.20635
##        Q                 R                 S                 T
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.02212   1st Qu.:0.01476   1st Qu.:0.04348   1st Qu.:0.03247
##  Median :0.03598   Median :0.03896   Median :0.05564   Median :0.05194
##  Mean   :0.03342   Mean   :0.03818   Mean   :0.06191   Mean   :0.04838
```

```
##  3rd Qu.:0.04545   3rd Qu.:0.05370   3rd Qu.:0.06964   3rd Qu.:0.06522
##  Max.   :0.18182   Max.   :0.24324   Max.   :0.22619   Max.   :0.18750
##        V                 W                 Y
##  Min.   :0.00000   Min.   :0.000000   Min.   :0.00000
##  1st Qu.:0.04575   1st Qu.:0.001899   1st Qu.:0.01463
##  Median :0.05844   Median :0.011492   Median :0.02865
##  Mean   :0.06512   Mean   :0.012327   Mean   :0.03644
##  3rd Qu.:0.07405   3rd Qu.:0.017889   3rd Qu.:0.04564
##  Max.   :0.20000   Max.   :0.133333   Max.   :0.14286
```

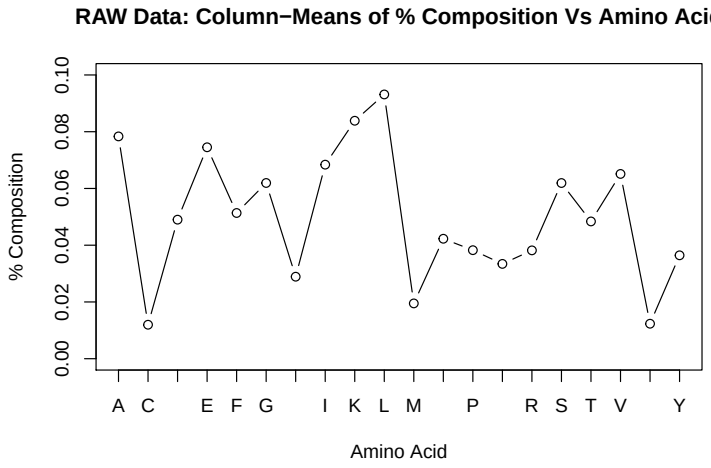
Visualize Descriptive Statistics using RAW Data

Formulas for mean:

$$E[X] = \sum_{i=1}^{n} x_i p_i \quad ; \qquad \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2}$$

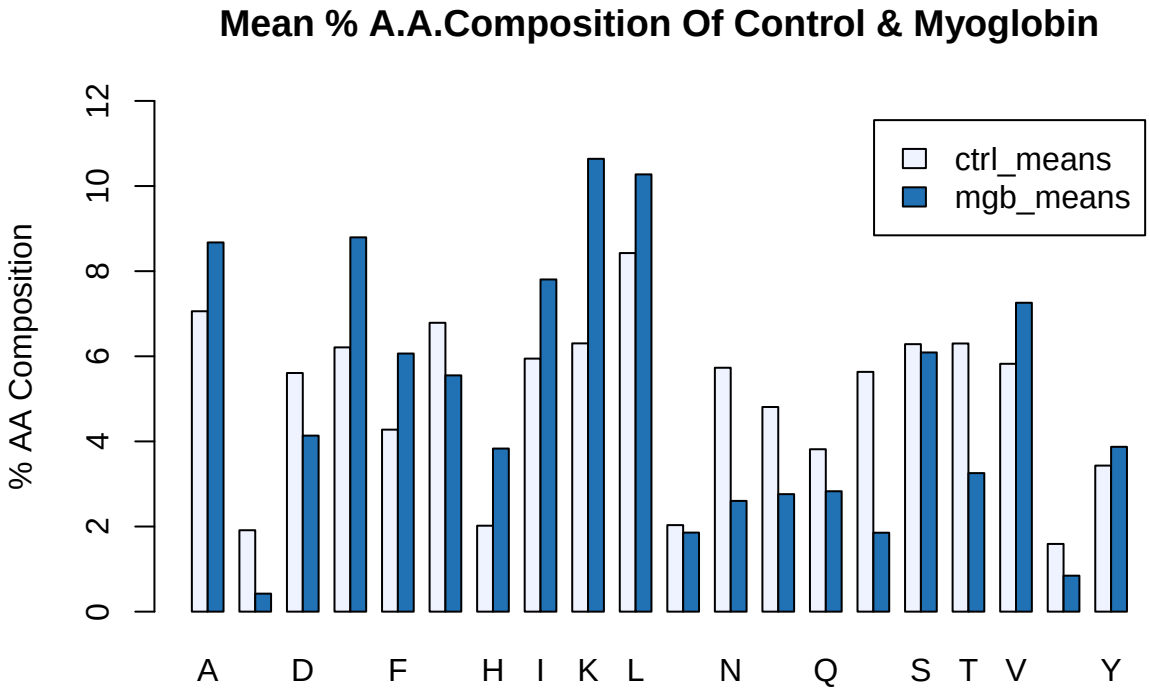
Scatter plot of means of *Myoglobin-Control* amino acid composition of RAW Data

- This Scatter-plot shows the means for each feature (column-means) in the dataset. The means represent the ungrouped or total of all proteins (where n = 2340) versus AA type.

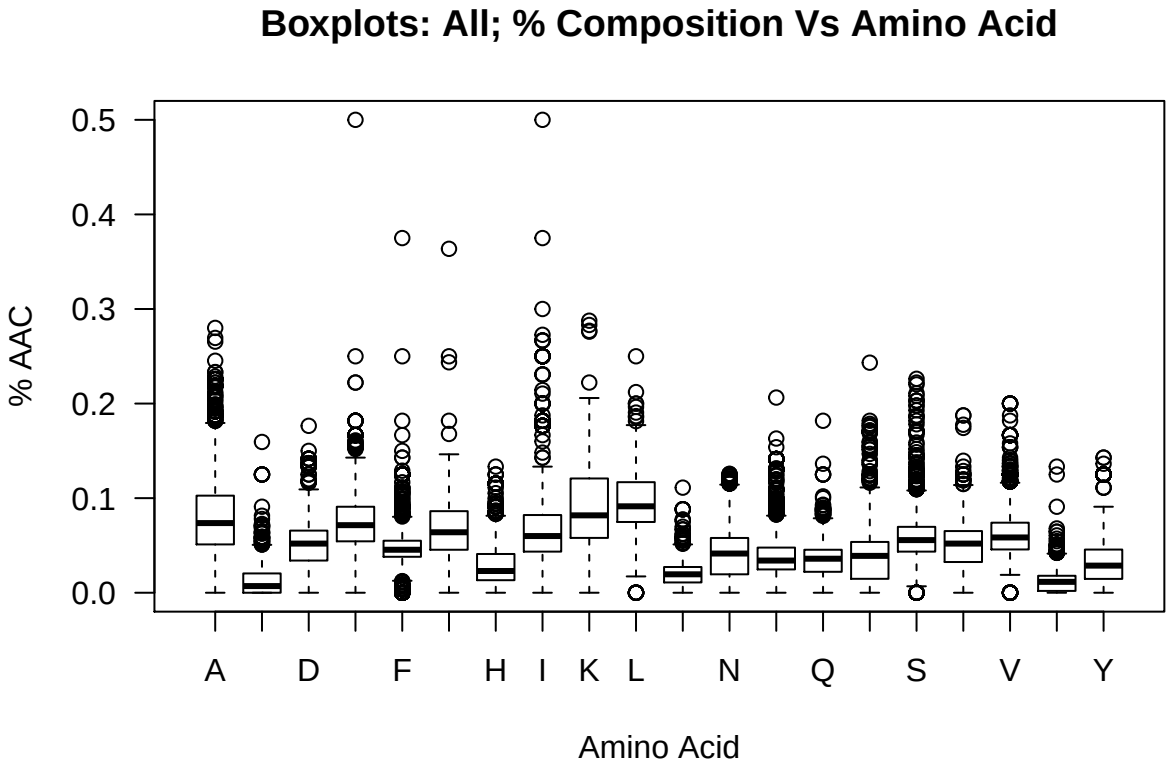



```
# A-4
### Grouped barchart of amino acid vs. protein category
barplot(percent_aa,
        main = "Mean % A.A.Composition Of 3 Protein Groupings",
        ylab = "% AA Composition",
        ylim = c(0, 12),
        col = colorRampPalette(brewer.pal(4, "Blues"))(3),
        legend = T,
        beside = T)
```

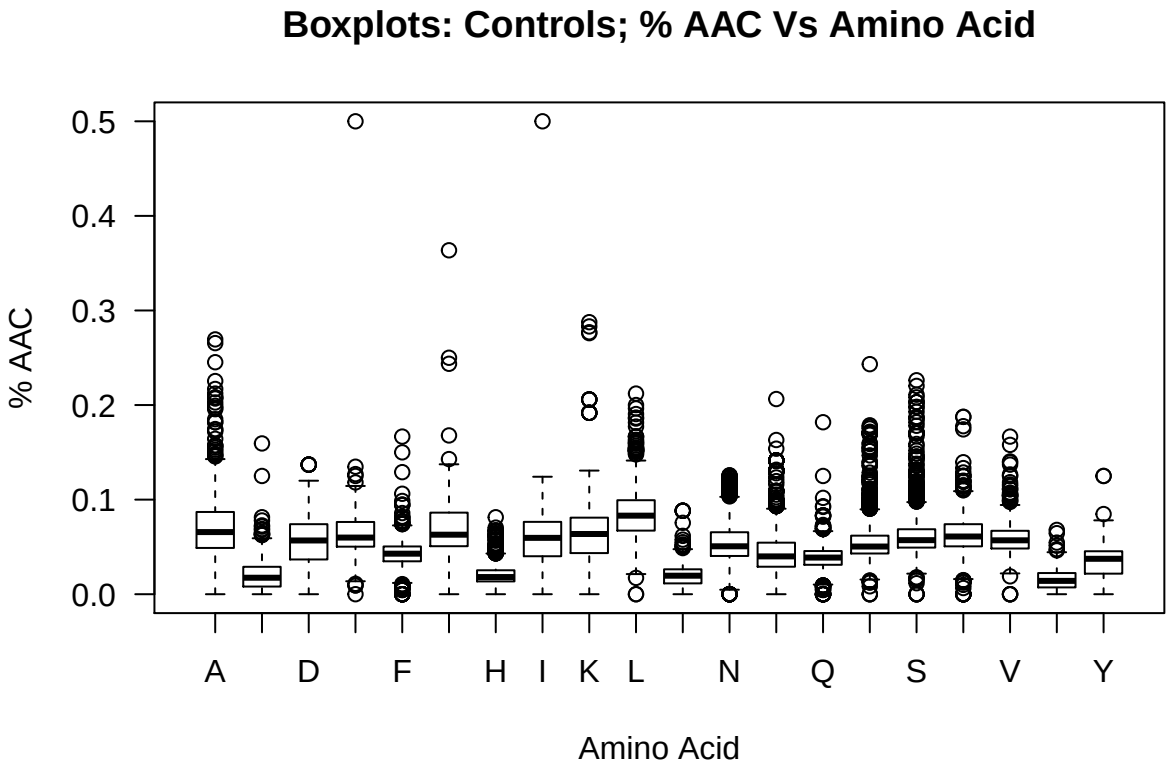
Means of percent amino acid composition of control & myoglobin categories, RAW data



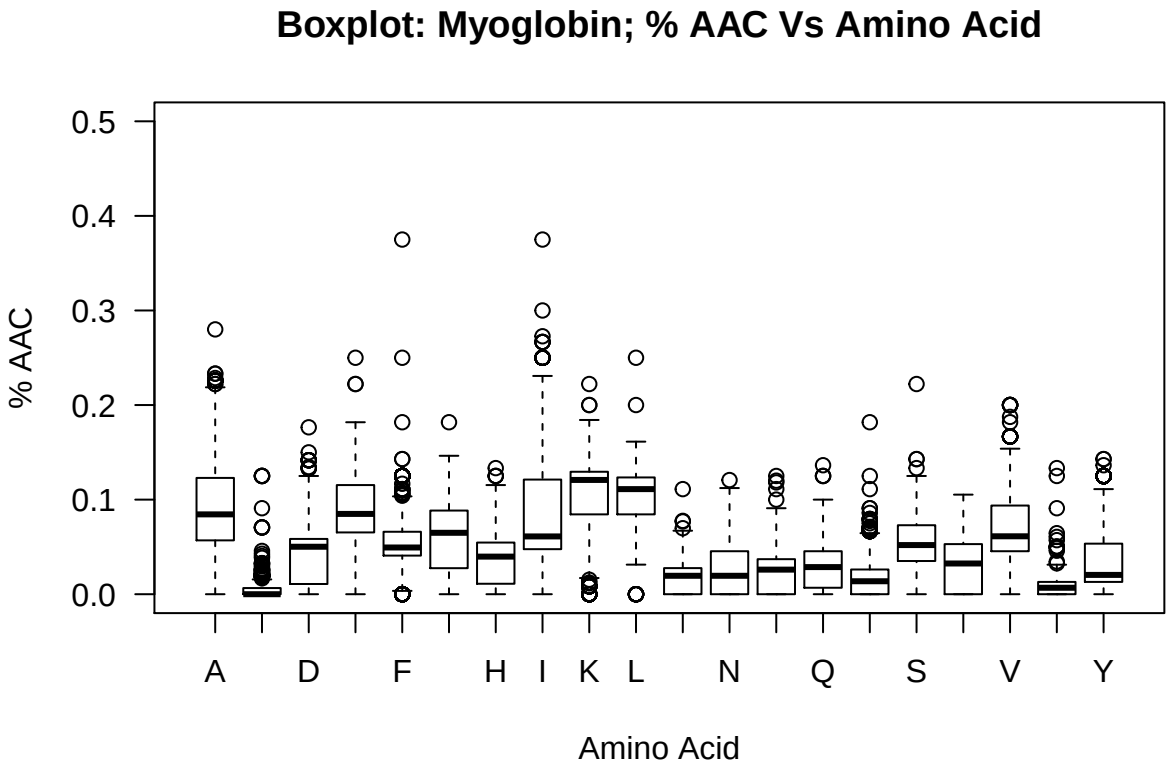
Boxplots of grand-means of overall amino acid composition, RAW data

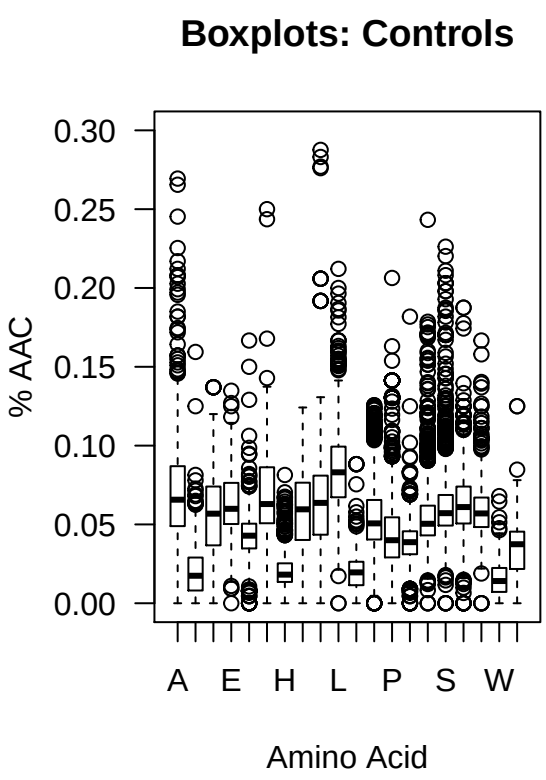


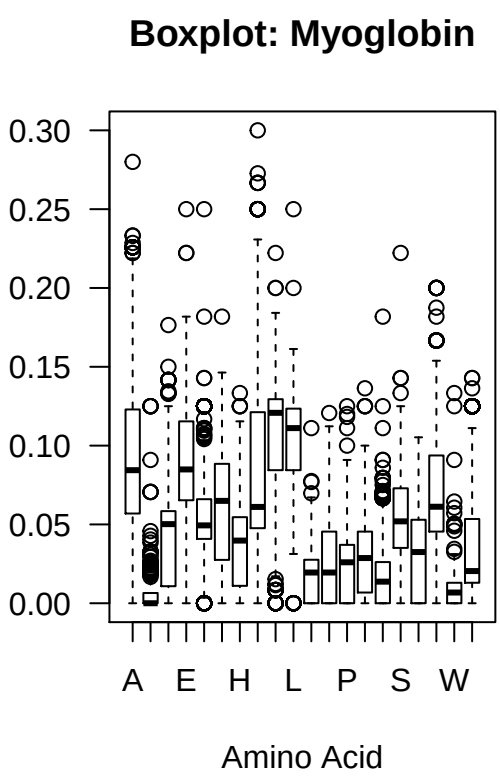
Boxplots of amino acid compositions for control (only), RAW data



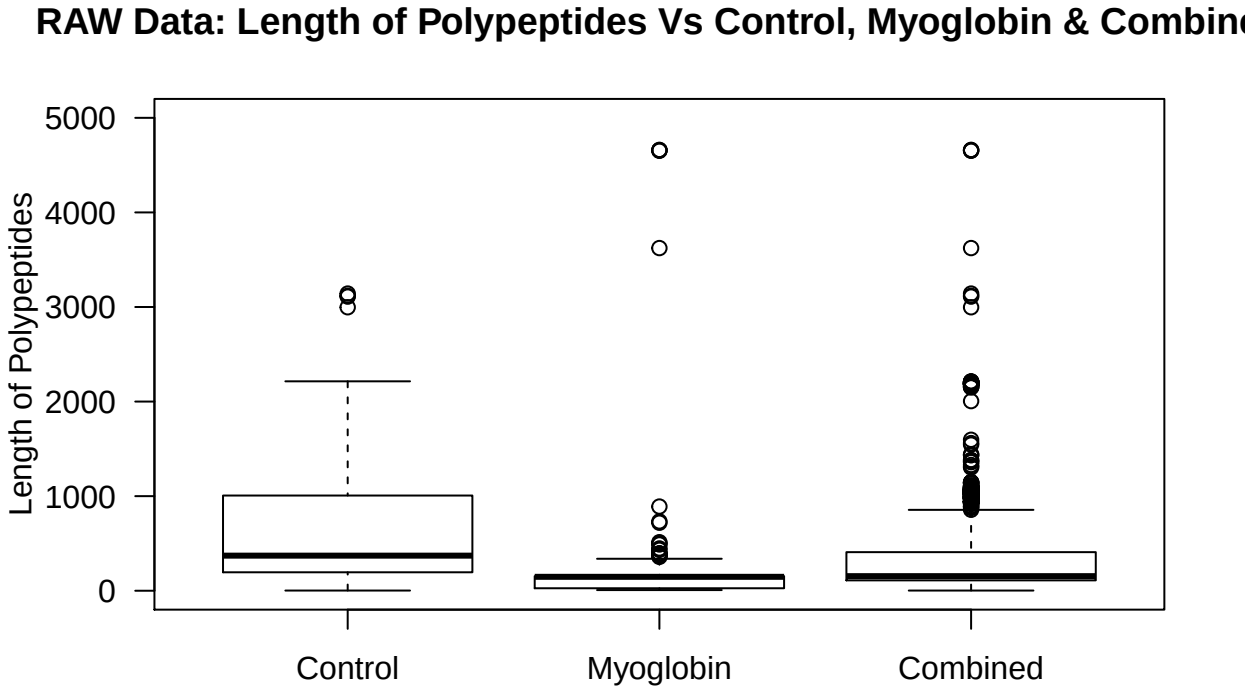
Boxplots of amino acid compositions for myoglobin (only), RAW data




Boxplots: Controls
% AAC
0.30
0.25
0.20
0.15
0.10
0.05
0.00
A E H L P S W
Amino Acid


Boxplot: Myoglobin
0.30
0.25
0.20
0.15
0.10
0.05
0.00
A E H L P S W
Amino Acid

Boxplots Of Length Of Polypeptides For RAW Data

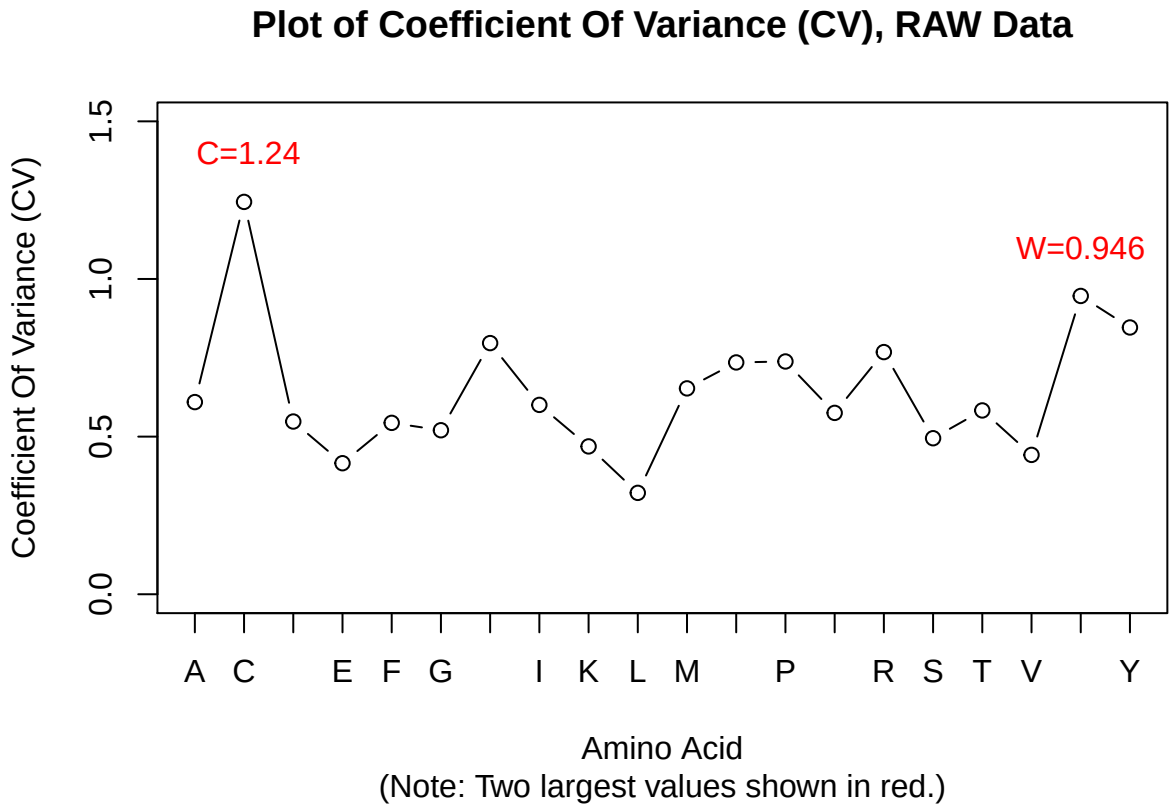


Plot Coefficient Of Variance For RAW Data

Standard deviations are sensitive to scale. Therefore I compare the normalized standard deviations. This normalized standard deviation is more commonly called the coefficient of variation (CV).

$$CV = \frac{\sigma(x)}{E[|x|]} \quad where \quad \sigma(x) \equiv \sqrt{E[x-\mu]^2} \tag{3}$$

$$CV = \frac{1}{\bar{x}} \cdot \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{4}$$




```
AA_var_norm
```

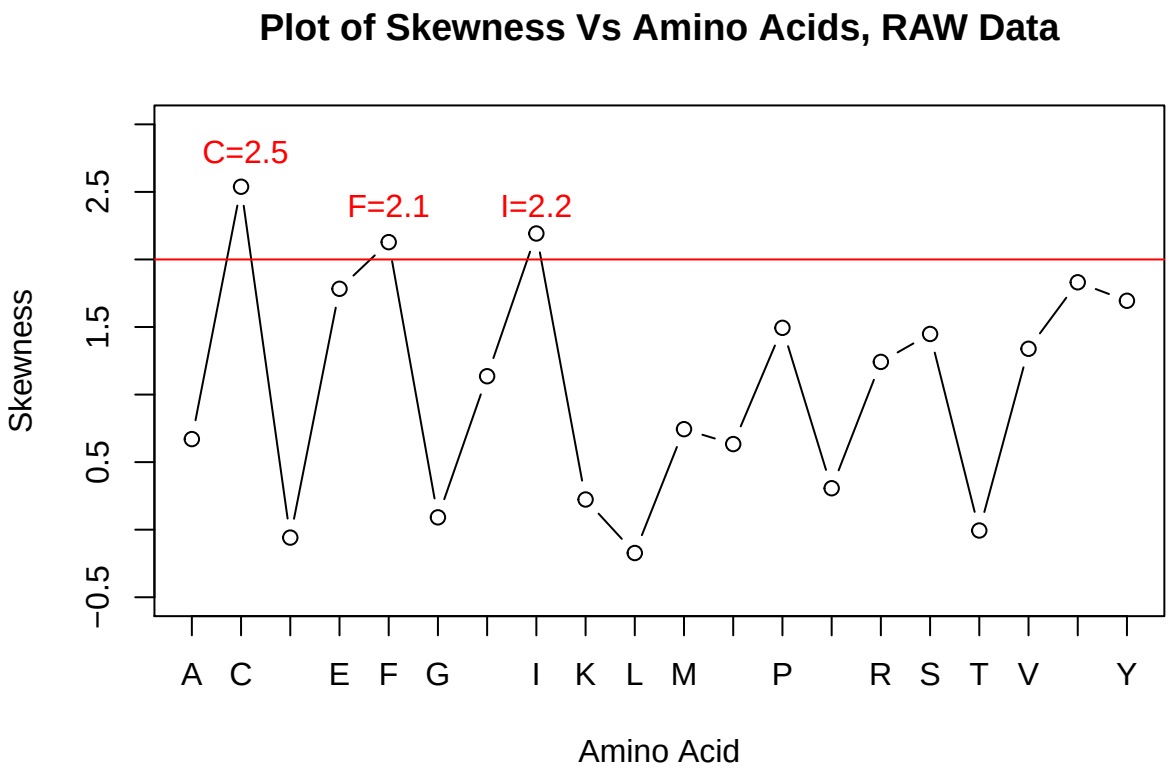
```
##         A         C         D         E         F         G         H         I
## 0.6095112 1.2444944 0.5478540 0.4156102 0.5436243 0.5201625 0.7966296 0.6005962
##         K         L         M         N         P         Q         R         S
## 0.4689544 0.3215591 0.6529752 0.7352478 0.7383244 0.5752622 0.7680977 0.4948690
##         T         V         W         Y
## 0.5830352 0.4420595 0.9461276 0.8461615
```

Skewness of distributions, RAW data

$$Skewness \ = E\left[\left(\frac{X-\mu}{\sigma(x)}\right)^3\right] \quad where \quad \sigma(x) \equiv \sqrt{E[x-\mu]^2} \tag{5}$$

$$Skewness \ = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^3}{\left(\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2}\right)^3} \tag{6}$$

Skewness values for each A.A. are determined in totality.




```
AA_skewness
```

```
##           A           C           D           E           F           G
##  0.670502595 2.538162400 -0.058540442 1.782876260 2.128117638 0.091338300
##           H           I           K           L           M           N
##  1.135783661 2.192145038 0.223433207 -0.172566877 0.744002991 0.633532783
##           P           Q           R           S           T           V
##  1.493903282 0.306716333 1.241930812 1.448521897 -0.006075043 1.338971930
##           W           Y
##  1.831047440 1.694362388
```

Determine coefficients of correlation, RAW data

An easily interpretable test is a correlation 2D-plot for investigating multicollinearity or feature reduction. Fewer attributes "means decreased computational time and complexity. Secondly, if two predictors are highly correlated, this implies that they measure the same underlying information. Removing one should not compromise the performance of the model and might lead to a more parsimonious and interpretable model. Third, some models can be crippled by predictors with degenerate distributions." [7]

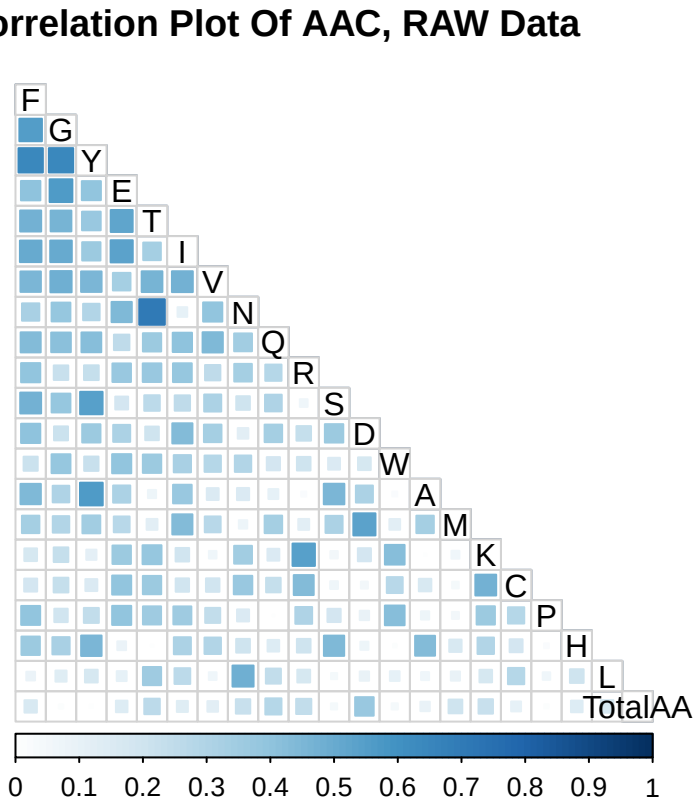
Pearson's correlation coefficient:

$$\rho_{x,y} = \frac{E\left[(X-\mu_x)(X-\mu_y)\right]}{\sigma_x \sigma_y} \tag{7}$$

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_1-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}} \tag{8}$$

[7] Max Kuhn and Kjell Johnson, Applied Predictive Modeling, Springer Publishing, 2018, P.43

```
c_m_corr_mat <- cor(c_m_RAW_AAC[, c(2, 4:23)], method = "p") # "p": Pearson test for continous variable

corrplot(abs(c_m_corr_mat),
         title = "Correlation Plot Of AAC, RAW Data",
         method = "square",
         type = "lower",
         tl.pos = "d",
         cl.lim = c(0, 1),
         addgrid.col = "lightgrey",
         cl.pos = "b",                    # Color legend position bottom.
         order = "FPC",                   # "FPC" = first principal component order.
         mar = c(1, 2, 1, 2),
         tl.col = "black")
```



NOTE: Amino acids shown in First Principal Component order, top to bottom.

1. Maximum value of Correlation between T & N.

```
## [1] 0.7098085
```

2. According to Max Kuhn[8], correlation coefficients need only be addressed if the |R| >= 0.75.
3. Therefore is **no reason to consider multicollinearity**.

[8]Max Kuhn and Kjell Johnson, Applied Predictive Modeling, Springer Publishing, 2018, P.47 (http://appliedpredictivemodeling.com/)

Boruta Random Forest Test, RAW data

> It finds relevant features by comparing original attributes' importance with importance achievable at random, estimated using their permuted copies (shadows).
>
> Miron Kursa [9]

```
c_m_class_20 <- c_m_RAW_AAC[, -c(2, 3)] # Remove TotalAA & PID
Class <- as.factor(c_m_class_20$Class) # Convert 'Class' To Factor
```

NOTE: $mcAdj = TRUE$, If True, multiple comparisons will be adjusted using the Bonferroni method to calculate p-values. Therefore, $p_i \leq \frac{\alpha}{m}$ where α is the desired p-value and m is the total number of null hypotheses.

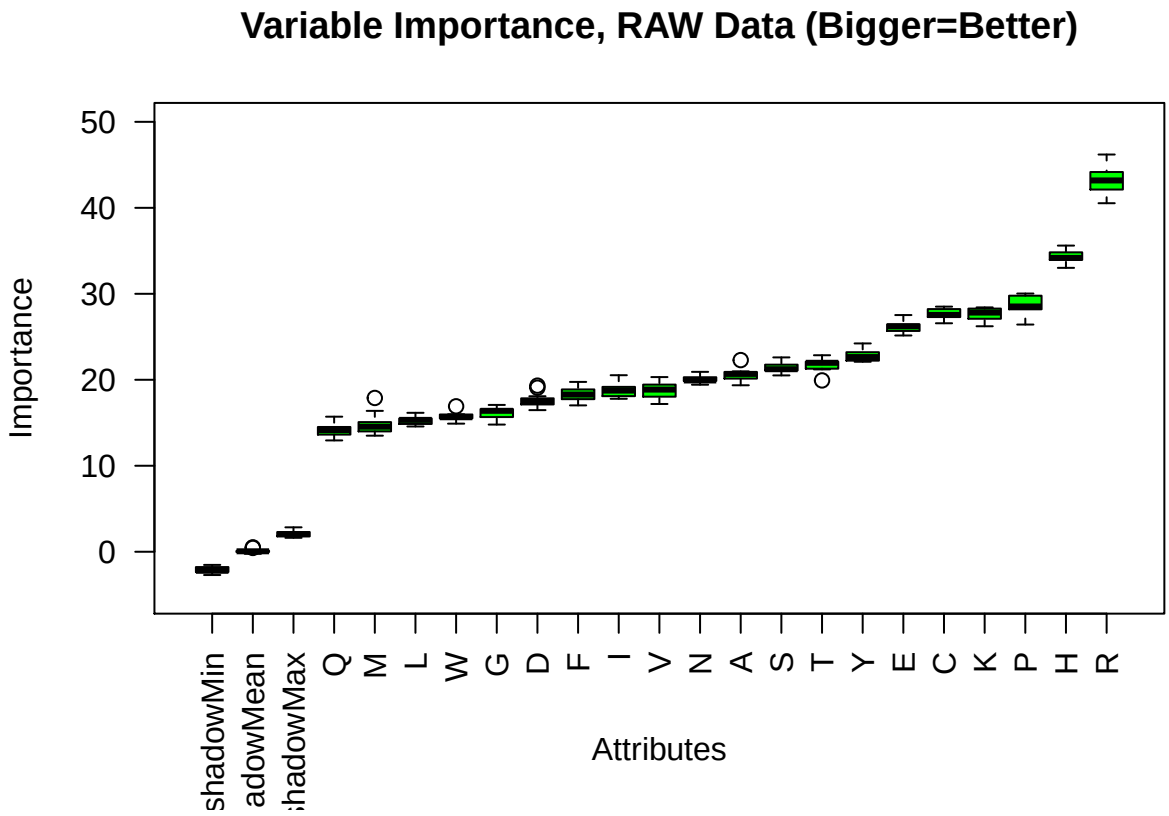
```
set.seed(1000)
#registerDoMC(cores = 3) # Start multi-processor mode
start_time <- Sys.time() # Start timer

boruta_output <- Boruta(Class ~ .,
                        data = c_m_class_20[, -1],
                        mcAdj = TRUE, # See Note above.
                        doTrace = 1) # doTrace = 1, represents non-verbose mode.

#registerDoSEQ() # Stop multi-processor mode
end_time <- Sys.time() # End timer
end_time - start_time # Display elapsed time
```

```
## Time difference of 31.45118 secs
```

```
names(boruta_output)
```

[9] https://notabug.org/mbq/Boruta/

Plot variable importance, RAW Data



Variable importance scores, RAW Data

```
## Warning in TentativeRoughFix(boruta_output): There are no Tentative attributes!
## Returning original object.
```

	meanImp	decision
R	43.18824	Confirmed
H	34.29757	Confirmed
P	28.70225	Confirmed
C	27.67710	Confirmed
K	27.60808	Confirmed
E	26.18884	Confirmed
Y	22.85337	Confirmed
T	21.67689	Confirmed
S	21.43716	Confirmed
A	20.53089	Confirmed
N	20.09681	Confirmed
V	18.77054	Confirmed
I	18.76492	Confirmed
F	18.31240	Confirmed
D	17.64592	Confirmed
G	16.15461	Confirmed
W	15.74107	Confirmed
L	15.27767	Confirmed
M	14.82861	Confirmed
Q	14.13939	Confirmed

Conclusion for Boruta random forest test, RAW Data

- All features are essential. None should be dropped.

Conclusions For EDA, RAW data

Three amino acids (C, F, I) from the single amino acid percent composition were deemed problematic due to their skewness were greater than 2.0. This suggests that a transformation should be carried out to rectify this issue.

Protein	Skewness
C, Cysteine	2.538162
F, Phenolalanine	2.128118
I, Isoleucine	2.192145

Analysis of TRANSFORMED data

This EDA section is a reevaluation square root transformed, `c_m_RAW_ACC.csv` data set, hence called `c_m_TRANSFORMED.csv`.

The $\sqrt{x_i}$ *Transformed* data is derived from `c_m_RAW_ACC.csv` where the amino acids C, F, I were transformed using a square root function. This transformation was done to reduce the skewness of these samples and avoid modeling problems arising from high skewness, as seen below.

Amino Acid	Initial skewness	Skew After Square-Root Transformation
C, Cysteine	2.538162	0.3478132
F, Phenolalanine	2.128118	-0.102739
I, Isoleucine	2.192145	0.2934749

```
# Import Transformed data
c_m_TRANSFORMED <- read_csv("./00-data/02-aac_dpc_values/c_m_TRANSFORMED.csv")
Class <- as.factor(c_m_TRANSFORMED$Class)
```

Check Transformed dataframe dimensions

```
dim(c_m_TRANSFORMED)
```

```
## [1] 2340   23
```

Check Transformed for missing values

```
apply(is.na(c_m_TRANSFORMED), 2, which)
```

```
## integer(0)
```

- No missing values found.

Count Transformed data for the number of polypeptides per class

Number of polypeptides per Class:

- Class 0 = Control,
- Class 1 = Myoglobin

```
##
##    0    1
## 1216 1124
```

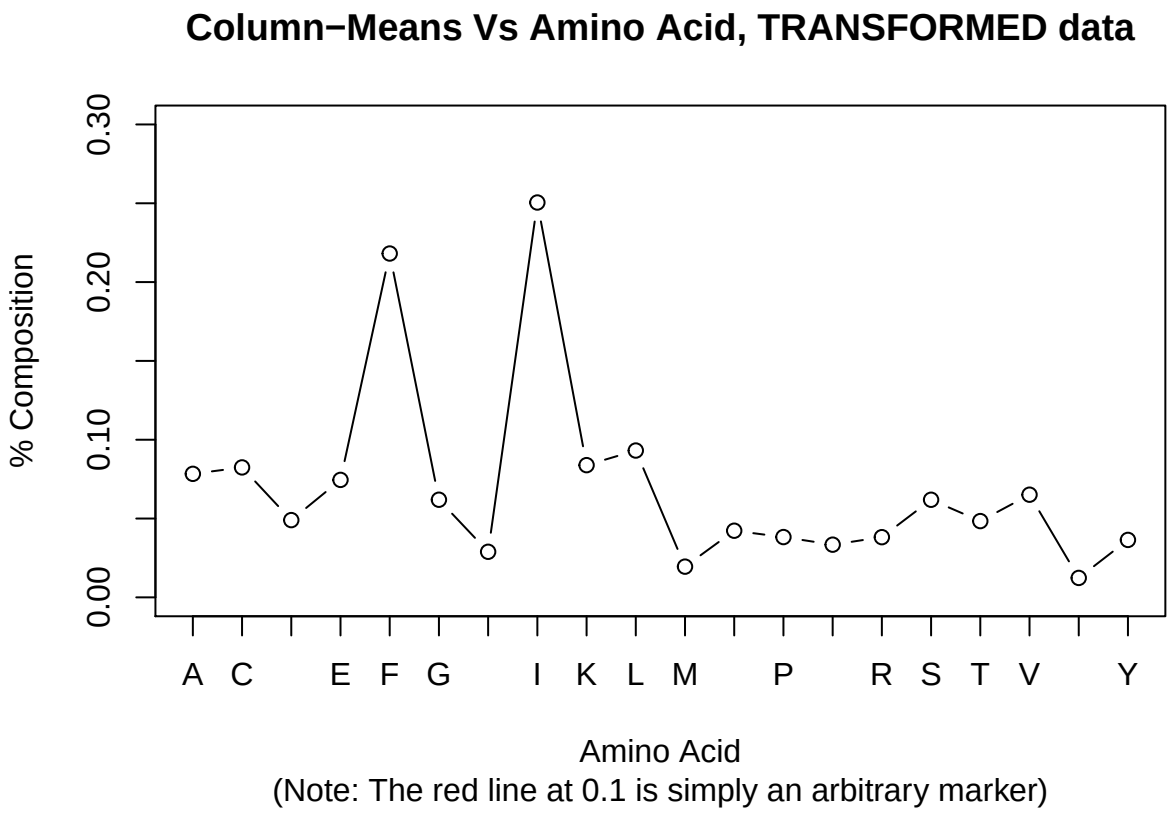
Visualization Descriptive Statistics, TRANSFORMED data

Formulas for mean:

$$E[X] = \sum_{i=1}^{n} x_i p_i \quad ; \qquad \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{9}$$

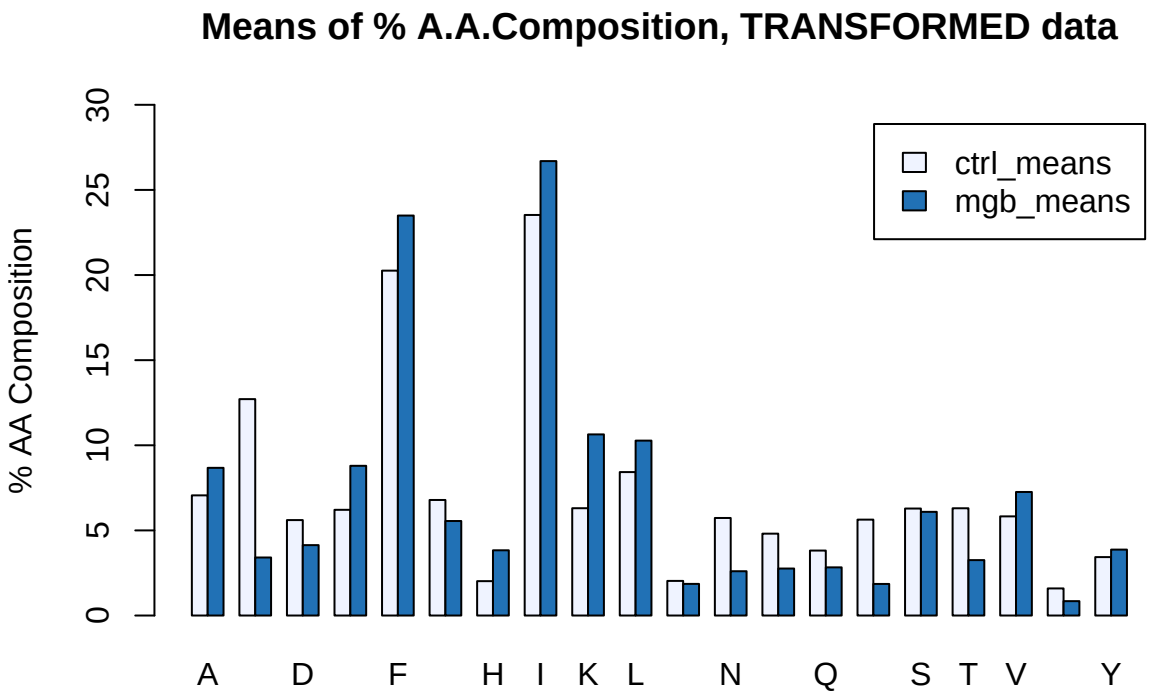
Scatter plot of means of *Myoglobin-Control* amino acid composition $\sqrt{x_i}$, TRANSFORMED data

- This plot shows the means for each feature (column-means) in the dataset. The means represent the ungrouped or total of all proteins (where n=2340) versus AA type.

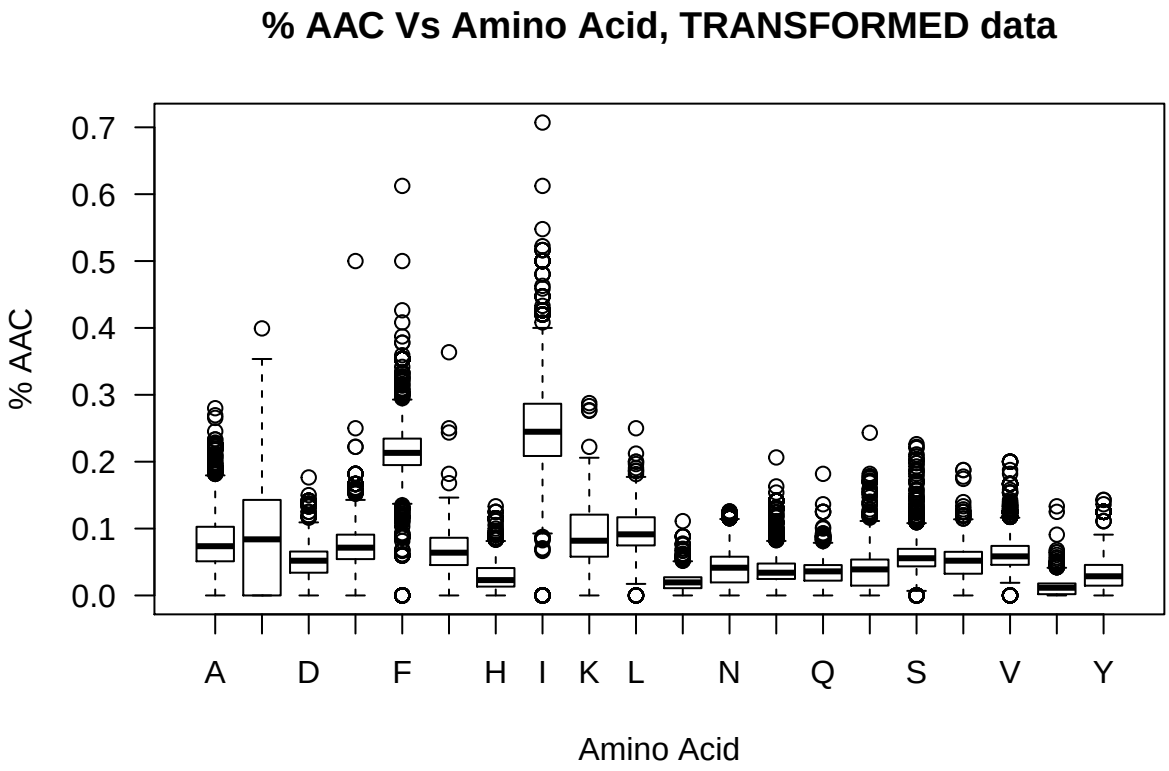



```
# A-4
## Grouped barchart of $\sqrt x_i$ Transformed amino acid vs.
## protein category data
barplot(percent_aa,
        main = "Mean % A.A.Composition, TRANSFORMED data",
        ylab = "% AA Composition",
        ylim = c(0, 30),
        col = colorRampPalette(brewer.pal(4, "Blues"))(3),
        legend = T,
        beside = T)
```

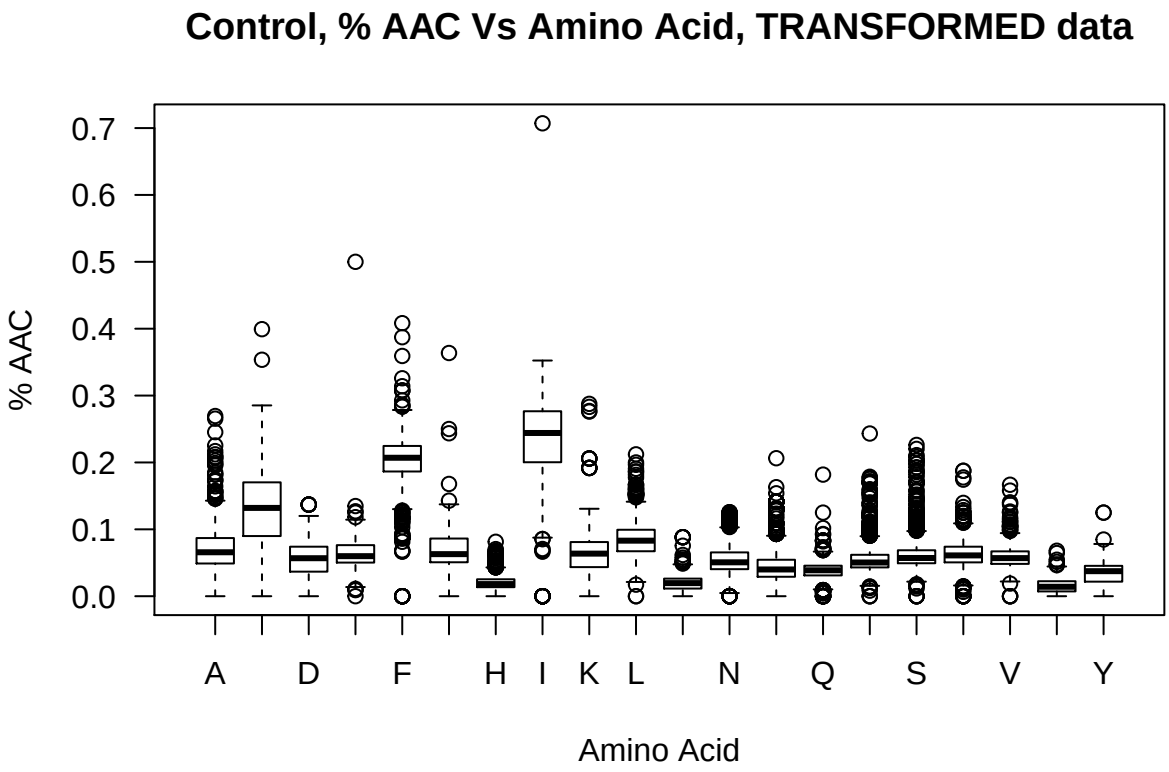
Grouped bar chart of means for percent amino acid composition of Transformed Data; control & myoglobin categories



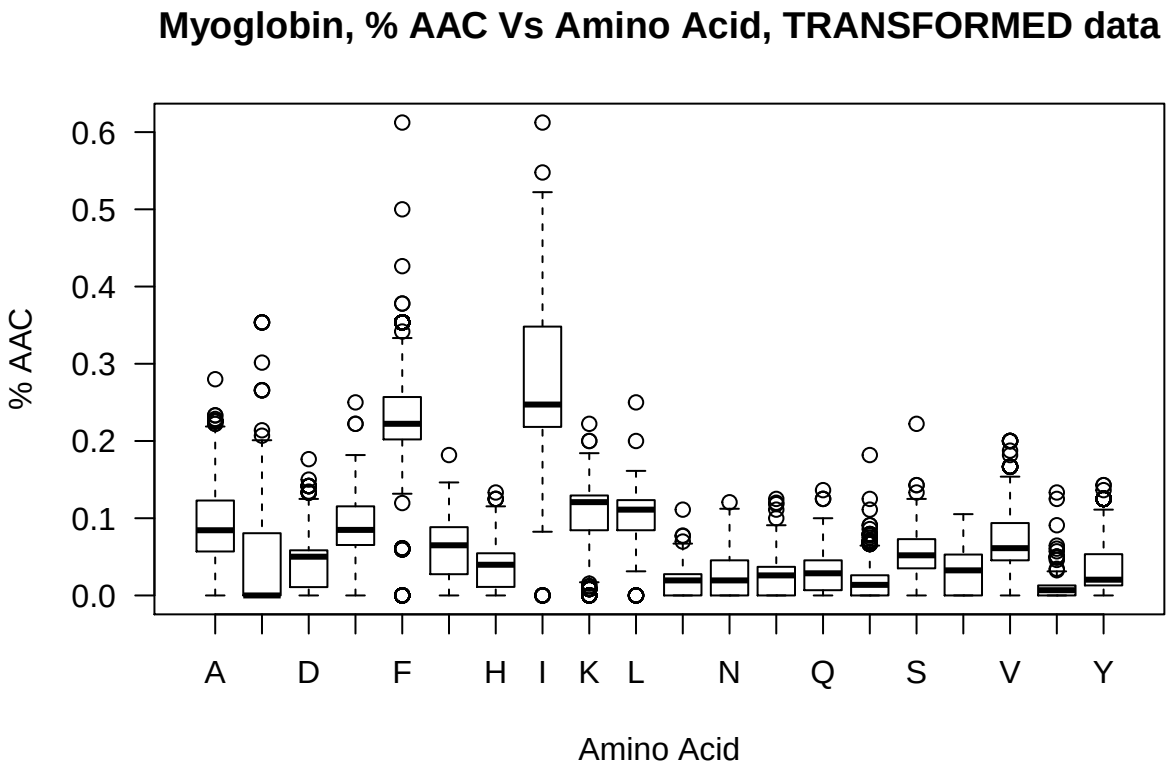
Boxplots of grand-means of the overall amino acid composition of square-root transformed data



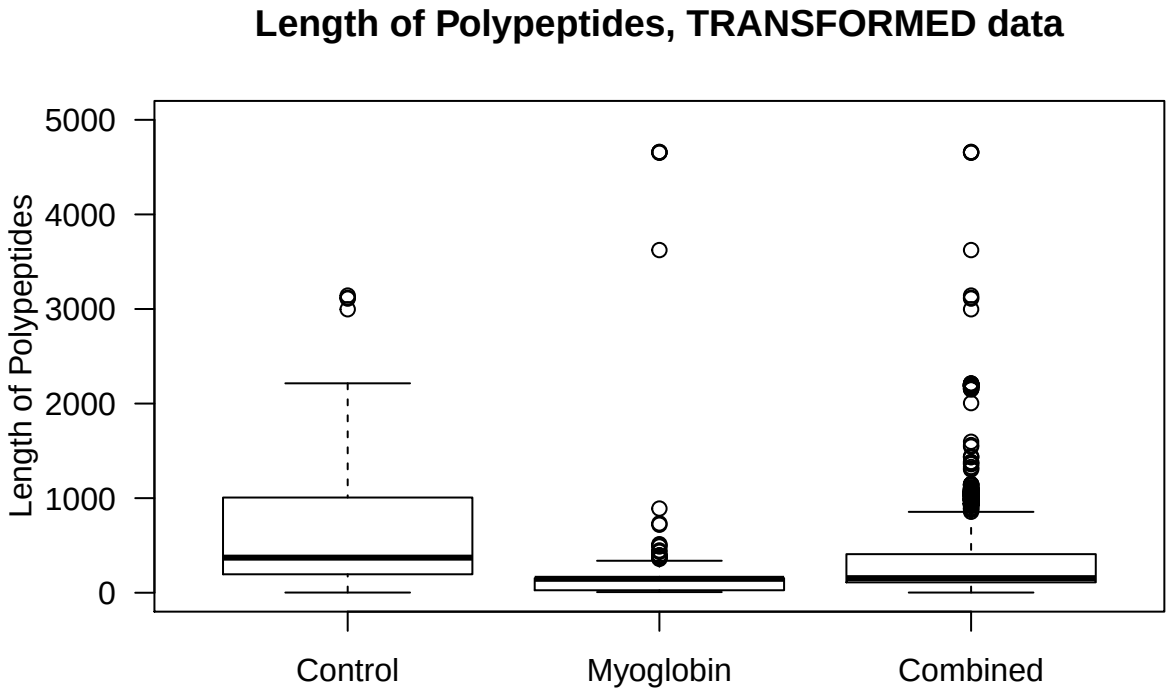
Boxplots of amino acid compositions for control (only) of square-root transformed data



Boxplots of amino acid compositions for myoglobin of square-root transformed Data(only), TRANSFORMED data



Boxplots Of Length Of Polypeptides Of Transformed Data; Myoglobin, Control & Combined



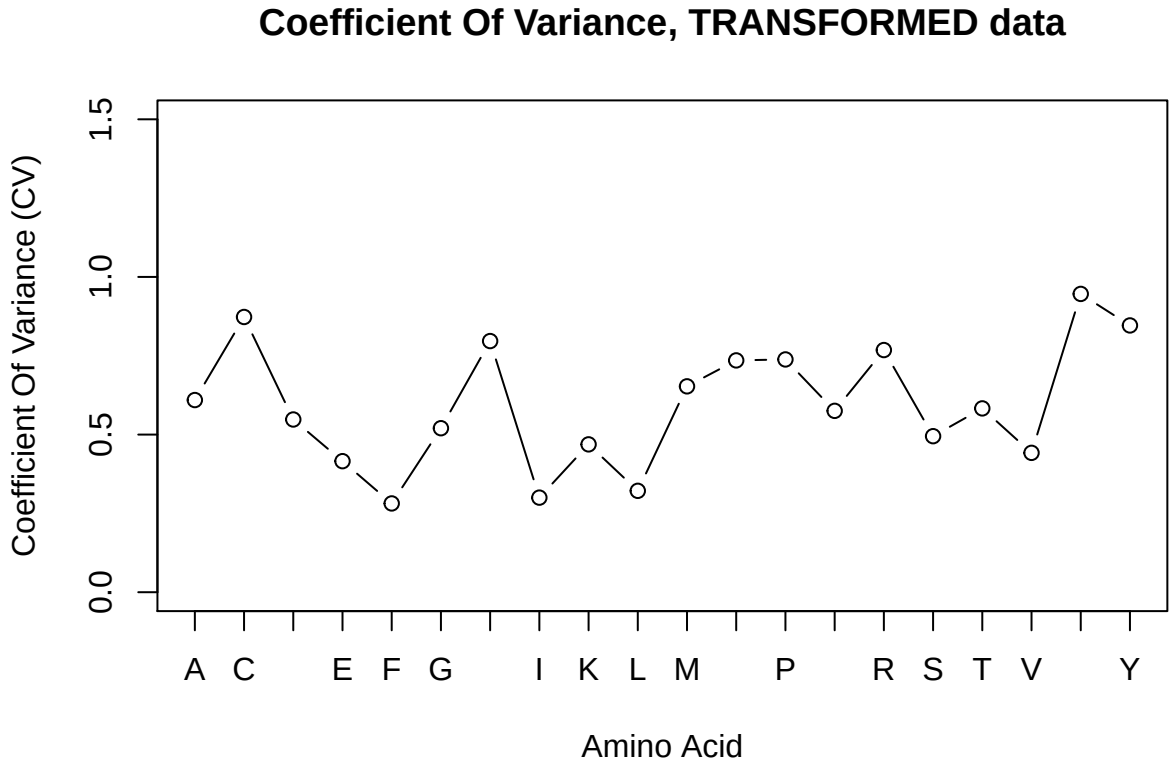
Coefficient of Variance (CV), TRANSFORMED data

Standard deviations are sensitive to scale. Therefore I compare the normalized standard deviations. This normalized standard deviation is more commonly called the coefficient of variation (CV).

$$CV = \frac{\sigma(x)}{E[|x|]} \quad where \quad \sigma(x) \equiv \sqrt{E[x-\mu]^2} \tag{10}$$

$$CV = \frac{1}{\bar{x}} \cdot \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{11}$$

Plot of Coefficient Of Variance (CV)




```
AA_var_norm
```

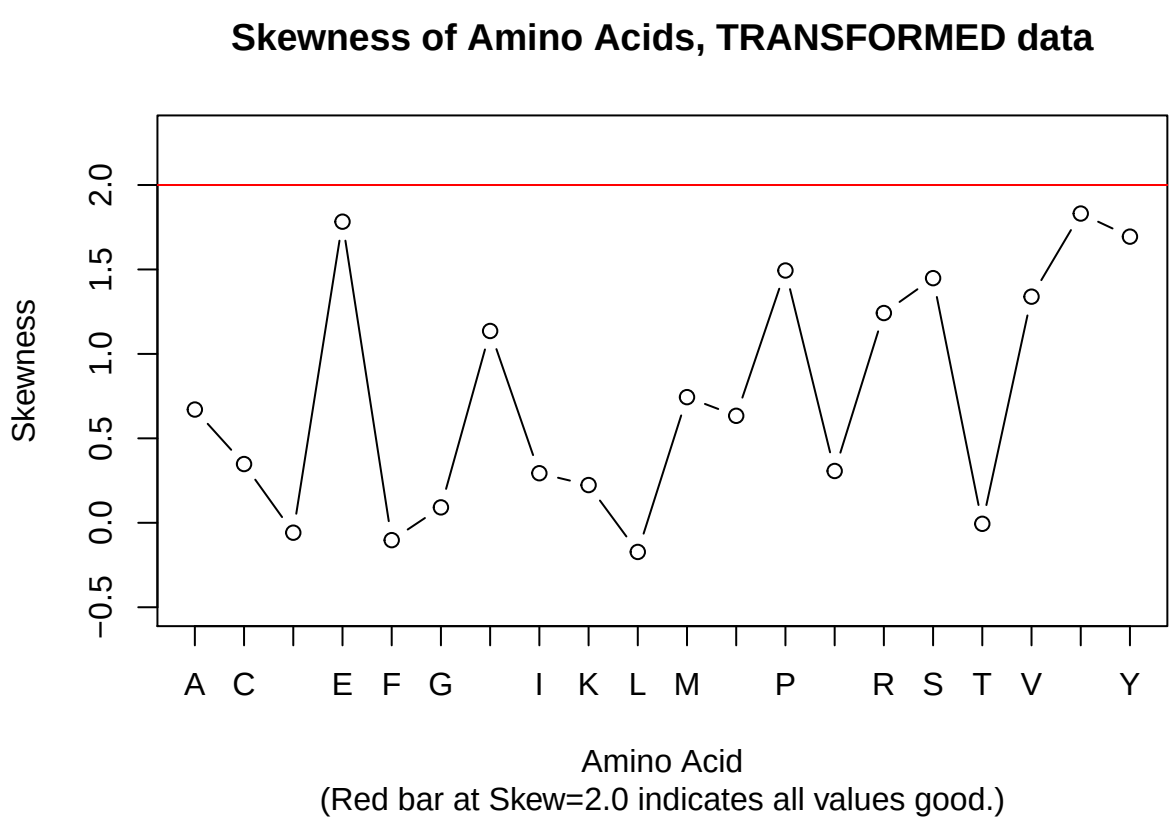
```
##         A         C         D         E         F         G         H         I
## 0.6095112 0.8729758 0.5478540 0.4156102 0.2815745 0.5201625 0.7966296 0.2999687
##         K         L         M         N         P         Q         R         S
## 0.4689544 0.3215591 0.6529752 0.7352478 0.7383244 0.5752622 0.7680977 0.4948690
##         T         V         W         Y
## 0.5830352 0.4420595 0.9461276 0.8461615
```

Skewness of distributions, TRANSFORMED data

$$Skewness \ = E\left[\left(\frac{X-\mu}{\sigma(x)}\right)^3\right] \quad where \quad \sigma(x) \equiv \sqrt{E[x-\mu]^2} \tag{12}$$

$$Skewness \ = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^3}{\left(\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2}\right)^3} \tag{13}$$

- Skewness values for each A.A. by Class of square-root transformed data




```
AA_skewness
```

```
##           A           C           D           E           F           G
##  0.670502595  0.347813248 -0.058540442  1.782876260 -0.102739748  0.091338300
##           H           I           K           L           M           N
##  1.135783661  0.293474879  0.223433207 -0.172566877  0.744002991  0.633532783
##           P           Q           R           S           T           V
##  1.493903282  0.306716333  1.241930812  1.448521897 -0.006075043  1.338971930
##           W           Y
##  1.831047440  1.694362388
```

Determine coefficients of correlation, TRANSFORMED data

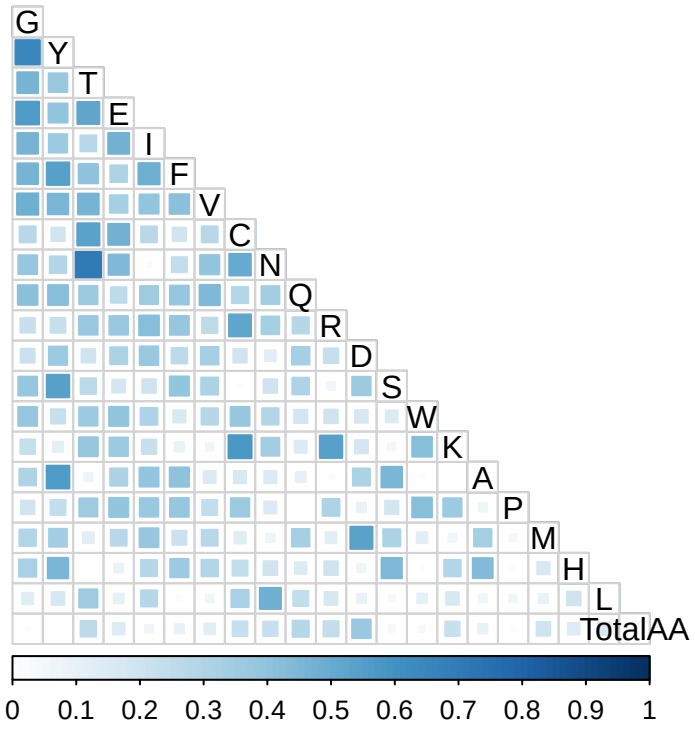
An easily interpretable test is a correlation 2D-plot for investigating multicollinearity or feature reduction. Fewer attributes "means decreased computational time and complexity. Secondly, if two predictors are highly correlated, this implies that they measure the same underlying information. Removing one should not compromise the performance of the model and might lead to a more parsimonious and interpretable model. Third, some models can be crippled by predictors with degenerate distributions." [10]

Pearson's correlation coefficient:

$$\rho_{x,y} = \frac{E\left[(X-\mu_x)(X-\mu_y)\right]}{\sigma_x \sigma_y} \tag{14}$$

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_1-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}} \tag{15}$$

[10] Max Kuhn and Kjell Johnson, Applied Predictive Modeling, Springer Publishing, 2018




```
c_m_corr_mat["T", "N"]
```

```
## [1] 0.7098085
```

No values in the correlation matrix meet the 0.75 cut off criteria for problems.

Boruta - Dimensionality Reduction, TRANSFORMED data

Perform Boruta search

NOTE: $mcAdj = TRUE$: If True, multiple comparisons will be adjusted using the Bonferroni method to calculate p-values. Therefore, $p_i \leq \frac{\alpha}{m}$ where α is the desired p-value and m is the total number of null hypotheses.

```
set.seed(1000)
#registerDoMC(cores = 3) # Start multi-processor mode
start_time <- Sys.time() # Start timer

boruta_output <- Boruta(Class ~ .,
                        data = c_m_class_20[, -1],
                        mcAdj = TRUE, # See Note above.
                        doTrace = 1) # doTrace = 1, represents non-verbose mode.
```

```
## After 11 iterations, +32 secs:
```

```
##  confirmed 20 attributes: A, C, D, E, F and 15 more;
```

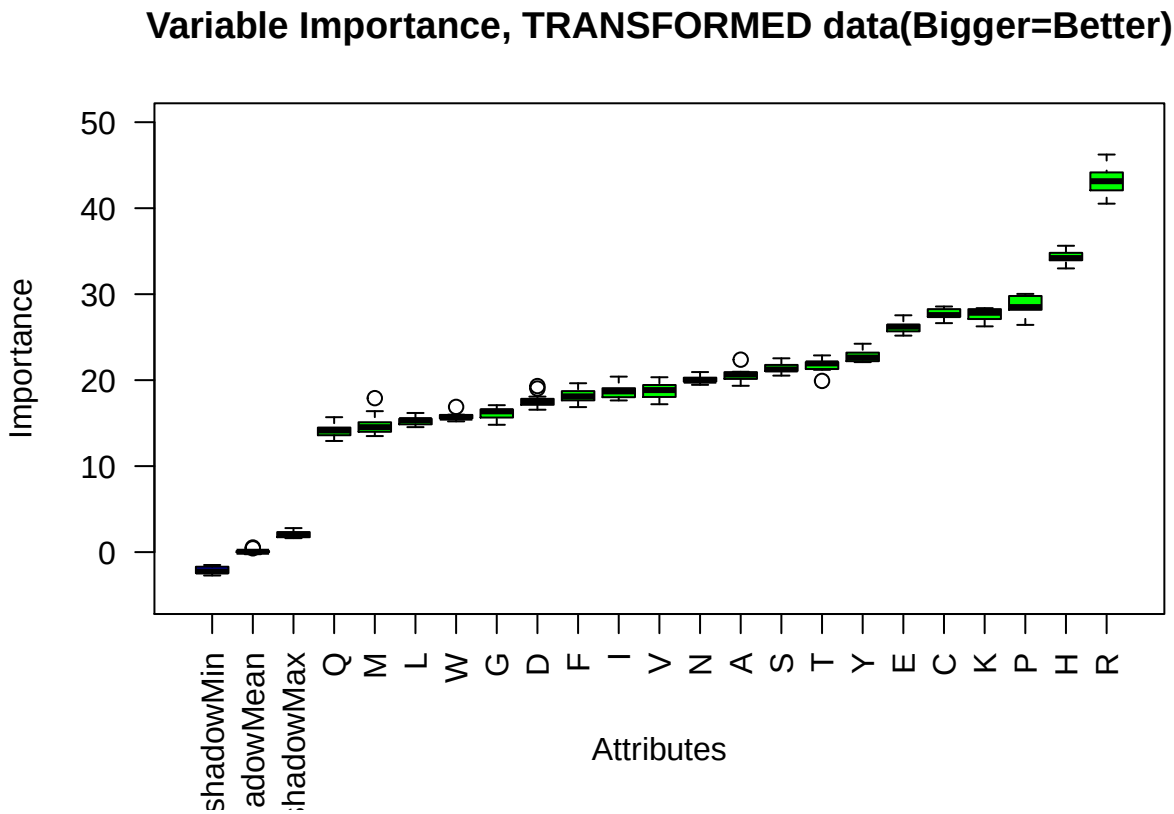
```
##  no more attributes left.
```

```
#registerDoSEQ() # Stop multi-processor mode
end_time <- Sys.time() # End timer
end_time - start_time # Display elapsed time
```

```
## Time difference of 31.72578 secs
```

Plot Variable Importance, TRANSFORMED data

```
plot(boruta_output,
     cex.axis = 1,
     las = 2,
     ylim = c(-5, 50),
     main = "Variable Importance, TRANSFORMED data(Bigger=Better)")
```



Variable Importance Scores, TRANSFORMED data

```
roughFixMod <- TentativeRoughFix(boruta_output)
```

```
## Warning in TentativeRoughFix(boruta_output): There are no Tentative attributes!
## Returning original object.
```

```
imps <- attStats(roughFixMod)
imps2 <- imps[imps$decision != "Rejected", c("meanImp", "decision")]
meanImps <- imps2[order(-imps2$meanImp), ] # descending sort

kable(meanImps) %>% kable_styling(bootstrap_options = c("striped", "hover"))
```

	meanImp	decision
R	43.17613	Confirmed
H	34.30370	Confirmed
P	28.70674	Confirmed
C	27.72357	Confirmed
K	27.60838	Confirmed
E	26.18872	Confirmed
Y	22.84975	Confirmed
T	21.66359	Confirmed
S	21.44119	Confirmed
A	20.54316	Confirmed
N	20.10100	Confirmed
V	18.77068	Confirmed
I	18.69155	Confirmed
F	18.18632	Confirmed
D	17.64435	Confirmed
G	16.15207	Confirmed
W	15.77085	Confirmed
L	15.27614	Confirmed
M	14.83421	Confirmed
Q	14.12976	Confirmed

```
# knitr::kable(meanImps,
# full_width = F,
# position = "left",
# caption = "Mean Importance Scores & Decision, TRANSFORMED data")
```

The *Boruta Random Rorest test* shows that all features are essential therefore none should be dropped from TRANSFORMED data.

EDA Conclusion

Feature Selection & Extraction

It was determined early on that three amino acids (C, F, I) from the data amino acid percent compositions (c_m_RAW_AAC.csv) had Skewness greater than two. It was found that tranforming the features using the square root function lowered the skewness to {-0.102739 $\leq$ skew after transformation $\leq$ 0.3478132}.

Table 7.1, Skewness Before And After Square-Root Transform

Amino Acid	Initial Skewness	Skew After Square-Root Transform
C, Cysteine	2.538162	0.347813248
F, Phenolalanine	2.128118	-0.102739748
I, Isoleucine	2.192145	0.293474879

The transformations of the three amino acids (C, F, I) did not appriciably change the Correlation coefficient, R. Therefore no R values were above 0.75 before or after testing. The highest coeffiecient of correlation being Threonine and Argnine with an R of 0.7098. This indicates that no features are collinear. Therefore the transformed data is used throughout this experiment.

Information Block**

How to: Dimension Reduction using High Correlation

How to reduce features given high correlation (|R| >= 0.75) {-}

1. Calculate the correlation matrix of the predictors.
2. If the correlation plot produced of any two variables is greater than or equal to (|R| >= 0.75), then we could consider feature elimination. This interesting heuristic approach would be used for determining which feature to eliminate.[11]
3. Determine if the two predictors associated with the most significant absolute pairwise correlation (R > |0.75|), call them predictors A and B.
4. Determine the average correlation between A and the other variables. Do the same for predictor B.
5. If A has a more significant average correlation, remove it; otherwise, remove predictor B.
6. Repeat Steps 2–4 until no absolute correlations are above the threshold.

An alternative test for variable importance carried out is called Boruta. Boruta builds Random Forests then "finds relevant features by comparing original attributes' importance with importance achievable at random." [12]

Boruta is used for dimensionality reduction of the **c__m__Transformed data**. Bortua showed that all dependent features are essential for the generation of a Random Forest Decision Tree. It would wise to keep all features for that model test and throughout the generation of other models. All features have decisive mean importance, which is generated by a Gini calculation.

[11] Max Kuhn and Kjell Johnson, Applied Predictive Modeling, Springer Publishing, 2018, (http://appliedpredictivemodeling.com/)

[12] https://notabug.org/mbq/Boruta/

Principle Component Analysis of A Binary Classification System

mcc

3/13/2020

Principle Component Analysis of A Binary Classification System

Introduction

This chapter describes the use and functional understanding of Principle Component Analysis (PCA). PCA is very popular and commonly used during the early phases of model development to provide information on variance. In particular, PCA is a transformative process that orders and maximizes variances found within a dataset.

> The primary goal of principal components analysis is to reveal the hidden structure in a dataset. In so doing, we may be able to; [1]
>
> 1. identify how different variables work together to create the dynamics of the system,
> 2. reduce the dimensionality of the data,
> 3. decrease redundancy in the data,
> 4. filter some of the noise in the data,
> 5. compress the data,
> 6. prepare the data for further analysis using other techniques.

Advantages Of Using PCA Include

1. PCA preserves the global structure among the data points,
2. It is efficiently applied to large data sets,
3. PCA may provide information on the importance of features found in the original datasets.

Disadvantages Of PCA Should Be Considered

1. PCA can easily suffer from scale complications,
2. Similarly to the point above, PCA is susceptible to significant outliers. If the number of samples is small or when values have many potential outliers, this can influence scaling and relative point placement,
3. Intuitive understanding can be tricky.

[1] Emily Mankin, Principal Components Analysis: A How-To Manual for R, http://people.tamu.edu/~alawing/materials/ESSM689/pca.pdf

Data centering / scaling / normalization

It is common for the first step when carrying out a PCA is to center, scale, normalize the data. This is important due the fact that PCA is sensitive to the scale of the features. If the features are quite different frome each other, i.e. different by one or more orders of magnitude then scaling is crucial.

While determining the variance of your dataset, it should be clear that the order of magnitude of your data features matters significantly. The reasons for this should be clear that if one axis is in 1,000's while the second axis is between 1 and 10, the larger scale will have a higher variance distorting the results.

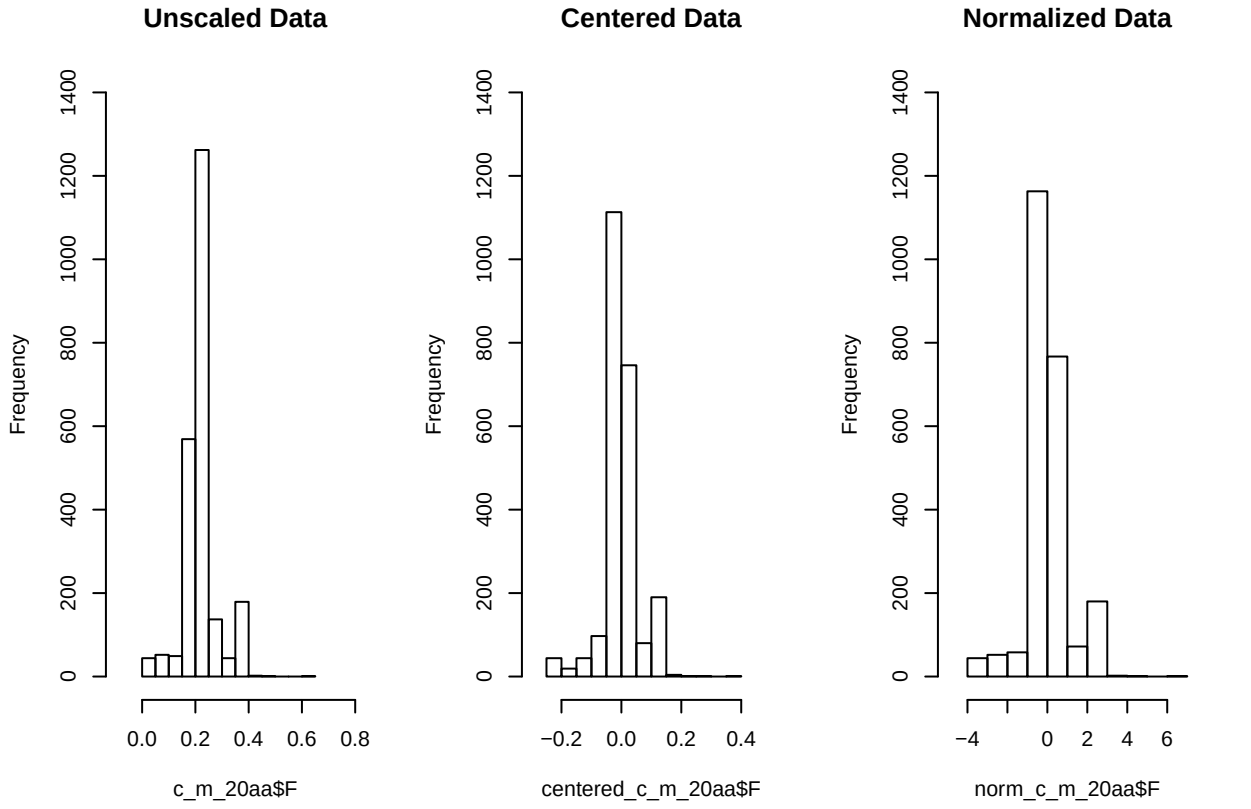
What do the center and scale arguments do in the `prcomp` command?

There are four common methods for scaling data:

Method	Formula
Centering	$f(x) \ = \ x - \bar{x}$
Scaling [0, 1]	$f(x) \ = \ \frac{x - min(x)}{max(x) - min(x)}$
Scaling [a, b]	$f(x) \ = \ (b - a) * \frac{x - min(x)}{max(x) - min(x)} + a$
Normalizing	$f(x) \ = \ \frac{x - mean(x)}{\sigma_x}$

Histograms of Scaled Vs. Unscaled data

Investigating the differences between the amino acid Phenylalanine (F) before and after 2 scaling methods.



Investigating the plots above, the main idea to recognize is that the data has not been fundamentally changed, simply 'shifted and stretched' or more accurately transformed. It appears that any visible changes of the distributions can be accounted for by differing binnings.

Although the differences are between all three histograms are minor, any transformation *would* be sufficient to use. However, I chose to use the *Normalized* dataset.

Finding the Covariance Matrix

The first step for calculating PCA is to determine the Covariance matrix. Covariance provides a measure of how strongly variables change together.[2] [3]

Covariance of two variables

Remember, this simplified formula is to determine covariance for a two-dimensional system.

$$cov(x, y) \;=\; \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y}) \tag{1}$$

Where N is the number of observations, $\bar{x}$ is the mean of the independent variable, $\bar{y}$ is the mean of the dependent variable.

Covariance of matrices

When dealing with a many feature variables one needs to determine the covariance of matrices, M using linear algebra. [4]

1. Find the column means of the matrix, M_{means}.
2. Find the difference matrix, $D \;=\; M - M_{means}$.
3. Finally calculate the covariance matrix:

$$cov\ (M) \;=\; \left(\frac{1}{N-1}\right)\ D^T \cdot D, \quad where \qquad D = M - M_{means} \tag{2}$$

Where D^T is the transpose of the difference matrix, N is the number of observations or rows in this case.

Finding PCA via singular value decomposition

The procedure below is an outline, not the full computation of PCA.

This procedure for PCA relies on the fact that it is similar to the singular value decomposition (SVD) used when determining eigenvectors and eigenvalues. [5]

> Singular value decomposition says that every n x p matrix can be written as the product of three matrices: $A = U\Sigma V^T$ where:
>
> 1. U is an orthogonal n x n matrix.
> 2. Σ is a diagonal n x p matrix. In practice, the diagonal elements are ordered so that $\Sigma_{ii} \;\geqq\; \Sigma_{jj}$ for all i $<$ j.
> 3. V is an orthogonal p x p matrix, and V^T represents a matrix transpose.
>
> The SVD represents the essential geometry of a linear transformation. It tells us that every linear transformation is a composition of three fundamental actions. Reading the equation from right to left:

[2] http://mathworld.wolfram.com/Covariance.html

[3] Trevor Hastie, Robert Tibshirani, Jerome Friedman, 'The Elements of Statistical Learning; Data Mining, Inference, and Prediction,' Second Edition, Springer, DOI:10.1007/978-0-387-84858-7, 2009

[4] http://mathworld.wolfram.com/Covariance.html

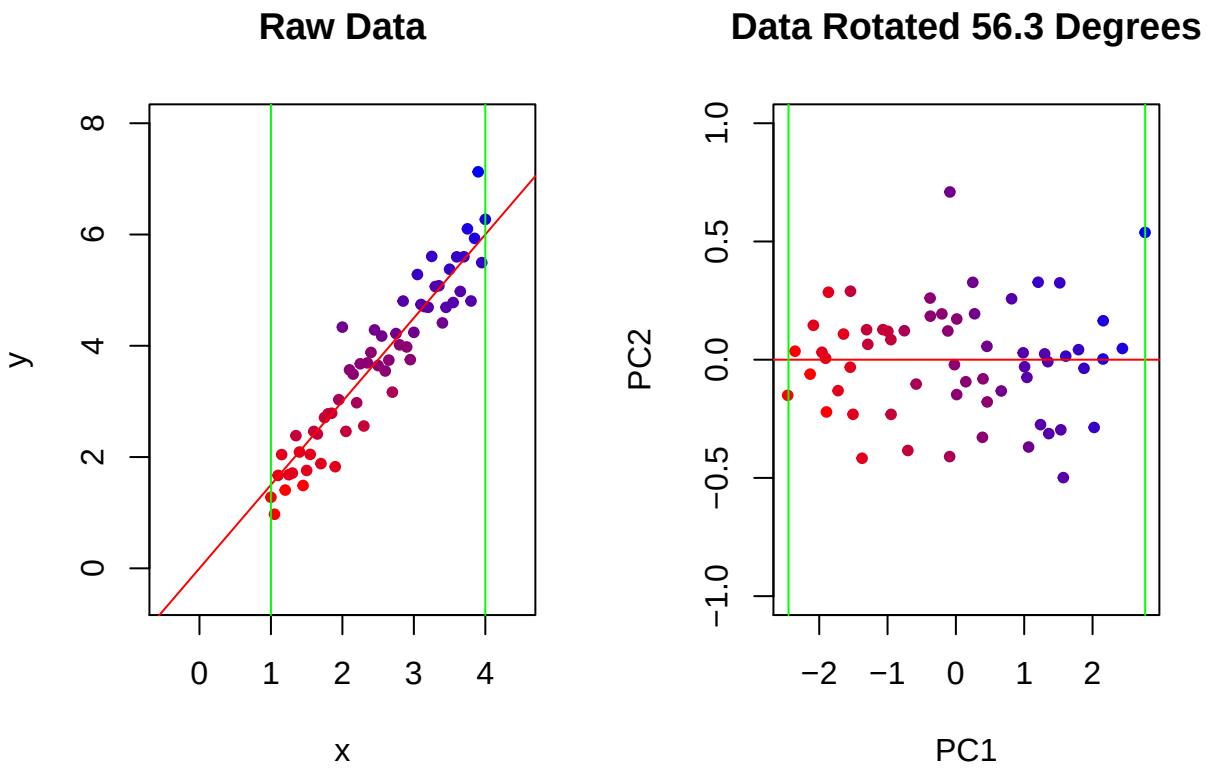
[5] https://blogs.sas.com/content/iml/2017/08/28/singular-value-decomposition-svd-sas.html

1. The matrix V represents a rotation or reflection of vectors in the p-dimensional domain.
2. The matrix Σ represents a linear dilation or contraction along each of the p coordinate directions. If n $\neq$ p, this step also canonically embeds (or projects) the p-dimensional domain into (or onto) the n-dimensional range.
3. The matrix U represents a rotation or reflection of vectors in the n-dimensional range.

The intuition for understanding PCA is reasonably straightforward. Consider the 2-dimensional data cloud of points or observations in a hypothetical experiment, as seen in the figure on the left. Variances along both the x and y dimensions are calculated. However, given the data shown, there is a rotation of that x-y plane, which will present the data showing its most significant variance. This variance will reside on an axis analogous to points on an Ordinary Least Squares (OLS) line. This axis is called the *first principle component* followed by the second principal component and so on.

Unlike an OLS calculation, PCA will determine not only the first and most significant variance of your data set, but it will, through the rotation and transform your dataset via linear algebra, calculating N variances within your dataset, where N is equal to the number of features in the dataset. The second principal component will be calculated only along a coordinate axis, which is perpendicular (orthogonal or orthonormal) to the first. Each subsequent principal component will then be calculated along axes which are orthogonal to each other. A further benefit of using PCA is that the variances it reports will be ranked in order from highest to lowest. [6]

Example of two-dimensional PCA using random data



[6]Brian Everitt, Torsten Hothorn, An Introduction to Applied Multivariate Analysis with R, Springer, DOI:10.1007/978-1-4419-9650-3, 2011

Graphic	Range (Green lines)	Differences
Raw Data (Left)	1 <= x <= 4	3 units
Transformed Data (Right)	-2.45 <= x <= 2.77	5.22 units

If we investigate the figures above we find that the range of the samples is (1 <= x <= 4), while the range for the transformed data is (-2.45 <= x <= 2.76). The differences between the two ranges are 3 and 5.21 units, respectively. The rotation should be no surprise since the PCA is essentially a maximization of variance.

Many R-packages will carry out the steps for PCA all behind the 'scenes' but giving no greater understanding for beginners. For example, `stats::prcomp` [7], `stats::princomp` [8] are most commonly used. However, there are dozens of similar packages. A keyword search for *PCA* at R-cran [9] provides 78 matches, as of November 2019.

Principle component analysis using `norm_c_m_20aa`

```
start_time <- Sys.time() # Start timer

c_m_20_PCA <- prcomp(norm_c_m_20aa)

Sys.time() - start_time # End timer & display time difference
```

```
## Time difference of 0.02079487 secs
```

Screeplot & Cumulative Proportion of Variance plot

Two plots are commonly used to determine the number of principal components that a researcher would generally accept as useful. The eigenvalues derived from PCA are proportional to the variances which they represent, and depending on the strategy used to calculate them, the eigenvalues are equal to the variances of the components.

The first of the two plots which I which is the scree plot. [10] The scree plot is a ranked list of the eigenvalues plotted against its principal components. An eigenvalue score of one is thought to provide a comparable amount of information as a single variable un-transformed by PCA.

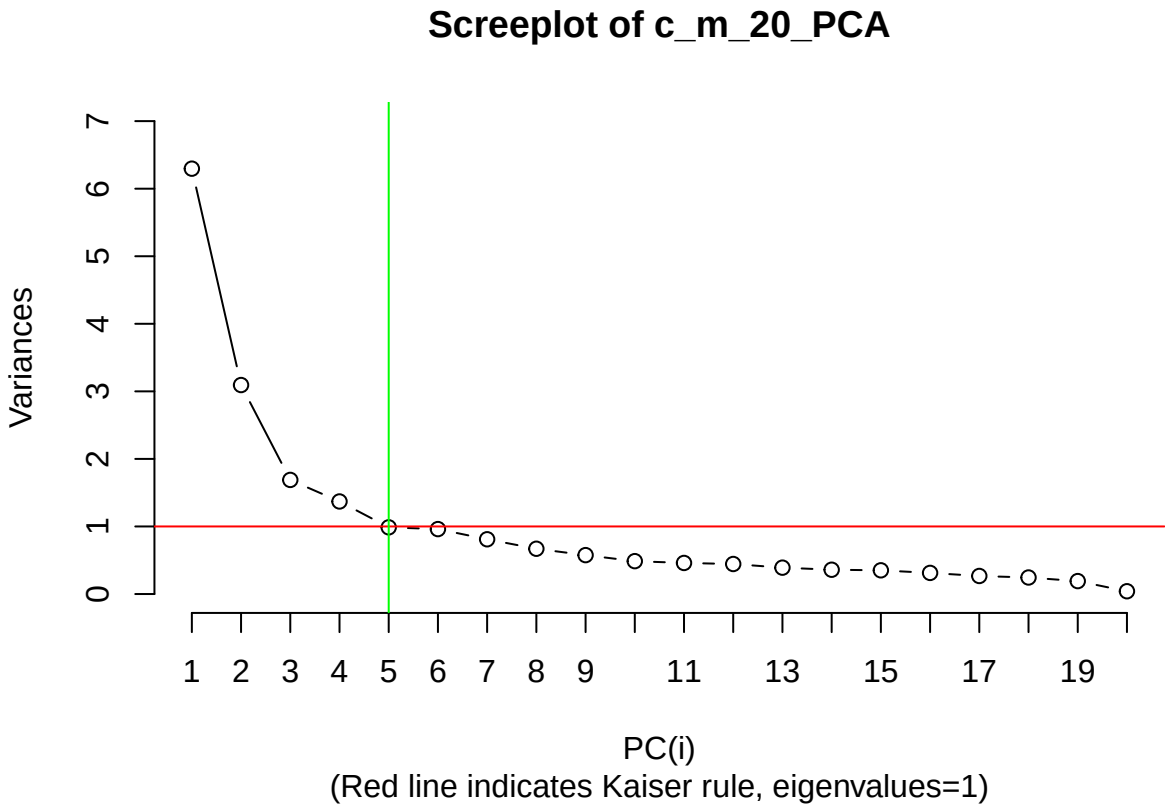
The second plot describes the cumulative proportion of variance versus the principal component. This graphic shows how much each principal component represents the entire cumulative variances or total squared error.

$$Cumlative\ Proportion\ of\ Variance\ =\ \frac{\sigma_i^2}{\sum_{i=1}^{N}\sigma_i^2} \tag{3}$$

Here again, there are several criteria regarding how best to use the information from the is plot. The first of which is Cattell's heuristic. Cattell advises using the principal component that is above the elbow of the curve. The second heuristic is keeping the total number of factors that best explains 80%-95% of the variance. There is no hard-fast rule at this time; a set of researchers only uses the first three factors or

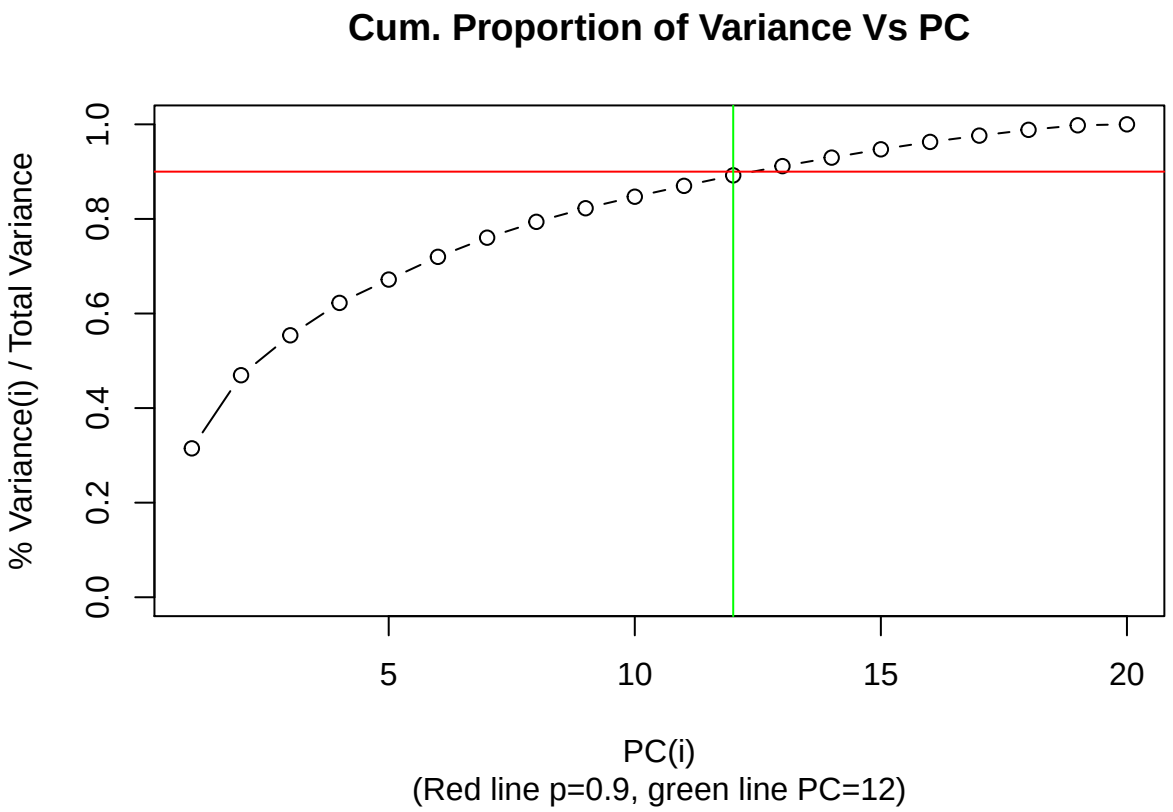
[7] https://stat.ethz.ch/R-manual/R-devel/library/stats/html/prcomp.html
[8] https://stat.ethz.ch/R-manual/R-devel/library/stats/html/princomp.html
[9] https://cran.r-project.org/web/packages/available_packages_by_name.html
[10] Raymond Cattell, "The scree test for the number of factors." Multivariate Behavioral Research. 1 (2): 245–76. DOI: 10.1207/s15327906mbr0102_10, 1966

none at all.[11] A second suggestion is to use the Kaiser rule, which states it is sufficient to use Principal Components, which have an eigenvalue greater than or equal to one. [12]



[11] Nicole Radzill, Ph.D., personal communication.

[12] https://stats.stackexchange.com/questions/253535/the-advantages-and-disadvantages-of-using-kaiser-rule-to-select-the-number-of-pr



If we investigate the 'cumulative proportion of variance' plot, we see an arbitrary line on the Y-axis, which denotes the 90% mark. At this point, the plot suggests that a researcher could use the most significant 12 of the variances from the PCA.

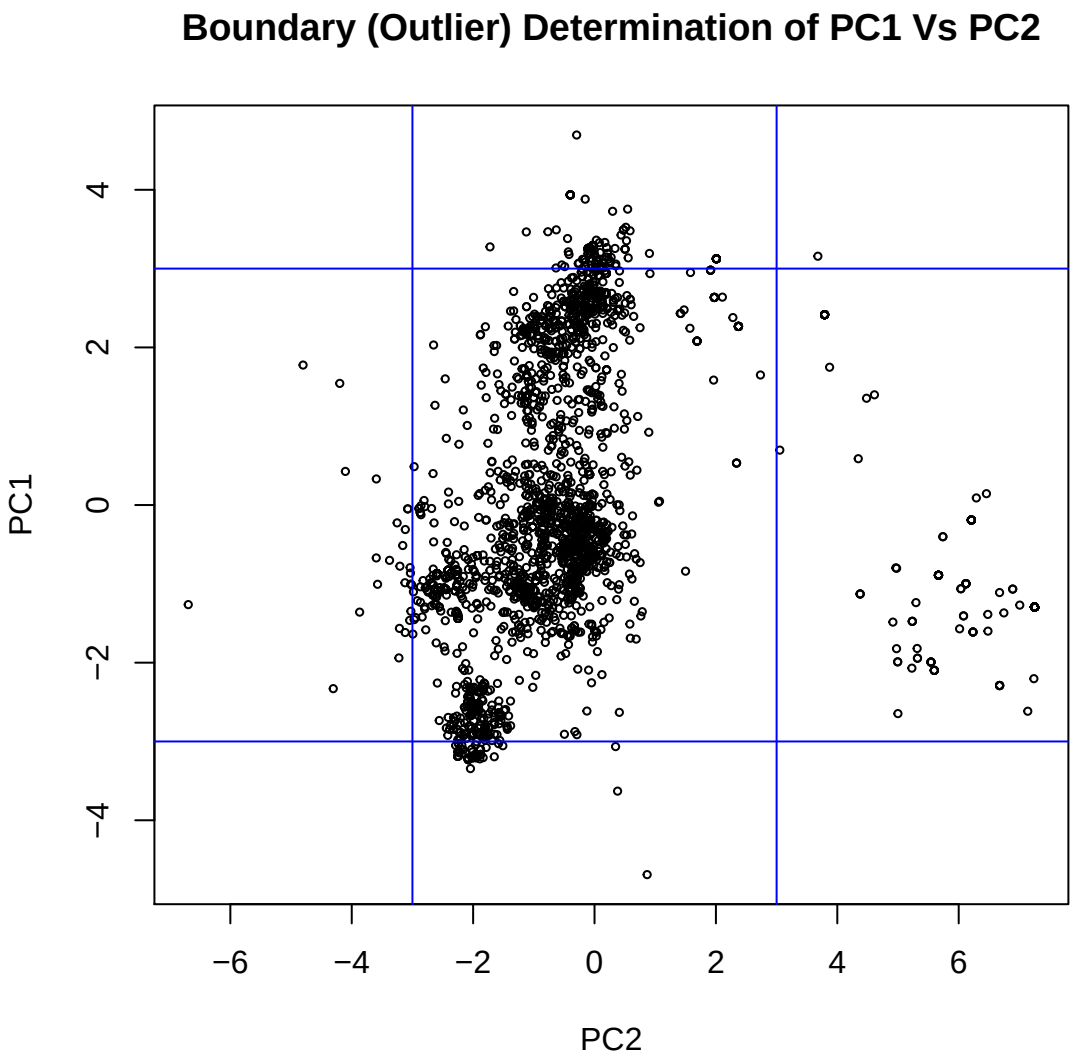
Biplots

Biplot 1: PC1 Vs. PC2 with 'Class' by color labels

- Black indicates control protein set, Class = 0
- Blue indicates myoglobin protein set, Class = 1

The first two principal components describe 46.95% of the variance.

Biplot 2: Determination Of 4 Rule Set For Outliers



Obtain Anomalous Points From Biplot #2: PC1 Vs. PC2

Anomalous data points are data that is greater than the absolute value of 3 sigma, Anomalous Point $> |3\sigma|$.

I have chosen to analyze the PCA biplot of the first and second principal components. The first and second components were used because they describe nearly 50% of the variance (46.95%).

Outliers from Principal Component-1

Rule Set Given PC1:

1. Outlier_1: c_m_20_PCA$x[, 1] > 3 standard deviations
2. Outlier_2: c_m_20_PCA$x[, 1] < -3 standard deviations

```
outliers_PC1 <- which((c_m_20_PCA$x[, 1] > 3) | (c_m_20_PCA$x[, 1] < -3))
length(outliers_PC1)
```

```
## [1] 285
```

Outliers from Principal Component-2

Rule Set Given PC2:

3. Outlier_3: c_m_20_PCA$x[, 2] > 3 standard deviations
4. Outlier_4: c_m_20_PCA$x[, 2] < -3 standard deviations

```
outliers_PC2 <- which((c_m_20_PCA$x[, 2] > 3) | (c_m_20_PCA$x[, 2] < -3))
length(outliers_PC2)
```

```
## [1] 177
```

List of all outliers (union and sorted) found using the ruleset 1 through 4

- The list of total outliers is derived by taking the `union` of `outliers_PC1` and `outliers_PC2` and then using `sort`.

```
total_pca_1_2_outliers <- union(outliers_PC1, outliers_PC2)
total_pca_1_2_outliers <- sort(total_pca_1_2_outliers)

length(total_pca_1_2_outliers)
```

```
## [1] 461
```

```
# Write out to Outliers folder
write.table(total_pca_1_2_outliers,
file = "./00-data/03-ml_results/pca_outliers.csv",
row.names = FALSE,
na = "",
col.names = "rowNum",
sep = ",")
```

It is important to remember and understand that this list of "total_pca_1_2_outliers" includes BOTH negative and positive controls. The groupings are as follows:

Group	Range of Groups
Controls	1, . . . , 1216
Positive (Myoglobin)	1217, . . . , 2341

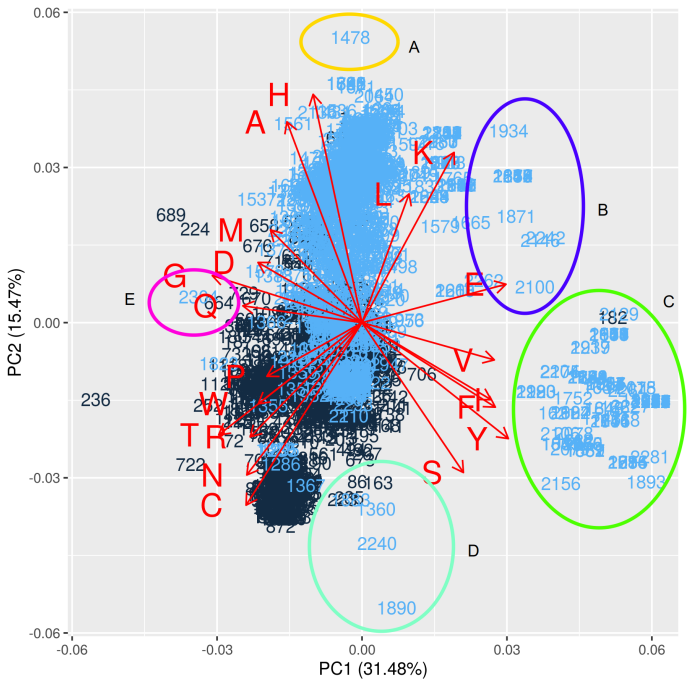
PCA Conclusion

Principal Component Analysis is very popular and an excellent choice to include during Exploratory Data. Analysis. One objective for using PCA is to filter noise from the dataset used and, in turn, increase any signal or to sufficiently delineate observations from each other. In fact, in the figure below, there are five colored groups outside the main body of observations that are marked at 'outliers.' The number of outliers obtained from PCA is 461 proteins. The premise of this experiment is to determine if PCA is an excellent representative measure for proteins that are categorized is false-positive, and false-negatives in the five subsequent machine learning model approach. It will be interesting to see if anyone of these groups will be present in the group of false-positives and false-negatives in any of the machine learning models.

Outliers derived from PC1 Vs PC2

The table and the figure below show a subset of outliers produced when the first and second principal component is graphed. My interest lies in finding if any one of the lettered groups (A-E) are part of the false-positives and false-negatives from each of the machine learning models. Each of the five groups is rich is a small number of amino acids. We hope that this information will shine a light on how the different machine models work. It is also expected that this will give help in constructing a model that is more interpretable for the more difficult opaque machine learning models, such as Random Forest, Neural Networks, and possibly Support Vector Machine using the Radial Basis Function.

Group	Increased concentration of amino acid	Example observations
A	H, L, K	1478
B	E, K	1934, 1870, 2100
C	V, I, F, Y	182, 1752, 2156
D	C, S	1360, 2240
E	G, D, Q	664, 2304



Logistic Regression For Binary Classification

mcc

3/13/2020

Logistic Regression For Binary Classification

Introduction

For individuals who have studied cell biology or biochemistry, logistic regression may be familiar as dose-response curves, enzyme kinetic curves, sigmoidal curves, median lethal dose curve (LD-50) or even an exponential growth curve given limited resources.

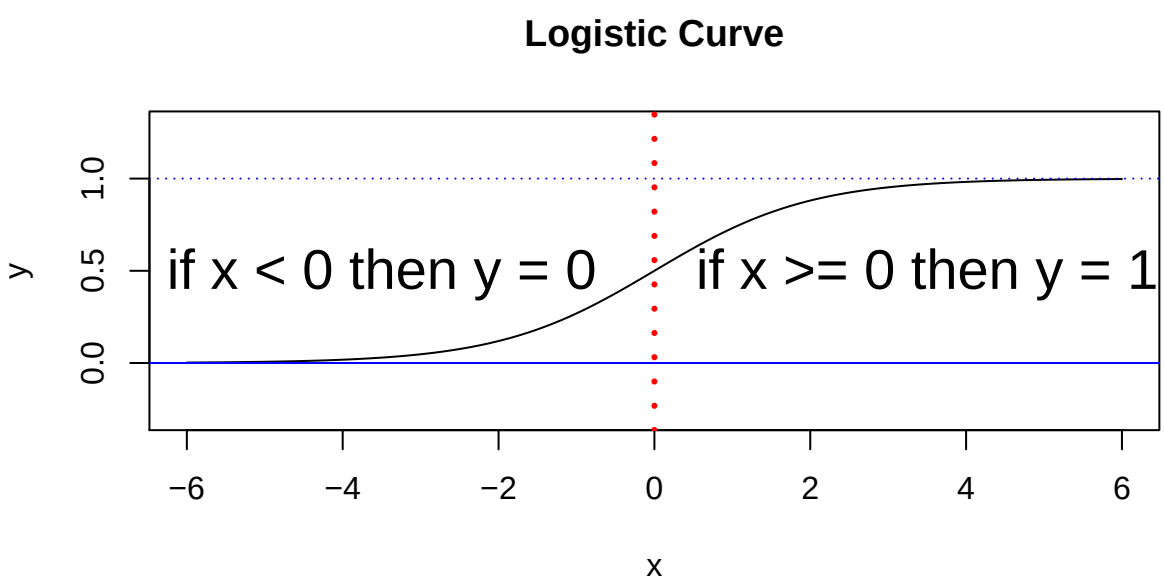
However, in the context of predictive modeling, Logistic Regression is used as a binary classifier that toggle between the logical values of zero or one.

Logistic regression (Logit) derives its name from its similarity to linear regression, as we shall see below. The input/independent variable for Logit is the set of real numbers, ($X \in \Re$). While, the output of a Logistic Regression is not represented by {0, 1}, ($Y \notin \Re$),

$$f(x) = \begin{cases} 0 \quad for \quad x < 0 \\ 1 \quad for \quad x \geq 0 \end{cases} \tag{1}$$

Using Logistic Regression, we may calculate the presence or absence of a product or quality that we wish to model given a difficult situation where the transition is not clear.

In the figure below, the function's domain, $X \in \{-\infty \text{ to } \infty\}$, whereby its range is {0, 1}. In the figure, the *decision boundary* is $x = 0$, denoted by the *red dotted line*. At the inflection point the curves range changes from *zero*, absence, to *one*, the presence of quality or item.



The logistic growth curve is commonly denoted by:

$$f(x) = \frac{M}{1 + Ae^{-r(x-x_0)}} \tag{2}$$

where M is the curve's maximum value, r is the maximum growth rate (also called the Malthusian parameter [1]), x_0 is the midpoint of the curve, A is the number of times that the initial population must double to reach M.[2]

In the specific case of *Logistic Regression for Binary Classification* where we have a probability between 0 and 1, M, and A take on the value one.

$$f(x) = \frac{1}{1 + e^{-(WX+b)}} \tag{3}$$

Since the logistic equation is exponential, it is easier to work with the formula in terms of its odds or *log-odds*. Odds are the probabilities of success over failure denoted as $\frac{p}{1-p}$ and more importantly, in this situation, log-odds are $ln\left(\frac{p}{1-p}\right)$.

Simply by using log-odds, logistic regression may be more easily expressed as a set of linear equations in x.[3] Hence we can now go from linear regression to logistic regression.

Step #1:

$$ln\left(\frac{Pr(y_i = 1|x_i)}{Pr(y_i = 0|x_i)}\right) = \beta_0 + \beta_1 x_1 + \ ... \ + \beta_n x_n \tag{4}$$

Step #2: Substitute (p for $Pr(y_i = 1|x_i)$) and ($1-p$ for $Pr(y_i = 0|x_i)$) and change notation to summation on the right hand side;

[1] https://en.wikipedia.org/wiki/Malthusian_growth_model
[2] https://en.wikipedia.org/wiki/Logistic_function
[3] http://juangabrielgomila.com/en/logistic-regression-derivation/

$$ln\left(\frac{p}{1-p}\right) = \sum_{i}^{k} \beta_i x_i \tag{5}$$

Step #3: Eliminate the natural log by taking the exponent on both sides;

$$\frac{p}{1-p} = exp\left(\sum_{i}^{k} \beta_i x_i\right) \tag{6}$$

Step #4: Substitute $u = \sum_{i}^{k} \beta_i x_i$;

$$\frac{p}{1-p} = e^u \tag{7}$$

Step #5: Rearrange to solve for p;

$$p(u) = \frac{e^u}{1+e^u} \tag{8}$$

Step #6: Incidentally, to find the probabilities, take the derivative of both sides using quotient rule;

$$p'(u) = \frac{(e^u)(1+e^u) - (e^u)(e^u)}{(1+e^u)^2} \tag{9}$$

Step #7: Simplify;

$$p'(u) = \frac{e^u}{(1+e^u)^2} \tag{10}$$

Step #8: Separate out to produce two fractions;

$$p'(u) = \left(\frac{e^u}{1+e^u}\right) \cdot \left(\frac{1}{1+e^u}\right) \tag{11}$$

Step #9: Substitute our previous success and failure variables back into place;

$$p'(u) = p(u) \cdot (1 - p(u)) \tag{12}$$

Now we can calculate the probabilities as well as the values for any given x value.

Logit-20 Training Using 20 Features

```
# Load Libraries
Libraries <- c("doMC", "knitr", "readr", "tidyverse", "caret")
for (p in Libraries) {
    library(p, character.only = TRUE)
}
```

```
# Import relevant data
c_m_TRANSFORMED <- read_csv("./00-data/02-aac_dpc_values/c_m_TRANSFORMED.csv",
                            col_types = cols(Class = col_factor(levels = c("0", "1")),
                                             PID = col_skip(),
                                             TotalAA = col_skip()))
```

```
# Partition data into training and testing sets
set.seed(1000)
index <- createDataPartition(c_m_TRANSFORMED$Class, p = 0.8, list = FALSE)
training_set.1 <- c_m_TRANSFORMED[index, ]
```

- The `test.set.1` and `Class.test` data sets are not produced since the Logit run with 20 features was not deemed useful. The reason for its dismissal was that is contained extraneous features.
- The first training run is to determine if all 20 features (amino acids) are necessary for our logistic regression model.

```
set.seed(1000)
registerDoMC(cores = 3)       # Start multi-processor mode
start_time <- Sys.time()      # Start timer

# Create model, 10X fold CV repeated 5X
tcontrol <- trainControl(method = "repeatedcv",
                         number = 10,
                         repeats = 5)

model_obj.1 <- train(Class ~ .,
                     data = training_set.1,
                     trControl = tcontrol,
                     method = "glm",
                     family = "binomial")

end_time <- Sys.time()       # End timer
end_time - start_time        # Display time
```

```
## Time difference of 3.231281 secs
```

```
registerDoSEQ()              # Stop multi-processor mode
```

Logit-20 Summary #1

```
summary(model_obj.1)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -5.9372  -0.2835  -0.0194   0.0516   3.6884
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   8.0525     9.2156   0.874 0.382234
## A             5.0438     9.6899   0.521 0.602699
## C           -14.2228     2.6949  -5.278 1.31e-07 ***
## D           -36.2676     8.0845  -4.486 7.25e-06 ***
## E            27.6016    11.1292   2.480 0.013135 *
## F             5.6174     5.2654   1.067 0.286034
## G           -22.1970    10.3043  -2.154 0.031229 *
## H            90.1101    12.1105   7.441 1.00e-13 ***
## I            -5.9795     4.3945  -1.361 0.173610
## K            -2.8961     9.8468  -0.294 0.768669
## L            -3.7417     9.2217  -0.406 0.684926
## M            -0.1427    12.0747  -0.012 0.990570
## N             3.3478     9.6749   0.346 0.729319
## P           -39.7466    11.1010  -3.580 0.000343 ***
## Q            -5.6804    11.2516  -0.505 0.613664
## R           -83.6045    11.8104  -7.079 1.45e-12 ***
## S            -9.9745    10.0872  -0.989 0.322750
## T           -36.5980     9.2791  -3.944 8.01e-05 ***
## V            16.3411     9.7859   1.670 0.094946 .
## W             9.0169    13.8870   0.649 0.516141
## Y           -31.9282    11.1167  -2.872 0.004078 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2593.68  on 1872  degrees of freedom
## Residual deviance:  657.72  on 1852  degrees of freedom
## AIC: 699.72
##
## Number of Fisher Scoring iterations: 8
```

The Akaike information criterion (AIC)[4] for model #1 is 699.72. This will be used later to compare the models generated to rate their ability to utilize the features best. - The list of probabilities for the estimates leaves us with only **9 important features** to try re-modeling, R, H, P, C, E, Y, T, D, G.

Logit-9 Training Using 9 Features

- This test uses **ONLY** 9 features: (R, H, P, C, E, Y, T, D, G)

```
# Data import & handling
c_m_9aa <- read_csv("./00-data/02-aac_dpc_values/c_m_TRANSFORMED.csv",
                    col_types = cols(Class = col_factor(levels = c("0", "1")),
                                     A = col_skip(), F = col_skip(),
                                     I = col_skip(), K = col_skip(),
                                     L = col_skip(), M = col_skip(),
                                     N = col_skip(), PID = col_skip(),
```

[4] https://en.wikipedia.org/wiki/Akaike_information_criterion

```
                                    Q = col_skip(), V = col_skip(),
                                    S = col_skip(), TotalAA = col_skip(),
                                    W = col_skip()))
```

```
# Partition data into training and testing sets
set.seed(1000)
index <- createDataPartition(c_m_9aa$Class, p = 0.8, list = FALSE)

training_set.2 <- c_m_9aa[ index, ]
test_set.2     <- c_m_9aa[-index, ]

Class_test.2 <- as.factor(test_set.2$Class)
```

```
set.seed(1000)
registerDoMC(cores = 3)           # Start multi-core
start_time <- Sys.time()          # Start timer

# Create model, 10X fold CV repeated 5X
fitControl <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 5,
                           savePredictions = "final") # IMPORTANT: Saves predictions

model_obj.2 <- train(Class ~ .,
                     data = training_set.2,
                     trControl = fitControl,
                     method = "glm",
                     family = "binomial")

end_time <- Sys.time()            # End timer
end_time - start_time             # Display time
```

```
## Time difference of 2.613658 secs
```

```
registerDoSEQ()                   # Stop multi-core
```

Logit-9 Summary

```
summary(model_obj.2)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -6.2083  -0.2984  -0.0204   0.0601   3.5666
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)    8.306      1.007   8.245  < 2e-16 ***
## C            -14.755      1.908  -7.733 1.05e-14 ***
## D            -31.411      4.949  -6.347 2.20e-10 ***
## E             21.932      5.092   4.307 1.66e-05 ***
## G            -23.259      5.071  -4.587 4.49e-06 ***
## H             94.580      8.431  11.218  < 2e-16 ***
## P            -29.394      6.264  -4.692 2.70e-06 ***
## R            -82.809      6.363 -13.015  < 2e-16 ***
## T            -40.915      5.624  -7.275 3.45e-13 ***
## Y            -37.860      6.291  -6.018 1.77e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2593.68  on 1872  degrees of freedom
## Residual deviance:  688.96  on 1863  degrees of freedom
## AIC: 708.96
##
## Number of Fisher Scoring iterations: 8
```

Logit-9 Confusion Matrix

```
Predicted_test_vals <- predict(model_obj.2, test_set.2[, -1])

confusionMatrix(Predicted_test_vals, Class_test.2, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 235  18
##          1   8 206
##
##                Accuracy : 0.9443
##                  95% CI : (0.9195, 0.9633)
##     No Information Rate : 0.5203
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8883
##
##  Mcnemar's Test P-Value : 0.07756
##
##             Sensitivity : 0.9196
##             Specificity : 0.9671
##          Pos Pred Value : 0.9626
##          Neg Pred Value : 0.9289
##              Prevalence : 0.4797
##          Detection Rate : 0.4411
##    Detection Prevalence : 0.4582
##       Balanced Accuracy : 0.9434
##
##        'Positive' Class : 1
##
```

- The Akaike information criterion (AIC) for model #2 is 708.96. This will be used later to compare the models generated to rate their ability to utilize the features best.
- The number of unique false-positives and false-negatives is 26.

Obtain List of False Positives & False Negatives From Logit-9

```
fp_fn_logit <- model_obj.2 %>% pluck("pred") %>% dplyr::filter(obs != pred)

# Write CSV in R
write.table(fp_fn_logit,
            file = "./00-data/03-ml_results/fp_fn_logit.csv",
            row.names = FALSE,
            na = "",
            col.names = TRUE,
            sep = ",")

nrow(fp_fn_logit) ## NOTE: NOT UNIQUE NOR SORTED
```

```
## [1] 536
```

- The logistic regression second test produced 536 protein samples, which are either false-positives or false-negatives. The list of 536 proteins may have duplicates. Therefore they are NOT UNIQUE NOR SORTED.

Logit Conclusion

Logit is easy to implement and understand and can be used for feature selection.

Considering the table Logit Models, below, it is clear that model #2 with nine features best describes the better of the two models.

Akaike Information Criterion [5]

$$AIC \ = \ 2K \ - \ 2ln(\widehat{L}) \tag{13}$$

Where $ln(\widehat{L})$ is the log-likelihood estimate, K is the number of parameters.

Two Logit Models

Model #	Features	AIC
1	20	699.72
2	9	708.96

Logit is a common machine learning method. It is easy to understand and explain. This supervised binary classification method is very useful for determining the importance of the features which can be applied. As we saw in Model#1, there were 11 features that had probabilities of the estimates used above the 5% threshold cut-off. In Model#2, only nine features were used to describe the model, and the AIC increased by 9.24.

The nine features which best described the logistic regression model were R, H, P, C, E, Y, T, D, G. If we compare this to the Boruta test carried out in the EDA, we find the overlap interesting.

Comparison of Boruta Vs Logit: Order of Importance

Test	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Boruta	R	H	P	K	C	E	Y	T	S	A	V	U	I	F	D	G	N	L	M	Q
Logit-9	R	H	P	.	C	E	Y	T	.	.	.	.	.	.	D	G	.	.	.	.

The first 7 out of 8 amino acid features are seen in the proper order, as described by the Boruta Random Forest model. This is confirmation that Logit can pick up the importance of features similar to Boruta.

Logit produced 536 proteins, which are false-negatives or false-positives. It should be noted that the 536 are NOT UNIQUE NOR SORTED. The number of unique FN/FP from the confusion matrix is 26. These proteins will be investigated further in the Outliers chapter, which compares these FN/FP proteins to the PCA outliers.

The two tests for Logit (using 20 then 9 features) is interesting. This shows that Logit is an alternatvie way

[5] https://en.wikipedia.org/wiki/Akaike_information_criterion

of choosing the importance of features. As can be seen in the table "*Comparison of Boruta Vs Logit: Order of Importance*" it was seen that the first seven features lined up very closely with the Boruta Random Forest feature order of importance.

Neural Networks For Binary Classification

mcc

3/13/2020

Neural Networks For Binary Classification

> "Machine learning is essentially a form of applied statistics with increased emphasis on the use of computers to statistically estimate complicated functions and a decreased emphasis on proving confidence intervals around these functions"
>
> – Ian Goodfellow, et al[1]

Introduction

If we discuss Neural Networks (NN), we should first consider the system we hope to emulate. Let us start with a simple count of neuronal cells in various organisms along the earth's phylogenetic tree. We might get a better idea of the type of "computing power" these living creatures possess. See table 6.1.

Table 6.1: Organisms Vs Number of Neurons In Each (Wikipedia)

Organism	Common Name	Approximate Number of Neurons
C. elegans	roundworm	302
Chrysaora fuscescens	jellyfish	5,600
Apis linnaeus	honey bee	960,000
Mus musculus	mouse	71,000,000
Felis silvestris	cat	760,000,000
Canis lupus familiaris	dog	2,300,000,000
Homo sapien sapien	humans	100,000,000,000

This table portrays a high-level overview of the computing power of neuronal clusters and brains produced throughout evolution. However, there is one missing number worth noting. The table above does not describe the connectivity between neurons. The connectivity of neurons varies greatly from lower to higher organisms. For example, some simple animals, such as the roundworm, have only "four to eight separate branches," [2] per nerve cell. While human neurons may have greater than 10,000 inter-connected synaptic junctions per neuron, thus resulting in a total of approximately 600 trillion synapses per human brain.[3]

Although neurons have differing morphologies, neurons in the human brain are extremely diverse. Indeed, size and shape may not be the definitive way of classifying neurons but instead by what neurotransmitters the cells secrete. "Neurotransmitters can be classified as either excitatory or inhibitory." [4] Currently

[1] Ian Goodfellow, Yoshua Bengio, Aaron Courville, 'Deep Learning', MIT Press, 2016, http://www.deeplearningbook.org
[2] https://www.wormatlas.org/hermaphrodite/nervous/Neuroframeset.html
[3] Shepherd, G. M. (2004), The synaptic organization of the brain (5th ed.), Oxford University Press, New York.
[4] https://www.kenhub.com/en/library/anatomy/neurotransmitters

the NeuroPep database "holds 5949 non-redundant neuropeptide entries originating from 493 organisms belonging to 65 neuropeptide families." [5]

[5] http://isyslab.info/NeuroPep/home.jsp

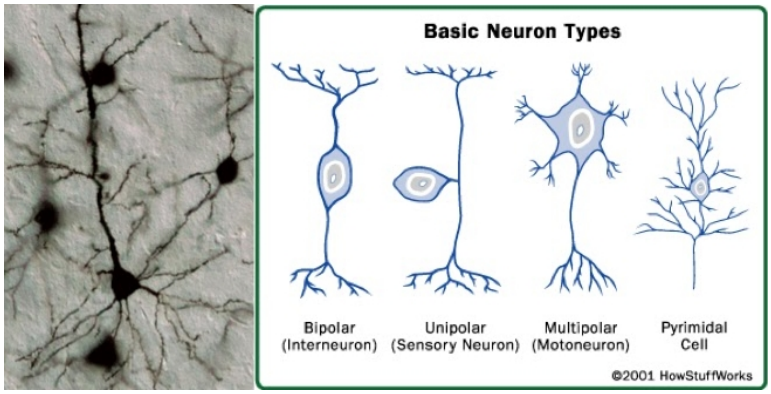


Figure 1: Basic Neuron Types and S.E.M. Image

6

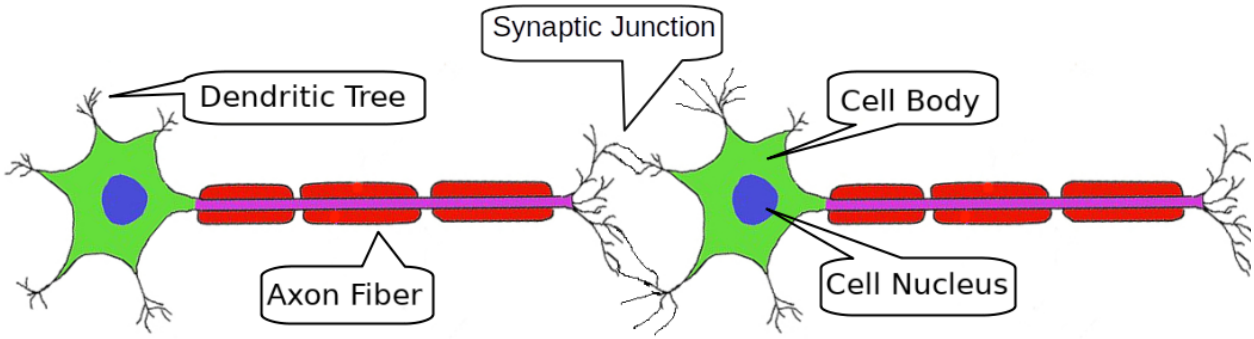


Figure 2: Two Neuron System (Image From The Public Domain)

Given an order of operation via:

Dendrite(s) $\Longrightarrow$ *Cell body* $\Longrightarrow$ *Fibrous Axon* $\Longrightarrow$ *Synaptic Junction or Synaptic Gap* $\Longrightarrow$ *Dendrite(s)* ... *Ad infinitum.*

However, nature is more subtle and intricate than to have neurons in a series, only blinking on and off, firing or not. NN are often programmed to classify dangerous road objects, as is the case of Tesla cars. The goal of a Tesla auto-piloted car is to use all available sensors to correctly classify all the conceivable circumstances on the road. On the road, a Tesla automobile uses dozens of senors which the computer needs to evaluate and weigh the values of all these sensors to formulate a 'decision.' The altitude of the auto, derived from the GPS, may weigh less heavily than the speed of the vehicle or Lidar estimates on how close objects are. However, our goal of safe driving can be thwarted when an artificial intelligence system decides a truck is a sign and does not apply the brakes.[7]

[6] https://www.howstuffworks.com/
[7] https://arstechnica.com/cars/2019/05/feds-autopilot-was-active-during-deadly-march-tesla-crash/

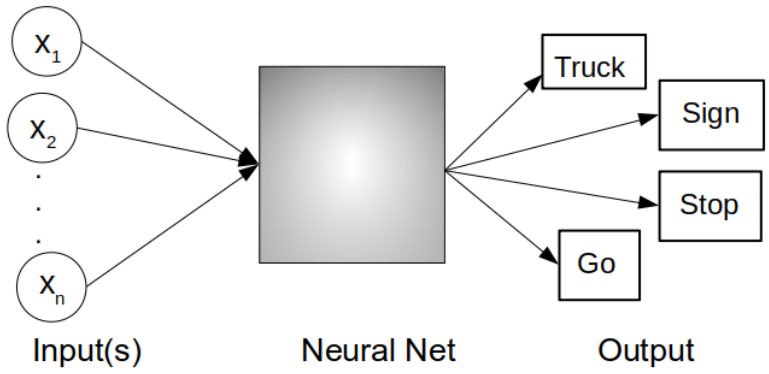


Figure 3: Goal of a Tesla Neural Networks is to generate the correct responses for its environment.

The One Neuron System

If we investigate a one neuron system, *our* neuron could be diagrammed in four sections.[8]

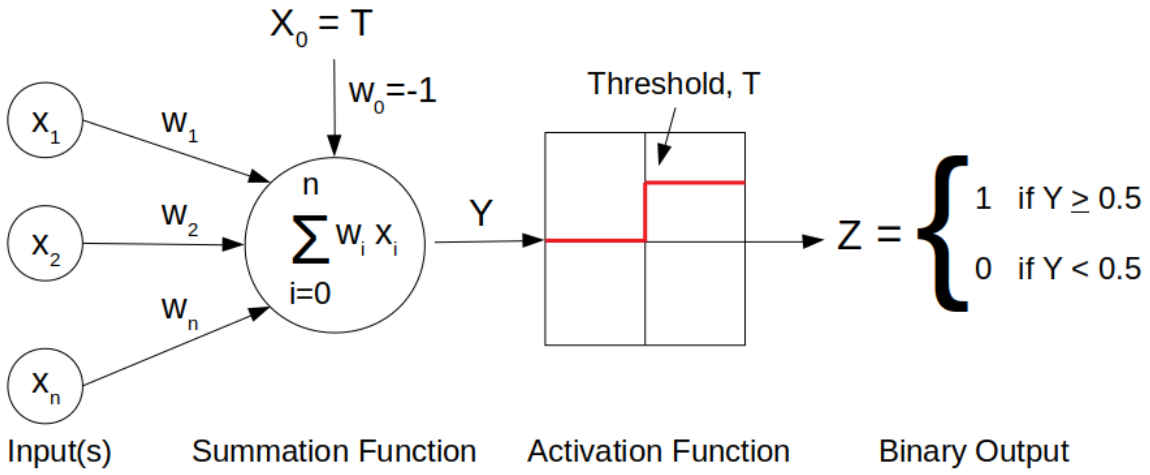


Figure 4: One Neuron Schema

If we investigate one neuron for a moment, we find two separate mathematical functions are being carried out by a single nerve cell.

Summation Function

The first segment is a summation function. It receives the real number values from, x_1 to x_n, all the branches of the dendritic trees, and multiplies them by a set of weights. These X inputs are multiplied by a set of corresponding unique weights from w_1 to w_n. An analogy I prefer is of small or large rivers joining giving a total current. The current moves through the branches giving a total signal or current of sodium ions. Interestingly the summation in each neuron, while dealing with the vectors of inputs and weights, is carrying out the dot product of these vectors.

[8]Tom Mitchell, Machine Learning, McGraw-Hill, 1997, ISBN: 0070428077

Initially, the NN researchers used the Heaviside-Threshold Function, as shown in figure 5.4, *One Neuron System*. The benefits of step functions were their simplicity and high signal to noise ratio. While the detriments were, it is a discontinuous function, therefore not differentiable and a mathematical problem.

Let us take into account the product, $x_0 \cdot w_0$. If we assign $x_0 = T$ and $w_0 = -1$ this simply becomes a bias. This bias allows us the ability to shift our Activation Function and its inflection point in the positive or negative x-direction.

$$\hat{Y} \;=\; X^T \cdot W - Bias \quad \equiv \quad \sum_{i=0}^{n} x_i w_i - T \tag{1}$$

Activation Functions

The second function is called an Activation Function. Once the Summation Function yields a value, its result is sent to the *Activation Function* or *Threshold Function*.

$$Z^{(1)} = f\left(\sum_{i=0}^{n} x_i w_i - T\right) = \{0, 1\} \tag{2}$$

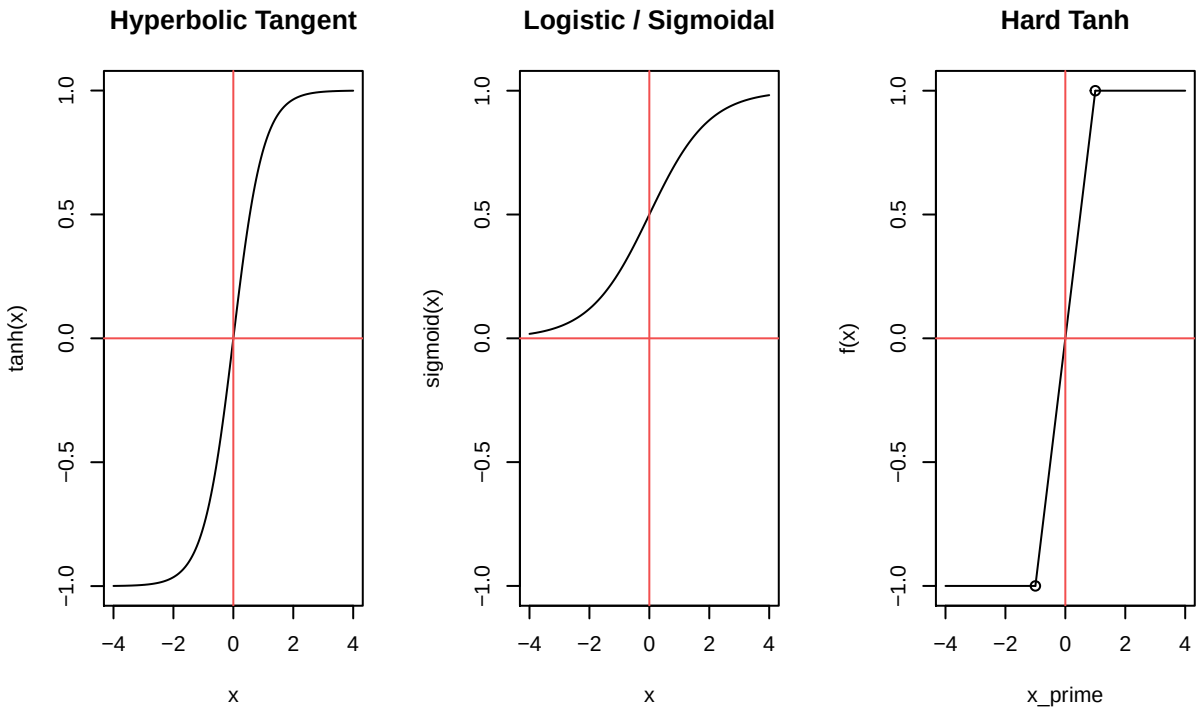
The function displayed in figure #6.4, One Neuron Schema, is a step function. However this step function has a problem mathematically, namely it is a discontinuous and therefore not differentiable. This fact is important.

Therefore several functions may be used in place of the step function. One is the hyperbolic tangent (*tanh*) function, the *sigmoidal* function, a *Hard Tanh*, a *reLU*, and *Softmax* Functions. These have certain advantages, namely they simplify the hyperbolic tangent function. Not only does the Hard Tanh and reLU simplify calculations it is useful for increasing the gain near the asymptotic limits of the sigmoidal and tanh functions. The derivatives of the sigmoidal and tanh functions are very small, near 0 and 1, while the reLU and Hard Tanh slopes are one or zero.

$$Z^{(2)} \;=\; tanh(x) = \frac{1 - e^{-\alpha}}{1 + e^{-\alpha}} \quad : \quad where \quad \alpha = \sum_{i=1}^{n} x_i w_i - T \tag{3}$$

$$Z^{(3)} \;=\; sigmoid(x) \;=\; \frac{1}{1 + e^{-\alpha}} \tag{4}$$

$$Z^{(4)} \;=\; Hard\ Tanh(x) \;=\; \begin{cases} 1 & x > 1 \\ x & -1 \le x \le 1 \\ -1 & x < -1 \end{cases} \tag{5}$$



Several alternative functions are useful for various reasons. The most common of which are Softmax and reLU functions.

Rectified Linear Activation Unit, (ReLU):

$$Z^{(5)} \;=\; ReLU \;= \begin{cases} x \geq 0 & y = x \\ x < 0 & y = 0 \end{cases} \tag{6}$$

Binary Output Or Probability

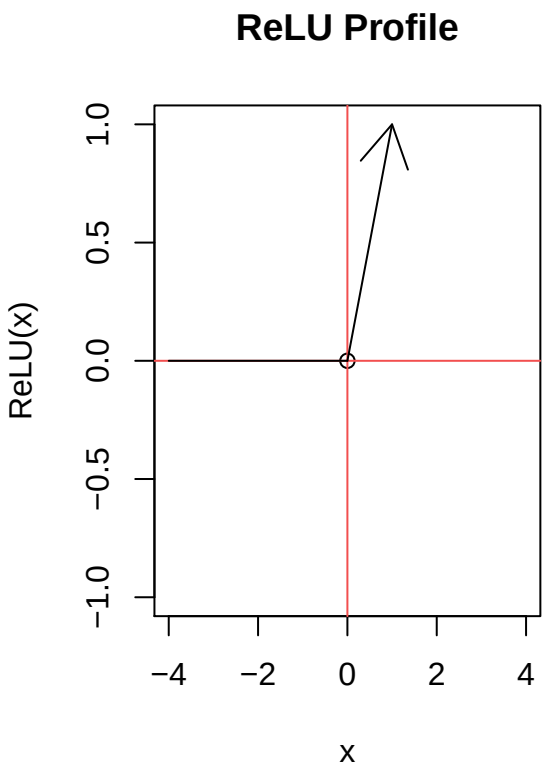
In the case of real neurons, the output is off or on, zero or one. However, in the case of our electronic model, it is advantageous to calculate a probability for greater interpretability.

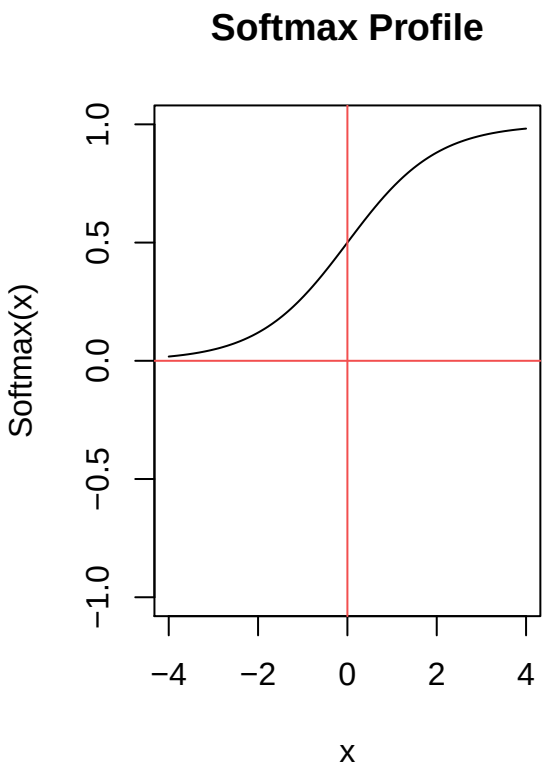
> The Softmax function may appear like the Sigmoid function from above but it differs in major ways.[9]
>
> - The softmax activation function returns the probability distribution over mutually exclusive output classes.
> - The calculated probabilities will be in the range of 0 to 1.
> - The sum of all the probabilities is equals to 1.

Typically the Softmax Function is used in binary or multiple classification logistic regression models and in building the final output layer of NN.

$$Z^{(6)} \;=\; Softmax(x) = \frac{e^{\alpha_i}}{\sum_{i=1}^{n} e^{\alpha_i}} \tag{7}$$

[9] Josh Patterson, Adam Gibson, Deep Learning; A Practitioner's Approach, 2017, O'Rreilly





The benefit of these activation functions is that they are now differentiable. This fact becomes important for *Back-Propagation*, which is discussed later.

The Two Neuron System

Building up in complexity, let us could consider our first Neural Network by using *only* two neurons. In two neuron systems, let us first generalize a bit more by adding that X is an array of all the inputs as is W_1 and W_2 is also an array of weights for each neuron. See figure #6.5.

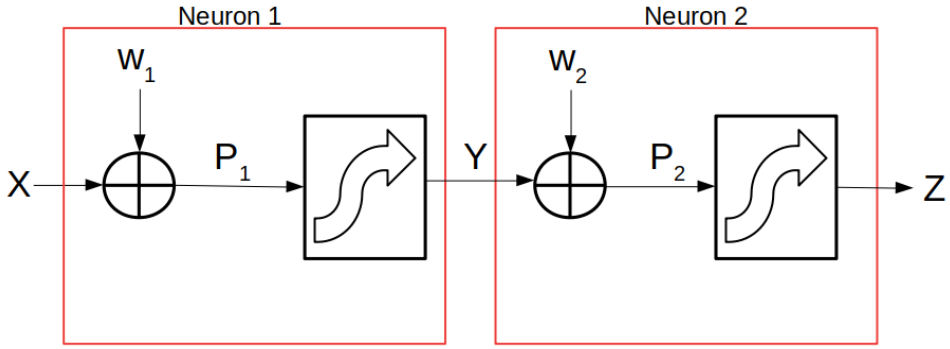


Figure 5: A Two Neuron System

Feed-Forward In A Two Neuron Network

In our two neuron network, we can now write out the mathematics for each step as it progresses in a "forward" (left to right) direction.

Step #1: To move from X to P_1

$$f^1(\vec{x}, \vec{w}) \equiv \quad P_1 = \left(X^T \cdot W_1 - T\right) \tag{8}$$

Step #2: P_1 feeds forward to Y

$$f^2(P_1) \quad \equiv \hat{Y} = \left(\frac{1}{1+e^{-\alpha}}\right) \quad : \quad where \quad \alpha = P_1 \tag{9}$$

Step #3: Y feeds forward to P_2

$$f^3(\vec{y}, \vec{w}) \quad \equiv \; P_2 = \left(Y^T \cdot W_2 - T\right) \tag{10}$$

Step #4: P_2 feeds forward to Z

$$f^4(P_2) \quad \equiv \; \hat{Z} = \left(\frac{1}{1+e^{-\alpha}}\right) \quad : \quad where \;\; \alpha = P_2 \tag{11}$$

Step #5: Our complicated function is simply a matter of chaining one result so that it may be used in the next step.

$$\hat{Z} \;=\; f^4\left(f^3\left(f^2\left(f^1\left(X, W\right)\right)\right)\right) \tag{12}$$

In our **Feed-Forward Propagation**, we can now take the values from any numerical system and produce zeros, ones, or probabilities. Remember, in this set of experiments, we are using the concentrations of the 20 amino acids to provide a categorical or binary output, belongs to a) Myoglobin protein family, or b) does not.

Error Back-propagation

Now that we have learned to calculate the output of our neurons using the Feed-Forward process, what if our final answer is incorrect? Can we build a feed back system to determine the weights needed to obtain our desired value of $\hat{z}$? The short answer is yes. The process for determining the weights is known as Error Back-Propagation. Error Back-Propagation, also known as Back-Propagation, is crucial to understanding and tuning a neural network.

Simply stated Back-Propagation is an optimization routine which iteratively calculates the errors that occur at each stage of a neural network. Starting from randomly seeded values for the initial weights, Back-Propagation uses the partial derivatives of the feed forward functions. The chain rule and gradient descent are also used to determine the weights (W_1 *and* W_2) which are propagated through the network to find weights used in the summation step of a neuron.[10]

This thumbnail sketch gives the building blocks to calculate W which can be run until we reach a value that we desire. However the first time the back-propagation is carried out all the weights are chosen randomly. If the weights were set to the same number there would be no change throughout the system.

In the two neuron system, our first step is to generate an error or performance (Perf) function to minimize. If we call d our desired value, we can minimize the square error, a common choice.[11]

Step #1: Performance (Perf)

$$\textbf{Perf} \quad = \quad c \cdot (d - \hat{z})^2 \tag{13}$$

Step #2:

$$\frac{dZ}{dx} \quad = \quad \frac{d\left\{f^4\left(f^3\left(f^2\left(f^1\left(X, W\right)\right)\right)\right)\right\}}{dx} \tag{14}$$

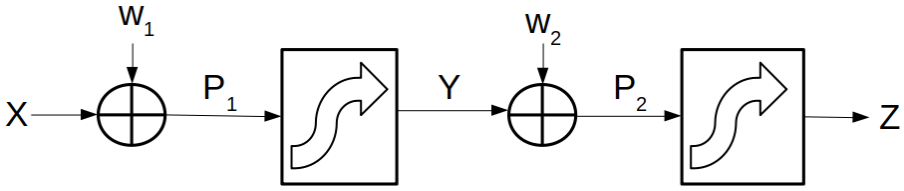


Figure 6: A Two Neuron System

Using the chain-rule, and figure 6.6, *Two Neuron System* as a guide, we can backwards to derive the formulas for error back-propagation. We find:

Step #3: Neuron 2 $\Rightarrow$ 1

$$\frac{\delta Perf}{\delta w_1} \quad = \quad \frac{\delta Perf}{\delta z} \cdot \frac{\delta z}{\delta P_2} \cdot \frac{\delta P_2}{\delta y} \cdot \frac{\delta y}{\delta P_1} \cdot \frac{\delta P_2}{\delta w_1} \tag{15}$$

Step #4: Performance

$$\frac{\delta Perf}{\delta z} \quad = \quad \frac{\delta\left\{\frac{1}{2}\|\overrightarrow{d} - \overrightarrow{z}\|^2\right\}}{\delta z} \quad = \quad d - z \tag{16}$$

Step #5: Substitute $P_2 = \alpha$

$$\frac{\delta z}{\delta P_2} \quad = \quad \frac{\delta\left((1 + e^{-\alpha})^{-1}\right)}{\delta \alpha} \quad = \quad e^{-\alpha} \cdot (1 + e^{-\alpha})^{-2} \tag{17}$$

[10]David Rumelhart, Geoffrey Hinton, & Ronald Williams, Learning Representations By Back-Propagating Errors, Nature, 323, 533-536, Oct. 9, 1986

[11]Ivan N. da Silva, Danilo H. Spatti, Rogerio A. Flauzino, Luisa H. B. Liboni, Silas F. dos Reis Alves, Artificial Neural Networks: A Practical Course, DOI 10.1007/978-3-319-43162-8, 2017

Step #6: Rearrange the right expression

$$\frac{e^{-\alpha}}{(1+e^{-\alpha})^{-2}} = \frac{e^{-\alpha}}{1+e^{-\alpha}} \cdot \frac{1}{1+e^{-\alpha}} \tag{18}$$

Step #7: Add 1 *and* subtract 1

$$= \frac{(1+e^{-\alpha})-1}{1+e^{-\alpha}} \cdot \frac{1}{1+e^{-\alpha}} \tag{19}$$

Step #8: Rearrange to find

$$= \left(\frac{1+e^{-\alpha}}{1+e^{-\alpha}} - \frac{1}{1+e^{-\alpha}}\right)\left(\frac{1}{1+e^{-\alpha}}\right) = \left(1 - \frac{1}{1+e^{-\alpha}}\right)\left(\frac{1}{1+e^{-\alpha}}\right) \tag{20}$$

Step #9: Therefore we find

$$\frac{\delta z}{\delta \alpha} = \frac{\delta\left((1+e^{-\alpha})^{-1}\right)}{\delta \alpha} = \left(1 - \frac{1}{1+e^{-\alpha}}\right)\left(\frac{1}{1+e^{-\alpha}}\right) \tag{21}$$

Nevertheless, we need one more part to ascertain the weights. As the error back-propagation is computed this process does not reveal how much the weights need to be adjusted/changed to compute the next round of weights given their current errors. For this we require one last equation or concept.

Once we compute the weights from our chain rule set of equations we must change the values in the direction proportional to the change in error. This is performed by using gradient descent.

Step #10: Learning Rate

$$\Delta W \; : \; W_{i+1} = W_i - \eta \cdot \frac{\delta Perf}{\delta W} \tag{22}$$

where η is the learning rate for the system. The key to the learning rate is that it must be sought and its range mapped for optimum efficiency. However smaller rates have the advantage of not overshooting the desired minimum/maximum. If the learning rate is too large the values of W may jump wildly and not settle into a max/min. There is a fine balance that must be considered such that the weights are not trapped in a local minimum and wildly oscillate unable to converge.

The last step of *error back-propagation* is simply setting up the derivatives mechanically and is not shown for brevity.

Neural Network Experiment For Binary Classification

```
# Load Libraries
Libraries <- c("dplyr", "knitr", "readr", "caret", "MASS", "nnet", "purrr", "doMC")
for (p in Libraries) {
    library(p, character.only = TRUE)
}
```

```
# Load Data
c_m_TRANSFORMED <- read_csv("./00-data/02-aac_dpc_values/c_m_TRANSFORMED.csv",
                            col_types = cols(Class = col_factor(levels = c("0","1")),
                                             PID = col_skip(),
                                             TotalAA = col_skip()))
```

```r
# Create Training Data
set.seed(1000)
# Stratified sampling
TrainingDataIndex <- createDataPartition(c_m_TRANSFORMED$Class, p = 0.8, list = FALSE)

# Create Training Data
trainingData <- c_m_TRANSFORMED[ TrainingDataIndex, ]
testData     <- c_m_TRANSFORMED[-TrainingDataIndex, ]

TrainingParameters <- trainControl(method = "repeatedcv",
                                   number = 10,
                                   repeats = 5,
                                   savePredictions = "final") # Saves predictions

TuneSizeDecay <- expand.grid(size = c(16, 18, 20),
                             decay = c(1, 0.1, 0.01))
```

Train model with neural networks

```r
end_time - start_time            # Display time
```

```
## Time difference of 5.439274 mins
```

Confusion Matrix and Statistics

```r
NNPredictions <- predict(NNModel, testData)

# Create confusion matrix
cmNN <-confusionMatrix(NNPredictions, testData$Class)
print(cmNN)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 239   9
##          1   4 215
##
##                Accuracy : 0.9722
##                  95% CI : (0.9529, 0.9851)
##     No Information Rate : 0.5203
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9442
##
##  Mcnemar's Test P-Value : 0.2673
##
##             Sensitivity : 0.9835
##             Specificity : 0.9598
```

```
##          Pos Pred Value : 0.9637
##          Neg Pred Value : 0.9817
##              Prevalence : 0.5203
##          Detection Rate : 0.5118
##    Detection Prevalence : 0.5310
##       Balanced Accuracy : 0.9717
##
##        'Positive' Class : 0
##
```

```
NNModel
```

```
## Neural Network
##
## 1873 samples
##   20 predictor
##    2 classes: '0', '1'
##
## Pre-processing: scaled (20), centered (20)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1685, 1686, 1686, 1686, 1686, 1685, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##   16    0.01   0.9675458  0.9349866
##   16    0.10   0.9724570  0.9448152
##   16    1.00   0.9609233  0.9216202
##   18    0.01   0.9703226  0.9405495
##   18    0.10   0.9708545  0.9416022
##   18    1.00   0.9618830  0.9235428
##   20    0.01   0.9703157  0.9405366
##   20    0.10   0.9716003  0.9430995
##   20    1.00   0.9612419  0.9222614
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 16 and decay = 0.1.
```

Obtain List of False Positives & False Negatives

```
fp_fn_NNModel <- NNModel %>% pluck("pred") %>% dplyr::filter(obs != pred)

# Write/save .csv
write.table(fp_fn_NNModel,
            file = "./00-data/03-ml_results/fp_fn_NN.csv",
            row.names = FALSE,
            na = "",
            col.names = TRUE,
            sep = ",")

nrow(fp_fn_NNModel) ## NOTE: NOT UNIQUE NOR SORTED
```

```
## [1] 258
```

False Positive & False Negative Neural Network set

```
keep <- "rowIndex"

fp_fn_NN <- read_csv("./00-data/03-ml_results/fp_fn_NN.csv")

NN_fp_fn_nums <- sort(unique(unlist(fp_fn_NN[, keep], use.names = FALSE)))

length(NN_fp_fn_nums)
```

```
## [1] 81
```

```
NN_fp_fn_nums
```

```
##  [1]    4    6   15   16   46   57   94   97  100  114  115  116  130  136  149
## [16]  150  170  179  182  183  185  249  445  449  453  503  518  522  526  530
## [31]  531  532  534  546  547  566  570  580  592  655  910  913  980 1033 1034
## [46] 1035 1093 1094 1100 1101 1117 1121 1130 1190 1219 1226 1233 1264 1300 1471
## [61] 1510 1522 1575 1576 1579 1585 1587 1594 1608 1618 1621 1693 1697 1734 1771
## [76] 1773 1780 1789 1831 1833 1873
```

```
write_csv(x = as.data.frame(NN_fp_fn_nums),
          path = "./00-data/04-sort_unique_outliers/NN_nums.csv")
```

Neural Network Conclusion

The Neural Network set included a total of 79 unique observations containing both FP and FN.

Accuracy was the primary criteria used to select the optimal model. There were 20 models tested. The neural network was configured with 20 inputs (one for each amino acid), one hidden layer and one output. The hidden layer was tested with either 10, 12, 14, 16, 18, 20 neurons and a array of different decays (1, 0.1, 0.01, 0.001). The caret software has a tuning parameter named `tuneGrid` that allows users to `expand` a set of arrays to a matrix of combinations to be tested. Therefore 20 models were tested with the training data set and the best values were size = 20 and decay = 0.1.

At this time, the author is not aware of any heuristic that gives the proper number of hidden layers and the proper number of neurons in each layer therefore one must search the experimental space for an optimized configuration. If a more thorough search of the experiemntal space was carried out using two or three hidden layers would be investigated. The poor showing of the Neural Network suggests that the data may have some additional decision boundary that is not yet represented by only 20 neurons.

Support Vector Machines For Binary Classification

mcc

3/13/2020

Support Vector Machines for Binary Classification

> "Support Vector Machines should be in the tool bag of every civilized person."
> Patrick Henry Winston [1]

Introduction

Support Vector Machine (SVM) learning is a supervised learning technique that may be used as a binary classification system or to find a regression formula. Support vector machines are a "maximum margin classifier. SVM finds the separating hyperplane with the maximum margin to its closest data points." [2]

Although there are many mathematical approaches to solving SVM, many of which include kernels, let us first describe the Linear SVM used for binary classification. Consider a situation where we need to distinguish between two circumstances, or classes.

In an imaginary biochemistry laboratory, researchers discover a novel enzyme found in different tissues throughout the human body. Biochemists purify the enzyme from several cadavers and several different tissue types. Literature suggests that this newly found enzyme has two isozymes, 1) Alpha has a high reaction rate and 2) Beta has a lower reaction rate. It now seems like a simple task to learn which isozymes you possess. Carry out kinetic enzyme analysis on the purified samples then attempt to classify them.

Once you have carried out the kinetic analysis you then determine the Michaelis–Menten constant, K_M. The K_M constant is plotted on a single axis and produces the graphic below.

Linearly Separable

In test #1, we can see the two isozymes can easily separated by activity alone. Figure 1 demonstrates that the data is *linearly separable.* The dataset is linearly separable if a single straight line can partition the data. In more general terms, "if the classes are linearly separable, it can be shown that the algorithm converges to a separating hyperplane." [3] As Cortes and Vapnik indicate the hyperplane is the decision boundary of any high dimension feature space, considering a hyperplane has one less dimension than its n-dimensional space.

Incidentally, in Patrick Winston's lecture on SVM, he calls SVM the "widest street approach." [4] Why does Professor Winston use this term? There are many possible streets which can be traced but the *goal* is to

[1] Patrick Henry Winston, 6.034 Artificial Intelligence, Fall 2010, Massachusetts Institute of Technology: MIT OpenCourseWare, http://ocw.mit.edu/6-034F10

[2] Nika Haghtalab & Thorsten Joachims, CS4780/5780 - Machine Learning for Intelligent Systems, Fall 2019, Cornell University, Department of Computer Science, https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote09.html

[3] Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning; Data Mining, Inference, and Prediction, https://web.stanford.edu/~hastie/ElemStatLearn/, 2017

[4] Patrick Winston, 6.034 Artificial Intelligence, Fall 2010, Massachusetts Institute of Technology: MIT OpenCourseWare, http://ocw.mit.edu/6-034F10

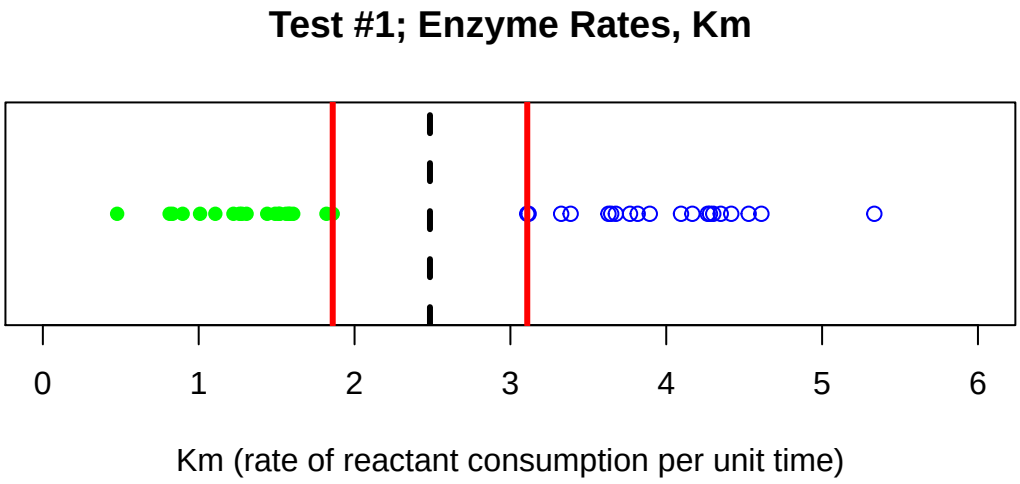


Figure 1: Is linearly separable data.

find the *widest street*. Many streets may be drawn in our example, but requiring the *widest street* leads to one. In fact, "an optimal hyperplane is here defined as the linear decision function with maximal margin between the vectors of the two classes." [5]

[6]

Adding the prosaic phrase *widest street* smartly leads to the idea that a widest decision boundary also has the greatest ability to generalize.

[7]

However in real life, linearly separabale data is rarely the case. Most often the activities are mixed as shown in test #2.

Understanding the hyperplane equation

If we were trying to find:

$$\frac{1}{2}\widehat{W}(X_{\oplus} \; - \; X_{\ominus}) \tag{1}$$

Where $\widehat{W} = \left(\frac{x_1}{||x||}, \frac{x_2}{||x||}\right)$ and $X_{\oplus}$ and $X_{\ominus}$ represent data points that are labeled either positive or negative. Suppose that $X_{\oplus}$ and $X_{\ominus}$ are equidistant from the decision boundary:

[5] C. Cortes, V. Vapnik, Machine Learning, 20, 273-297, 1995
[6] C. Cortes, V. Vapnik, Machine Learning, 20, 273-297, 1995
[7] Allison Horst, University of California, Santa Barbara, https://github.com/allisonhorst/stats-illustrations

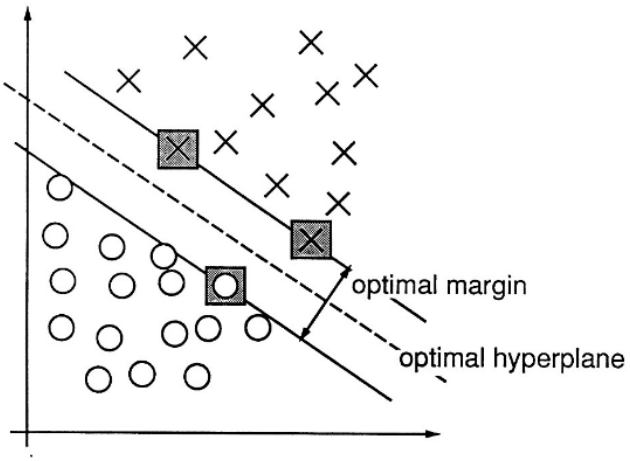


Figure 2. An example of a separable problem in a 2 dimensional space. The support vectors, marked with grey squares, define the margin of largest separation between the two classes.

Figure 2: Vapnick SVM Diagram

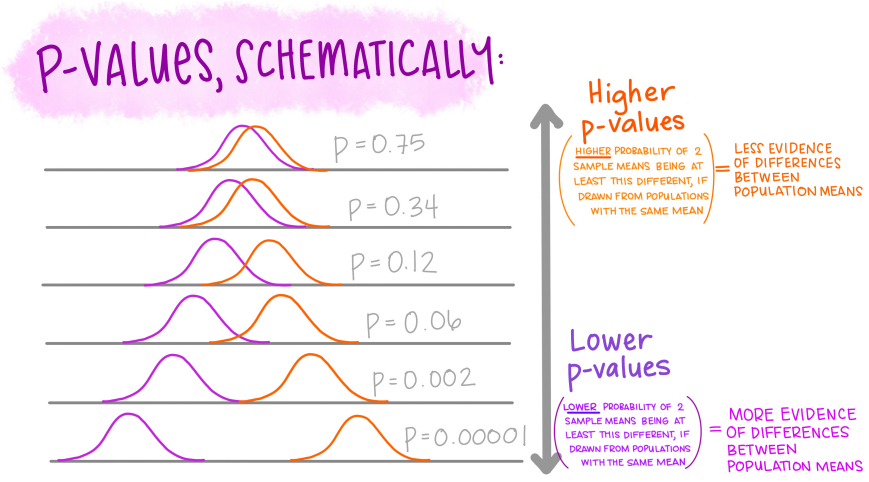


Figure 3: P Values Schematic

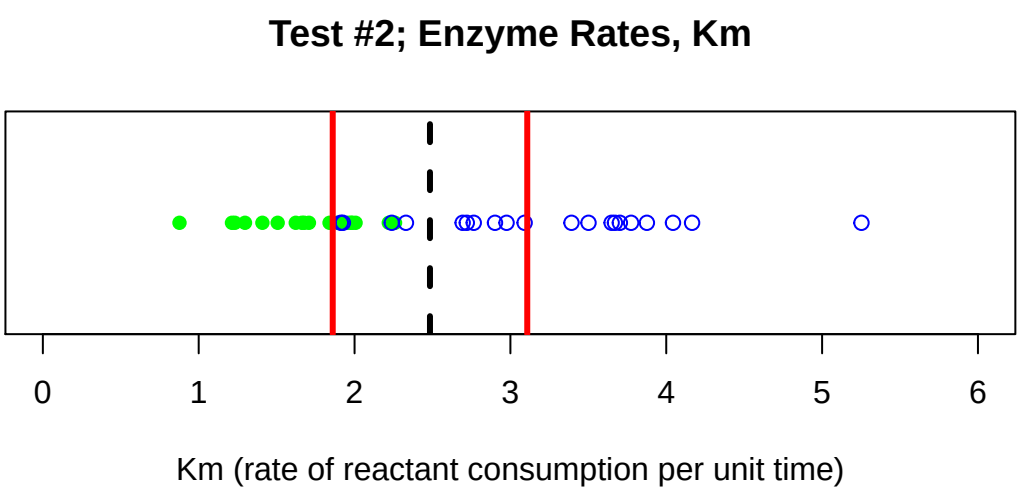


Figure 4: Is not linearly separable data.

Where a represents the region above the hyperplane;

$$W^T X_{\oplus} + b = a \tag{2}$$

and where $-a$ represents the region below the hyperplane or decision boundary.

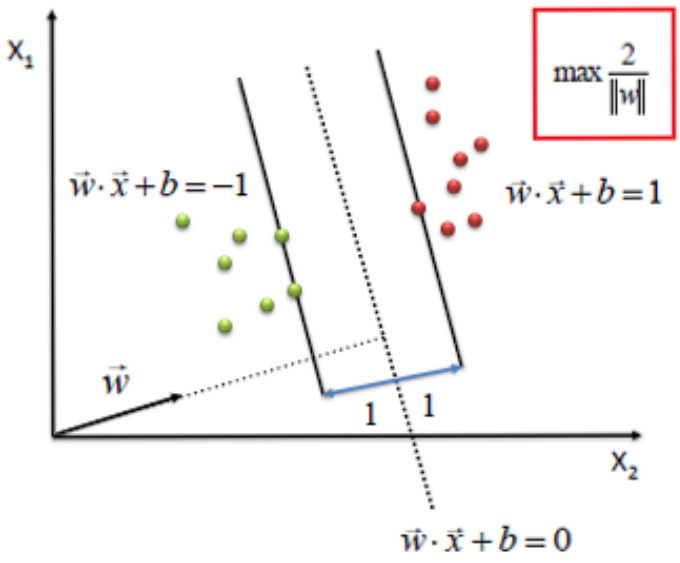
$$W^T X_{\ominus} + b = -a \tag{3}$$

Subtracting the two equations:

$$W^T (X_{\oplus} \; - \; X_{\ominus}) = 2a \tag{4}$$

Divide by the norm of w:

$$\widehat{W}^T (X_{\oplus} \; - \; X_{\ominus}) = \frac{2a}{||W||} \tag{5}$$



8

Soft Margins

In the case above, the activities overlap hence determining which isozyme is Alpha or Gamma is more difficult. In 1995, C. Cortes and V. Vapnik introduced the mathematics and ideas for "Soft Margins" or non-separable training data.[9]

The same is true of an n-dimensional system.

The first mention of an SVM like system is by Vapnik and Lerner in 1963, where the two described an implementation of a non-linear generalization called a Generalized Portrait algorithm.[10] As research has progressed, the types and complexity of SVM implementations have grown to encompass many circumstances. The ability of SVM to deal with different problems and handle different decision boundary shapes has made SVM a potent tool.

Kernel Use

For example, this experiment has chosen to investigate three possible decision boundary shapes for the two-class protein data. The three mathematical constructs which will be tested are:

1. Linear hyperplane (also known as "plain-vanilla"),
2. Curvilinear or polynomial hyperplane and,
3. A radial basis function hyperplane,
4. Sigmoidal.

Linear: $K(x,\ y)\ =\ w^T x + b$

- The linear kernel does not transform the data at all.

Three common SVM kernel formulae investigated are:

[8]Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning; Data Mining, Inference, and Prediction, https://web.stanford.edu/~hastie/ElemStatLearn/, 2017

[9]C.Cortes, V.Vapnik, Machine Learning, 20, 273-297, 1995

[10]V. Vapnik and A. Lerner, 1963. Pattern recognition using generalized portrait method. Automation and Remote Control, 24, 774–780

Polynomial: $K(x_i,\ y)\ =\ (\gamma\ x_i^T\ x_j\ +\ r)^d,\quad \gamma > 0$

- The polynomial kernel has a straightforward non-linear transform of the data.
- Such that $\gamma,\ r$, and d are kernel parameters.

Radial Basis Function (RBF): $K(x_i, x_j)\ =\ exp(-\gamma\ \parallel x_i^T - x_j\ \parallel^2),\quad \gamma > 0$

- The Gaussian RBF kernel which performs well on many data and is a good default

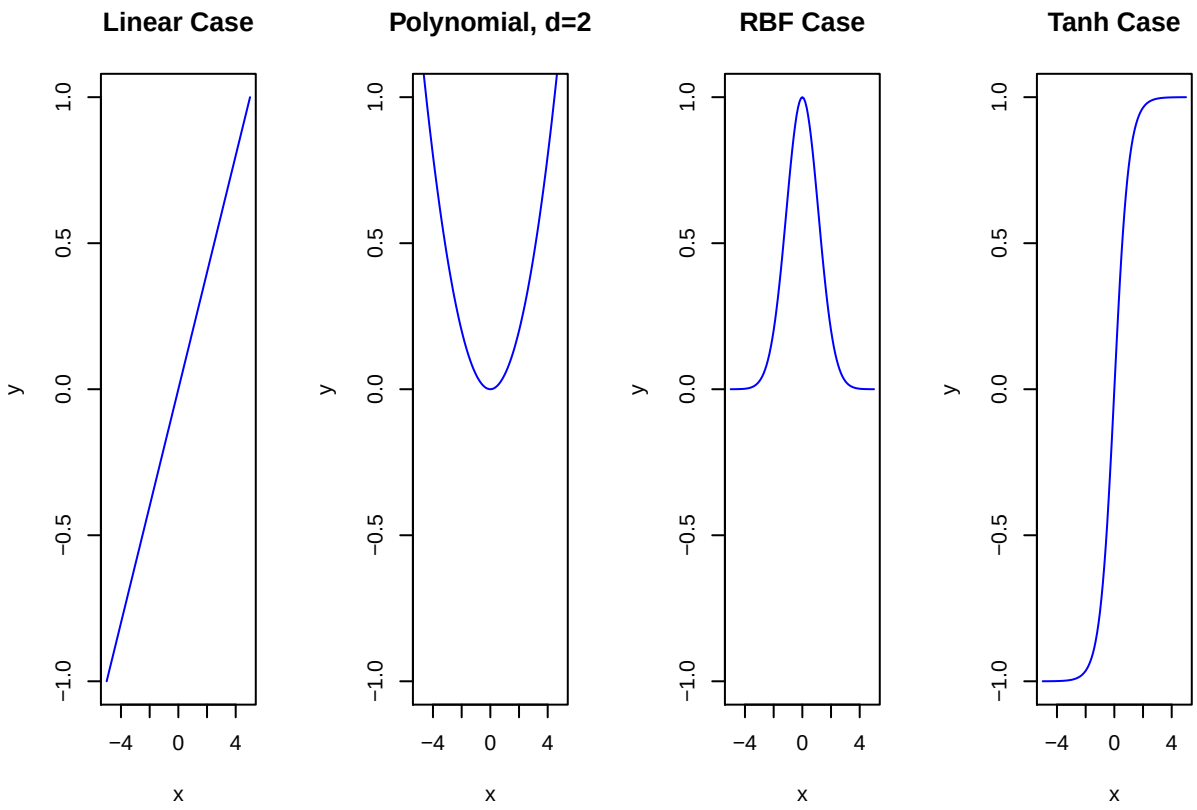
Sigmoidal: $K(x, y)\ =\ \tanh(\gamma\ x^T\ y\ +\ r),\quad \gamma > 0$

- Incidentally, The sigmoid kernel produces an SVM analogous to the activation function similar to a [perceptron] with a sigmoid activation function.[11]

It is essential to note, at this time, there are no reliable rules for which kernel, i.e., boundary shape, to use with any given data set.

[11](https://data-flair.training/blogs/svm-kernel-functions/)

Plots of 4 common SVM boundary shapes:



SVM-Linear Intuition

The simplest form of SVM utilizes a hyperplane as a separating element between the positive and control protein observations. This type of implementation is denoted as SVM-Linear (svm-lin) in this report. Here the mathematics is more easily described and can even be shown with a simple 2-dimensional graphic.

SUPPORT-VECTOR NETWORKS

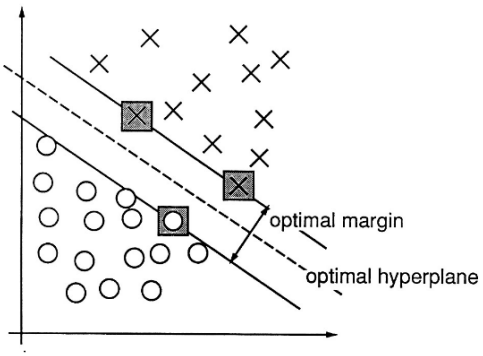


Figure 2. An example of a separable problem in a 2 dimensional space. The support vectors, marked with grey squares, define the margin of largest separation between the two classes.

[12]

Given a set of labeled pairs of data:

$$\{(X_1,\ y),\ ...,\ (X_m,\ y)\}, \quad y \in \{1,\ -1\}, \quad where\ X^{m\ x\ n} \in \Re \tag{6}$$

For mathematical convenience, the labels are a set of values 1 or -1.

Therefore, we may write.

$$f(x_i) = \begin{cases} x \geq 0;\ y = 1 \\ x < 0;\ y = -1 \end{cases} \tag{7}$$

This is no different than is currently done in beginner level algebra. As is shown in the example below, the same is true for higher-dimensional problems.

Will be described and calculated in more detail in this report. However, there are alternative implementations of SVM.

In this experiment, three implementations of SVM have been used. The three are denoted as SVM-Linear (svm-lin), SVM-Polynomial (svm-poly), and SVM-Radial Basis Function (svm-rbf).

The switches in the R/caret software are easy such that one can use a number of kernels by changing the name of method..

with differing amounts of hyperparameters to modify. The intuition for the svm-poly and svm-rbf is also fairly straightforward. Instead of using a linear hyperplane to bisect the hi-dimensional space, which describes the decision boundary, the mathematics for a polynomial curvilinear function or a radial basis function may be utilized.

Yet another measurable difference that was investigated in this experiment was the use of a kernel transformation. It is conceivable to envision a hyperplane with no transformations utilized. Alternatively, the kernel transformations of original data can be used to increase the ability of the function to differentiate between positively and negatively labeled samples. A mathematical treatment can be found by Christopher Burges.[13]

As the usage of SVM grew, different issues presented problems for defining and coding the decision boundary were found. In the simplest case, the data points that sit along the support vector are nicely and neatly on the positive or the negative side. This is known as a hard margin which delineates the decision boundary. In reality, the decision boundary may include positive or negative datapoints that sporadically cross the boundary. In the circumstance where the decision boundary has similar points on either side, a penalty may be enlisted to deter the mathematics from choosing a boundary that includes too many misfit datapoints. In 1995, Support Vector Machines were described by Vladimir Vapnik and Corinna Cortes while at Bell Labs dealt with the soft-margin that occurs in the above situation.[14]

SVM is a non-parametric approach to regression and classification models.

What is Non-parametric?

For that matter, what is parametric learning and models. Just as we have learned that machine learning models can be supervised, unsupervised, or even semi-supervised another characteristic between machine learning models is whether they are parametric or not.

In Webster's dictionary [15] states a *parameter* is

a. Estimation of values which enter into the equation representing the chosen relation

[12]Vladimir Vapnik & Corinna Cortes, Machine Learning, 20, 273-297, 1995
[13]Christopher Burges, Tutorial on Support Vector Machines for Pattern Recognition, D.M. & Knowl. Dis., 2, 121-167, 1998
[14]Vladimir Vapnik & Corinna Cortes, Machine Learning, 20, 273-297, 1995
[15]Webster's third new international dictionary, ISBN 0-87779-201-1, 1986

b. "[An] independent variable through functions of which other functions may be expressed", Frank Yates, a 20th-century statistician

Another excellent explanation of this idea includes;

> Does the model have a fixed number of parameters, or does the number of parameters grow with the amount of training data? The former is called a parametric model, and the latter is called a non-parametric model. Parametric models have the advantage of often being faster to use, but the disadvantage of making stronger assumptions about the nature of the data distributions. Non-parametric models are more flexible, but often computationally intractable for large datasets.[16]

Since Support Vector Machines are best described as a system where increasing the amount of training data, the numbers of parameters may grow as well. Therefore SVM is a non-parametric technique. Considering this idea in more detail, the estimation of the decision boundary does not entirely rely on the estimation of independent values (i.e., the values of the parameters). SVM is fascinating because the decision boundary may only rely on a small number of data points, otherwise known as support vectors.

In short, one guiding idea of SVM is a geometric one. In a binary-class learning system, the metric for the concept of the "best" classification function can be realized geometrically[17] by using a line or a plane (more precisely called a hyperplane when discussing multi-dimensional datasets) to separate the two labeled groups. The hyperplane that separates the labeled sets is also known as a decision boundary.

This decision boundary can be described as having a hard or soft margin. As one might suspect, there are instances where the delineation between the labels is pronounced when this occurs decision boundary produces a hard margin. Alternatively, when the demarcation between the labeled groups is not so well defined by a straight and rigid line, the decision boundary provided is a soft margin. In either case, researchers have built up mathematics to deal with hard and soft margins. As an aside, the use of penalization is one method for coping with data points that impinge on the boundary hyperplane.

By introducing a "soft margin" instead of a hard boundary, we can add a slack variable xi to account for the amount of a violation by the classifier, which later can be minimized.

In short, one guiding idea of SVM is a geometric one. In a binary-class learning system, the metric for the concept of the "best" classification function can be realized geometrically[18] by using a line or a plane (more precisely called a hyperplane when discussing multi-dimensional datasets) to separate the two labeled groups. The hyperplane that separates the labeled sets is also known as a decision boundary.

[Big O notation]Big O notation [19]

Algorithm	Training	Prediction
SVM (Kernel)	$O(n^2p + n^3)$	$O(n_s v_p)$

Where p is the number of features, n_s v_p is the number of support vectors

There are three properties that make SVMs attractive for data scientists:[20]

1. SVMs construct a maximum margin separator—a decision boundary with the largest possible distance to example points. This helps them generalize well.

[16]Kevin P. Murphy, Machine Learning, A Probabilistic Perspective, MIT Press, ISBN 978-0-262-01802-9, 2012
[17]Xindong Wu, et al., Top 10 algorithms in data mining, Knowl Inf Syst, 14:1–37, DOI:10.1007/s10115-007-0114-2, 2008
[18]Xindong Wu, et al., Top 10 algorithms in data mining, Knowl Inf Syst, 14:1–37, DOI:10.1007/s10115-007-0114-2, 2008
[19]Alexandros Karatzoglou, David Meyer, Kurt Hornik, 'Support Vector Machines in R', Journal of Statistical Software, April 2006, Volume 15, Issue 9.
[20]Stuart Russell and Peter Norvig, Artificial Intelligence, A Modern Approach, Third Edition, Pearson, ISBN-13: 978-0-13-604259-4, 2010

2. SVMs create a linear separating hyperplane, but they have the ability to embed the data into a higher-dimensional space, using the so-called kernel trick. Often, data that are not linearly separable in the original input space are easily separable in the higher- dimensional space. The high-dimensional linear separator is actually nonlinear in the original space. This means the hypothesis space is greatly expanded over methods that use strictly linear representations.
3. SVMs are a nonparametric method—they retain training examples and potentially need to store them all. On the other hand, in practice they often end up retaining only a small fraction of the number of examples—sometimes as few as a small constant times the number of dimensions. Thus SVMs combine the advantages of nonparametric and parametric models: they have the flexibility to represent complex functions, but they are resistant to overfitting.

SVM-Linear Model

```
# Load Libraries
Libraries <- c("doMC", "knitr", "readr", "tidyverse", "caret", "kernlab")
for (p in Libraries) {  # Install Library if not present
    if (!require(p, character.only = TRUE)) { install.packages(p) }
    library(p, character.only = TRUE)
}
```

```
# Import data & data handling
c_m_TRANSFORMED <- read_csv("./00-data/02-aac_dpc_values/c_m_TRANSFORMED.csv",
                            col_types = cols(Class = col_factor(levels = c("0", "1")),
                                             PID = col_skip(),
                                             TotalAA = col_skip()))
```

Partition data into training and testing sets

```
set.seed(1000)
index <- createDataPartition(c_m_TRANSFORMED$Class, p = 0.8, list = FALSE)

training_set <- c_m_TRANSFORMED[ index,]
test_set     <- c_m_TRANSFORMED[-index,]

Class_test <- as.factor(test_set$Class)
```

SVM-Linear Training

```
set.seed(1000)
registerDoMC(cores = 3)  # Start multi-processor mode
start_time <- Sys.time() # Start timer

# tuneGrid = svmLinearGrid
svmLinearGrid <- expand.grid(C = c(2^(4.5), 2^5, 2^(5.5)))

# Create model, 10X fold CV repeated 5X
tcontrol <- trainControl(method = "repeatedcv",
```

```
                         number = 10,
                         repeats = 5,
                         savePredictions = "final") # Saves predictions

lin_model_obj <- train(Class ~ .,
                       data = training_set,
                       method = "svmLinear",
                       trControl = tcontrol,
                       tuneGrid = svmLinearGrid)

end_time <- Sys.time()   # End timer
end_time - start_time    # Display time
```

```
## Time difference of 1.845401 mins
```

```
registerDoSEQ()          # Stop multi-processor mode
```

SVM-Linear Model Summary

```
lin_model_obj
```

```
## Support Vector Machines with Linear Kernel
##
## 1873 samples
##   20 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1685, 1686, 1686, 1686, 1686, 1685, ...
## Resampling results across tuning parameters:
##
##   C         Accuracy   Kappa
##   22.62742  0.9482199  0.8961196
##   32.00000  0.9484338  0.8965504
##   45.25483  0.9485402  0.8967687
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 45.25483.
```

SVM-Linear Predict test_set

```
Predicted_test_vals <- predict(lin_model_obj, test_set[, -1])

summary(Predicted_test_vals)
```

```
##   0   1
## 250 217
```

SVM-Linear Confusion Matrix

```
confusionMatrix(Predicted_test_vals, Class_test, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 234  16
##          1   9 208
##
##                Accuracy : 0.9465
##                  95% CI : (0.922, 0.9651)
##     No Information Rate : 0.5203
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8926
##
##  Mcnemar's Test P-Value : 0.2301
##
##             Sensitivity : 0.9286
##             Specificity : 0.9630
##          Pos Pred Value : 0.9585
##          Neg Pred Value : 0.9360
##              Prevalence : 0.4797
##          Detection Rate : 0.4454
##    Detection Prevalence : 0.4647
##       Balanced Accuracy : 0.9458
##
##        'Positive' Class : 1
##
```

SVM-Linear Obtain False Positives & False Negatives

```
fp_fn_svm_linear <- lin_model_obj %>% pluck("pred") %>% dplyr::filter(obs != pred)

# Write out to Outliers folder
write.table(fp_fn_svm_linear,
            file = "./00-data/03-ml_results/fp_fn_svm_linear.csv",
            row.names = FALSE,
            na = "",
            col.names = TRUE,
            sep = ",")

nrow(fp_fn_svm_linear)
```

```
## [1] 482
```

```r
head(fp_fn_svm_linear)
```

```
##          C pred obs rowIndex    Resample
## 1 45.25483    0   1     1223 Fold01.Rep1
## 2 45.25483    0   1     1873 Fold03.Rep1
## 3 45.25483    0   1     1101 Fold05.Rep1
## 4 45.25483    0   1     1618 Fold04.Rep1
## 5 45.25483    0   1     1866 Fold01.Rep1
## 6 45.25483    0   1     1831 Fold05.Rep1
```

SVM-Polynomial Model

Partition data into training and testing sets

```r
set.seed(1000)
index <- createDataPartition(c_m_TRANSFORMED$Class, p = 0.8, list = FALSE)

training_set <- c_m_TRANSFORMED[ index,]
test_set     <- c_m_TRANSFORMED[-index,]

Class_test <- as.factor(test_set$Class)
```

SVM-Poly Training

```r
set.seed(1000)
registerDoMC(cores = 3)  # Start multi-processor mode
start_time <- Sys.time() # Start timer

# Create model, 10X fold CV repeated 5X
tcontrol <- trainControl(method = "repeatedcv",
                         number = 10,
                         repeats = 5,
                         savePredictions = "final") # Saves predictions

poly_model_obj <- train(Class ~ .,
                        data = training_set,
                        method = "svmPoly",
                        trControl= tcontrol)

end_time <- Sys.time()   # End timer
end_time - start_time    # Display time
```

```
## Time difference of 3.100607 mins
```

```r
registerDoSEQ()          # Stop multi-processor mode
```

SVM-Poly Model Summary

```
poly_model_obj
```

```
## Support Vector Machines with Polynomial Kernel
##
## 1873 samples
##   20 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1685, 1686, 1686, 1686, 1686, 1685, ...
## Resampling results across tuning parameters:
##
##   degree  scale  C     Accuracy   Kappa
##   1       0.001  0.25  0.8927921  0.7837449
##   1       0.001  0.50  0.8931101  0.7842397
##   1       0.001  1.00  0.8995164  0.7972922
##   1       0.010  0.25  0.9069985  0.8125193
##   1       0.010  0.50  0.9182148  0.8352852
##   1       0.010  1.00  0.9243031  0.8476521
##   1       0.100  0.25  0.9266532  0.8524643
##   1       0.100  0.50  0.9326311  0.8645594
##   1       0.100  1.00  0.9340187  0.8674061
##   2       0.001  0.25  0.8935368  0.7851036
##   2       0.001  0.50  0.8997298  0.7977161
##   2       0.001  1.00  0.9046473  0.8077461
##   2       0.010  0.25  0.9208824  0.8406987
##   2       0.010  0.50  0.9316686  0.8625657
##   2       0.010  1.00  0.9356167  0.8705585
##   2       0.100  0.25  0.9672198  0.9342616
##   2       0.100  0.50  0.9670042  0.9338334
##   2       0.100  1.00  0.9649755  0.9297728
##   3       0.001  0.25  0.8969559  0.7920424
##   3       0.001  0.50  0.9014433  0.8012428
##   3       0.001  1.00  0.9110610  0.8207620
##   3       0.010  0.25  0.9339083  0.8670639
##   3       0.010  0.50  0.9365781  0.8724848
##   3       0.010  1.00  0.9464057  0.8923335
##   3       0.100  0.25  0.9688252  0.9375066
##   3       0.100  0.50  0.9704278  0.9407533
##   3       0.100  1.00  0.9690403  0.9379858
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were degree = 3, scale = 0.1 and C = 0.5.
```

SVM-Poly Predict test_set

```
Predicted_test_vals <- predict(poly_model_obj, test_set[, -1])
```

```r
summary(Predicted_test_vals)
```

```
##   0   1
## 246 221
```

SVM-Poly Confusion Matrix

```r
confusionMatrix(Predicted_test_vals, Class_test, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 238   8
##          1   5 216
##
##                Accuracy : 0.9722
##                  95% CI : (0.9529, 0.9851)
##     No Information Rate : 0.5203
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9442
##
##  Mcnemar's Test P-Value : 0.5791
##
##             Sensitivity : 0.9643
##             Specificity : 0.9794
##          Pos Pred Value : 0.9774
##          Neg Pred Value : 0.9675
##              Prevalence : 0.4797
##          Detection Rate : 0.4625
##    Detection Prevalence : 0.4732
##       Balanced Accuracy : 0.9719
##
##        'Positive' Class : 1
##
```

SVM-Poly Obtain False Positives & False Negatives

```r
fp_fn_svm_poly <- poly_model_obj %>% pluck("pred") %>% dplyr::filter(obs != pred)

# Write CSV in R
write.table(fp_fn_svm_poly,
            file = "./00-data/03-ml_results/fp_fn_svm_poly.csv",
            row.names = FALSE,
            na = "",
            col.names = TRUE,
            sep=",")

nrow(fp_fn_svm_poly)
```

```
## [1] 277
```

```
head(fp_fn_svm_poly)
```

```
##   degree scale   C pred obs rowIndex    Resample
## 1      3   0.1 0.5    0   1     1576 Fold01.Rep2
## 2      3   0.1 0.5    1   0      182 Fold09.Rep4
## 3      3   0.1 0.5    1   0      445 Fold06.Rep5
## 4      3   0.1 0.5    1   0      531 Fold09.Rep4
## 5      3   0.1 0.5    1   0      115 Fold05.Rep2
## 6      3   0.1 0.5    0   1     1780 Fold02.Rep2
```

SVM-RBF Model

```
# Load Libraries
rm(list = ls())
Libraries = c("doMC", "knitr", "readr", "tidyverse", "caret", "kernlab")

for(p in Libraries){  # Install Library if not present
    if(!require(p, character.only = TRUE)) { install.packages(p) }
    library(p, character.only = TRUE)
}
opts_chunk$set(cache = TRUE)
```

Import data & data handling

```
c_m_TRANSFORMED <- read_csv("./00-data/02-aac_dpc_values/c_m_TRANSFORMED.csv",
                            col_types = cols(Class = col_factor(levels = c("0","1")),
                                             PID = col_skip(), TotalAA = col_skip()))
#View(c_m_TRANSFORMED)
```

Partition data into training and testing sets

```
set.seed(1000)
index <- createDataPartition(c_m_TRANSFORMED$Class, p = 0.8, list = FALSE)

training_set <- c_m_TRANSFORMED[ index,]
test_set     <- c_m_TRANSFORMED[-index,]

Class_test <- as.factor(test_set$Class)
```

SVM-RBF Training

```
set.seed(1000)
registerDoMC(cores = 3)  # Start multi-processor mode
start_time <- Sys.time() # Start timer

# Create tuneGrid: Cost
```

```
tune.Grid = data.frame(expand.grid(C = 2^(seq(-5, 15, 2))))

# Create: 10X fold CV repeated 5X
tcontrol <- trainControl(method = "repeatedcv",
                         number = 10,
                         repeats = 5,
                         savePredictions = "final") # Save predictions

rbf_model_obj <- train(Class ~ .,
                   data = training_set,
                   method = "svmRadialCost",
                   tuneGrid = tune.Grid,
                   trControl= tcontrol)

end_time <- Sys.time()    # End timer
end_time - start_time     # Display time
```

```
## Time difference of 1.12283 mins
```

```
registerDoSEQ() # Stop multi-processor mode
```

SVM-RBF Model Summary

```
rbf_model_obj
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1873 samples
##   20 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1685, 1686, 1686, 1686, 1686, 1685, ...
## Resampling results across tuning parameters:
##
##   C           Accuracy   Kappa
##   3.1250e-02  0.9144721  0.8278653
##   1.2500e-01  0.9511031  0.9018251
##   5.0000e-01  0.9704221  0.9406791
##   2.0000e+00  0.9699937  0.9398269
##   8.0000e+00  0.9740562  0.9480186
##   3.2000e+01  0.9720247  0.9439679
##   1.2800e+02  0.9711714  0.9422795
##   5.1200e+02  0.9715980  0.9431324
##   2.0480e+03  0.9714916  0.9429197
##   8.1920e+03  0.9715980  0.9431330
##   3.2768e+04  0.9714916  0.9429209
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 8.
```

SVM-RBF Predict test_set

```
Predicted_test_vals <- predict(rbf_model_obj, test_set[, -1])

summary(Predicted_test_vals)
```

```
##   0   1
## 245 222
```

SVM-RBF Confusion Matrix

```
confusionMatrix(Predicted_test_vals, Class_test, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 238   7
##          1   5 217
##
##                Accuracy : 0.9743
##                  95% CI : (0.9555, 0.9867)
##     No Information Rate : 0.5203
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9485
##
##  Mcnemar's Test P-Value : 0.7728
##
##             Sensitivity : 0.9688
##             Specificity : 0.9794
##          Pos Pred Value : 0.9775
##          Neg Pred Value : 0.9714
##              Prevalence : 0.4797
##          Detection Rate : 0.4647
##    Detection Prevalence : 0.4754
##       Balanced Accuracy : 0.9741
##
##        'Positive' Class : 1
##
```

SVM-RBF Obtain False Positives & False Negatives

```
fp_fn_svmRadialCost <- rbf_model_obj %>% pluck("pred") %>% dplyr::filter(obs != pred)

# Write CSV in R
write.table(fp_fn_svmRadialCost,
            file = "./00-data/03-ml_results/fp_fn_svmRbf.csv",
```

```
            row.names = FALSE,
            na = "",
            col.names = TRUE,
            sep=",")

nrow(fp_fn_svmRadialCost)
```

```
## [1] 243
```

```
head(fp_fn_svmRadialCost)
```

```
##   C pred obs rowIndex    Resample
## 1 8    1   0      522 Fold01.Rep1
## 2 8    0   1     1579 Fold07.Rep1
## 3 8    0   1     1585 Fold02.Rep1
## 4 8    0   1     1587 Fold07.Rep1
## 5 8    1   0      141 Fold07.Rep3
## 6 8    1   0       94 Fold03.Rep1
```

SVM Conclusion

SVM has shown that it is very versatile. Using the same amino acid dataset and by changing the type of function used to build the decision boundary one can get results in the high 90%. Even the linear SVM with no kernel transformation is able to find a hyperplane that with no more tuning than choosing a large enough range for the cost function.

It was seen that using a range for the cost starting at 2^5 and progressing toward 2^{-15}, stepping by one log, a wide range can be tested. Once the cost function range is narrowed a finer tuning can be carried out using one-quarter or one-half log steps. This further narrowing of the cost function may not need to be done since the one log steps seems anecdotally sufficient. This is an easy and powerful tool.

Results

mcc

3/13/2020

Results

I have studied six different M.L. algorithms using protein amino acid percent composition data from two classes. Class number 1 is Myoglobin proteins, which is the positive control set. While the second class is a control group (0) of human proteins that do not have Fe binding centers.

Group	Class	N of Class	Range of Groups
Controls	0 or (-)	1216	1, ..., 1216
Myoglobin	1 or (+)	1124	1217, ..., 2340

The Six M.L Algorithms consist of:

Name	Type	Output Used For Graphing
Principal Component Analysis	Unsupervised	Anomalies > Abs(3σ)
Logistic Regression	Supervised	FP & FN
SVM-linear	Supervised	FP & FN
SVM-polynomial kernel	Supervised	FP & FN
SVM-radial basis function kernel	Supervised	FP & FN
Neural Network	Supervised	FP & FN

==========

Scatter Plots of Anomalies Vs. FP & FN Outputs

To obtain False-Positive (FP) and False-Negatives (FN) from sets:

1. False-Postives $\stackrel{\text{def}}{=}$ $\{\text{obs} = 0 \quad \wedge \quad \text{pred} = 1\}$
2. False-Negatives $\stackrel{\text{def}}{=}$ $\{\text{obs} = 1 \quad \wedge \quad \text{pred} = 0\}$

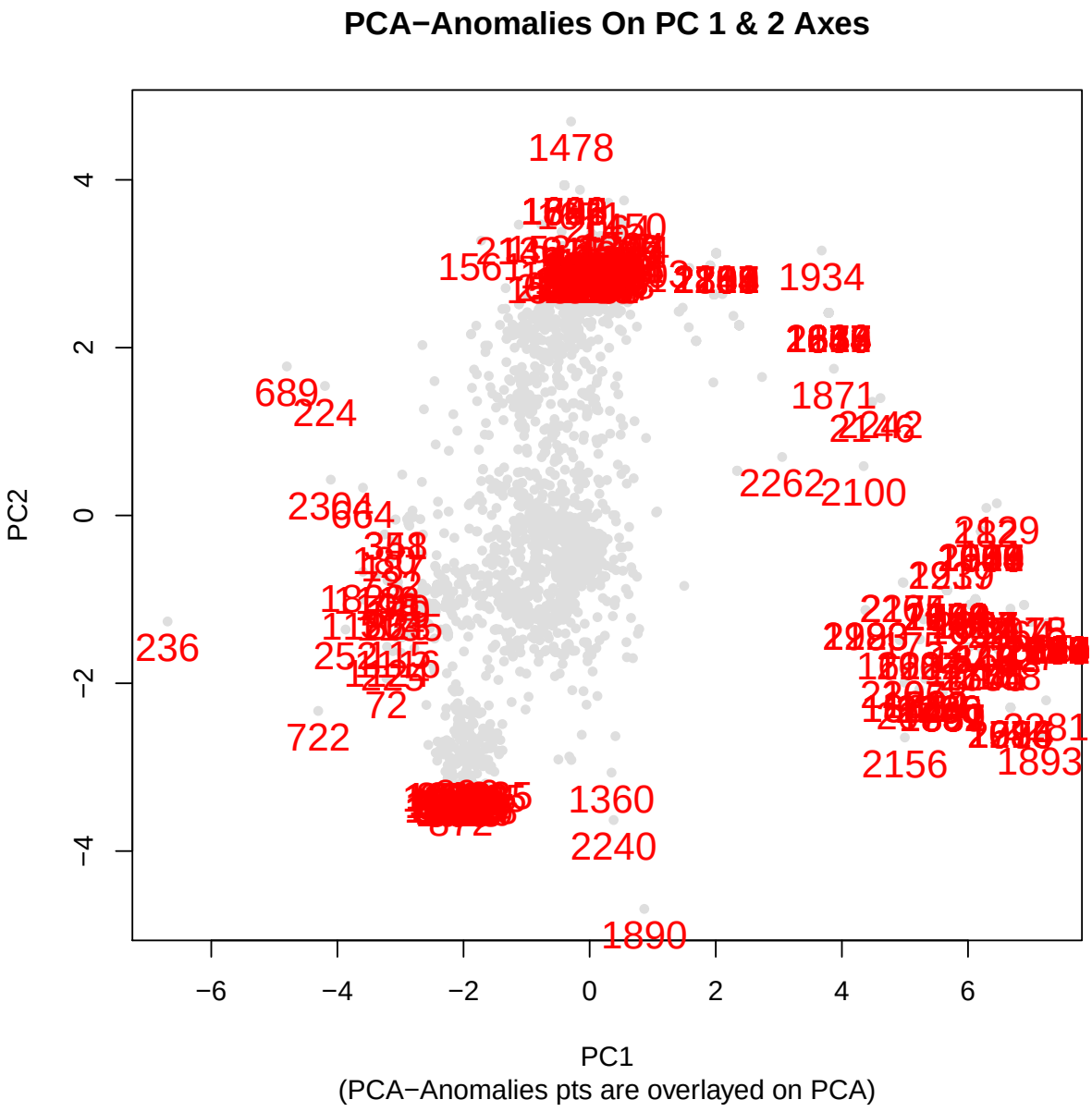
Anomalies Inner Joined with PC

```
# Load Libraries
Libraries = c("knitr", "readr")
for(p in Libraries){
    library(p, character.only = TRUE)
}
opts_chunk$set(fig.align = "center", cache=TRUE)
```

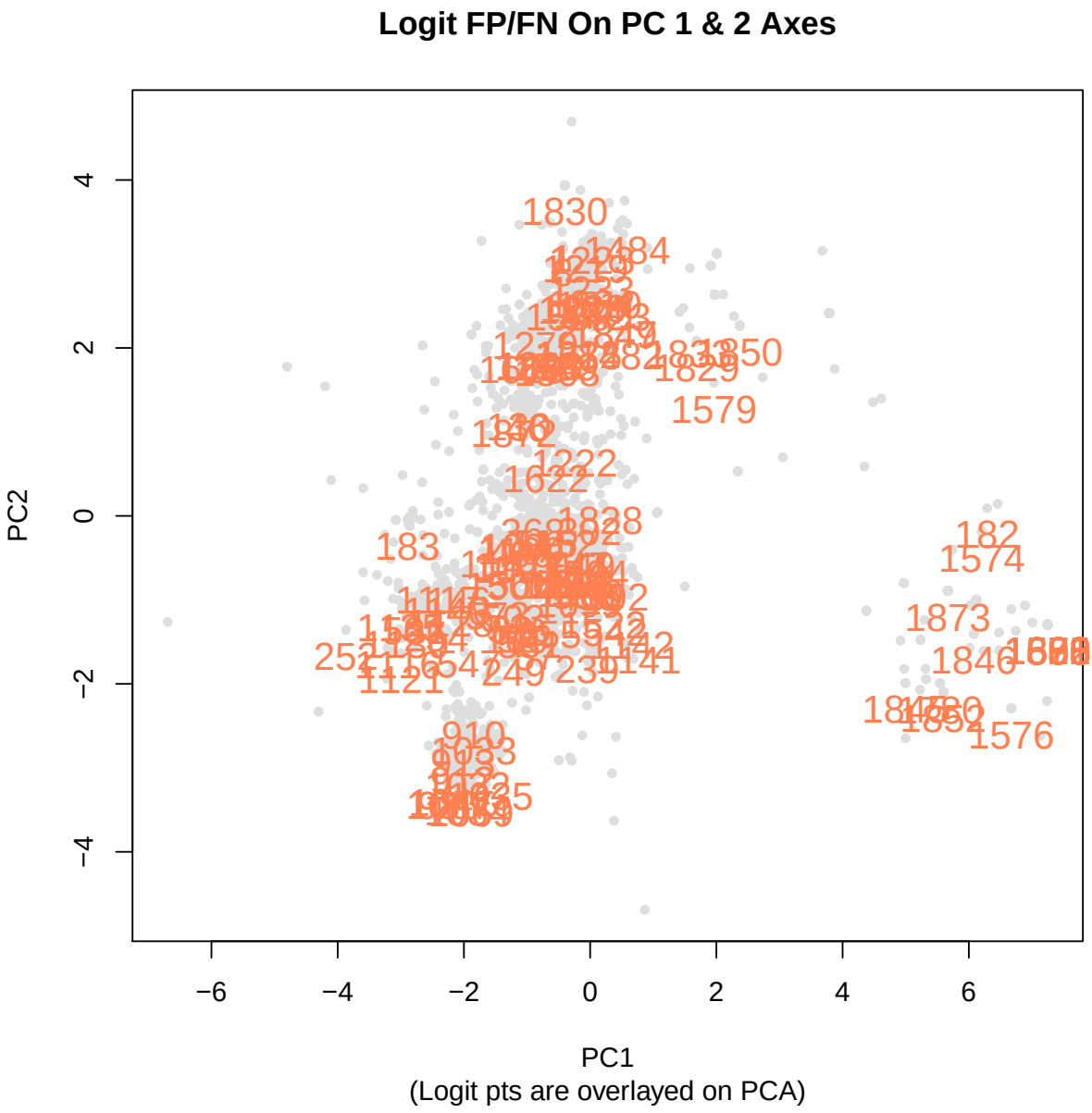
```r
# Prepare PCA: PC1 and PC2 for all 2340 proteins
norm_c_m_20aa <- read_csv("./00-data/03-ml_results/norm_c_m_20aa.csv")

pca_values <- prcomp(norm_c_m_20aa)

row_pc12 <- cbind(rowNum = 1:2340, PC1 = pca_values$x[,1], PC2 = pca_values$x[,2])
# dim(row_pc12)
write.table(row_pc12,
            file = "./00-data/05-joins_plots/row_pc12.txt",
            sep = ",",
            row.names = F)
```

```r
# Inner-join (merge) with PC 1&2 (row_pc12)
logit_pc    <- merge(x = logit_nums,    y = row_pc12, by = "rowNum")
pca_pc      <- merge(x = pca_outliers,  y = row_pc12, by = "rowNum")
nn_pc       <- merge(x = nn_nums,       y = row_pc12, by = "rowNum")
svm_lin_pc  <- merge(x = svm_lin_nums,  y = row_pc12, by = "rowNum")
svm_poly_pc <- merge(x = svm_poly_nums, y = row_pc12, by = "rowNum")
svm_rbf_pc  <- merge(x = svm_rbf_nums,  y = row_pc12, by = "rowNum")
```

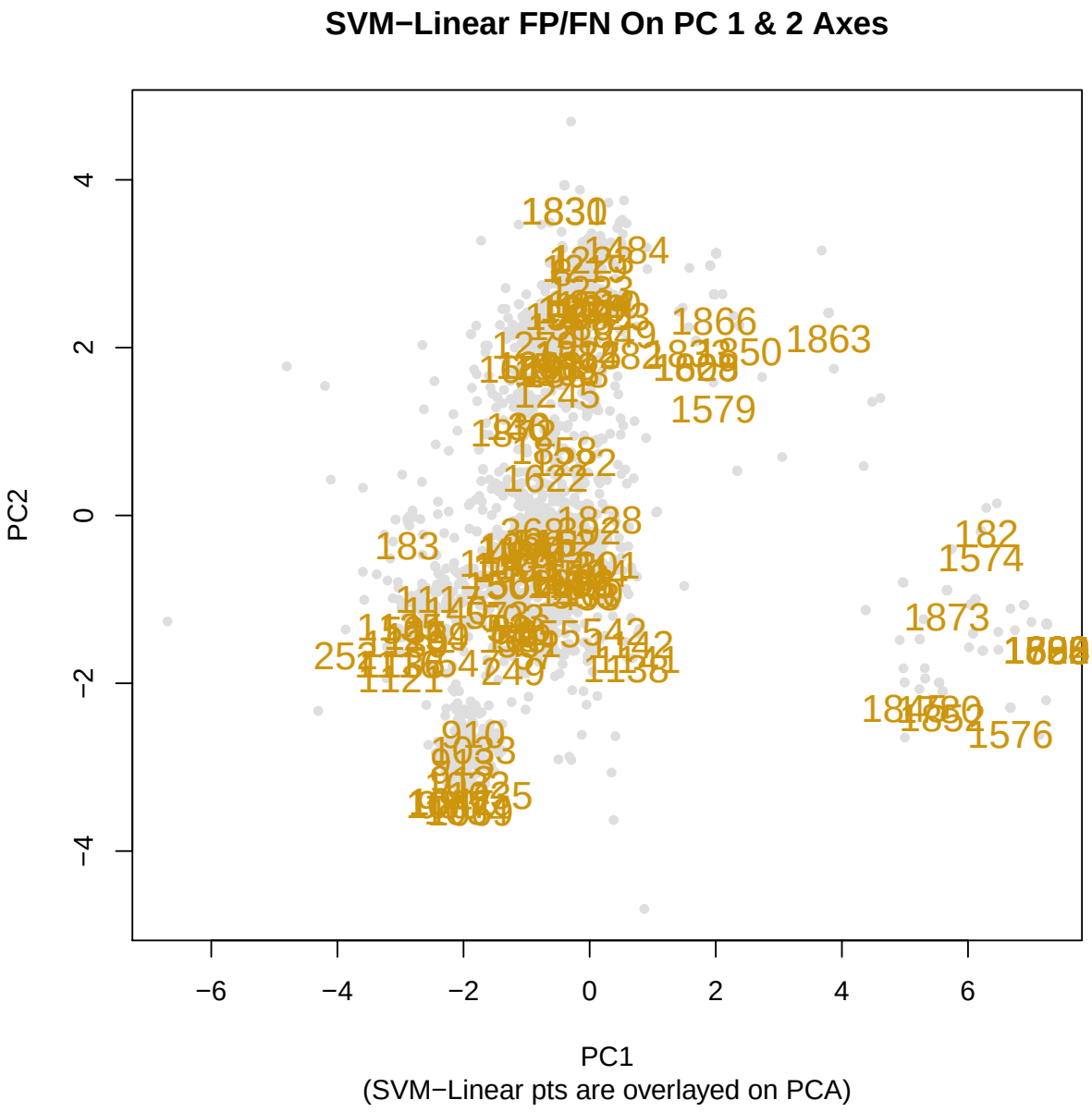
PCA-Anomalies Plot



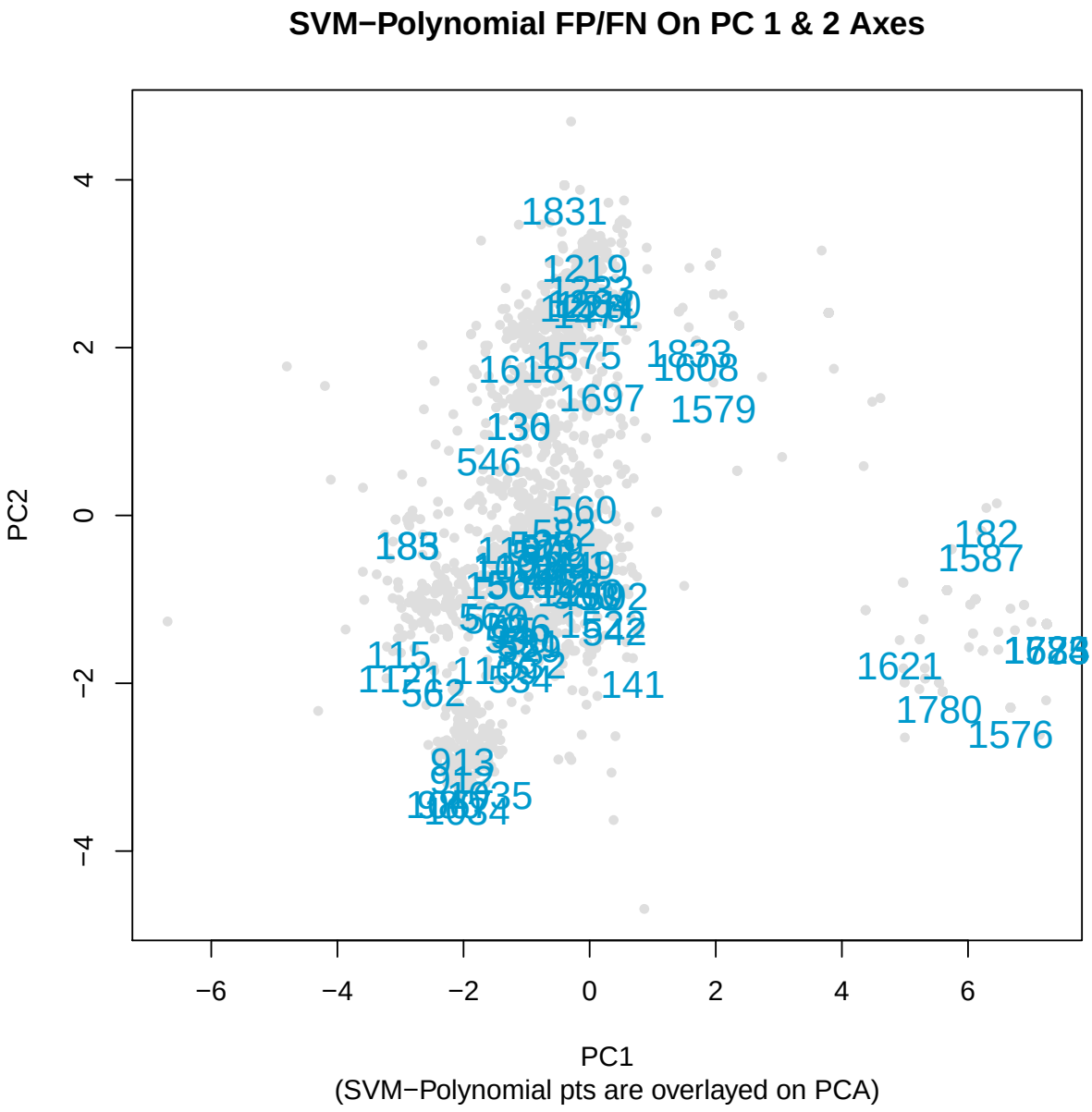
Logit Plot


Logit FP/FN On PC 1 & 2 Axes
PC2
PC1
(Logit pts are overlayed on PCA)
4
2
0
−2
−4
−6
−4
−2
0
2
4
6
1830
1484
1829
1579
1222
1622
183
182
1574
1873
1576

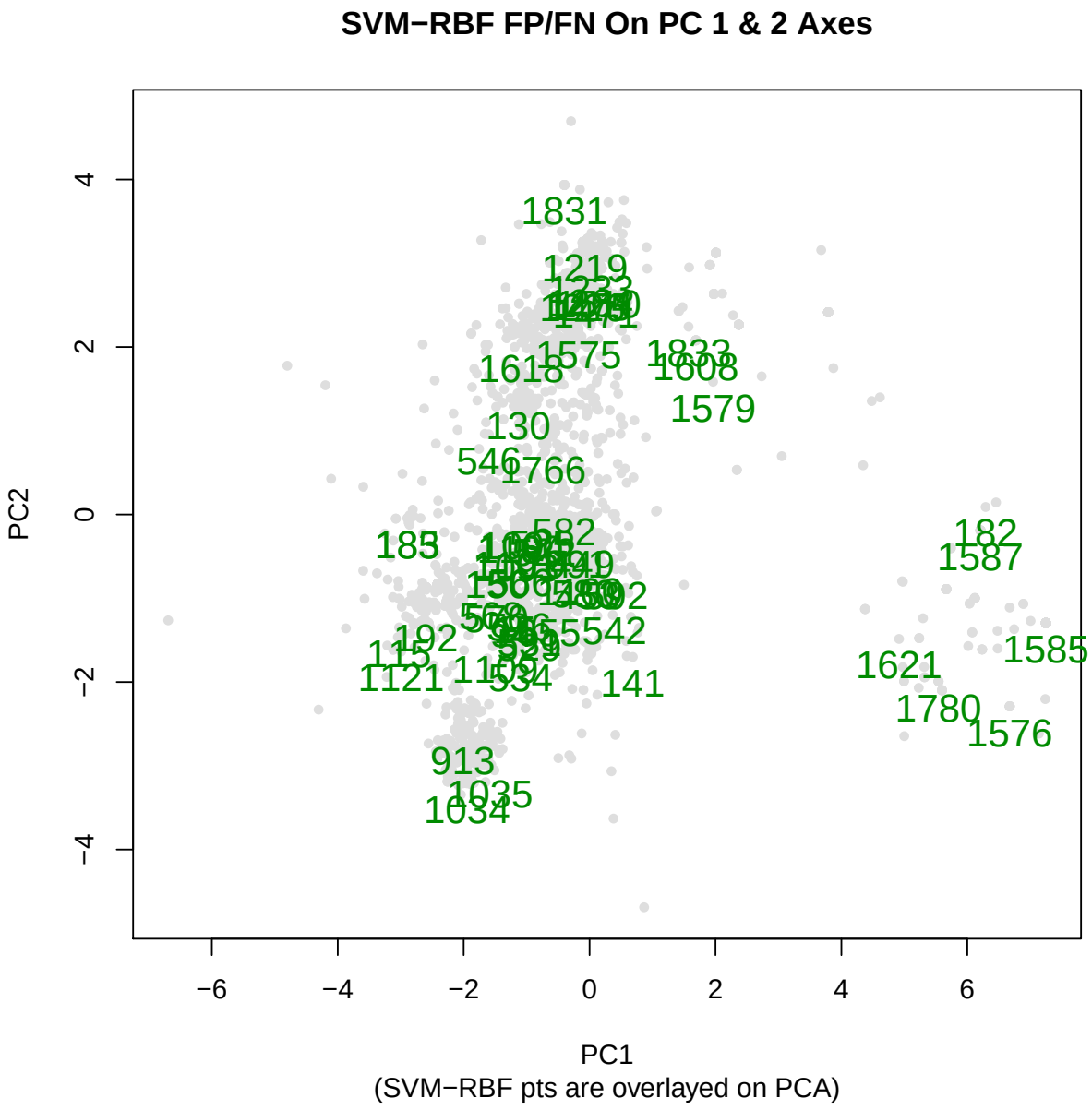
SVM-Linear Plot



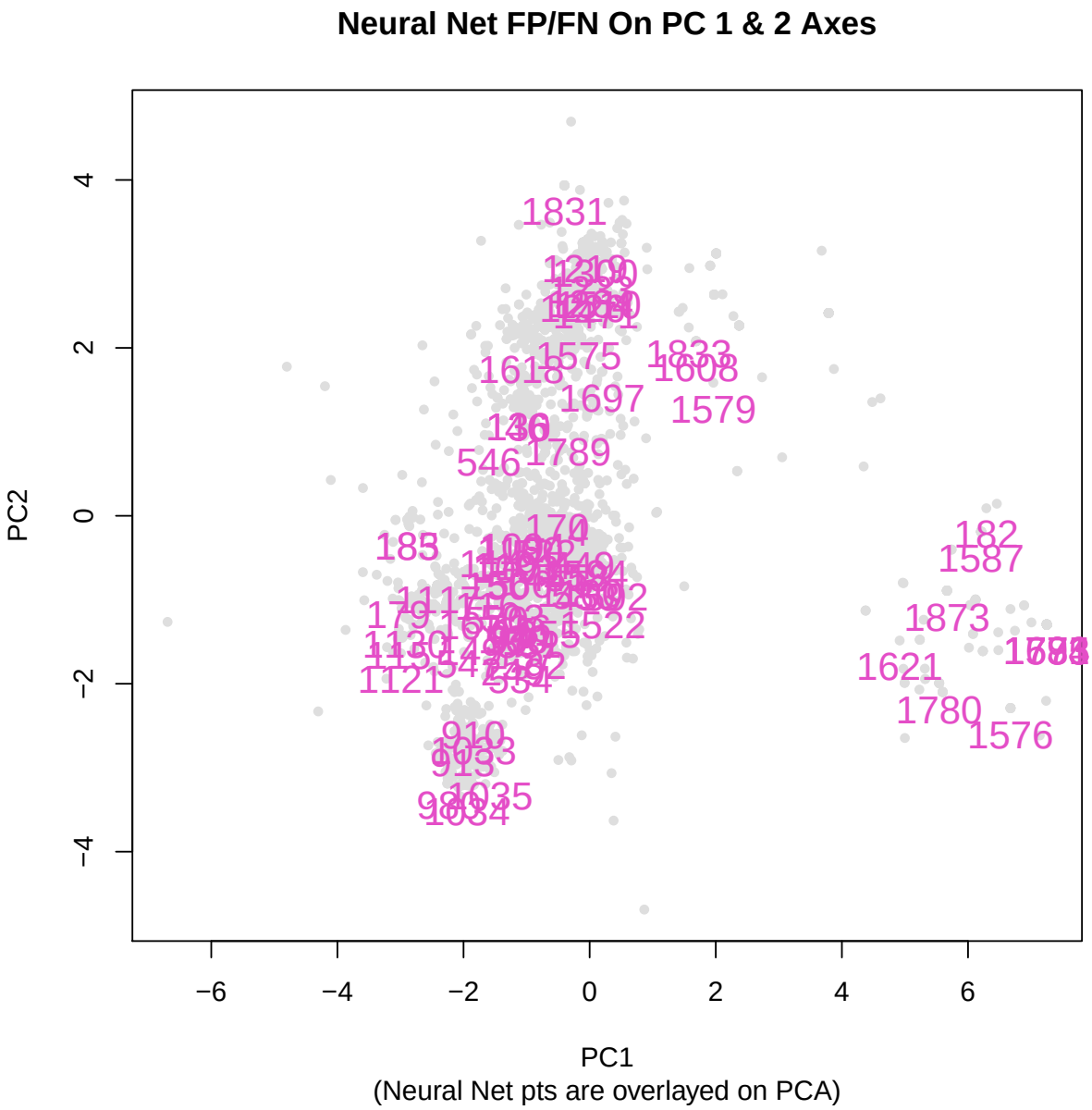
SVM-Polynomial Plot



SVM-Radial Basis Function Plot



Neural Network Function Plot



Statistical Learning Method Vs Total Number of FP/FN

Statistical Method	Unique
Principal Componnent Analysis	460
Logit	119
SVM Linear	125
SVM Polynomial	70
SVM Radial Basis Function	58
Deep Learning	79

Comparison of Machine Learning Accuracies

```
# Load Libraries
Libraries <- c("doMC", "knitr", "readr", "caret", "nnet", "caretEnsemble", "e1071", "kernlab")
for (p in Libraries) {
    library(p, character.only = TRUE)
}
knitr::opts_chunk$set(echo = TRUE, fig.align="center")
```

```
# Import data & data handling
c_m_TRANSFORMED <- read_csv("./00-data/02-aac_dpc_values/c_m_TRANSFORMED.csv",
                            col_types = cols(Class = col_factor(levels = c("0", "1")),
                                             PID = col_skip(),
                                             TotalAA = col_skip()))
```

```
# Partition data into training and testing sets
set.seed(1000)
index <- createDataPartition(c_m_TRANSFORMED$Class, p = 0.8, list = FALSE)

training_set <- c_m_TRANSFORMED[ index,]
test_set     <- c_m_TRANSFORMED[-index,]

Class_test <- as.factor(test_set$Class)
```

Stacking Algorithms - Run multiple algorithms in one call.

```
## Warning in caretList(Class ~ ., data = training_set, trControl = trainControl, :
## Duplicate entries in methodList. Using unqiue methodList values.
```

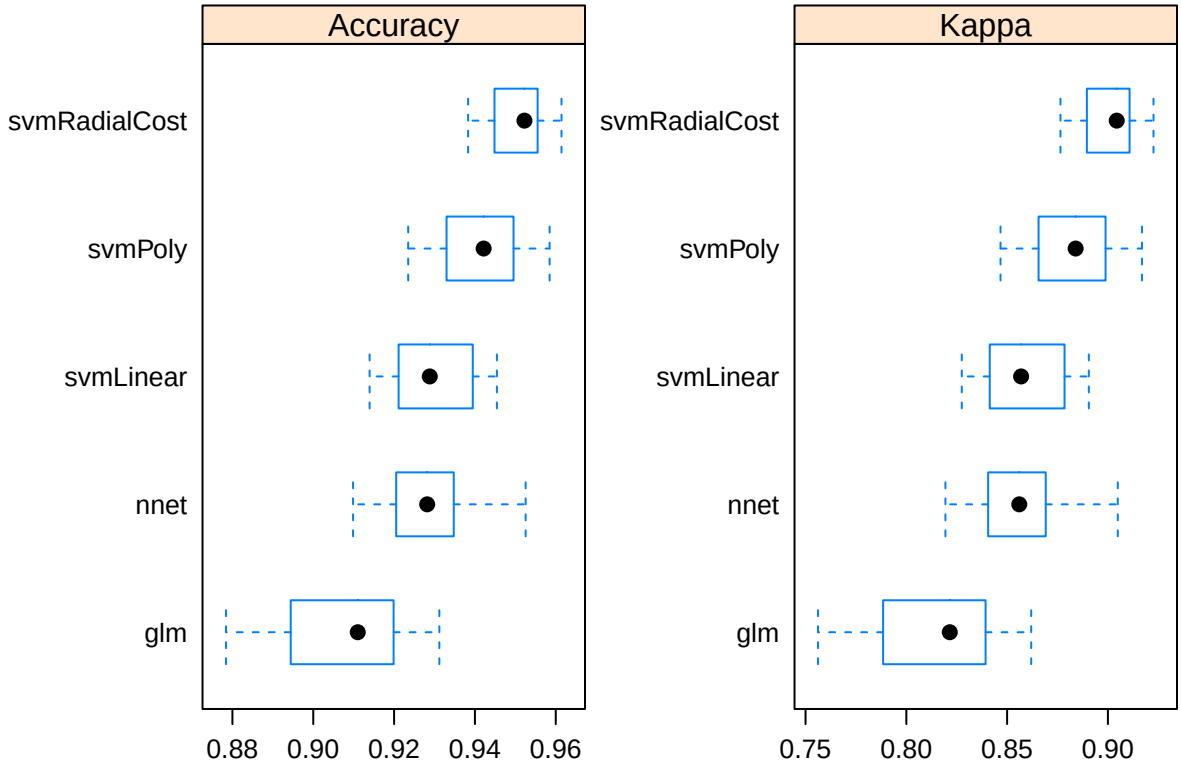
```
## # weights:  287
## initial  value 1622.518111
## iter  10 value 591.200854
## iter  20 value 331.103547
## iter  30 value 281.561829
## iter  40 value 215.474048
## iter  50 value 179.866748
## iter  60 value 111.221062
## iter  70 value 76.163156
## iter  80 value 55.582062
```

```
## iter  90 value 46.458483
## iter 100 value 39.618132
## final  value 39.618132
## stopped after 100 iterations
```

```
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: glm, nnet, svmLinear, svmPoly, svmRadialCost
## Number of resamples: 10
##
## Accuracy
##                    Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## glm           0.8784104 0.8954626 0.9110056 0.9078127 0.9184104 0.9311981    0
## nnet          0.9098458 0.9208185 0.9281899 0.9286350 0.9334118 0.9525504    0
## svmLinear     0.9139466 0.9215485 0.9288256 0.9292275 0.9373989 0.9454330    0
## svmPoly       0.9234875 0.9346085 0.9421534 0.9415091 0.9485244 0.9584816    0
## svmRadialCost 0.9383155 0.9454330 0.9522539 0.9510603 0.9553610 0.9614243    0
##
## Kappa
##                    Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## glm           0.7561869 0.7905961 0.8215603 0.8152905 0.8365124 0.8619539    0
## nnet          0.8193980 0.8411102 0.8560213 0.8569694 0.8665330 0.9048933    0
## svmLinear     0.8275022 0.8422492 0.8569400 0.8578486 0.8743642 0.8905356    0
## svmPoly       0.8467331 0.8688294 0.8839959 0.8827193 0.8967519 0.9169053    0
## svmRadialCost 0.8764532 0.8906928 0.9043832 0.9019157 0.9104916 0.9226237    0
```

Plot the resamples output to compare the models.



Mean Accuracies of M.L. Techniques, n=10

Rank	M.L. Technique	Mean Accuracy
1	SVM-RBF	0.9510603
2	SVM-Poly	0.9415091
3	SVM-Lin	0.9292275
4	NN w 20 Neurons	0.9286350
5	Logit	0.9078127

Conclusion

mcc

3/13/2020

Conclusion

Biplot1.annotated.png

This research was carried out to investigate if there is any relationship between the unsupervised machine learning technique of Principal Component Analysis and five alternate methods. The five alternative methods included 3 Support Vector Machines, a Neural Network and Logistic Regression.

The comparison between PCA and the five alternate M.L. methods was done be using only the first two Principal Components, since these two comprise approximately 50% of the error of total sum of squares error. See table 8.1.

Table 8.1, The Six M.L Algorithms consist of:

Name	Type	Output Used For Graphing
Principal Component Analysis	Unsupervised	Anomalies > Abs(3σ)
Logistic Regression	Supervised	FP & FN
SVM-linear	Supervised	FP & FN
SVM-polynomial kernel	Supervised	FP & FN
SVM-radial basis function kernel	Supervised	FP & FN
Neural Network w 20 Neurons	Supervised	FP & FN

Comparison of PCA Anomalies

of this study was that there would be a correlation or overlap of the anomalies which occurred

Statistical Learning Method Vs Total Number of FP/FN

Statistical Method	Unique
Principal Component Analysis	460
Logit	119
SVM Linear	125
SVM Polynomial	70
SVM Radial Basis Function	58
Deep Learning	79

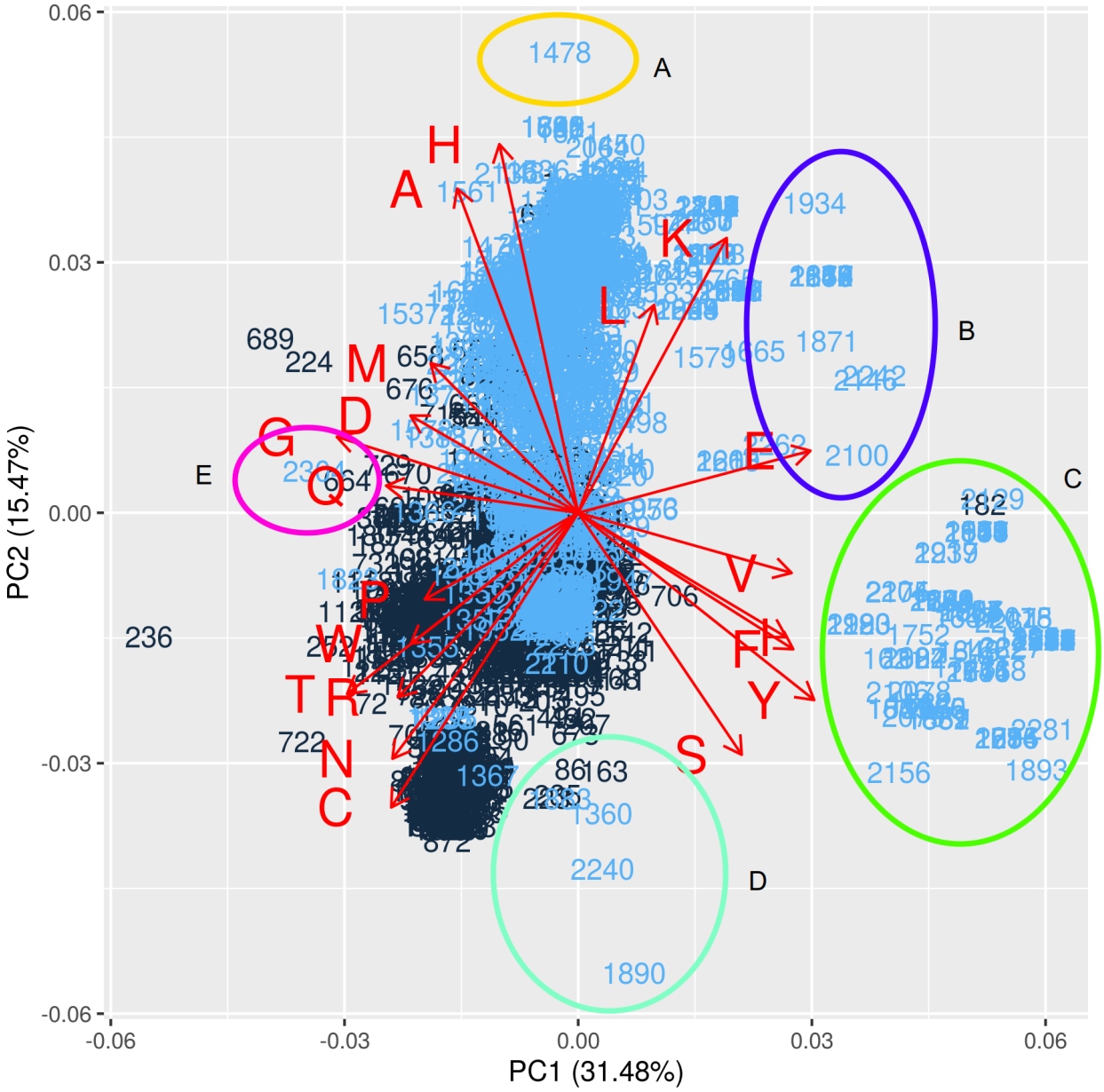


Figure 1: Biplot of Problem regions for all M.L. models

Comparison of Accuracy Measurements

Mean Accuracies of M.L. Techniques, n=10

Rank	M.L. Technique	Mean Accuracy
1	SVM-RBF	0.9510603
2	SVM-Poly	0.9415091
3	SVM-Lin	0.9292275
4	NN w 20 Neurons	0.9286350
5	Logit	0.9078127

One main tenet of fitting any model to a set of number is that it is helpful and even important to know about the distribution(s) of the number set(s). However without fore knowledge about the distribution(s) modeling may be difficult. Therefore the basic assumption throughout this exercise is that we are teaching models without knowing the probability distributions. Also, using several different machine learning approaches is advantageous.

To help us with the task of fitting unknown distributions must use cost or error functions to minimize problems with fitting a model. In many cases in the research the cost function is a summation of square errors. However it is a fine line between over-fitting and under-fitting or model to our data set. To achieve a best fit the model must be flexible. There is a trade off, between reducing the variance and a commensurate increase of bias.

> There are many ways to define such models, but the most important distinction is this: does the model have a fixed number of parameters, or does the number of parameters grow with the amount of training data? The former is called a parametric model, and the latter is called a non-parametric model. Parametric models have the advantage of often being faster to use, but the disadvantage of making stronger assumptions about the nature of the data distributions. Non-parametric models are more flexible, but often computationally intractable for large data sets.
>
> Kevin Murphy[1]

We have seen that it is helpful to transform the raw data for several reasons. The first example was during the Exploratory Data Analysis chapter that found skewness was found in three of the amino acid columns or features. The skewness of these three features was greater than 2.0. Using common techniques such as taking the square-root or taking the reciprocal or log transformations are common and easy to achieve.

Alternatively, transformations such as the kernel transformation is used strategically with Support Vector Machines. Of the three examples of SVM, the SVM-Linear did not use a kernel transformation and its performance while being good compared to the Logistic Regression had accuracy less than the SVM which used Kernel transformations. It is possible to use any number of transformations such as the polynomial and the radial-basis function. These alternatives allow non-linear decision boundaries.

[1]Kevin P.Murphy, Machine learning : a probabilistic perspective, 2012, ISBN 978-0-262-01802-9

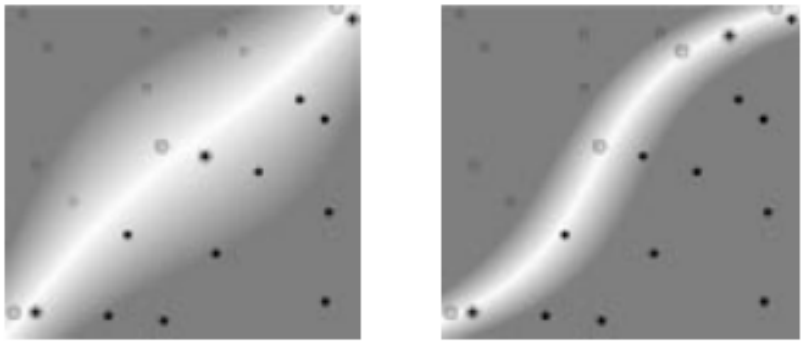


Figure 9. Degree 3 polynomial kernel. The background colour shows the shape of the decision surface.

[2]

A Neural Network was set up and experimented with the protein data set. The NN was set to explore a small set of the total possible experimental space. The number of neurons was tested between 10 and 20 with an interval of 2. It was surprising to see that the NN was fourth out of five in accuracy. It is surprising due to the idea that two neurons should be somewhat equivalent to a decision boundary with 20 linear boundaries. However only one hidden layer was attempted. In the future more attention would have been paid to Restricted Boltzmann Machines or other types of neurons which employ memory or types of feed-back.

One additional strategy for building M.L. models is to use an Ensemble approach. Ensembles are very easy to implement. For example, a model consisting of a Logistic Regression followed by SVM model could be paired in the hope that it would achieve higher accuracies. This researcher also is aware of several other M.L. methods that could be very useful with data that is between a finite range such as the percent amino acid composition. The percent amino acid composition is by definition between 0 and 1. Also, this protein data set has a rather small set of predictors or independent variables.

The impetus for this work was a paper published in 2007 by Xindong Wu, J. Ross Quinlan, et al.[3] I found this paper to be an amazing starting point since it ranked most common M.L. techniques. I chose my M.L. methods with this paper in mind. Starting with the Logistic Regression as the entry point for understanding Neural Networks and even other binary classifiers. One area that was not discussed here is Decision Trees and CART. These use Entropy and Information theory as the ideas governing there behaviour. In depth review of these M.L. approaches woould havve been an excellent background for Random Forests and all of its spin-offs.

One item which was interesting was that the Principal Component Analysis had very little bearing on the set of observations that were found as false-positives and false-negatives. This could be due to the fact that the first two principal components which were chosen have very little bearing on how the alternative M.L. methods produce their decision boundaries. It was an interesting experiment nonetheless, since it is thought that PCA can reduce dimensional. The fact that 12 of the principal components were needed to achieve 90-95% variance in the system. It was hoped that the 50% which was represented by the first two components would be functional. This could also suggest that the decision boundary that best fits the protein data set is not linear.

[2]Kevin P.Murphy, Machine learning : a probabilistic perspective, 2012, ISBN 978-0-262-01802-9
[3]Xindong Wu, J. Ross Quinlan, et al., Knowl Inf Syst (2008) 14:1–37, DOI 10.1007/s10115-007-0114-2

Appendices

mcc

3/13/2020

Appendices

```
knitr::opts_chunk$set(cache = TRUE)
```

Word To The Wise

- LEARN GIT. Get a Github account!
  - Put Every computer program you write and Every stick of knowledge related to your work in a clean format on Git. You will benefit from it in the long run...
- To Biologists and Biochemists (which I also consider myself), it is your task to become more familiar with computer languages, computer concepts and math/statistics.

As of the writing of this booklet, Data Science and Bioinformatics are now centered around the computer languages R and Python.

To many researchers in data science and bioinformatics the field now includes such languages as, *in no particular order*,

- R
- Python
- Bash shell scripting
- SQL
- Julia
- Javascript
- RMarkdown, (This one is fun and easy)

Just start with one language

- Stick with it for a time and learn it. Learn the ins-and-outs of one language first.

Is R or Python better?

1. R & Python are both FREE,
2. Both have great integrated development environments (IDEs),
    - RStudio is great & FREE,
    - Spyder is also great & FREE,
    - PyCharm
    - Sublime
    - Visual Studio Code
    - Jupyter Notebook
3. Both languages have been around for $>$ 20 years, therefore both have tons of FREE information & tutorials on YouTube,

Install R & RStudio

NOTE: I use Ubuntu.

1. Go to: https://cran.r-project.org/
2. Choose R for your operating system.
3. If you are using Linux, I recommend that you download/install 4 R files.
    1. r-base-core_#.#.#.*.deb (approx. 30 MB)
    2. r-base-dev_#.#.#.*.deb (approx. 45 KB)
    3. r-base_#.#.#*.deb (approx. 90 KB)
    4. r-base-html-#.#.#*.deb (approx. 90 KB)
4. If you do have Linux you may try this video: How to install R.
5. Go to: https://www.rstudio.com/
6. From RStudio's homepage click, *Products* then click *RStudio* from the drop-down menu.
7. Click the Download of the FREE version of RStudio Desktop
8. Click *RStudio #.#* to download a version for your machine
9. Have Linux? Try this - How to install RStudio.
10. If you are looking for instructions for Mac & Windows machines try:
    - FreeCodeCamp

Load Libraries Used In This Project

If you are using Ubuntu/Linux you may need these Linux libraries first.

```
sudo apt-get install libcurl4-openssl-dev libssl-dev libxml2-dev build-essential
```

To install `car` & `rgl`

```
sudo apt-get install xorg libx11-dev libglu1-mesa-dev libfreetype6-dev
```

```
install.packages("rlang")
library(rlang)

load_or_install <- function(package_names) {
    for(package_name in package_names) {
        if(!is_installed(package_name)) {
            install.packages(package_name,
            repos = "http://lib.stat.cmu.edu/R/CRAN",
            dependencies = TRUE)
        }
        library(package_name,character.only=TRUE,quietly=TRUE,verbose=FALSE)
        print("OK")
        }
}

load_or_install(c("doMC", "corrplot", "knitr", "caret", "tidyverse"))

load_or_install(c("ggplot2", "rmarkdown", "bookdown", "blogdown", "kernlab"))

load_or_install(c("e1071", "plyr", "RColorBrewer", "neuralnet", "ggfortify"))

load_or_install(c("rpart", "MASS", "tidyr", "ggplot2", "seqinr", "Boruta", "kableExtra"))

load_or_install(c("LogicReg", "randomForest", "foreach", "caretEnsemble"))

load_or_install(c("import", "dplyr", "stringr", "stringi", "readr", "tinytex"))
```

Calculate the amino acid compositions (AAC) and Di-peptide compositions (DPC)

from .fasta formats, {Myoglobin, Non-Myoglobin}

Calculating the Amino Acid and Di-peptide composition of a protein string is a simple calculation requiring the total amino acid length of the peptide or poly-peptide of interest and a count of substrings. Initially, the command `seqinr::read.fasta` reads .fasta file formats and returns a list of proteins stripping away all other information. Secondly, the command `stringr::str_count()` produces an integer value of the number of substrings in a larger string, i.e. `peptide`.

For example, `aa_nums[j] = str_count(peptide, col_titles[j]) / total_aa`,

Where; `aa_nums[j]` is an array to saving values for later writing to file, `peptide` is the string to check, i.e. protein of interest, `col_titles[j]` is the substring which is either a single amino acid or di-peptide.

Input: .fasta Output: .csv

Libraries

```
Libraries = c("stringr", "knitr", "seqinr")

for (p in Libraries) {  # Install Libraries
    library(p, character.only = TRUE)
}

opts_chunk$set(cache = TRUE,
               warning = FALSE,
```

```
                message = FALSE,
                align = "center")
```

Import uniprot-myoglobin.fasta - Read peptide lines

```
read_fasta <- function(file) {
    listo_proteins <- read.fasta(file = file,
                                 seqtype = "AA",
                                 as.string = TRUE,
                                 seqonly = FALSE,
                                 strip.desc = TRUE)
    return(listo_proteins)
}

file = "./00-data/ORIGINAL_DATA/uniprot-myoglobin.fasta"
myoglobins <- read_fasta(file)
```

Column_titles

```
column_titles = function() {
    peptides = c("A", "C", "D", "E", "F",
                 "G", "H", "I", "K", "L",
                 "M", "N", "P", "Q", "R",
                 "S", "T", "V", "W", "Y")

    # Add DIPEPTIDES column titles
    di_titles = vector(mode = "character", length = 400)
    k = 1
    for (i in 1:20) {
        for (j in 1:20) {
            di_titles[k] <- paste(peptides[i], peptides[j], sep = "")
            k = k + 1
        }
    }
    aa_di_titles <- c("Class","TotalAA","PID", peptides, di_titles)
    return(aa_di_titles)
}

col_titles <- column_titles()
col_titles
```

Write empty .csv

```
write_empty_csv <- function(protein_class = "C") {
    col_titles <- column_titles()
    file_name <- paste(protein_class, "_aac_dpc.csv", sep = "")
    write.table(t(col_titles),
                file_name,
                sep = ",",
                col.names = FALSE,
                row.names = FALSE,
                eol = "\n")
```

```
    return(file_name)
}

file_name <- write_empty_csv()
```

Calculate AAC and DPC values function

```
calc_aac_dpc <- function(peptide, protein_class = "C", i, file_name) {
    aa_nums = matrix(0, ncol = 423)
    ##############################
    # First column is class
    aa_nums[1] = ifelse(protein_class == "C", 0, 1)
    # Second column is total number of amino acids
    total_aa = nchar(peptide)
    aa_nums[2] = total_aa
    # Third line is 'Protein ID', PID
    aa_nums[3] = paste(protein_class, i, sep = "")
    # Column 4:423 - Calculate AAC/DPC
    for (j in 4:423) {
        aa_nums[j] = str_count(peptide, col_titles[j]) / total_aa
        }
    write(t(aa_nums), file = file_name, append = TRUE, ncolumns = 423, sep = ",")
}
```

Run Myoglobin

```
# RUN Myoglobin
for (i in 1:1124) {
    peptide <- myoglobins[[i]][1]
    calc_aac_dpc(peptide, protein_class = "M", i, file_name)
}
```

Run Control / Human-NOT-myoglobin

- Import data - Read peptide lines

```
read_fasta <- function(file) {
    listo_proteins <- read.fasta(file = file,
                            seqtype = "AA",
                            as.string = TRUE,
                            seqonly = FALSE,
                            strip.desc = TRUE)
    return(listo_proteins)
}

file = "./00-data/ORIGINAL_DATA/uniprot-human+NOT+hemoglobin+NOT+myoglobin+random.fasta"
controls <- read_fasta(file)
```

Run Controls

```
for (i in 1:1216) {
```

```
    peptide <- controls[[i]][1]
    calc_aac_dpc(peptide, protein_class = "C", i, file_name)
}
```

KEEP AAC ONLY FOR RAW DATA

```
file = "./00-data/aac_dpc_values/C+M_aac_dpc.csv"
C+M_aac_dpc <- read.csv(file,
                        stringsAsFactors=FALSE)
# View(`C+M_aac_dpc`)

# Select 1st thru 23rd variables
c_m_RAW_AAC <- C+M_aac_dpc[c(1:23)]
```

- To A Comma Delimited Text File

```
setwd("../00-data/02-aac_dpc_values/")

write.table(c_m_RAW_AAC,
            file = "./00-data/02-aac_dpc_values/c_m_RAW_AAC.csv",
            sep = ",",
            row.names = F)
```

Transform {C, F, I} from c__m__RAW__AAC

```
library(readr)

file = "../00-data/02-aac_dpc_values/c_m_RAW_AAC.csv"
c_m_RAW_AAC <- read_csv(file,
                        col_types = cols(Class = col_factor(levels = c("0","1"))))
c_m_TRANSFORMED_AAC <- c_m_RAW_AAC
```

1. Transfrom C,F,I using sqrt(x)
2. Columns: C=5, F=8, I=11

```
c_m_TRANSFORMED_AAC[, 5] <- sqrt(c_m_TRANSFORMED_AAC[, 5]) # C
c_m_TRANSFORMED_AAC[, 8] <- sqrt(c_m_TRANSFORMED_AAC[, 8]) # F
c_m_TRANSFORMED_AAC[,11] <- sqrt(c_m_TRANSFORMED_AAC[,11]) # I

file = "./00-data/02-aac_dpc_values/c_m_TRANSFORMED.csv"
write_csv(c_m_TRANSFORMED_AAC,
          file = file,
          col_names = T)
```

Where To Find Help

1. Cheat Sheets
2. https://community.rstudio.com
3. https://www.reddit.com/r/RStudio/
4. https://R-bloggers.com/

5. https://resources.rstudio.com/
6. Rpubs.com

- **Rpubs.com** contains R/RStudio notebooks and Markdown pages, VERY HELPFUL work from other peoples online R documents. It is a way to learn from others and share your work. - Sign up, it is FREE! then press: *Get Started*

NOTE: If you are interested in seeing what others have published search Google, Rpubs.com does not have its own search function. In Google, Search: `site:rpubs.com eda`

Other sites:

1. Coursera
2. Stack Overflow
3. Quora
4. Roger Peng's EDA
5. Bookdown - **terrible** yet necessary resource

The Lean Publishing (https://leanpub.com) company contains a library in the form of FREE down-loadable books/pdfs. I recommend;

1. How to be a modern scientist[1] by Jeffrey Leek[2]
2. R Programming for Data Science[3] by Roger Peng[4]
3. Exploratory Data Analysis with R[5] by Roger Peng
4. Data Analysis for the Life Sciences[6] by Rafael Irizarry & Michael Love

Machine Setting & Session Info

```
Sys.info()[c(1:3,5)]
```

```
##                                                 sysname
##                                                 "Linux"
##                                                 release
##                                       "5.3.0-40-generic"
##                                                 version
## "#32~18.04.1-Ubuntu SMP Mon Feb 3 14:05:59 UTC 2020"
##                                                 machine
##                                                "x86_64"
```

```
sessionInfo()
```

```
## R version 3.6.0 (2019-04-26)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.4 LTS
##
```

[1] https://leanpub.com/modernscientist
[2] http://jtleek.com
[3] https://leanpub.com/rprogramming
[4] https://simplystatistics.org
[5] https://leanpub.com/exdata
[6] https://leanpub.com/dataanalysisforthelifesciences

```
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.28
##
## loaded via a namespace (and not attached):
##  [1] compiler_3.6.0   magrittr_1.5     tools_3.6.0      htmltools_0.4.0
##  [5] yaml_2.2.1       Rcpp_1.0.3       codetools_0.2-16 stringi_1.4.6
##  [9] rmarkdown_2.1    stringr_1.4.0    xfun_0.12        digest_0.6.25
## [13] rlang_0.4.5      evaluate_0.14
```