Tirgul 3 - Dimensionality Reduction and Feature Selection

Romi Goldner Kabeli

3/28/2023

What is dimensionality reduction?

Dimensionality reduction is a useful and powerful tool in data science, allowing us to make sense of our data by reducing its complexity and feature size while preserving important structure and information. It comes in two forms: - Feature Elimination, where redundant and unnecessary variables are removed. - Simpler to implement. - Disadvantages - possibility of losing information. - Feature Extraction, where new variables are formed from the old ones. - Preserves the original structure better. - Disadvantages - newly formed variables may be more difficult to interpret.

Main uses of dimensionality reduction: - simplification of data - denoising - variable selection - visualization. It allows us to explore our data with fewer variables, helping to uncover the most relevant aspects of it in an efficient manner.

We will work on cereal data:

```
cereals <- read.csv("cereal.csv", header = TRUE, row.names = 1)
str(cereals)
```

```
## 'data.frame':    77 obs. of  12 variables:
##  $ Manufacturer: chr  "N" "Q" "K" "K" ...
##  $ Cold.or.Hot : chr  "C" "C" "C" "C" ...
##  $ calories    : int  70 120 70 50 110 110 110 130 90 90 ...
##  $ protein     : int  4 3 4 4 2 2 2 3 2 3 ...
##  $ fat         : int  1 5 1 0 2 2 0 2 1 0 ...
##  $ sodium      : int  130 15 260 140 200 180 125 210 200 210 ...
##  $ fiber       : num  10 2 9 14 1 1.5 1 2 4 5 ...
##  $ carbo       : num  5 8 7 8 14 10.5 11 18 15 13 ...
##  $ sugars      : int  6 8 5 0 8 10 14 8 6 5 ...
##  $ potass      : int  280 135 320 330 -1 70 30 100 125 190 ...
##  $ vitamins    : int  25 0 25 25 25 25 25 25 25 25 ...
##  $ rating      : num  68.4 34 59.4 93.7 34.4 ...
```

Looks like we have 77 brands of cereal with 12 variables describing them, which is a lot. Lets leave only the numerical features:

```
numeric_vars <- c(
  "calories", "protein", "fat", "sodium", "fiber", "carbo",
  "sugars", "potass", "vitamins", "rating"
)

cereals_num <- cereals[, numeric_vars]
ncol(cereals_num)
```

```
## [1] 10
```

Now, lets say we want to categorize these cereals. We'll call some the "fun" ones, and some the "shredded" ones

```
fun_cereals <- c(
  "Lucky_Charms", "Count_Chocula", "Froot_Loops",
  "Frosted_Flakes", "Cocoa_Puffs", "Cinnamon_Toast_Crunch"
)

shredded_wheat_cereals <- c(
  "Shredded_Wheat", "Shredded_Wheat_'n'Bran",
  "Shredded_Wheat_spoon_size"
)

cereals_num_sub <- match(
  c(fun_cereals, shredded_wheat_cereals),
  rownames(cereals_num)
)

cereals_num$classification <- "normal"
cereals_num$classification[match(fun_cereals, rownames(cereals_num))] <- "fun"
cereals_num$classification[match(
  shredded_wheat_cereals,
  rownames(cereals_num)
)] <- "shredded"

cereals_num$label <- ""
cereals_num$label[cereals_num_sub] <- rownames(cereals_num)[cereals_num_sub]
```

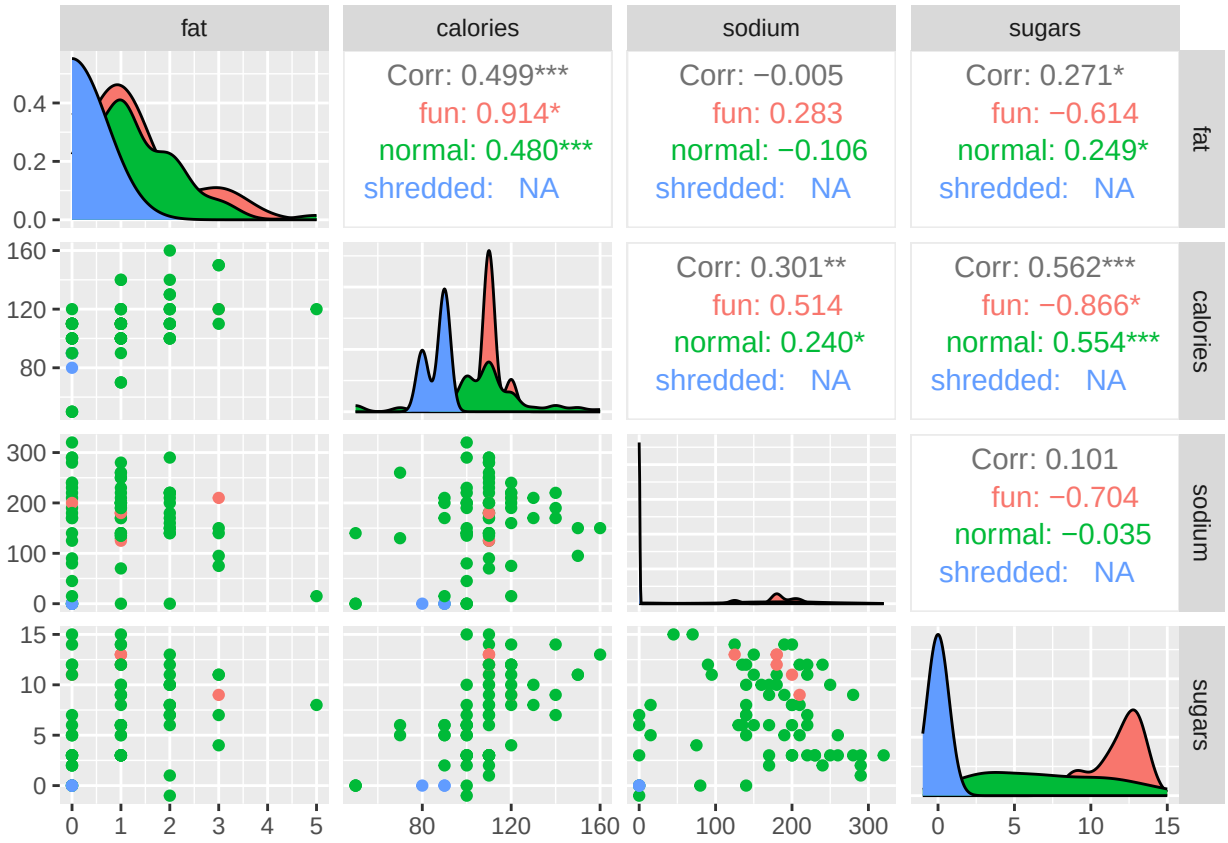
Now, we can think of some questions to ask about our data:

Are all fun cereals the same? what about the shredded ones? Can some cereals not currently classified be added to any category? How different are the cereals in each category?

A quick look at our data is still pretty confusing:

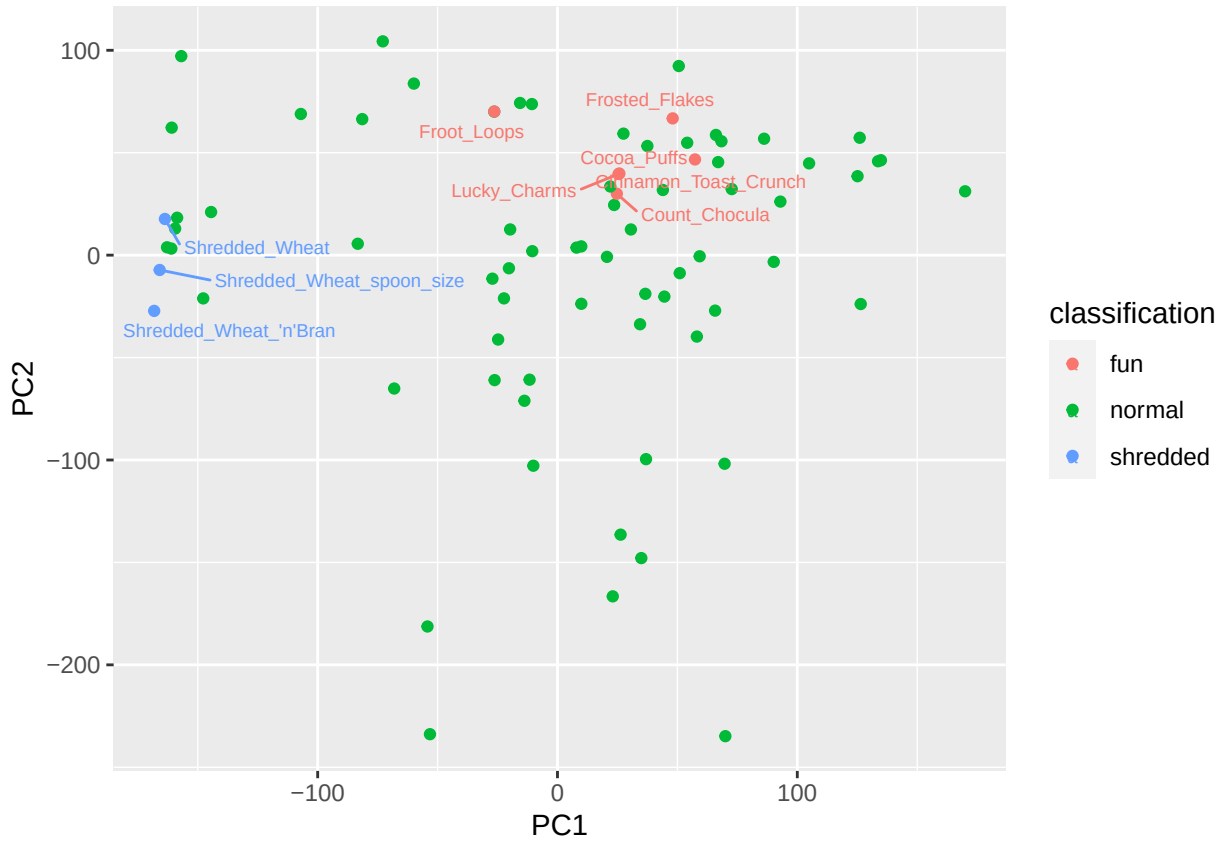
```
ggpairs(cereals_num,
  columns = c("fat", "calories", "sodium", "sugars"),
  ggplot2::aes(colour = classification)
)
```

```
## Warning in cor(x, y): the standard deviation is zero

## Warning in cor(x, y): the standard deviation is zero

## Warning in cor(x, y): the standard deviation is zero

## Warning in cor(x, y): the standard deviation is zero

## Warning in cor(x, y): the standard deviation is zero

## Warning in cor(x, y): the standard deviation is zero
```



Because our data contains many columns, we need a way to visualize most of the complexity represented in all dimensions in just two or three dimensions. Fortunately, that is exactly what dimension reduction does. Here, we apply a simple dimension reduction technique called principal component analysis (PCA) to all 10 numerical variables in the data set. * Important to mention that before doing any dimensionality reduction, it is important to preprocess your data.

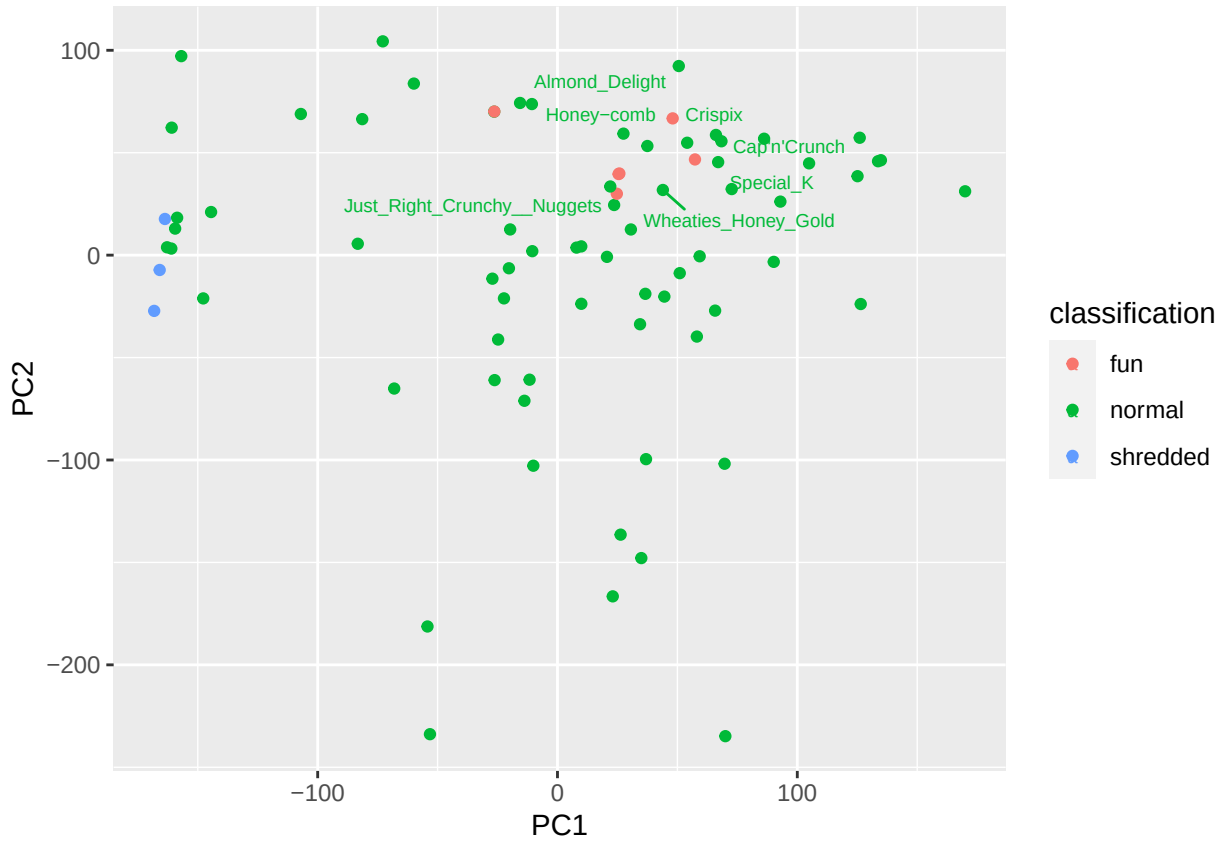
```
# Preform PCA on our data
prcomp_cereals_num <- prcomp(cereals_num[, 1:10])
pca_cereals_num <- data.frame(
  PC1 = prcomp_cereals_num$x[, 1],
  PC2 = prcomp_cereals_num$x[, 2],
  label = cereals_num$label,
  classification = cereals_num$classification
)

ggplot(pca_cereals_num, aes(x = PC1, y = PC2, label = label, col = classification)) +
  geom_point() +
  ggrepel::geom_text_repel(cex = 2.5)
```



Using the PCA, we can now easily find cereals that are similar to some of the fun cereals. These will have similar values of the principal components as the fun cereals. Let's find out who they are:

```
cereals_num_int <- which(pca_cereals_num$PC1 > 0 &
  pca_cereals_num$PC1 < 75 &
  pca_cereals_num$PC2 > 25)

pca_cereals_num$label2 <- ""
pca_cereals_num$label2[cereals_num_int] <- rownames(cereals_num)[cereals_num_int]
pca_cereals_num$label2[cereals_num_sub] <- ""

ggplot(pca_cereals_num, aes(x = PC1, y = PC2, label = label2, col = classification)) +
  geom_point() +
  ggrepel::geom_text_repel(cex = 2.5)
```

```
## Warning: ggrepel: 3 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



According to the first two principal components other fun cereals could be Honeycomb, Cap'n'Crunch and Crispix.

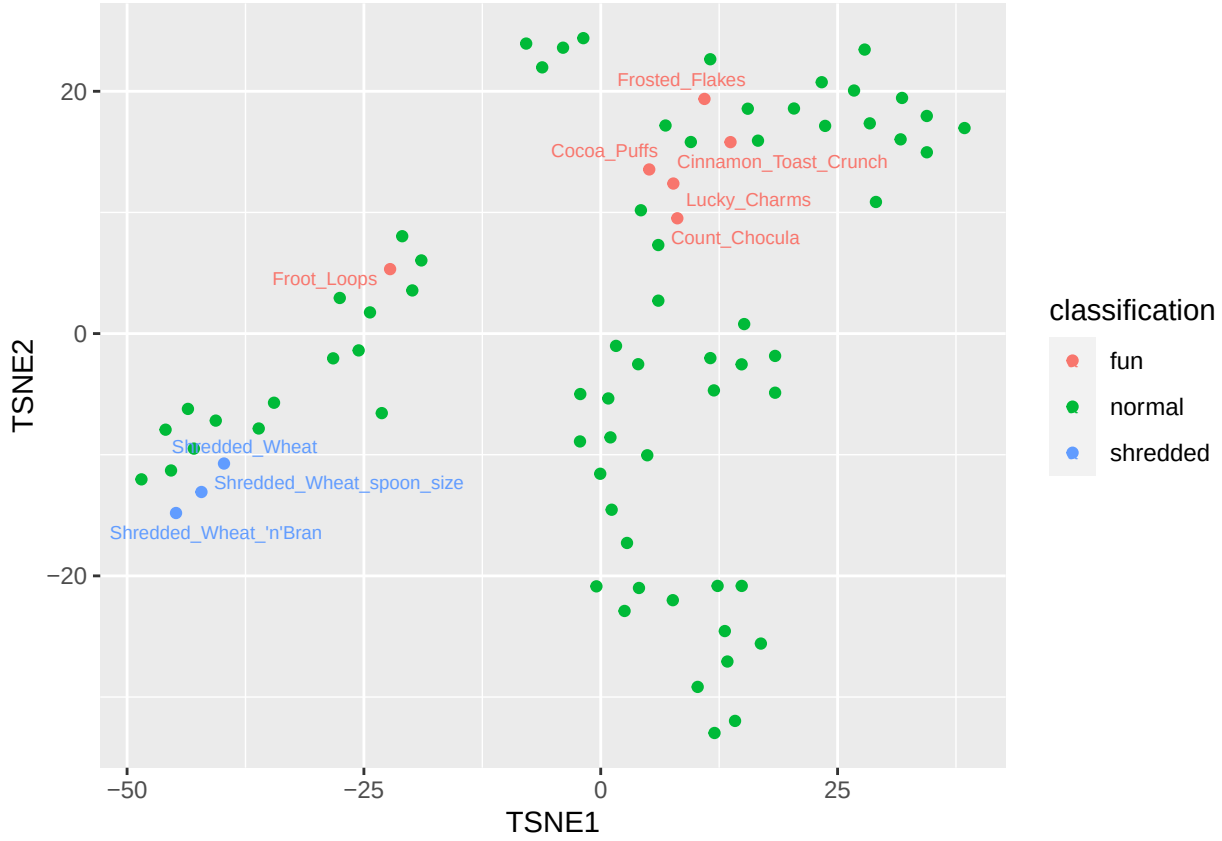
Some more dimensionality reduction techniques

TSNE

T-distributed Stochastic Neighbor Embedding (t-SNE) is a nonlinear dimension reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions. Specifically, it models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability. The main application of t-SNE is for visualizations of large amount of data, as it can recover well-separated clusters.

```
library(Rtsne)
tsne_cereals_num <- Rtsne(cereals_num[, 1:12],
  pca = FALSE, perplexity = 10,
  theta = 0.0
)

tsne_cereals_num <- data.frame(
  TSNE1 = tsne_cereals_num$Y[, 1],
  TSNE2 = tsne_cereals_num$Y[, 2],
  label = cereals_num$label,
```

```
  classification = cereals_num$classification
)

ggplot(tsne_cereals_num, aes(
  x = TSNE1, y = TSNE2,
  label = label, col = classification
)) +
  geom_point() +
  ggrepel::geom_text_repel(cex = 2.5)
```



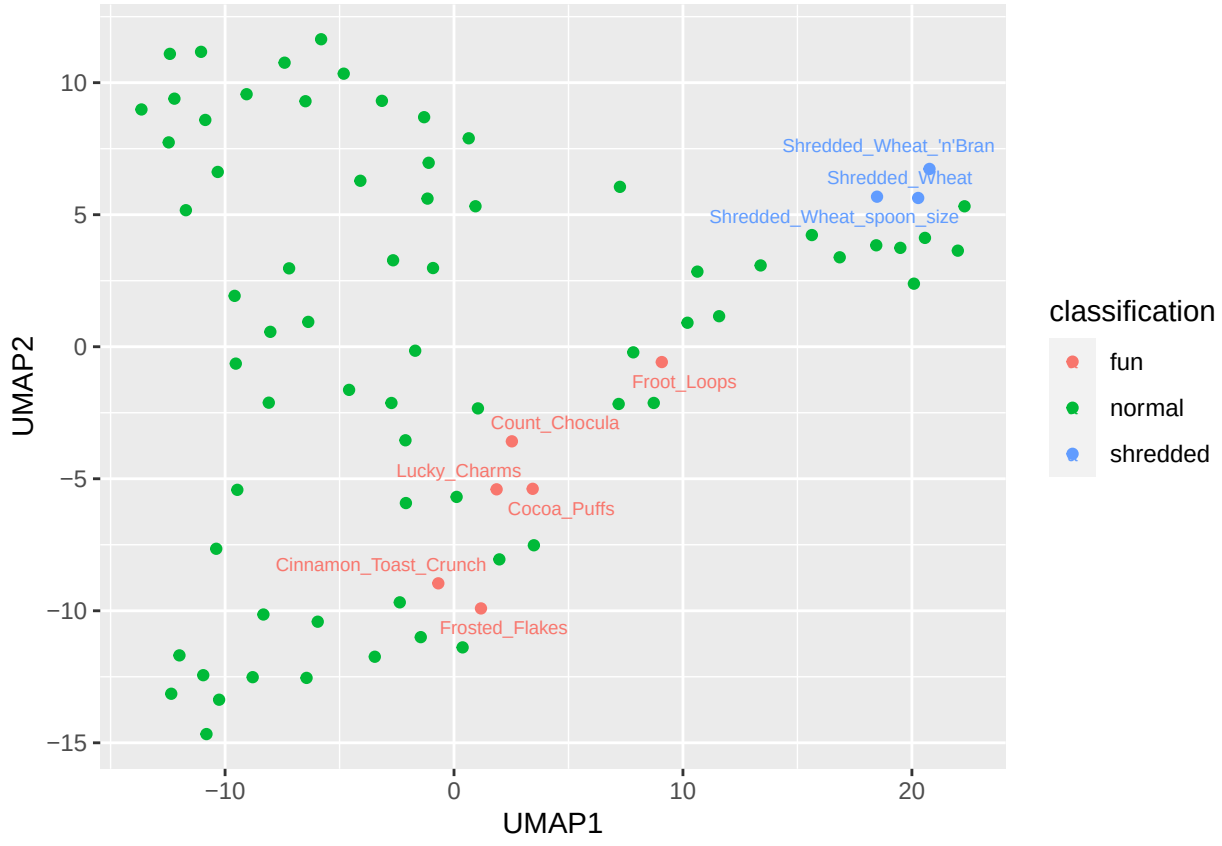
UMAP

Uniform Manifold Approximation and Projection (UMAP) produce a low dimensional representation of high dimensional data that preserves relevant structure. It searches for a low dimensional projection of the data that has the closest possible equivalent topological structure to the original high dimensional data. UMAP is similar to t-SNE but preserves more global structure in the data. Its main use is visualization of a large amount of observations as it recovers well-separated clusters.

```
library(uwot)
```

```
## Loading required package: Matrix
```

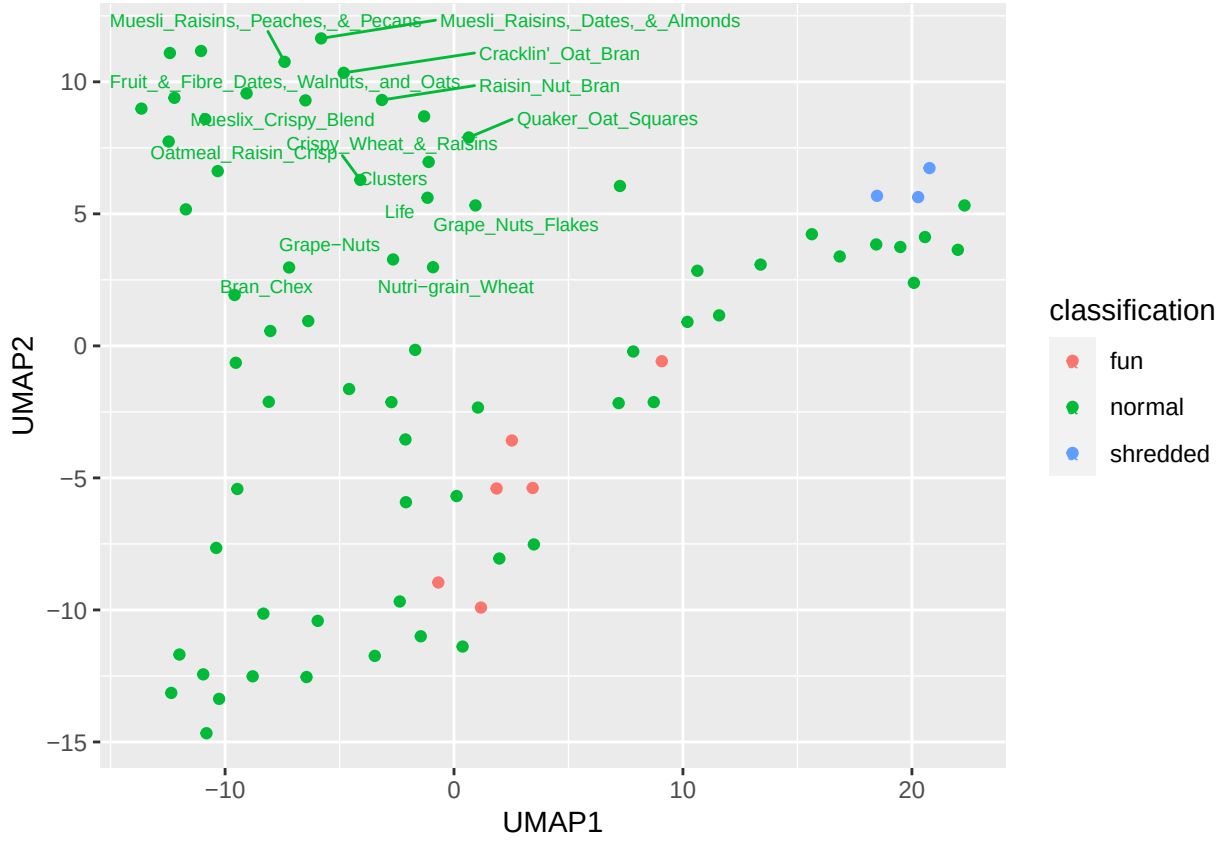
```
umap_cereals_num <- umap(cereals_num[, 1:12],
  n_neighbors = 15,
  min_dist = 1, spread = 5
)

umap_cereals_num <- data.frame(
  UMAP1 = umap_cereals_num[, 1],
  UMAP2 = umap_cereals_num[, 2],
  label = cereals_num$label,
  classification = cereals_num$classification
)

ggplot(umap_cereals_num, aes(
  x = UMAP1, y = UMAP2,
  label = label, col = classification
)) +
  geom_point() +
  ggrepel::geom_text_repel(cex = 2.5)
```



Let's use the UMAP results to see what normal cereals are close to the fun ones.

```
cereals_num_int <- which(umap_cereals_num$UMAP1 < 2 &
  umap_cereals_num$UMAP1 > -10 &
  umap_cereals_num$UMAP2 > 2 &
  umap_cereals_num$UMAP2 < 12)
```

```
umap_cereals_num$label2 <- ""
umap_cereals_num$label2[cereals_num_int] <- rownames(cereals_num)[cereals_num_int]
umap_cereals_num$label2[cereals_num_sub] <- ""

ggplot(umap_cereals_num, aes(
  x = UMAP1, y = UMAP2,
  label = label2, col = classification
)) +
  geom_point() +
  ggrepel::geom_text_repel(cex = 2.5)
```



Number of dimensions to retain

So far we have really only looked at the first two dimensions obtained after dimension reduction. However, this was merely a choice. In fact dimensions beyond the first two, often contain vital information. One way to check this is to look at the variance explained by each principle component. In fact it is good practice when using PCA to state the explained variance by every principle component plotted.

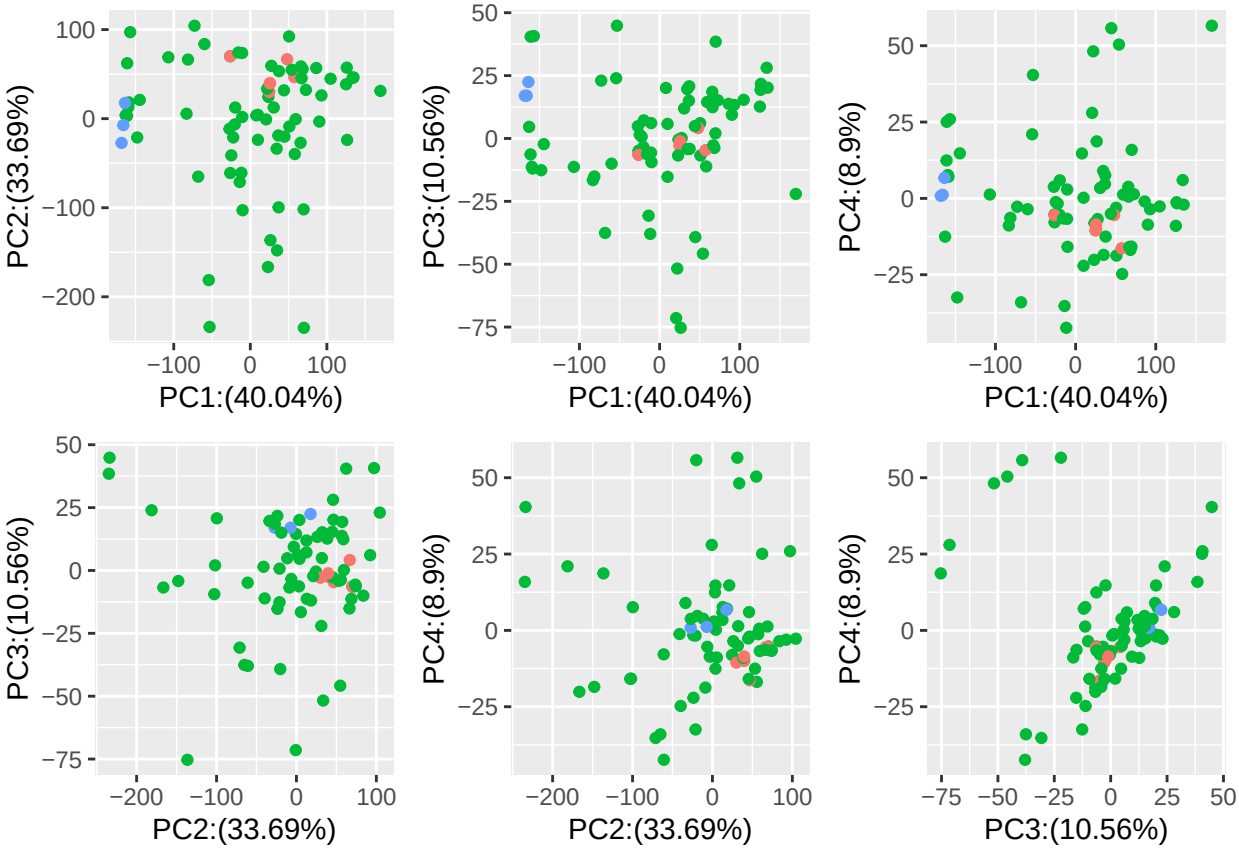
```
prcomp_cereals_num <- prcomp(cereals_num[, 1:10])
var_exp <- round(prcomp_cereals_num$sdev[1:4] / sum(prcomp_cereals_num$sdev) * 100, 2)
pca_cereals_num <- data.frame(
  PC1 = prcomp_cereals_num$x[, 1],
  PC2 = prcomp_cereals_num$x[, 2],
  PC3 = prcomp_cereals_num$x[, 3],
```

```
  PC4 = prcomp_cereals_num$x[, 4],
  label = cereals_num$label,
  classification = cereals_num$classification
)

comb <- list(
  first = c(1, 2), second = c(1, 3), third = c(1, 4), fourth = c(2, 3),
  fifth = c(2, 4), sixth = c(3, 4)
)

gg <- lapply(comb, function(x) {
  ggplot(pca_cereals_num, aes_string(
    x = paste0("PC", x[1]),
    y = paste0("PC", x[2]), col = "classification"
  )) +
    geom_point() + guides(col = FALSE) + ylab(paste0(
      "PC", x[2], ":(",
      var_exp[x[2]], "%)"
    )) + xlab(paste0("PC", x[1], ":(", var_exp[x[1]], "%)"))
})
```

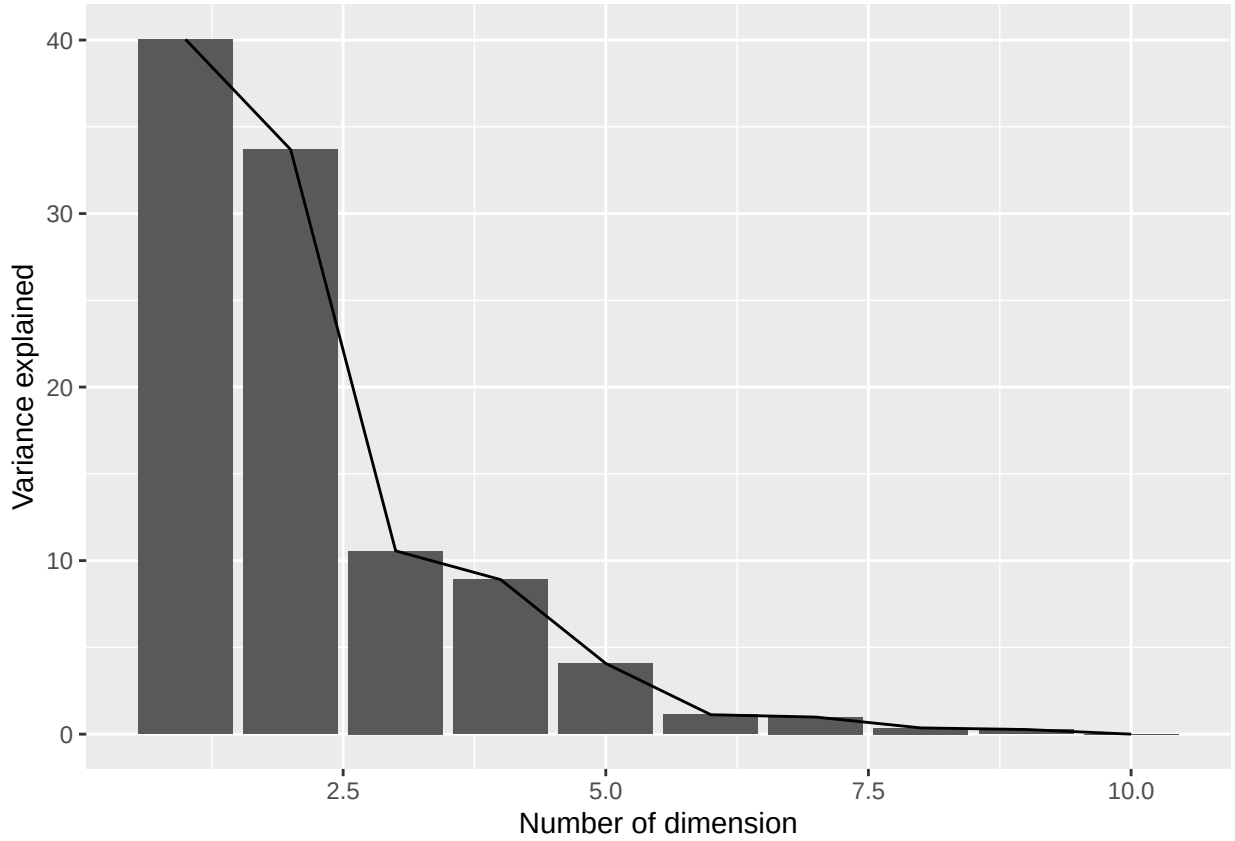
```
## Warning: ‘aes_string()‘ was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation ideoms with ‘aes()‘
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
## Warning: The ‘<scale>‘ argument of ‘guides()‘ cannot be ‘FALSE‘. Use "none" instead as
## of ggplot2 3.3.4.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
plot_grid(plotlist = gg)
```



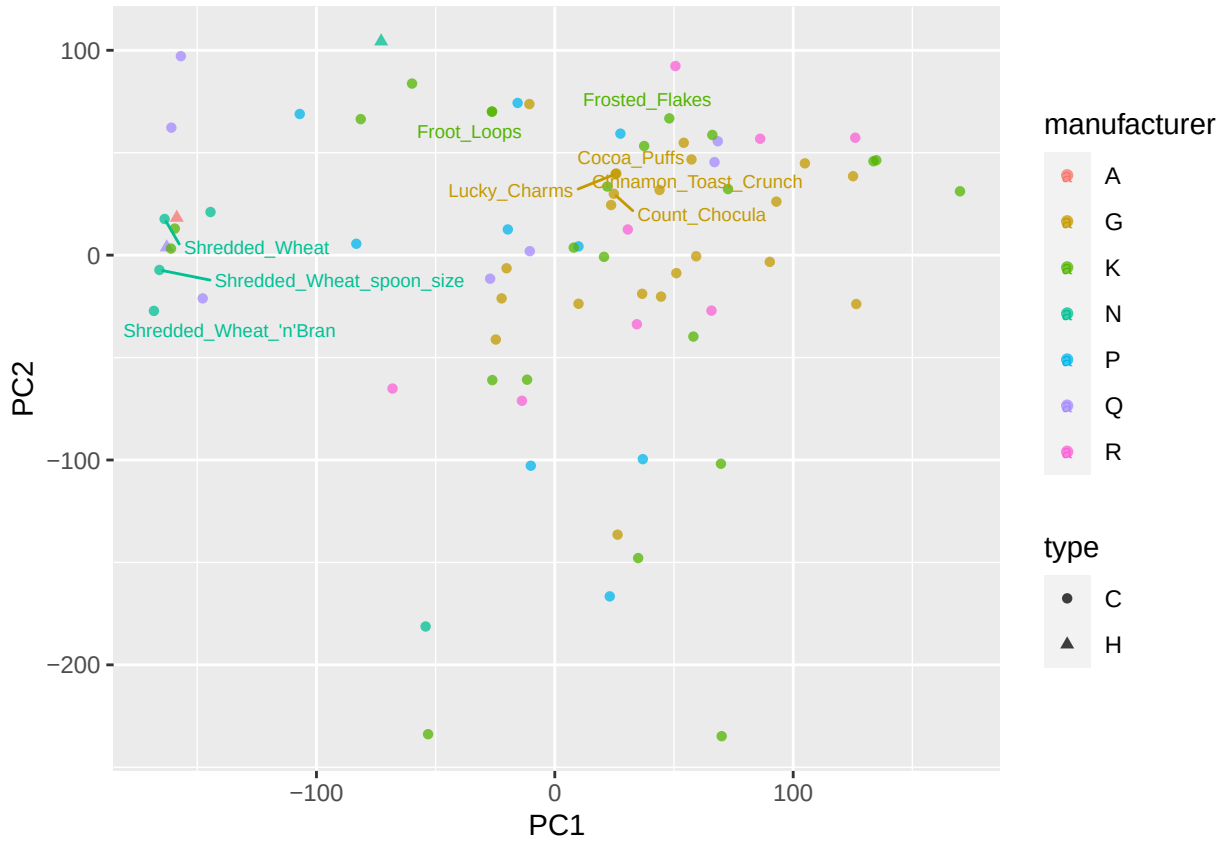

```
var_exp <- data.frame(
  "Variance explained" =
    prcomp_cereals_num$sdev / sum(prcomp_cereals_num$sdev) * 100,
  "Number of dimension" = 1:length(prcomp_cereals_num$sde),
  check.names = FALSE
)

ggplot(var_exp, aes(x = `Number of dimension`, y = `Variance explained`)) +
  geom_col() + geom_line()
```



Hiden signals

The primary objective of dimension reduction is to compress data to uncover patterns. Often after identifying patterns, it is of interest to explain their appearance, i.e. find the "hidden signal". In data collection, we often collect additional information that are not included in the dimension reduction calculations but can hold the key to understanding these patterns. The simplest way to reveal such patterns is to use these external covariates is to include them in dimension reduction visualizations — with their values encoded as color, shape, size, or even transparency of corresponding points on the plot.

```
pca_cereals_num$manufacturer <- cereals$Manufacturer
pca_cereals_num$type <- cereals$Cold.or.Hot

ggplot(pca_cereals_num, aes(
  x = PC1, y = PC2, label = label, col = manufacturer,
  shape = type
)) +
  geom_point(alpha = 0.75) +
  ggrepel::geom_text_repel(cex = 2.5)
```



We can see that the cereal are also clustering according to the manufacturer!

Feature Selection

Now we will go over feature selection, a very important task that improves the performance of machine learning models.

Remove Redundant Features - Correlation Matrix

Different datasets will contain different features. Some will be highly correlated with each-other and some won't. We want to understand the different correlations between the features - highly correlated features will have a better affect on many methods.

Generally, you want to remove attributes with an absolute correlation of 0.75 or higher.

```
# load packages
library(mlbench) # for diabetes dataset
library(caret)
```

```
## Loading required package: lattice
```

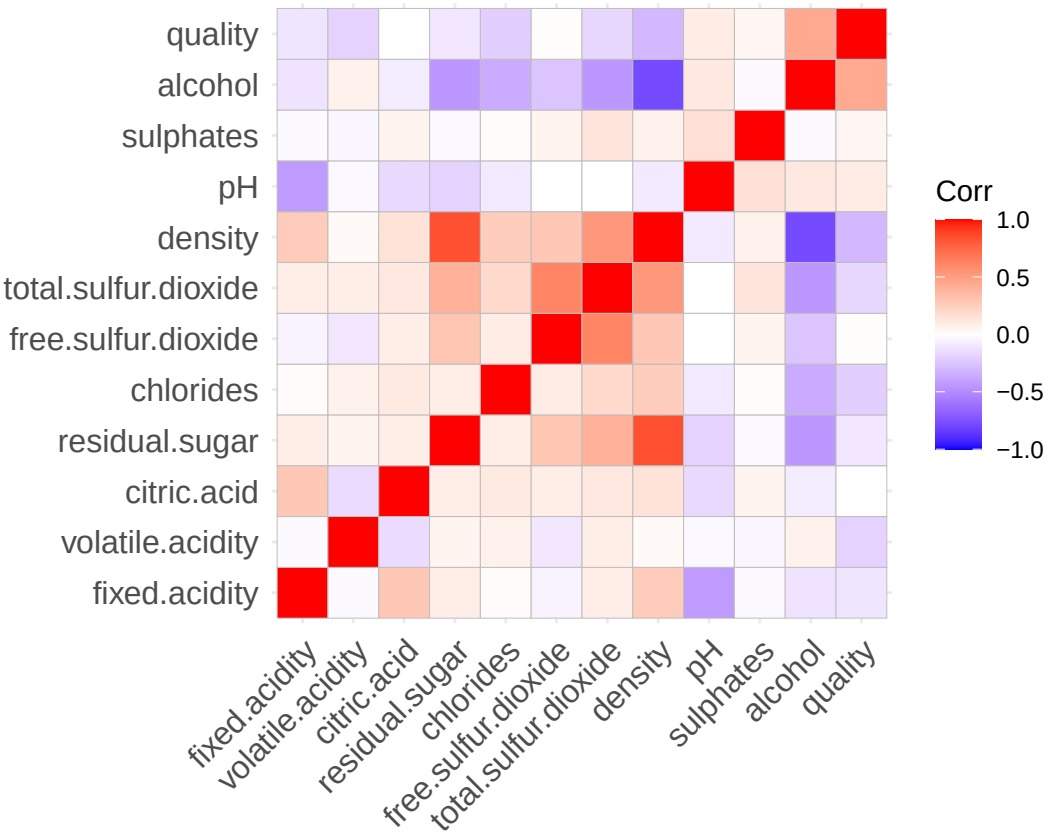
```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(ggcorrplot)
```

```
# ensure the results are repeatable
set.seed(7)
# load the data
winequality <- read.csv("http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequali
# calculate correlation matrix
correlationMatrix <- cor(winequality)
# display the correlation matrix
ggcorrplot(correlationMatrix)
```




```
# find attributes that are highly correlated
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.75)
# print indexes of highly correlated attributes
print(highlyCorrelated)
```

```
## [1] 8
```

In the above results we see that attribute #8, density, is highly correlated with the rest. For this reason we could remove it.

Rank Features By Importance

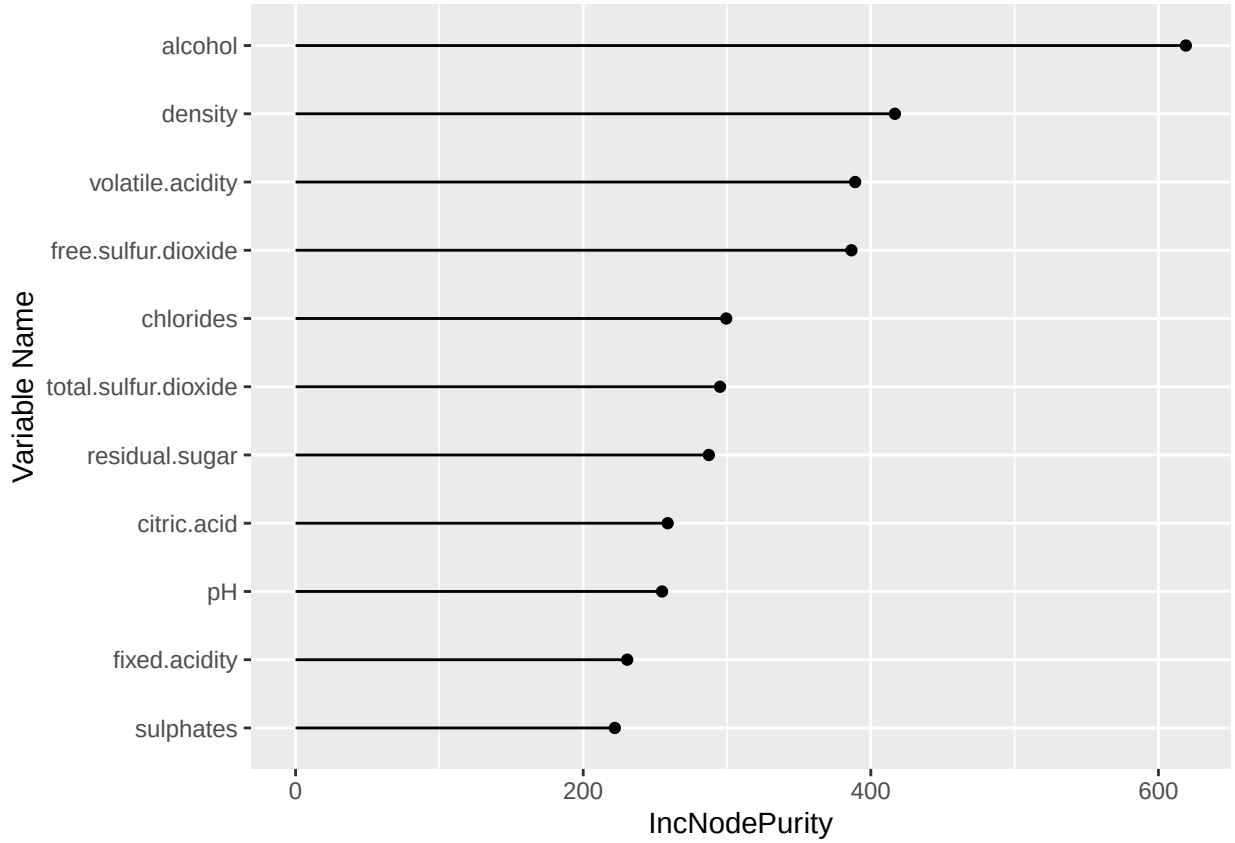
The importance of features can be estimated from data by building a model. Some methods have this built in, and some measure it using ROC curve analysis.

We will continue with the same dataset from above. The varImp is then used to estimate the variable importance, which is printed and plotted. Here we are using the logistic regression method to train, but we won't talk about it in further detail. We will start learning about training and ML algorithms starting next practice.

```
#train random forest model and calculate feature importance
rf = randomForest(x= winequality[,1:11],y= winequality[,12])
# estimate variable importance
importance <- varImp(rf, scale=FALSE)
#sort the score in decreasing order
var_imp_df <- data.frame(cbind(variable = rownames(importance), score = importance[,1]))
var_imp_df$score <- as.double(var_imp_df$score)
var_imp_df[order(var_imp_df$score,decreasing = TRUE),]
```

```
##                variable    score
## 11              alcohol 619.0751
## 8               density 416.7203
## 2      volatile.acidity 389.1000
## 6    free.sulfur.dioxide 386.5097
## 5              chlorides 299.4458
## 7   total.sulfur.dioxide 295.1347
## 4         residual.sugar 287.3480
## 3            citric.acid 258.7359
## 9                     pH 254.8268
## 1          fixed.acidity 230.5983
## 10            sulphates 221.9580
```

```
# plot ranking of features
ggplot(var_imp_df, aes(x=reorder(variable, score), y=score)) +
  geom_point() +
  geom_segment(aes(x=variable,xend=variable,y=0,yend=score)) +
  ylab("IncNodePurity") +
  xlab("Variable Name") +
  coord_flip()
```



We see that the alcohol, density and volatile.acidity are the top 3 most important attributes in the dataset and the sulfates attribute is the least important.

Feature Selection

The third technique for feature selection we'll see is automatic feature selection. It can be used to build many models with different subsets of a dataset and identify those attributes that are and are not required to build an accurate model. A popular method for this is called Recursive Feature Elimination or RFE (caret package).

A Random Forest algorithm is used on each iteration to evaluate the model (we will learn about it in a future practice). The algorithm is configured to explore all possible subsets of the attributes. All 11 attributes are selected in this example, although in the plot showing the accuracy of the different attribute subset sizes, we can see that just 6 attributes gives almost comparable results.

```
# define the control using a random forest selection function
filterCtrl <- rfeControl(functions=rfFuncs, method="cv", number=3)
# run the RFE algorithm
results <- rfe(x= winequality[,1:11],y= winequality[,12], sizes=c(1:11), rfeControl=filterCtrl)
# summarize the results
print(results)
```

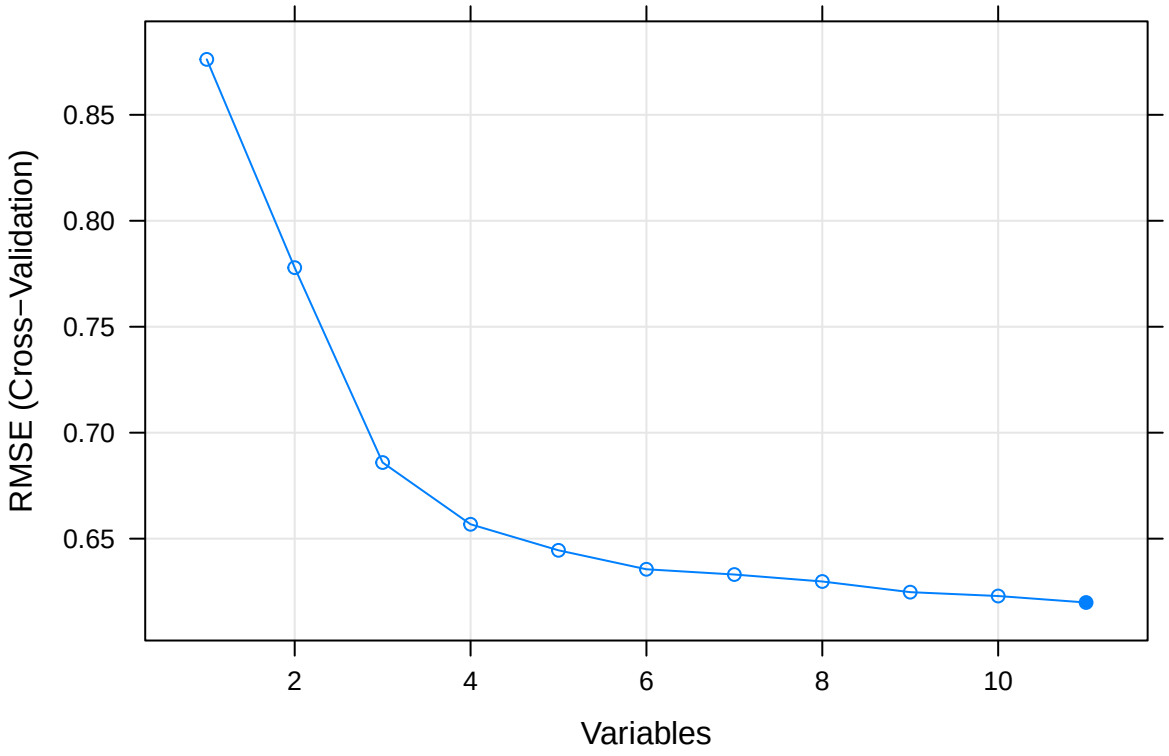
```
##
## Recursive feature selection
##
```

```
## Outer resampling method: Cross-Validated (3 fold)
##
## Resampling performance over subset size:
##
##  Variables   RMSE Rsquared    MAE   RMSESD RsquaredSD    MAESD Selected
##          1 0.8762  0.03125 0.6708 0.017792   0.002959 0.009911
##          2 0.7779  0.23661 0.5981 0.057750   0.112859 0.046384
##          3 0.6859  0.40122 0.5217 0.005709   0.018902 0.005134
##          4 0.6568  0.45432 0.4951 0.008925   0.015798 0.008548
##          5 0.6445  0.47949 0.4831 0.013437   0.007519 0.008910
##          6 0.6355  0.48860 0.4677 0.011085   0.010066 0.006957
##          7 0.6331  0.49376 0.4650 0.012377   0.013215 0.010213
##          8 0.6298  0.50117 0.4640 0.014179   0.007488 0.008688
##          9 0.6248  0.50807 0.4572 0.011944   0.010477 0.007147
##         10 0.6229  0.51210 0.4546 0.014253   0.005748 0.008258
##         11 0.6199  0.51744 0.4529 0.013609   0.007337 0.009360        *
##
## The top 5 variables (out of 11):
##    volatile.acidity, alcohol, free.sulfur.dioxide, pH, residual.sugar
```

```
# list the chosen features
predictors(results)
```

```
##  [1] "volatile.acidity"     "alcohol"              "free.sulfur.dioxide"
##  [4] "pH"                   "residual.sugar"       "citric.acid"
##  [7] "chlorides"            "total.sulfur.dioxide" "fixed.acidity"
## [10] "sulphates"            "density"
```

```
# plot the results
plot(results, type=c("g", "o"))
```


RMSE (Cross−Validation)
0.85
0.80
0.75
0.70
0.65
2
4
6
8
10
Variables