Otto-von-Guericke-Universität Magdeburg
Advanced Machine Learning Semester Project

Saagar Gaikwad, Naveen Raj Datha, Sumit Kundu

July 7, 2017

Separation of Signals from High Energy Physics Experiments

By investigating the structure of matter and the laws that govern its interactions, physicists try to discover the fundamental properties of the physical universe. The primary tools of experimental high-energy physicists are modern accelerators, which collide protons and/or antiprotons to create exotic particles that occur only at extremely high-energy densities. Observing these particles and measuring their properties may yield critical insights about the very nature of matter. The Monte Carlo simulations of the collisions generate these particles along with decay products. The task is to distinguish collisions which produce particles of interest from those producing other particles (background).

In this paper we have proposed solutions using Machine Learning Algorithms to classify the particles using three different classifiers: -Nearest Neighbour, Naive Bayes and Decision Tree. We have also explained our idea about its Online variant.[1][2]

1. Collect Data

Data collection is the process of gathering and measuring information on targeted variables in an established systematic fashion, which then enables one to answer relevant questions and evaluate outcomes.

The data used in this paper is collected from the UCI Machine Learning Repository. The original source of the HEPMASS Data Set was Daniel Whiteson daniel '@' uci.edu, Assistant Professor, Physics and Astronomy, Univ. of California Irvine.[3]

1. P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *arXiv preprint arXiv:1402.4735*, 2014,
2. P. Baldi et al., "Parameterized machine learning for high-energy physics," *arXiv preprint arXiv:1601.07913*, 2016,
3. , `http://archive.ics.uci.edu/ml/datasets/HEPMASS`.

```
## Set Work Directory
setwd("C:\\Users\\Sumit\\Documents\\R")

## Read data from the directory as a data frame, a data structure
## which can contain different data types in different columns

## Test data
#hepmass_test_raw <- read.csv(file = "all_test.csv", header = TRUE, sep = ",")

## Train data
#hepmass_train_raw <- read.csv(file = "all_train.csv", header = TRUE, sep = ",")
```

2. Sample data

Data sampling is a statistical analysis technique used to select, manipulate and analyze a representative subset of data points in order to identify patterns and trends in the larger data set being examined.

Sampling allows one to work with a small, manageable amount of data in order to build and run analytical models more quickly, while still producing accurate findings. In some cases, a very small sample can tell all of the most important information about a data set. In others, using a larger sample can increase the likelihood of accurately representing the data as a whole, even though the increased size of the sample may impede ease of manipulation and interpretation. Either way, samples are best drawn from data sets that are as large and close to complete as possible.[4]

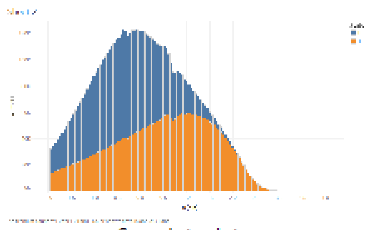
There are many different methods for drawing samples from data, and the ideal one depends on the data set and situation. Sampling can be based on probability, an approach that uses random numbers that correspond to points in the data set. This approach ensures that there is no correlation between points that are chosen for the sample. Further, variations in probability sampling include simple, stratified and systematic random sampling and multi-stage cluster sampling.

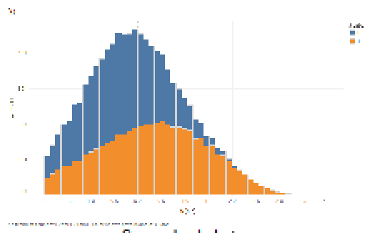
Due to hardware limitation we selected Random Sampling method to sample our data. While selecting the random number our objective was to make sure that the data is not big enough for the Computer RAM to get overwhelmed and also not end up with very small sample size. We observed that the data set works best in our machine when the sample size is 18% of original data.

```
## Sample data using Random Sampling method

## Test data
#sampled_hepmass_test_raw <- sample(hepmass_test_raw, 1000000, replace = FALSE, prob = NULL)
#write.csv(sampled_hepmass_test_raw, file = "sampled_train.csv")

## Train data
#sampled_hepmass_train_raw <- sample(hepmass_train_raw, 1000000, replace = FALSE, prob = NULL)
#write.csv(sampled_hepmass_test_raw, file = "sampled_train.csv")
```

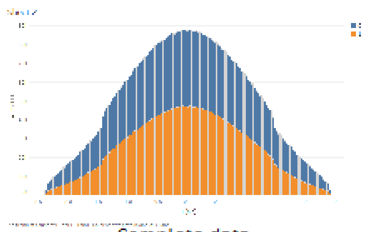
4. , http://searchbusinessanalytics.techtarget.com/definition/data-sampling.

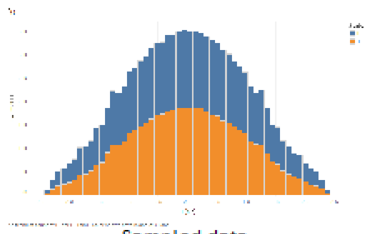
F0:


Complete data

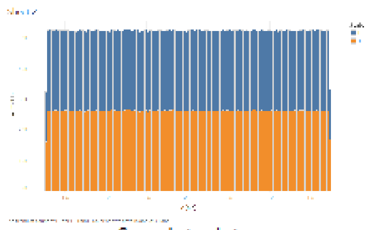

Sampled data

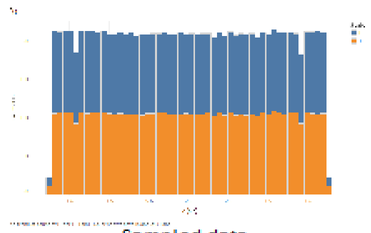
F1:


Complete data

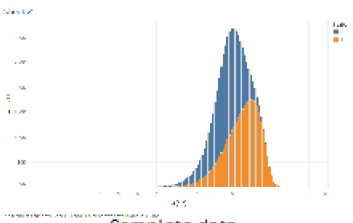

Sampled data

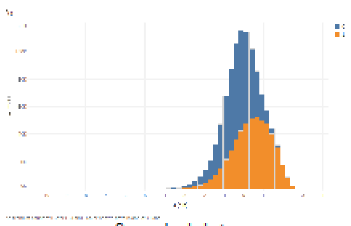
F2:


Complete data

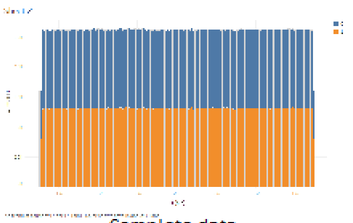

Sampled data

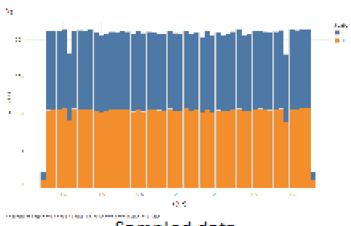
F3:


Complete data

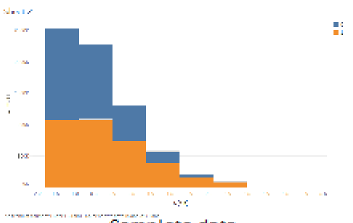

Sampled data

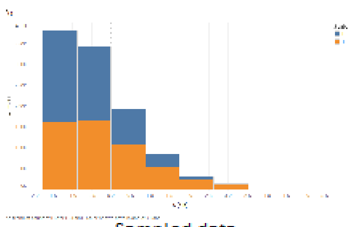
F4:


Complete data

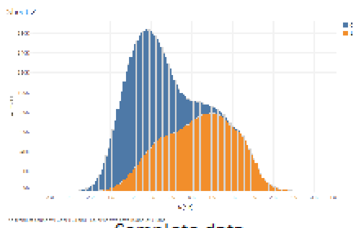

Sampled data

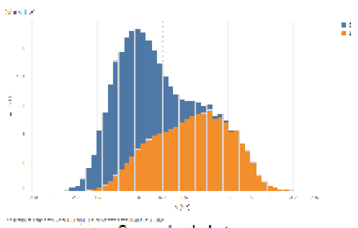
F5:


Complete data

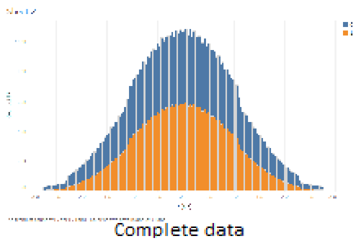

Sampled data

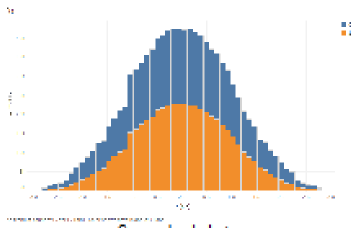
F6:


Complete data

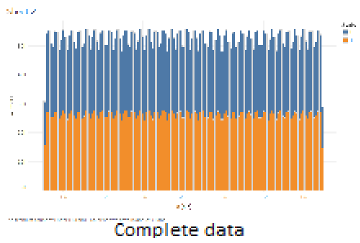

Sampled data

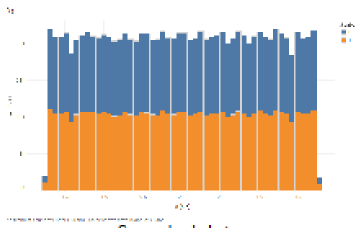
F7:


Complete data

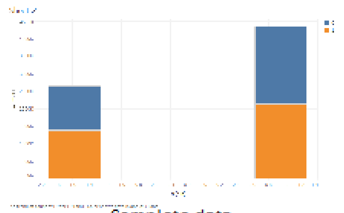

Sampled data

F8:

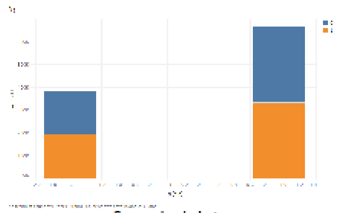

Complete data


Sampled data

F9:

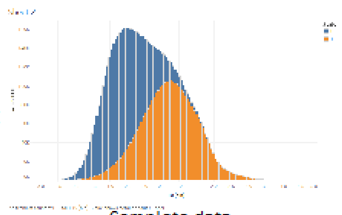


Complete data

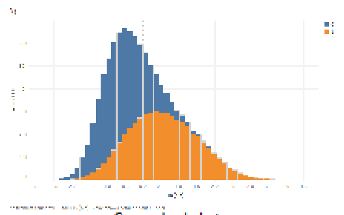


Sampled data

F10:

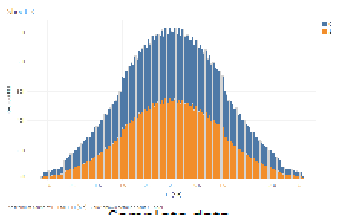


Complete data

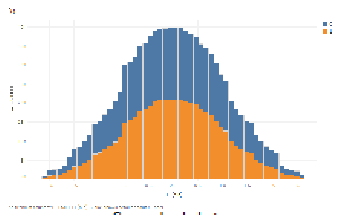


Sampled data

F11:

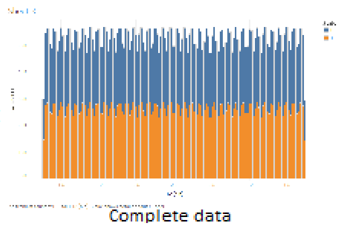


Complete data

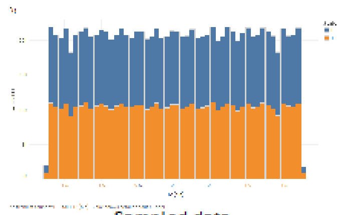


Sampled data

F12:

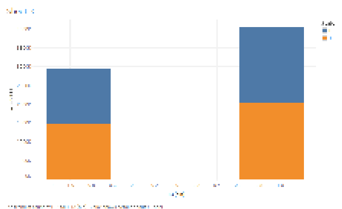


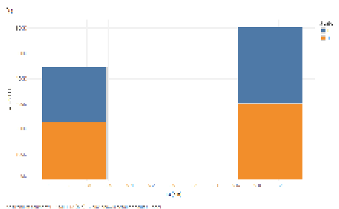
Complete data



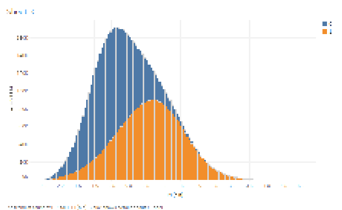
Sampled data

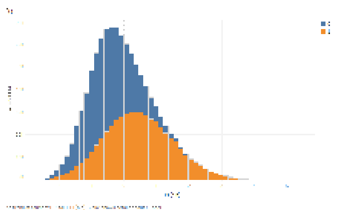
F13:


Complete data

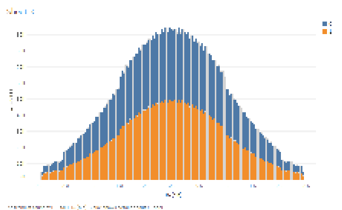

Sampled data

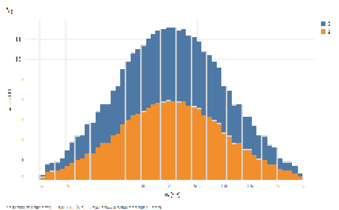
F14:


Complete data

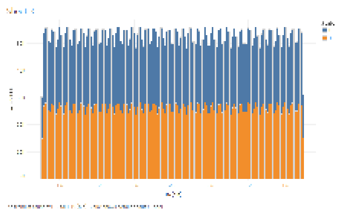

Sampled data

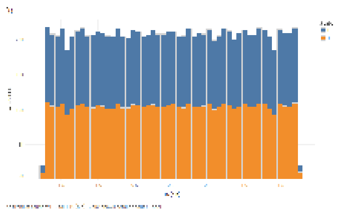
F15:


Complete data

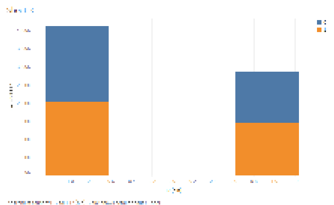

Sampled data

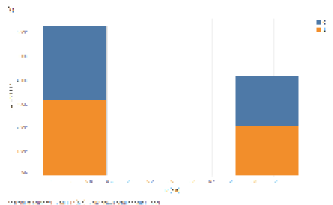
F16:


Complete data

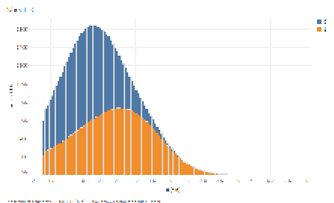

Sampled data

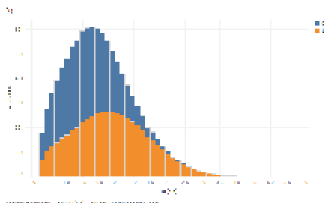
F17:


Complete data

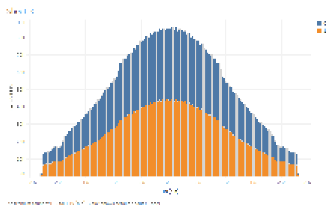

Sampled data

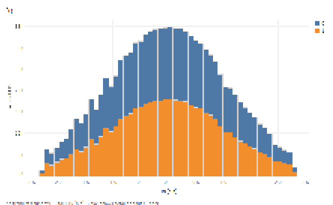
F18:


Complete data

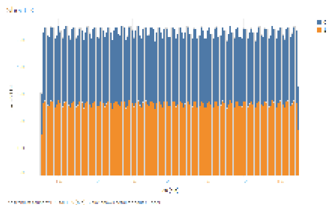

Sampled data

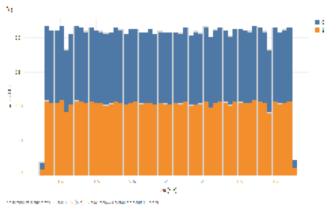
F19:

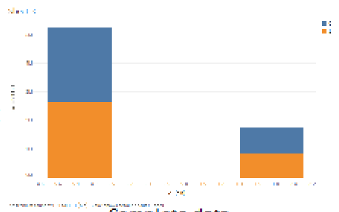

Complete data

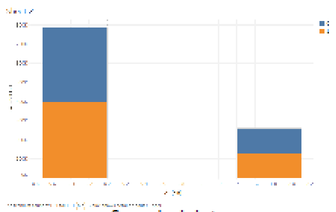

Sampled data

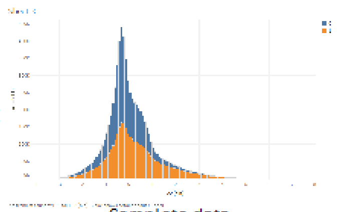
F20:

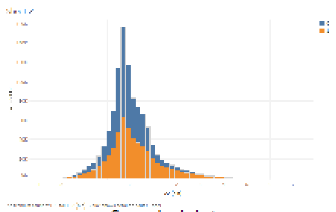

Complete data

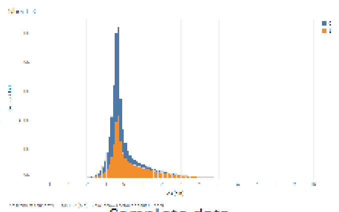

Sampled data

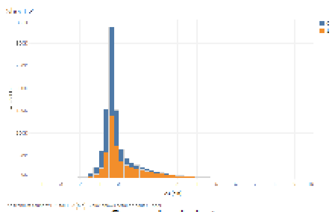
F21: 
Complete data

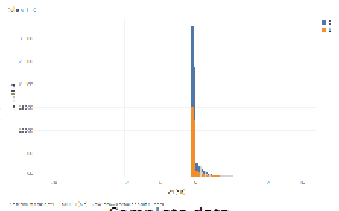

Sampled data

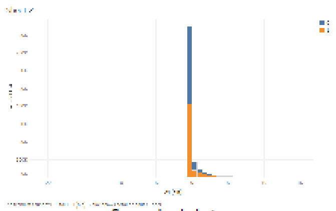
F22: 
Complete data

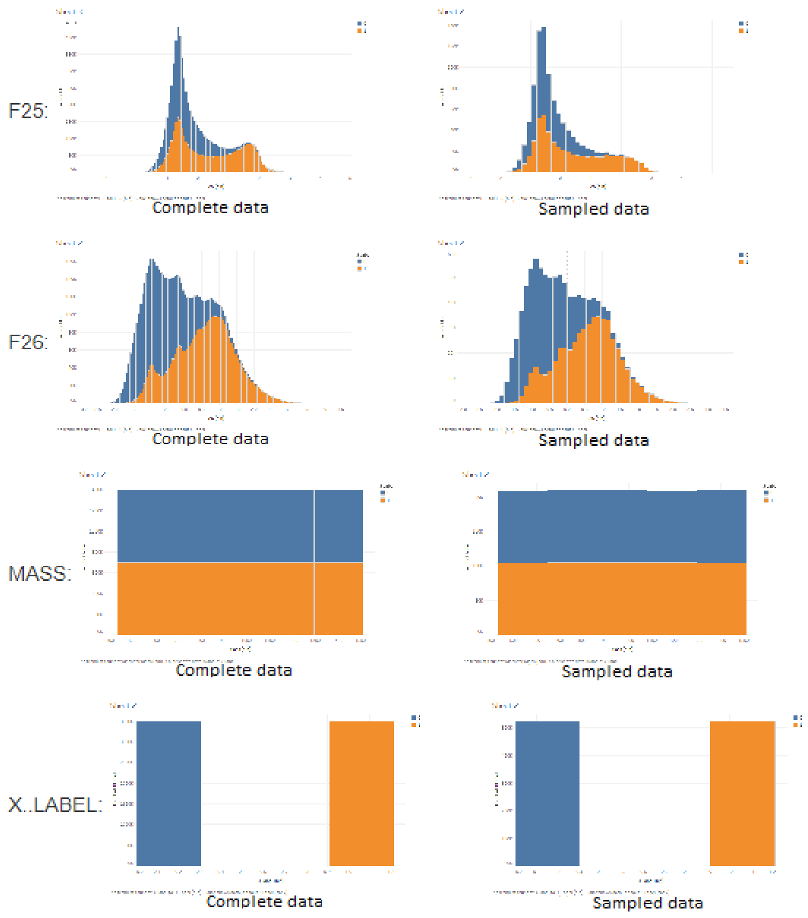

Sampled data

F23: 
Complete data


Sampled data

F24: 
Complete data


Sampled data



3. Clean Data

Data cleansing is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data using data wrangling tools, or as batch processing through scripting.[5][6]

- 3.1 Exploring Raw Data
  In any process involving data, the first goal should always be understanding the data, i.e. the dimensionality and structure of it.

5. , `https://www.datacamp.com/courses/cleaning-data-in-r`.
6. , `http://www.kdnuggets.com/2016/03/doing-data-science-kaggle-walkthrough-cleaning-data.html`.

```r
## Read data
sampled_hepmass_train <- read.csv(file = "sampled_train.csv", header = TRUE, sep = ",")
sampled_hepmass_test <- read.csv(file = "sampled_test.csv", header = TRUE, sep = ",")

## Determine the number of rows and columns present in the data frame
dim(sampled_hepmass_train)
```

```r
## Determine the structure of the data frame
str(sampled_hepmass_train)
```

We observed that the data is well structured and ready for the next step which arranges tidies everything up and organizes in a standard format.

- 3.2 Tidying Data
  The process of tidying data involves the steps which can be summarized as given below:

  1. Fixing up Formats

     Often when data is saved or translated from one format to another, some data may not be translated correctly. A typical job when it comes to cleaning data is correcting these types of issues.

     ```r
     ## Convert the data type of columns from integer to numerical
     X..LABEL <- as.factor(sampled_hepmass_train$X..label)

     ## Replace the column in the data frame with the new vector
     sampled_hepmass_train$X..label <- X..LABEL
     ```

  2. Correcting Erroneous Values
     Once the format of all the columns present in the data frame got corrected, we checked the data for - spelling mistakes, erroneous values, etc and standardized them.

```
## Check the columns for erroneous values
```

F0:

Hide

```
## F0
table(sampled_hepmass_train$f0)
```

F1:

Hide

```
## F1
table(sampled_hepmass_train$f1)
```

F2:

Hide

```
## F2
table(sampled_hepmass_train$f2)
```

F3:

Hide

```
## F3
table(sampled_hepmass_train$f3)
```

F4:

Hide

```
## F4
table(sampled_hepmass_train$f4)
```

F5:

Hide

```
## F5
table(sampled_hepmass_train$f5)
```

F6:

Hide

```
## F6
table(sampled_hepmass_train$f6)
```

F7:

Hide

```
## F7
table(sampled_hepmass_train$f7)
```

F8:

Hide

```
## F8
table(sampled_hepmass_train$f8)
```

F9:

Hide

```
## F9
table(sampled_hepmass_train$f9)
```

F10:

Hide

```
## F10
table(sampled_hepmass_train$f10)
```

F11:

Hide

```
## F11
table(sampled_hepmass_train$f11)
```

F12:

Hide

```
## F12
table(sampled_hepmass_train$f12)
```

F13:

Hide

```
## F13
table(sampled_hepmass_train$f13)
```

F14:

Hide

```
## F14
table(sampled_hepmass_train$f14)
```

F15:

Hide

```
## F15
table(sampled_hepmass_train$f15)
```

F16:

Hide

```r
## F16
table(sampled_hepmass_train$f16)
```

F17:

Hide

```r
## F17
table(sampled_hepmass_train$f17)
```

F18:

Hide

```r
## F18
table(sampled_hepmass_train$f18)
```

F19:

Hide

```r
## F19
table(sampled_hepmass_train$f19)
```

F20:

Hide

```r
## F20
table(sampled_hepmass_train$f20)
```

F21:

Hide

```r
## F21
table(sampled_hepmass_train$f21)
```

F22:

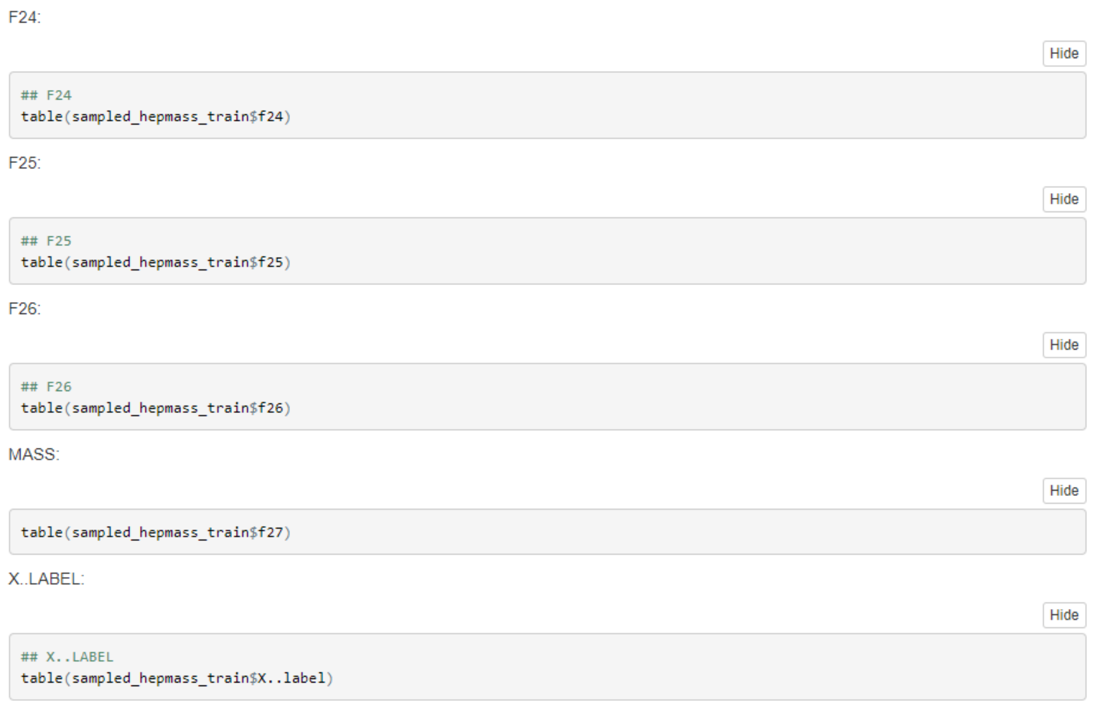
Hide

```r
## F22
table(sampled_hepmass_train$f22)
```

F23:

Hide

```r
## F23
table(sampled_hepmass_train$f23)
```



F24:

```
## F24
table(sampled_hepmass_train$f24)
```

F25:

```
## F25
table(sampled_hepmass_train$f25)
```

F26:

```
## F26
table(sampled_hepmass_train$f26)
```

MASS:

```
table(sampled_hepmass_train$f27)
```

X..LABEL:

```
## X..LABEL
table(sampled_hepmass_train$X..label)
```

With the help of above displayed visualizations, we got to know that no erroneous values exists and hence, proceeded to check the presence of missing values.

3. Filling in Missing Values
It is quite common for some values to be missing from data sets. This typically means that a piece of information was simply not collected. Rubin (1976) differentiated between three types of missingness mechanisms:
a. **Missing completely at random (MCAR):** When cases with missing values can be thought of as a random sample of all the cases.
b. **Missing at random (MAR):** When conditioned on all the available data, any remaining missingness is completely random; i.e. it does not depend on some missing variables.
c. **Missing not at random (MNAR):** When data is neither MCAR nor MAR and this is difficult to handle because it requires strong assumptions about the patterns of missingness.

Missing data in general is one of the trickier issues that is dealt with when cleaning data. Broadly, there are two solutions:
1. Deleting/Ignoring Rows with Missing Values:
The simplest solution available is to delete all cases for which a value is missing. This method is called Complete Case Analysis (CC). There are some issues which pops up when CC is implemented on data sets.

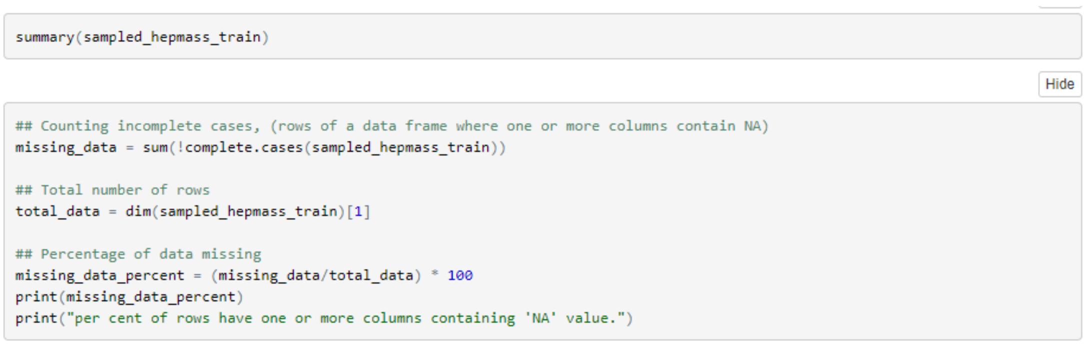
The first is that this approach only makes sense if the number of rows with missing data is relatively small compared (say 10%) to the data set.

The second issue is that in order to delete the rows containing missing data, one needs to be confident that the rows to be deleted do not contain information that is not contained in other rows.

2. Filling in the Values
The second broad option for dealing with missing data is to fill the missing values with a value. This method is known as Multiple Imputation (MI) which simulates multiple values to impute (fill-in) each missing value, then analyses each imputed data set separately and finally pools the results together.

Two general approaches for imputing multivariate data have emerged: Joint Modeling (JM) and Fully Conditional Specification (FCS), also known as Multivariate Imputation by Chained Equations (MICE).

In order to decide whether there are any missing values and which method to use for imputing missing data, if there is any, we looked into the summary of the data available, and observed that values are not missing in our data set.



```
summary(sampled_hepmass_train)
```

```
## Counting incomplete cases, (rows of a data frame where one or more columns contain NA)
missing_data = sum(!complete.cases(sampled_hepmass_train))

## Total number of rows
total_data = dim(sampled_hepmass_train)[1]

## Percentage of data missing
missing_data_percent = (missing_data/total_data) * 100
print(missing_data_percent)
print("per cent of rows have one or more columns containing 'NA' value.")
```

- Preparing Data for Analysis
  Data Preparation is an important aspect of model building as it helps in building predictive models free from correlated variables, biases and unwanted noise. There are many methods for preparing data out of which we used Principal Component Analysis in this paper.

  1. Principal Component Analysis Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the original variables or the number of observations. This transformation is defined in such a way that the first principal component has the largest possible variance, and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set.[7]

7. , https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/.

Hide

```
## Combine data into one data frame
sampled_hepmass_combined <- rbind(sampled_hepmass_train, sampled_hepmass_test)

## Remove the dependent and identifier variables
sampled_hepmass_combined_wt_label <- subset(sampled_hepmass_combined, select = -c(X..label))

## Divide the combined data
pca_train <- sampled_hepmass_combined_wt_label[1:nrow(sampled_hepmass_train),]
pca_test <- sampled_hepmass_combined_wt_label[-(1:nrow(sampled_hepmass_train)),]

## principal component analysis
princ_comp <- prcomp(pca_train, scale. = T)
names(prin_comp)
```

Mean:

Hide

```
## Mean
prin_comp$center
```

Standard Deviation:

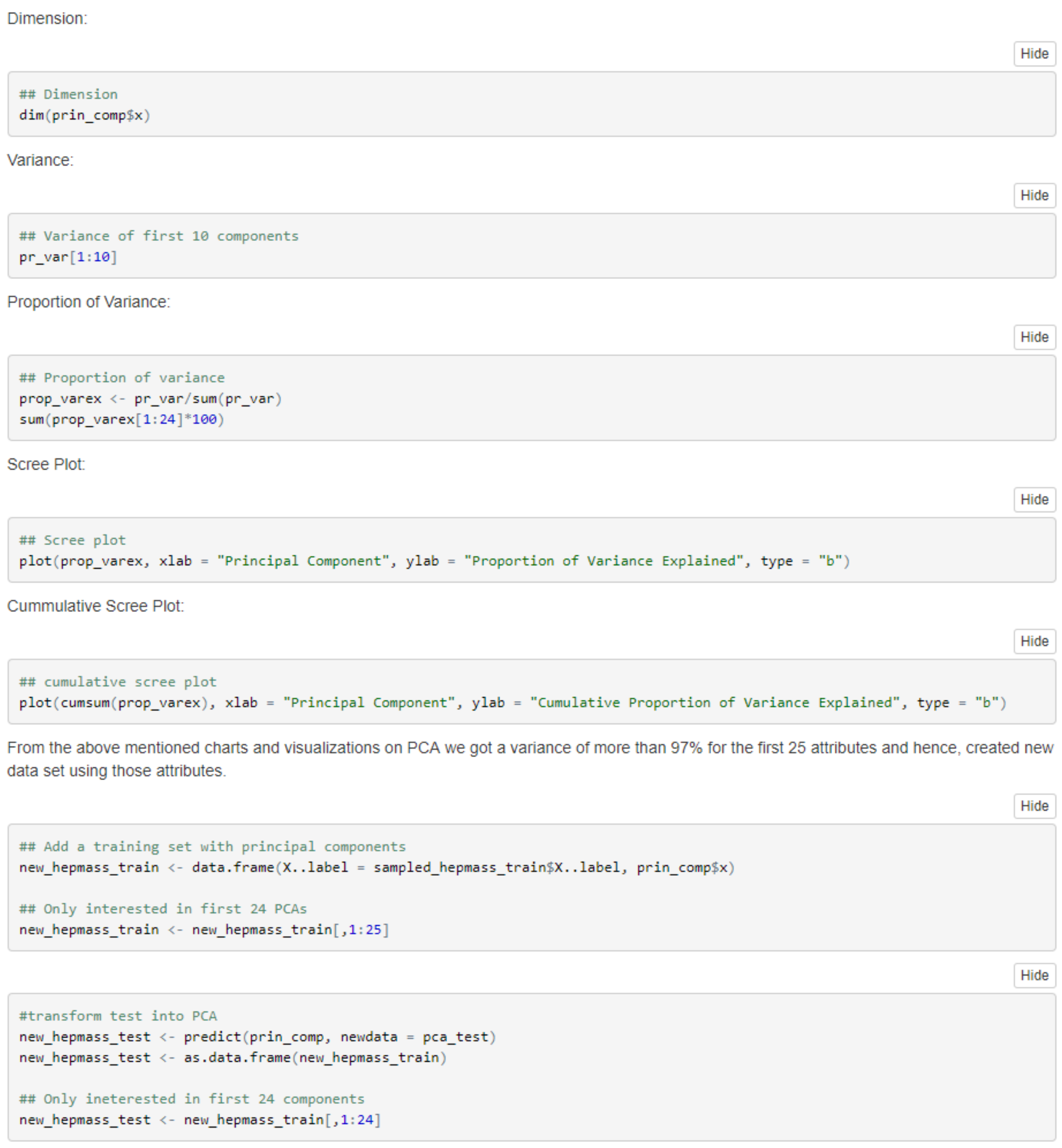
Hide

```
## Standard deviation
prin_comp$scale
```

Rotation:

Hide

```
## Rotation
prin_comp$rotation
```


Dimension:

```
## Dimension
dim(prin_comp$x)
```

Variance:

```
## Variance of first 10 components
pr_var[1:10]
```

Proportion of Variance:

```
## Proportion of variance
prop_varex <- pr_var/sum(pr_var)
sum(prop_varex[1:24]*100)
```

Scree Plot:

```
## Scree plot
plot(prop_varex, xlab = "Principal Component", ylab = "Proportion of Variance Explained", type = "b")
```

Cummulative Scree Plot:

```
## cumulative scree plot
plot(cumsum(prop_varex), xlab = "Principal Component", ylab = "Cumulative Proportion of Variance Explained", type = "b")
```

From the above mentioned charts and visualizations on PCA we got a variance of more than 97% for the first 25 attributes and hence, created new data set using those attributes.

```
## Add a training set with principal components
new_hepmass_train <- data.frame(X..label = sampled_hepmass_train$X..label, prin_comp$x)

## Only interested in first 24 PCAs
new_hepmass_train <- new_hepmass_train[,1:25]
```

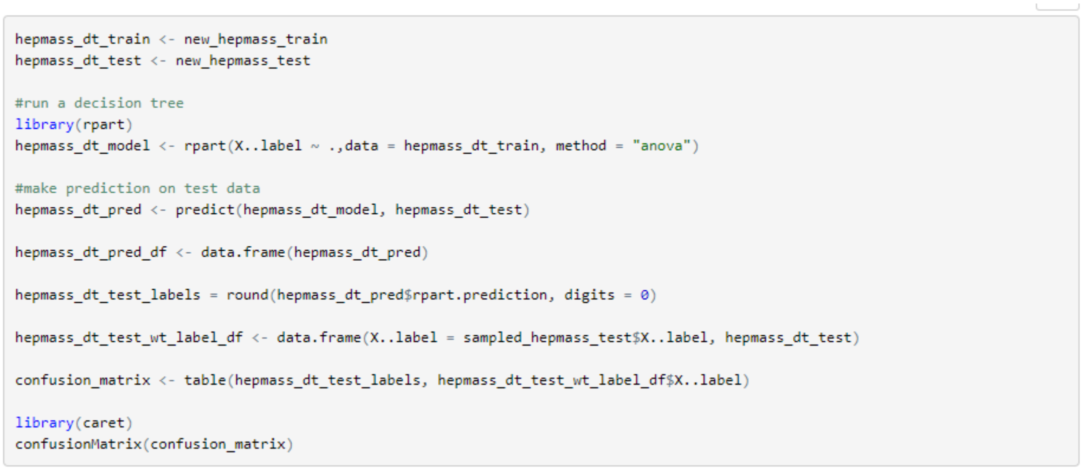
```
#transform test into PCA
new_hepmass_test <- predict(prin_comp, newdata = pca_test)
new_hepmass_test <- as.data.frame(new_hepmass_train)

## Only ineterested in first 24 components
new_hepmass_test <- new_hepmass_train[,1:24]
```

- Analyse Data
  We used the below mentioned three Machine Learning algorithms to classify the data.

  - **4.1 Decision Tree**
    Decision tree learning uses a decision tree as a predictive model to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Decision trees where the target variable can take continuous values are called regression trees, here the input variables can either be continuous or categorical.[8][9]

8. , `https://en.wikipedia.org/wiki/Decision_tree_learning`.
9. , `https://www.techwalla.com/articles/advantages-disadvantages-of-decision-trees`.

4.1.1 Algorithm:
A decision tree is built through a iterative process that splits the data into branches, and then continues splitting each branch into smaller groups as the method moves up each branch. The algorithm selects the split that minimizes the sum of the squared deviations from the mean in the two separate branches.This splitting rule is applied to each of the new branches. This process continues until each node reaches a terminal node or leaf node.
4.1.2 Pros:
Decision Trees are easy to understand, even a layperson can look at a simple tree and perform classification. They are very useful in data exploration as they are robust to missing values and outliers and therefore little or no effort for data preparation. They are non-parametric methods and therefore we do not have to worry about tuning parameters
4.1.3 Cons:
We may loose information when implementing decision on continuous variables because of the splitting rule and it also leads to over fitting on training data.
4.1.4 Case Study:
In this section we applied Decision Tree Algorithm on the data set which we obtained after performing PCA. It is one of the best suited algorithms for numerical data and as we had 27 out of 28 attributes as numerical in our data set, we implemented this algorithm to leverage this feature.



```
hepmass_dt_train <- new_hepmass_train
hepmass_dt_test <- new_hepmass_test

#run a decision tree
library(rpart)
hepmass_dt_model <- rpart(X..label ~ .,data = hepmass_dt_train, method = "anova")

#make prediction on test data
hepmass_dt_pred <- predict(hepmass_dt_model, hepmass_dt_test)

hepmass_dt_pred_df <- data.frame(hepmass_dt_pred)

hepmass_dt_test_labels = round(hepmass_dt_pred$rpart.prediction, digits = 0)

hepmass_dt_test_wt_label_df <- data.frame(X..label = sampled_hepmass_test$X..label, hepmass_dt_test)

confusion_matrix <- table(hepmass_dt_test_labels, hepmass_dt_test_wt_label_df$X..label)

library(caret)
confusionMatrix(confusion_matrix)
```

– **4.2 K Nearest Neighbor Classifier**

In pattern recognition, the k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification. It is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The input consists of the k closest training examples in the feature space. The output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.[10][11][12]

4.2.1 Algorithm: The training examples are vectors in a multidimensional feature space, each with a class label. In the training phase of the algorithm, the feature vectors and class labels of the training samples are stored.

In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidean Distance. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric (or Hamming Distance).

4.2.2 Pros The algorithm is highly unbiased in nature and makes no prior assumption of the underlying data. It is also very simple and effective and easy to implement.

4.2.3 Cons The training process is really fast as the data is stored verbatim (hence lazy learner) but the prediction time is pretty high with useful insights missing at times. Therefore, building this algorithm requires time to be invested in data preparation (especially treating the missing data and categorical features) to obtain a robust model.

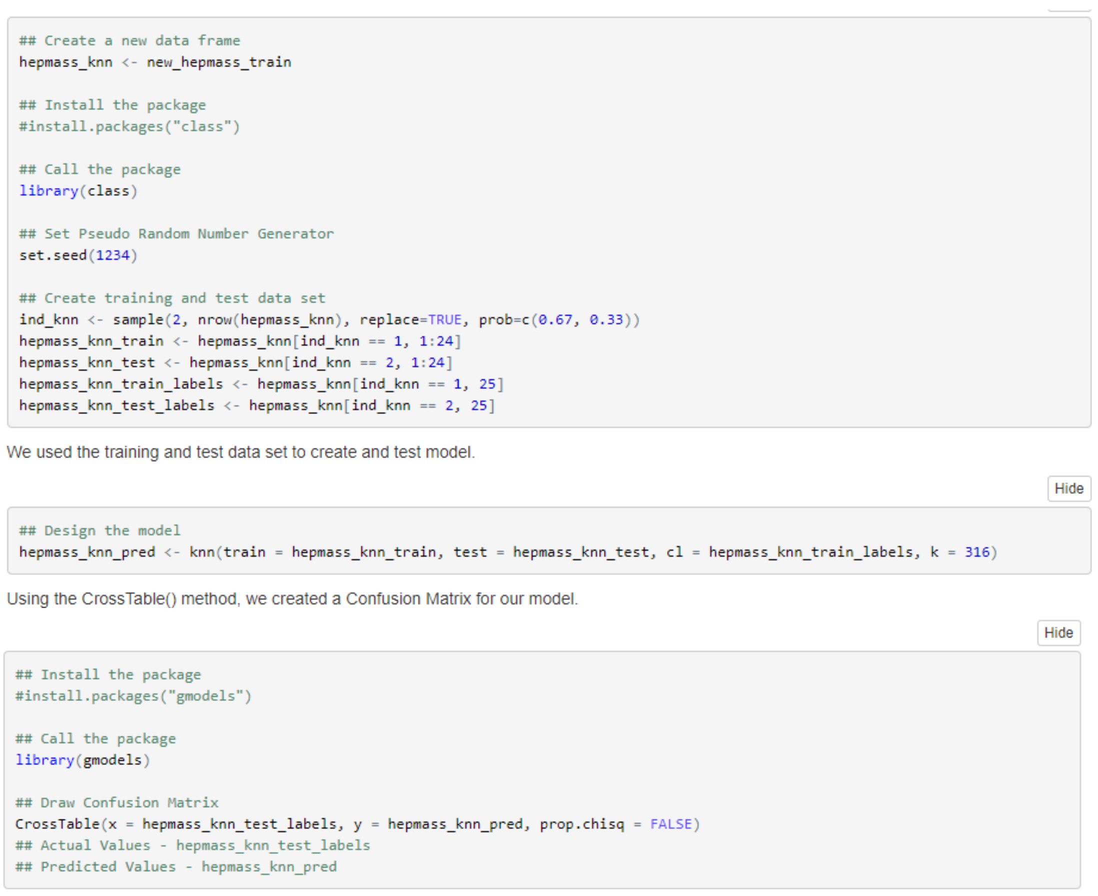
Distance usually relates to all the attributes and assumes all of them have the same effects on distance. The similarity metrics do not consider the relation of attributes which result in inaccurate distance and then impact on classification precision. Wrong classification due to presence of many irrelevant attributes is often termed as the curse of dimensionality and can be solved by performing feature selection beforehand.

Majority voting is used to classify data and this becomes a major drawback when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the K-Nearest Neighbors due to their large number. One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of its k nearest neighbors. The class (or value, in regression problems) of each of the k nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point. Another way to overcome skew is by abstraction in data representation. For example, in a Self-organizing Map (SOM), each node is a representative (a center) of a cluster of similar points, regardless of their density in the original training data. K-NN can then be applied to the SOM.

4.2.4 Case Study In this section we applied K-Nearest Neighbor Classifier Algorithm on the data set which we obtained after performing PCA.

KNN being a lazy learner, allows model to be trained very fast as all computation is left for classification phase. And HEPMASS data set being very large, we implemented this algorithm to leverage it's lazy learner feature.

10. , `https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm`.
11. , `http://people.revoledu.com/kardi/tutorial/KNN/Strength%20and%20Weakness.htm`.
12. , `http://www.cs.upc.edu/~bejar/apren/docum/trans/03d-algind-knn-eng.pdf`.



```r
## Create a new data frame
hepmass_knn <- new_hepmass_train

## Install the package
#install.packages("class")

## Call the package
library(class)

## Set Pseudo Random Number Generator
set.seed(1234)

## Create training and test data set
ind_knn <- sample(2, nrow(hepmass_knn), replace=TRUE, prob=c(0.67, 0.33))
hepmass_knn_train <- hepmass_knn[ind_knn == 1, 1:24]
hepmass_knn_test <- hepmass_knn[ind_knn == 2, 1:24]
hepmass_knn_train_labels <- hepmass_knn[ind_knn == 1, 25]
hepmass_knn_test_labels <- hepmass_knn[ind_knn == 2, 25]
```

We used the training and test data set to create and test model.

```r
## Design the model
hepmass_knn_pred <- knn(train = hepmass_knn_train, test = hepmass_knn_test, cl = hepmass_knn_train_labels, k = 316)
```

Using the CrossTable() method, we created a Confusion Matrix for our model.

```r
## Install the package
#install.packages("gmodels")

## Call the package
library(gmodels)

## Draw Confusion Matrix
CrossTable(x = hepmass_knn_test_labels, y = hepmass_knn_pred, prop.chisq = FALSE)
## Actual Values - hepmass_knn_test_labels
## Predicted Values - hepmass_knn_pred
```

- **4.3 Naive Bayes Classifier**

  In machine learning, Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

  Bayes Theorem: Bayes' theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event. For example, if cancer is related to age, then, using Bayes' theorem, a person's age can be used to more accurately assess the probability that they have cancer, compared to the assessment of the probability of cancer made without knowledge of the person's age.

  Bayes' theorem can be stated mathematically as the following equation:
  $P(h|d) = (P(d|h) * P(h)) / P(d)$

  Where, $P(h|d)$ is the probability of hypothesis h given the data d. This is called the posterior probability. $P(d|h)$ is the probability of data d given that the hypothesis h was true. $P(h)$ is the probability of hypothesis h being true (regardless of the data). This is called the prior probability of h. $P(d)$ is the probability of the data (regardless of the hypothesis).

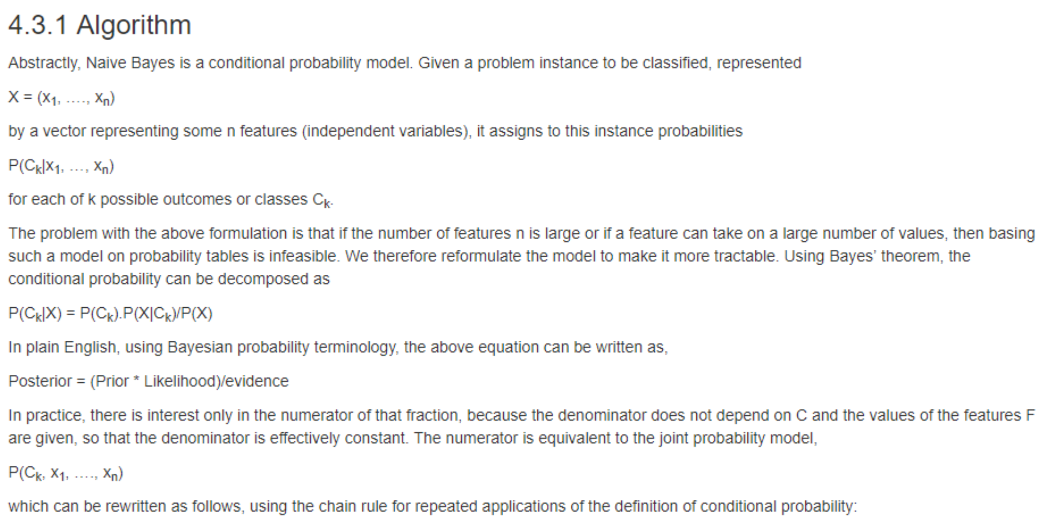
  After calculating the posterior probability for a number of different hypotheses, we can select the hypothesis with the highest probability. This is the maximum probable hypothesis and may formally be called the maximum a posteriori (MAP) hypothesis.

This can be written as:

MAP(h) = max((P(d|h) * P(h)) / P(d))

The P(d) is a normalizing term which allows us to calculate the probability.

Naive Bayes is a simple technique for constructing classifiers, models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle. All naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A Naive Bayes Classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.



4.3.1 Algorithm

Abstractly, Naive Bayes is a conditional probability model. Given a problem instance to be classified, represented

$X = (x_1, ..., x_n)$

by a vector representing some n features (independent variables), it assigns to this instance probabilities

$P(C_k|x_1, ..., x_n)$

for each of k possible outcomes or classes C_k.

The problem with the above formulation is that if the number of features n is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. We therefore reformulate the model to make it more tractable. Using Bayes' theorem, the conditional probability can be decomposed as

$P(C_k|X) = P(C_k).P(X|C_k)/P(X)$

In plain English, using Bayesian probability terminology, the above equation can be written as,

Posterior = (Prior * Likelihood)/evidence

In practice, there is interest only in the numerator of that fraction, because the denominator does not depend on C and the values of the features F are given, so that the denominator is effectively constant. The numerator is equivalent to the joint probability model,

$P(C_k, x_1, ..., x_n)$

which can be rewritten as follows, using the chain rule for repeated applications of the definition of conditional probability:

$$\begin{aligned}p(C_k, x_1, \dots, x_n) &= p(x_1, \dots, x_n, C_k)\\ &= p(x_1 | x_2, \dots, x_n, C_k) p(x_2, \dots, x_n, C_k)\\ &= p(x_1 | x_2, \dots, x_n, C_k) p(x_2 | x_3, \dots, x_n, C_k) p(x_3, \dots, x_n, C_k)\\ &= \dots\\ &= p(x_1 | x_2, \dots, x_n, C_k) p(x_2 | x_3, \dots, x_n, C_k) \dots p(x_{n-1} | x_n, C_k) p(x_n | C_k) p(C_k)\end{aligned}$$

Now the “naive” conditional independence assumptions come into play: assume that each feature F_i is conditionally independent of every other feature, F_j for j != i, given the category, C. This means that

$$p(x_i | x_{i+1}, \dots, x_n, C_k) = p(x_i | C_k)$$

Thus, the joint model can be expressed as,

$$\begin{aligned}p(C_k | x_1, \dots, x_n) &\propto p(C_k, x_1, \dots, x_n)\\ &\propto p(C_k)\ p(x_1 | C_k)\ p(x_2 | C_k)\ p(x_3 | C_k)\ \cdots\\ &\propto p(C_k) \prod_{i=1}^{n} p(x_i | C_k)\,.\end{aligned}$$

This means that under the above independence assumptions, the conditional distribution over the class variable, C is:

$$p(C_k | x_1, \dots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i | C_k)$$

where the evidence Z = p(x) is a scaling factor dependent only on $x_1, \dots, x_n$, that is, a constant if the values of the feature variables are known.

The Naive Bayes Classifier combines this model with a decision rule. One common rule is to pick the hypothesis that is most probable; this is known as the Maximum a Posteriori or MAP decision rule. The corresponding classifier, a Bayes Classifier, is the function that assigns a class label y^ = C_k for some k as follows:

$$\hat{y} = \underset{k \in \{1, \dots, K\}}{\operatorname{argmax}}\ p(C_k) \prod_{i=1}^{n} p(x_i | C_k).$$

[13]

4.3.2 Pros

If the Naive Bayes conditional independence assumption actually holds, a Naive Bayes classifier will converge quicker than discriminative models like logistic regression, so you need less training data. And even if the Naive Bayes assumption doesn’t hold, a Naive Bayes classifier still often does a great job in practice. A good bet for something which is fast, easy and performs pretty well.

4.3.3 Cons

Its main disadvantage is that it can’t learn interactions between features. E.g.: It can’t learn that although one loves movies with Brad Pitt and Tom Cruise, one hates movies where they’re together.

If categorical variable has a category (test data set), which was not observed in training data set, then model will assign a zero probability and will be unable to make a prediction.

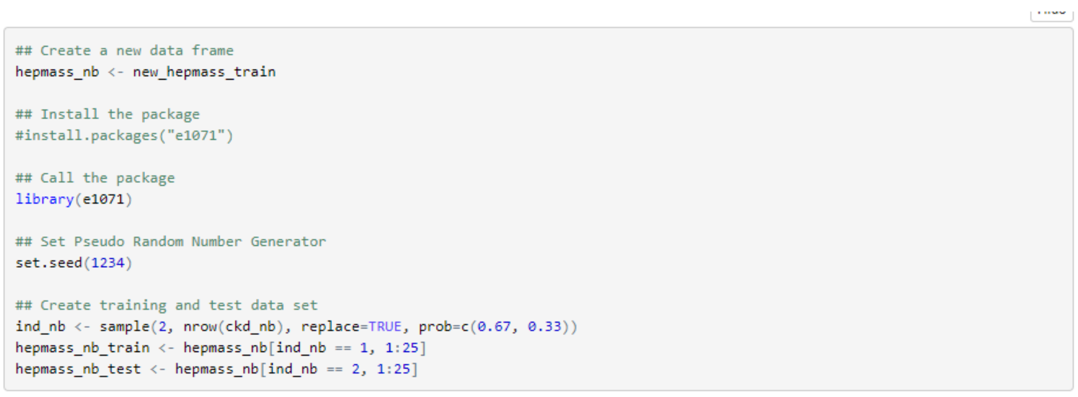
Another limitation of Naive Bayes is the assumption of independent predictors. In real life it is almost impossible that we get a set of predictors which are completely independent.

4.3.3 Case Study

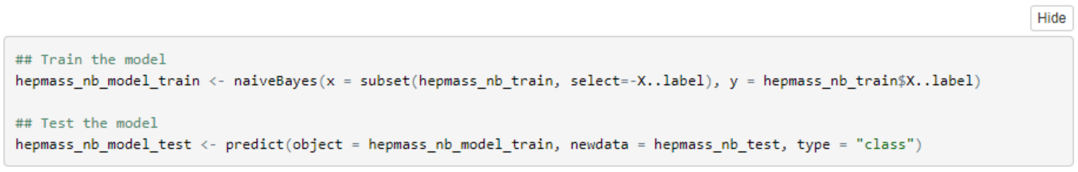
In this section we applied Naive Bayes Classifier Algorithm on the data set we obtained after performing PCA.

The new components that we obtained after performing PCA became orthgonal with no co-relation to each other. And Naive Bayes being an algorithm which assumes independence between features, we wanted to leverage this feature and hence, implemented it.

13. , `https://en.wikipedia.org/wiki/Naive_Bayes_classifier`.

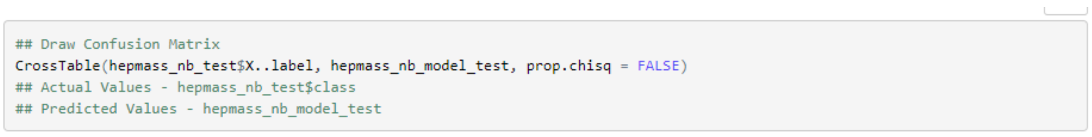

```
## Create a new data frame
hepmass_nb <- new_hepmass_train

## Install the package
#install.packages("e1071")

## Call the package
library(e1071)

## Set Pseudo Random Number Generator
set.seed(1234)

## Create training and test data set
ind_nb <- sample(2, nrow(ckd_nb), replace=TRUE, prob=c(0.67, 0.33))
hepmass_nb_train <- hepmass_nb[ind_nb == 1, 1:25]
hepmass_nb_test <- hepmass_nb[ind_nb == 2, 1:25]
```

We used the training and test data set to create and test model.

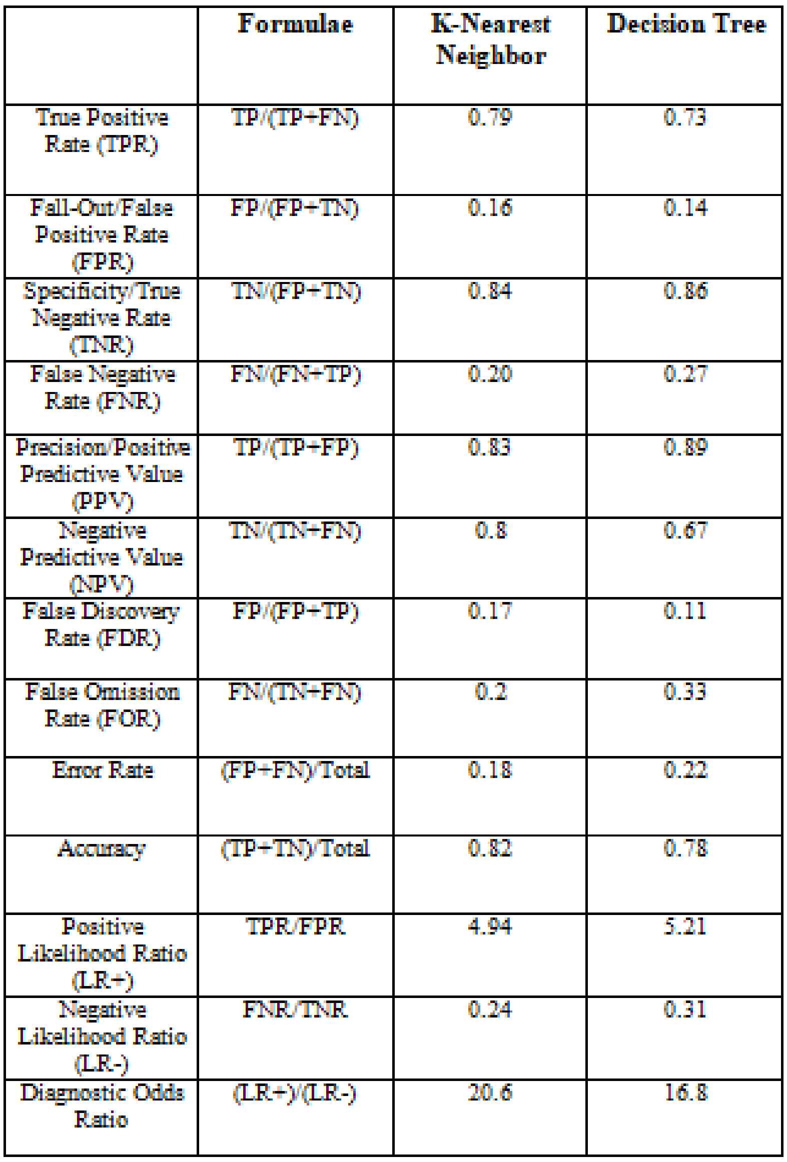

Hide

```
## Train the model
hepmass_nb_model_train <- naiveBayes(x = subset(hepmass_nb_train, select=-X..label), y = hepmass_nb_train$X..label)

## Test the model
hepmass_nb_model_test <- predict(object = hepmass_nb_model_train, newdata = hepmass_nb_test, type = "class")
```

Using the CrossTable() method, we created a Confusion Matrix for our model.


```
## Draw Confusion Matrix
CrossTable(hepmass_nb_test$X..label, hepmass_nb_model_test, prop.chisq = FALSE)
## Actual Values - hepmass_nb_test$class
## Predicted Values - hepmass_nb_model_test
```

- **Report**
  We compared the Confusion Matrix for all the algorithms and observed the following:



	Formulae	K-Nearest Neighbor	Decision Tree
True Positive Rate (TPR)	TP/(TP+FN)	0.79	0.73
Fall-Out/False Positive Rate (FPR)	FP/(FP+TN)	0.16	0.14
Specificity/True Negative Rate (TNR)	TN/(FP+TN)	0.84	0.86
False Negative Rate (FNR)	FN/(FN+TP)	0.20	0.27
Precision/Positive Predictive Value (PPV)	TP/(TP+FP)	0.83	0.89
Negative Predictive Value (NPV)	TN/(TN+FN)	0.8	0.67
False Discovery Rate (FDR)	FP/(FP+TP)	0.17	0.11
False Omission Rate (FOR)	FN/(TN+FN)	0.2	0.33
Error Rate	(FP+FN)/Total	0.18	0.22
Accuracy	(TP+TN)/Total	0.82	0.78
Positive Likelihood Ratio (LR+)	TPR/FPR	4.94	5.21
Negative Likelihood Ratio (LR-)	FNR/TNR	0.24	0.31
Diagnostic Odds Ratio	(LR+)/(LR-)	20.6	16.8

We calculated the above mentioned parameters from the confusion matrix, but not all parameters are important for a given problem. For example when dealing with a medical diagnosis data set we may emphasize more on getting better FNR (False negative rate) even though we may have to compromise a bit on accuracy. For the HEPMASS data set we figured out that the parameter ‘Accuracy’ matters the most and therefore the KNN algorithm performs better as compared to the other algorithms.

- **Online Variant**
  It is a form of learning in which the data becomes available in a sequential order and is used to update the best predictor for future data at each step. It is a very common technique in situations where its infeasible to get all data at once and/or computationally infeasible

to train on the entire data set at once and/or necessary for the algorithm to dynamically adapt to the new patterns in data. It is prone to catastrophic interference i.e. tendency of ANN to completely and abruptly forget already learned information upon learning new information.

Active Learning is a special case of semi-supervised Machine Learning Algorithm which is able to interactively query the user to obtain the desired outputs at new data points. It reduces the dimensionality of the data by removing noise and redundancy. However, it may get overwhelmed by uninformative examples.[14][15]

Ways to decide on which data points to be labelled: * Uncertainty Sampling - Labels those points for which the current model is least certain as to what the correct output should be * Query by Committee - Variety of models are trained on the current labelled data, and vote on the output for unlabelled data.Those points for which the "committee" disagrees the most are then labeled. * Expected Model Changec - Label those points that would most change the current model.

We can implement the online version on our data set using either of the ways mentioned above.

- **Improvements**
  We could have worked more on checking the quality of variables using visualizations such as Factor Map which displays variable contribution in terms of percantage, Biplot of Variables and Cos2 of Variables. We also could have performed more rounds of PCA and evaluate its impact on the classification of data.

14. B. Settles, "Active learning literature survey," *University of Wisconsin, Madison* 52, nos. 55-66 (2010): 11.
15. , `http://stat.ethz.ch/R-manual`.