Statistical Models in R: Part 1

MEF - BDA 503

Nov 21, 2017

Contents

Principle Components Analysis (PCA) **1**
- Properties of PCA . . . 3
- Examples . . . 4
  - Made-up Example regarding Transformations . . . 4
  - Young People Survey . . . 5
- References . . . 6

Multidimensional Scaling (MDS) **6**
- Properties . . . 12
- References . . . 12

K-Means **12**
- Properties . . . 16

Hierarchical Clustering **16**
- Properties . . . 21
- References . . . 21

Supervised Learning (Regression and Classification) **21**
- Linear Regression . . . 22
- Logistic regression . . . 24
- K Nearest Neighbors (KNN) . . . 27
- References . . . 28

Principle Components Analysis (PCA)

Let's recall the example from the lecture notes. Our survey asks about how much they care about price, software, aesthetics and brand when choosing a new computer. There are 16 respondents with differing operating systems (OS). (e.g. 0 is Windows, 1 is Mac). Data set is as follows. Likert answers from 1 to 7 is a scale from strongly disagree to strongly agree.

```
#Customer Survey Data
survey_data <- data.frame(
customer = 1:16,
OS = c(0,0,0,0,1,0,0,0,1,1,0,1,1,1,1,1),
Price = c(6,7,6,5,7,6,5,6,3,1,2,5,2,3,1,2),
Software = c(5,3,4,7,7,4,7,5,5,3,6,7,4,5,6,3),
Aesthetics = c(3,2,4,1,5,2,2,4,6,7,6,7,5,6,5,7),
Brand = c(4,2,5,3,5,3,1,4,7,5,7,6,6,5,5,7)
)

#Let's do some exploratory analysis
summary(survey_data[,3:6])
```

```
##      Price          Software       Aesthetics       Brand
##  Min.   :1.000   Min.   :3.000   Min.   :1.00   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:4.000   1st Qu.:2.75   1st Qu.:3.750
##  Median :5.000   Median :5.000   Median :5.00   Median :5.000
##  Mean   :4.188   Mean   :5.062   Mean   :4.50   Mean   :4.688
##  3rd Qu.:6.000   3rd Qu.:6.250   3rd Qu.:6.00   3rd Qu.:6.000
##  Max.   :7.000   Max.   :7.000   Max.   :7.00   Max.   :7.000
```

```
#Correlation Matrix
cor(survey_data[,3:6])
```

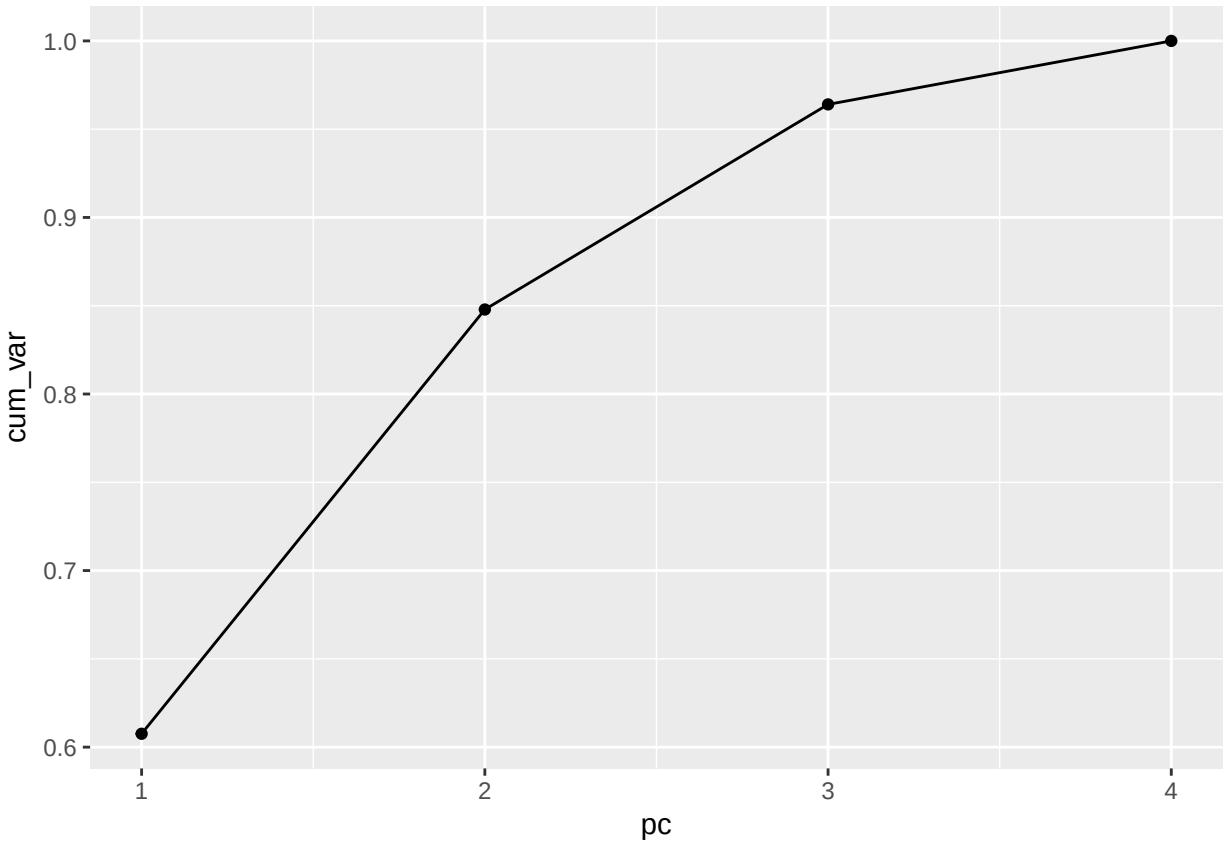
```
##                 Price   Software Aesthetics      Brand
## Price       1.0000000  0.1856123 -0.6320222 -0.5802668
## Software    0.1856123  1.0000000 -0.1462152 -0.1185864
## Aesthetics -0.6320222 -0.1462152  1.0000000  0.8528544
## Brand      -0.5802668 -0.1185864  0.8528544  1.0000000
```

```
#Do PCA with princomp function and use correlation matrix to create components
survey_pca <- princomp(as.matrix(survey_data[,3:6]),cor=T)
summary(survey_pca,loadings=TRUE)
```

```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3     Comp.4
## Standard deviation     1.5589391 0.9804092 0.6816673 0.37925777
## Proportion of Variance 0.6075727 0.2403006 0.1161676 0.03595911
## Cumulative Proportion  0.6075727 0.8478733 0.9640409 1.00000000
##
## Loadings:
##            Comp.1 Comp.2 Comp.3 Comp.4
## Price      -0.523         0.848
## Software   -0.177  0.977 -0.120
## Aesthetics  0.597  0.134  0.295 -0.734
## Brand       0.583  0.167  0.423  0.674
```

As you can clearly see, the first 2 PC is responsible for 85% of the variation. First PC considers Price and Software as negative effects and Aesthetics and Brand as a positive effect, second PC is highly affected by Software. With the third PC, 96.4% of the variation is explained.

```
library(ggplot2)
ggplot(data.frame(pc=1:4,cum_var=c(0.6075727,0.8478733,0.9640409,1.00000000)),
    aes(x=pc,y=cum_var)) + geom_point() + geom_line()
```



You see, there is a trade-off between explanatory power and number of explanatory variables (dimensions).

Properties of PCA

- PCA is a method to reduce the number of explanatory variables. The resulting principle components (PC) are actually linear transformations of your explanatory variables. (e.g. $PC_i = a_1 * x_1 + a_2 * x_2 + \cdots + a_p * x_p$).
- If some or most of your explanatory variables are highly correlated, then PCA is the way to go.
- Total number of PCs are equal to total number of explanatory variables. Though, it is possible to eliminate some PCs without considerable loss in explanatory power.
- PCs are fairly independent (i.e. low correlation).
- You cannot use categorical or binary variables in PCA.
- PCA is more of an exploration tool than an explanation or prediction tool. Though, it is possible to use PCs as input to other methods (i.e. regression).
- PCA considers linear relationship, so it is not good for non-linear relations (i.e. $x_1 = x_2^2$). It also assumes normally distributed errors, so fat tailed distributions are a poor fit to PCA.
- It is possible to use covariance matrix rather than the correlation matrix.
- Centering and scaling can be done to get better results. Centering sets the mean values of the covariates to 0 and scaling converts explanatory variables to result in unit variance.
- In R you can use both `prcomp` and `princomp` functions. While the former uses singular value decomposition method to calculate principle components, the latter uses eigenvalues and eigenvectors.

Examples

Made-up Example regarding Transformations

```
set.seed(58)
#Randomly create data points around the normal distribution
x1=rnorm(30,mean=2,sd=4)
#Get one linear transformation and one nonlinear transformation of the data
pca_data<-data.frame(x1,x2=x1*2,x3=(x1^2),x4=abs(x1)^(0.5)+rnorm(30))
#See the correlation matrix
pca_cor<-cor(pca_data)
pca_cor
```

```
##            x1         x2        x3         x4
## x1 1.00000000 1.00000000 0.6648541 0.08208783
## x2 1.00000000 1.00000000 0.6648541 0.08208783
## x3 0.66485406 0.66485406 1.0000000 0.40646124
## x4 0.08208783 0.08208783 0.4064612 1.00000000
```

```
#See the eigenvalues and eigenvectors
pca_eigen<-eigen(pca_cor)
pca_eigen
```

```
## eigen() decomposition
## $values
## [1] 2.625027e+00 1.067917e+00 3.070557e-01 4.510095e-16
##
## $vectors
##            [,1]       [,2]       [,3]          [,4]
## [1,] -0.5856026  0.2603411  0.2988179  7.071068e-01
## [2,] -0.5856026  0.2603411  0.2988179 -7.071068e-01
## [3,] -0.5269453 -0.2545741 -0.8108765  6.106227e-16
## [4,] -0.1909657 -0.8942243  0.4048395 -9.714451e-17
```

```
#See the standard deviations and proportion of variances
sqrt(pca_eigen$values)
```

```
## [1] 1.620194e+00 1.033401e+00 5.541261e-01 2.123698e-08
```

```
pca_eigen$values/sum(pca_eigen$values)
```

```
## [1] 6.562569e-01 2.669792e-01 7.676393e-02 1.127524e-16
```

```
cumsum(pca_eigen$values/sum(pca_eigen$values))
```

```
## [1] 0.6562569 0.9232361 1.0000000 1.0000000
```

```
#Run PCA
pca_result<-princomp(pca_data,cor=T)
#See the PCA
summary(pca_result,loadings=TRUE)
```

```
## Importance of components:
##                           Comp.1    Comp.2     Comp.3 Comp.4
## Standard deviation     1.6201937 1.0334006 0.55412609      0
## Proportion of Variance 0.6562569 0.2669792 0.07676393      0
## Cumulative Proportion  0.6562569 0.9232361 1.00000000      1
```

```
##
## Loadings:
##    Comp.1 Comp.2 Comp.3 Comp.4
## x1 -0.586  0.260  0.299  0.707
## x2 -0.586  0.260  0.299 -0.707
## x3 -0.527 -0.255 -0.811
## x4 -0.191 -0.894  0.405
```

```
pca_result2<-princomp(pca_data[,c("x1","x3","x4")],cor=T)
summary(pca_result2,loadings=TRUE)
```

```
## Importance of components:
##                           Comp.1    Comp.2     Comp.3
## Standard deviation     1.3481348 0.9628649 0.50539453
## Proportion of Variance 0.6058225 0.3090363 0.08514121
## Cumulative Proportion  0.6058225 0.9148588 1.00000000
##
## Loadings:
##    Comp.1 Comp.2 Comp.3
## x1 -0.601 -0.517  0.609
## x3 -0.690        -0.722
## x4 -0.403  0.855  0.327
```

Young People Survey

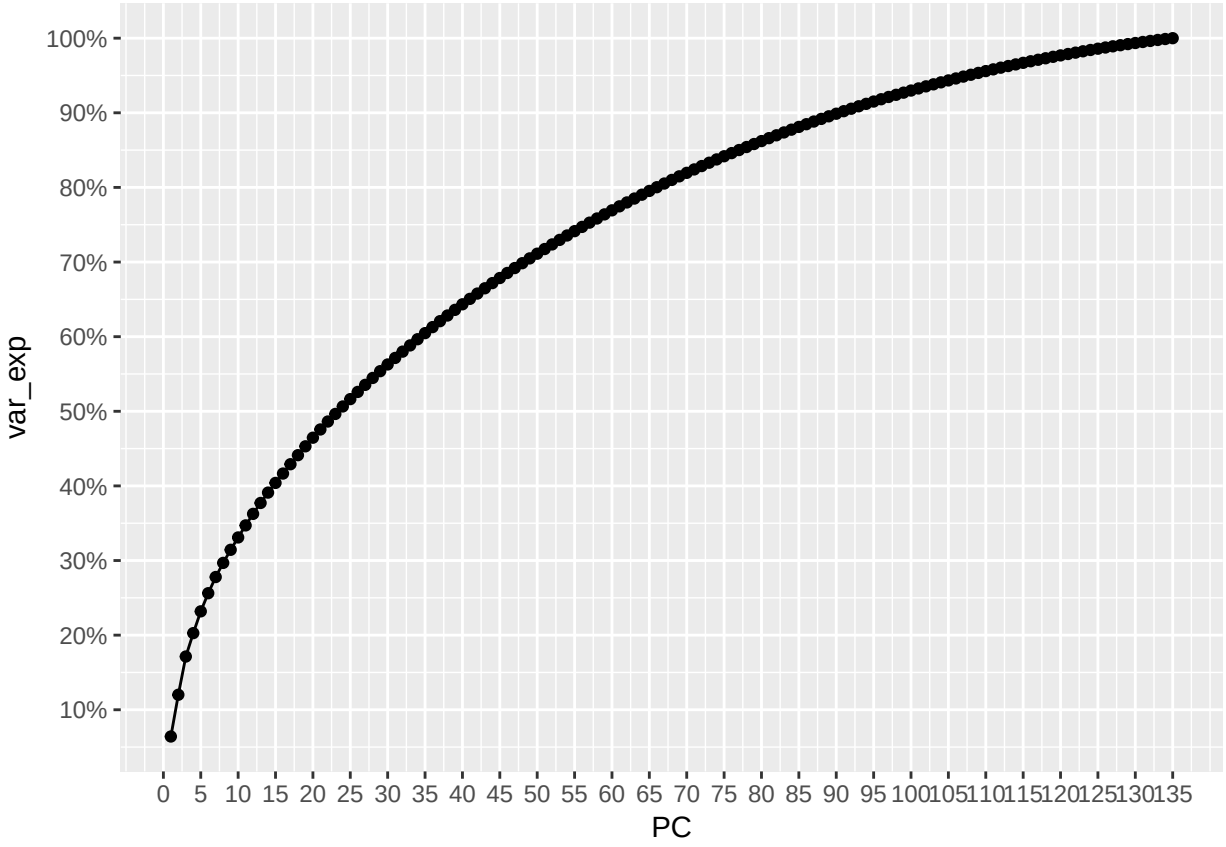
Following data is from Kaggle. Data set is named Young People Survey. It is a simple survey with lots of questions and some demographic data. Resulting PCA analysis helps us to maintain 90% of the variance with just two-thirds of the survey questions.

```
#Prepare data
yr_data <-
read.csv("data/youth_responses.csv",sep=",") %>%
filter(complete.cases(.)) %>%
# mutate(id=row_number()) %>%
tbl_df()

#Prepare PCA data
yr_pca<-
yr_data[,sapply(yr_data,class)=="integer"] %>%
select(Music:Spending.on.healthy.eating)

#Run PCA analysis
yr_pca_result<-princomp(yr_pca,cor=T)

#See the PCA output
ggplot(data=data.frame(PC=1:length(yr_pca_result$sdev),
    var_exp=cumsum(yr_pca_result$sdev^2/sum(yr_pca_result$sdev^2))),
aes(x=PC,y=var_exp)) + geom_line() +
geom_point() +
scale_y_continuous(labels = scales::percent,breaks=seq(0,1,length.out=11)) +
scale_x_continuous(breaks=seq(0,135,by=5))
```



References

- http://www.rpubs.com/aaronsc32/principal-component-analysis
- https://onlinecourses.science.psu.edu/stat505/node/49
- http://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch18.pdf
- https://www.kaggle.com/c/santander-customer-satisfaction/data
- https://www.kaggle.com/c/santander-customer-satisfaction/data
- http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf
- http://www.rpubs.com/koushikstat/pca
- https://www.kaggle.com/miroslavsabo/young-people-survey

Multidimensional Scaling (MDS)

Classical multidimensional scaling is kind of a reverse engineering method. Basically, from a distance matrix it can generate approximate locations of nodes in the coordinate system.

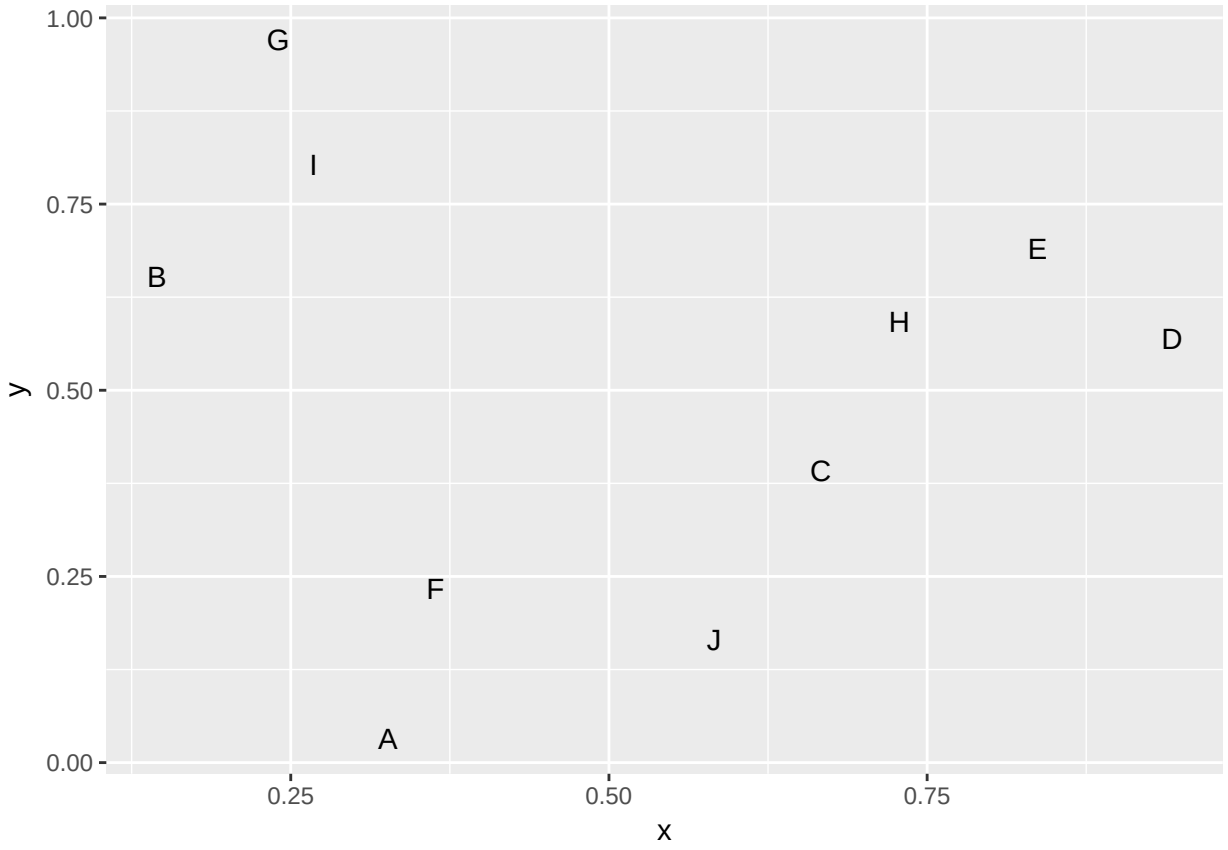
The most classical example is if a distance matrix between cities are used as an input, MDS gives the coordinates as the output. In the following example we are going to "generate" 10 cities, plot them, get distance matrix, run MDS on the distance matrix and get coordinates back.

```
#Set the seed for reproducibility
set.seed(58)
#Create a coordinate matrix (all coordinates are between 0 and 1).
#Suppose the places of cities A to J
coord_mat<-matrix(round(runif(20),6),ncol=2)
```

```r
#Column names of the coordinate matrix, say x and y
colnames(coord_mat)<-c("x","y")
#Row names of the coordinates are cities.
#LETTERS is a predefined vector with letters of the English alphabet.
rownames(coord_mat)<-LETTERS[1:10]
#Create distance matrix
dist_mat<-dist(coord_mat)
#Display coordinate matrix
print(coord_mat)
```

```
##          x        y
## A 0.326454 0.031954
## B 0.144960 0.653293
## C 0.666349 0.392102
## D 0.942411 0.569635
## E 0.836936 0.690193
## F 0.363652 0.234271
## G 0.240290 0.970383
## H 0.728265 0.592039
## I 0.267996 0.802786
## J 0.582656 0.164829
```

```r
ggplot(as.data.frame(coord_mat),aes(x=x,y=y)) + geom_text(label=rownames(coord_mat))
```




```r
print(dist_mat)
```

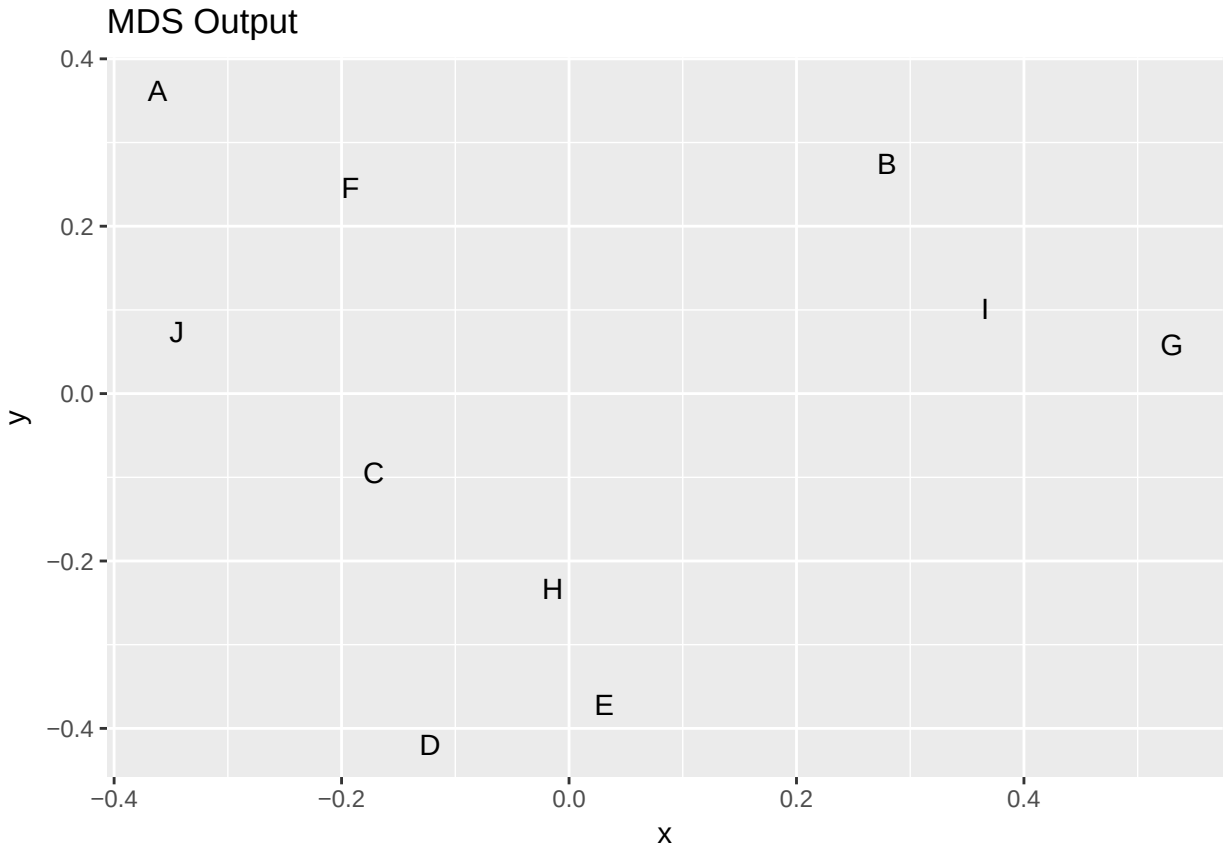
```
##          A         B         C         D         E         F         G
## B 0.6473038
```

```
## C 0.4952123 0.5831528
## D 0.8176209 0.8018271 0.3282197
## E 0.8329889 0.6929592 0.3434504 0.1601849
## F 0.2057082 0.4726580 0.3413738 0.6689028 0.6571625
## G 0.9423764 0.3311101 0.7182863 0.8084385 0.6591607 0.7463773
## H 0.6893093 0.5865124 0.2093046 0.2153148 0.1464363 0.5108234 0.6174656
## I 0.7730455 0.1936131 0.5721420 0.7135790 0.5799741 0.5765062 0.1698716
## J 0.2886091 0.6558772 0.2421932 0.5415640 0.5836657 0.2297497 0.8752895
##           H         I
## B
## C
## D
## E
## F
## G
## H
## I 0.5062231
## J 0.4513428 0.7113368
```

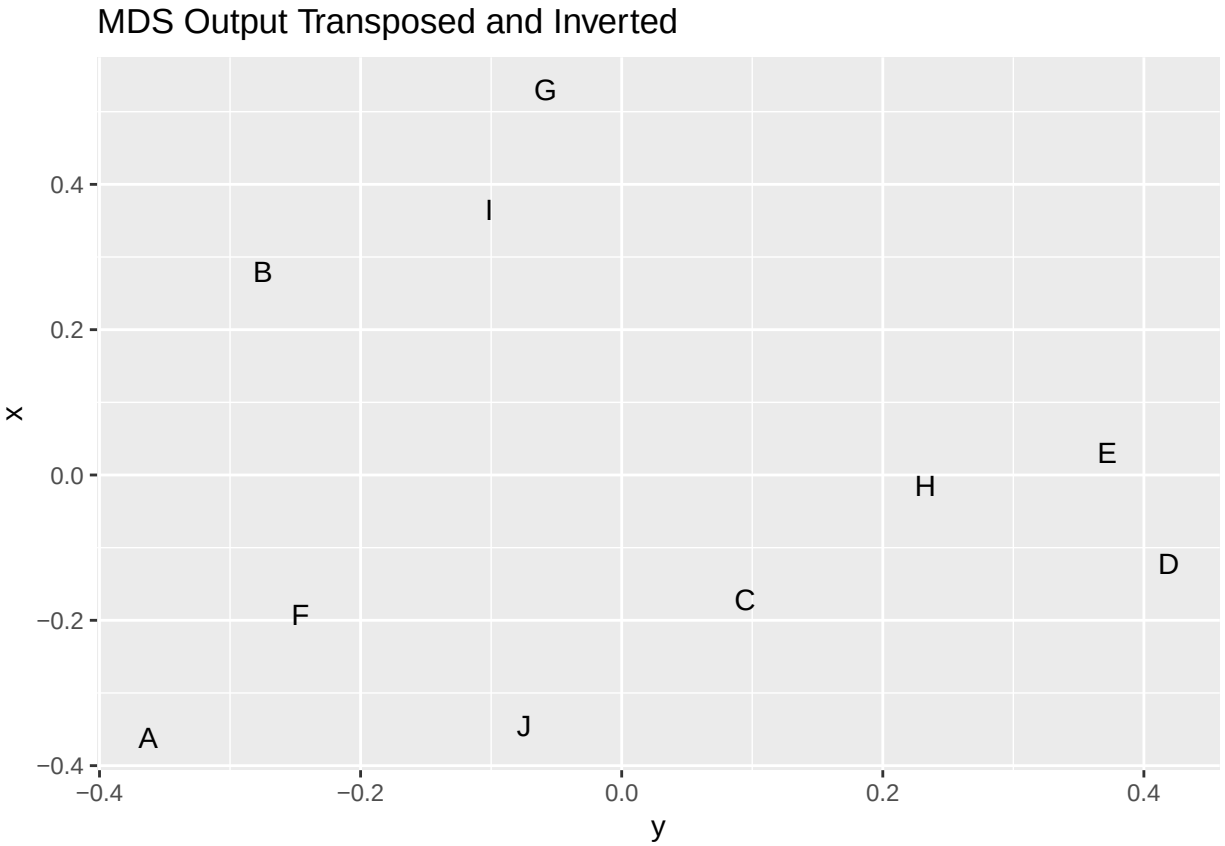
```
#Now let's employ Multidimensional Scaling (MDS)
#Base R has a lovely command called cmdscale (Classical multidimensional scaling)
mds_data<-cmdscale(dist_mat,k=2)
colnames(mds_data)<-c("x","y")
#Print the output
print(mds_data)
```

```
##             x           y
## A -0.36165938  0.36271265
## B  0.27965223  0.27483980
## C -0.17157590 -0.09456567
## D -0.12215882 -0.41904394
## E  0.03094720 -0.37195089
## F -0.19213662  0.24618843
## G  0.53023233  0.05840622
## H -0.01431325 -0.23268465
## I  0.36591977  0.10150796
## J -0.34490755  0.07459010
```

```
#Let's plot the output
ggplot(as.data.frame(mds_data),aes(x=x,y=y)) +
geom_text(label=rownames(mds_data)) + labs(title="MDS Output")
```




```
#Not quite similar? Let's manipulate a bit.
ggplot(as.data.frame(mds_data) %>% mutate(y=-y),aes(x=y,y=x)) +
geom_text(label=rownames(mds_data)) +
labs(title="MDS Output Transposed and Inverted")
```



The output is not perfect but a close representation of the original coordinates.

Following example is from the Young People Survey data, again. We would like to see the how respondants music preferences correlate among different genres. We are going to use the correlation matrix as a similarity measure.

```
#Get the Young People Survey data
yr_mds_data <- yr_pca %>% select(Dance:Opera)
print(head(as.data.frame(yr_mds_data)))
```

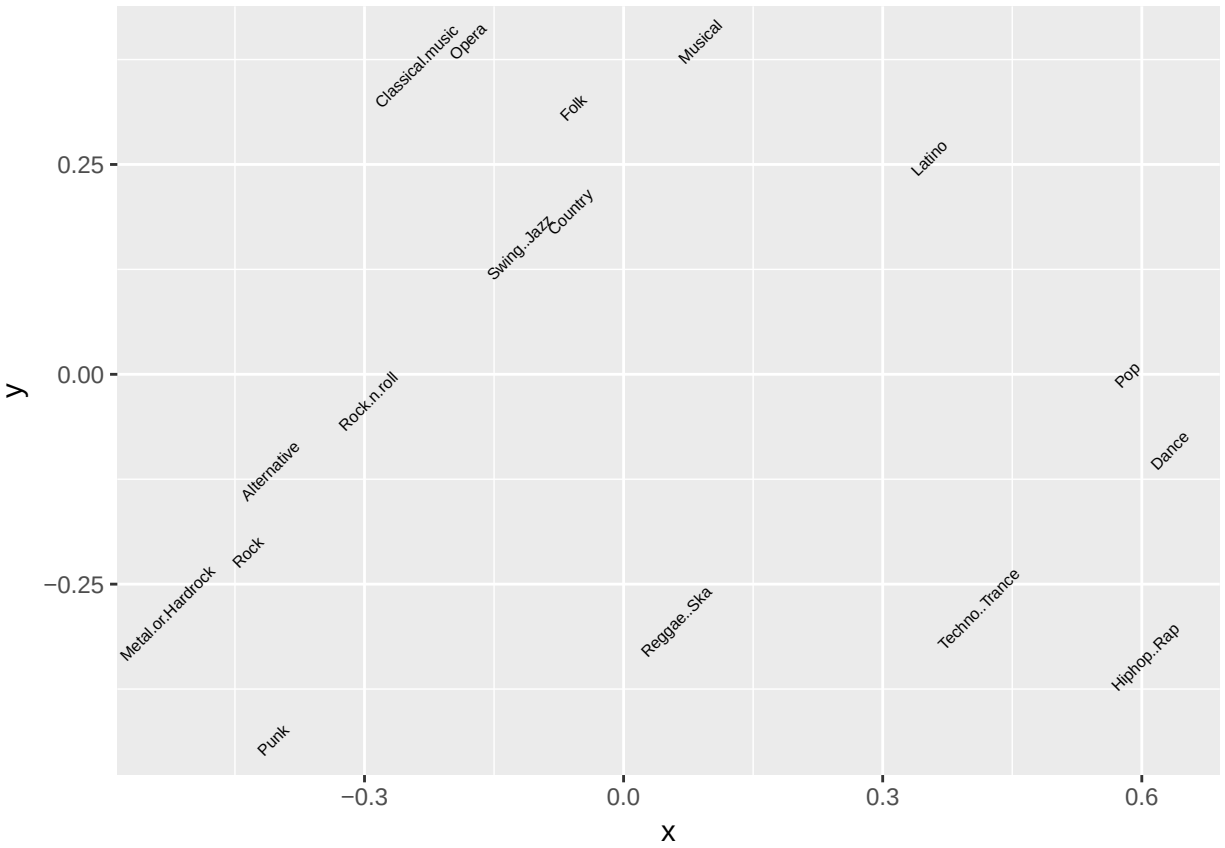
```
##   Dance Folk Country Classical.music Musical Pop Rock Metal.or.Hardrock
## 1     2    1       2               2       1   5    5                 1
## 2     2    1       1               1       2   3    5                 4
## 3     2    2       3               4       5   3    5                 3
## 4     4    3       2               4       3   5    3                 1
## 5     2    3       2               3       3   2    5                 5
## 6     5    3       1               2       2   5    3                 1
##   Punk Hiphop..Rap Reggae..Ska Swing..Jazz Rock.n.roll Alternative Latino
## 1    1           1           1           1           3           1      1
## 2    4           1           3           1           4           4      2
## 3    4           1           4           3           5           5      5
## 4    2           5           3           2           1           2      4
## 5    3           4           3           4           4           5      3
## 6    1           3           1           1           2           3      3
##   Techno..Trance Opera
## 1              1     1
## 2              1     1
## 3              1     3
```

```
## 4              2     2
## 5              1     3
## 6              5     2
```

```
#Correlation is a similarity measure between -1 and 1
#Get the negative of it as a distance measure and add 1 to make sure distances start from 0
yr_dist <- 1 - cor(yr_mds_data)
#Apply MDS
yr_mds <- cmdscale(yr_dist,k=2)
#Provide column names
colnames(yr_mds) <- c("x","y")
print(yr_mds)
```

```
##                           x            y
## Dance              0.63228003 -0.0905717504
## Folk              -0.05815591  0.3181027833
## Country           -0.06170712  0.1938031939
## Classical.music   -0.23970265  0.3672066570
## Musical            0.08869545  0.3964227778
## Pop                0.58317094 -0.0005306494
## Rock              -0.43508434 -0.2122796539
## Metal.or.Hardrock -0.52816298 -0.2846440216
## Punk              -0.40583563 -0.4357945663
## Hiphop..Rap        0.60401086 -0.3370311749
## Reggae..Ska        0.06095831 -0.2946989516
## Swing..Jazz       -0.12028127  0.1515518683
## Rock.n.roll       -0.29557875 -0.0318547555
## Alternative       -0.40931490 -0.1148703704
## Latino             0.35378213  0.2579434372
## Techno..Trance     0.41130757 -0.2796089505
## Opera             -0.18038173  0.3968541272
```

```
#Plot
ggplot(data.frame(yr_mds),aes(x=x,y=y)) + geom_text(label=rownames(yr_mds),angle=45,size=2)
```



Rock and related music are clustered on the left bottom, with Punk a bit distant. On the top classical music and opera goes together; swing-jazz, country, folk and musical genres are close. Pop, dance, techno and hip-hip/rap music are loosely together. Not bad, eh? Perhaps it is not the most accurate representation, but we can see that MDS captured considerable information in just two dimensions.

Properties

- Non-parametric (except you need to specify the number of components).
- Another dimension reduction technique.
- MDS is frequently used in psychology, sociology, archeology, biology, medicine, chemistry, network analysis and economics.

References

- http://carme2011.agrocampus-ouest.fr/slides/Groenen.pdf

K-Means

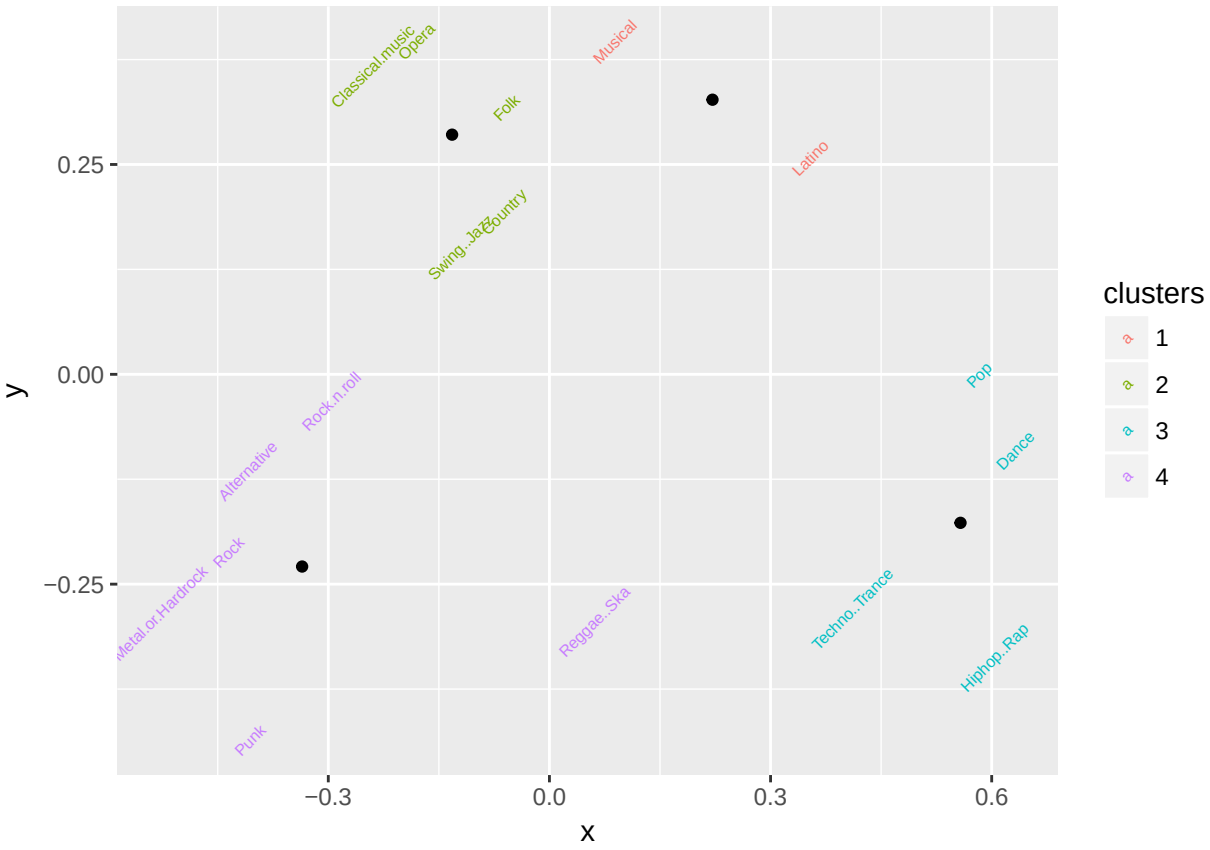
One of the most popular clustering algoritms is, without doubt, K-Means. In the classical form, given a number of (say, k) clusters, the algorithm adds nodes to the clusters to minimize the variance within cluster (i.e. squared distance from the cluster center to the nodes).

Let's apply K-Means to the Young People Survey MDS output.

```r
#Set the seed because K-Means algorithm uses a search based method
set.seed(58)
#Apply K-Means
genre_cluster<-kmeans(yr_mds,centers=4)
##Get the clusters
mds_clusters<-
data.frame(genre=names(genre_cluster$cluster),
cluster_mds=genre_cluster$cluster) %>% arrange(cluster_mds,genre)
mds_clusters
```

```
##                genre cluster_mds
## 1             Latino           1
## 2            Musical           1
## 3    Classical.music           2
## 4            Country           2
## 5               Folk           2
## 6              Opera           2
## 7        Swing..Jazz           2
## 8              Dance           3
## 9        Hiphop..Rap           3
## 10               Pop           3
## 11    Techno..Trance           3
## 12       Alternative           4
## 13 Metal.or.Hardrock           4
## 14              Punk           4
## 15       Reggae..Ska           4
## 16              Rock           4
## 17       Rock.n.roll           4
```

```r
#Plot the output
ggplot(
    data.frame(yr_mds) %>% mutate(clusters=as.factor(genre_cluster$cluster),
    genres=rownames(yr_mds)),aes(x=x,y=y)) +
    geom_text(aes(label=genres,color=clusters),angle=45,size=2) +
    geom_point(data=as.data.frame(genre_cluster$centers),aes(x=x,y=y)
)
```



As long as columns are numeric, you can easily apply K-Means to the data set. Interpretation of K-Means can change with the data. Let's also use the raw data.

```
#Set the seed because K-Means algorithm uses a search based method
set.seed(58)
#Apply K-Means to the raw data.
genre_cluster_raw<-kmeans(t(yr_mds_data),centers=4)

#Build the comparison data
compare_kmeans<-
left_join(mds_clusters,
data.frame(genre=names(genre_cluster_raw$cluster),
cluster_raw=genre_cluster_raw$cluster),by="genre")

print(compare_kmeans)
```

```
##                genre cluster_mds cluster_raw
## 1             Latino           1           1
## 2            Musical           1           2
## 3    Classical.music           2           2
## 4            Country           2           2
## 5               Folk           2           2
## 6              Opera           2           2
## 7        Swing..Jazz           2           2
## 8              Dance           3           1
## 9        Hiphop..Rap           3           1
## 10               Pop           3           1
## 11    Techno..Trance           3           3
```

```
## 12       Alternative           4           4
## 13 Metal.or.Hardrock           4           4
## 14              Punk           4           4
## 15       Reggae..Ska           4           1
## 16              Rock           4           4
## 17       Rock.n.roll           4           4
```

Now, let's get genre average scores and apply K-Means again.

```
#Set the seed because K-Means algorithm uses a search based method
set.seed(58)
#Prepare data (get mean scores) and apply K-Means
yr_means_data<-yr_mds_data %>% summarise_each(funs(mean)) %>% t()
```

```
## `summarise_each()` is deprecated.
## Use `summarise_all()`, `summarise_at()` or `summarise_if()` instead.
## To map `funs` over all variables, use `summarise_all()`
```

```
yr_means_data
```

```
##                       [,1]
## Dance             3.069971
## Folk              2.258017
## Country           2.112245
## Classical.music   2.981050
## Musical           2.759475
## Pop               3.440233
## Rock              3.787172
## Metal.or.Hardrock 2.355685
## Punk              2.451895
## Hiphop..Rap       2.889213
## Reggae..Ska       2.774052
## Swing..Jazz       2.758017
## Rock.n.roll       3.161808
## Alternative       2.887755
## Latino            2.806122
## Techno..Trance    2.298834
## Opera             2.153061
```

```
means_kmeans<-kmeans(yr_means_data,centers=4)
#Add to compare kmeans
compare_kmeans<-
left_join(compare_kmeans,
data.frame(genre=names(means_kmeans$cluster),cluster_mean=means_kmeans$cluster),by="genre")
print(compare_kmeans)
```

```
##                genre cluster_mds cluster_raw cluster_mean
## 1             Latino           1           1            4
## 2            Musical           1           2            4
## 3    Classical.music           2           2            3
## 4            Country           2           2            2
## 5               Folk           2           2            2
## 6              Opera           2           2            2
## 7        Swing..Jazz           2           2            4
## 8              Dance           3           1            3
## 9        Hiphop..Rap           3           1            4
## 10               Pop           3           1            1
```

```
## 11    Techno..Trance           3           3           2
## 12       Alternative          4           4           4
## 13 Metal.or.Hardrock          4           4           2
## 14              Punk          4           4           2
## 15       Reggae..Ska          4           1           4
## 16              Rock          4           4           1
## 17       Rock.n.roll          4           4           3
```

The latest K-Means iteration is not the best descriptive model. So, be mindful about how you prepare your data.

Properties

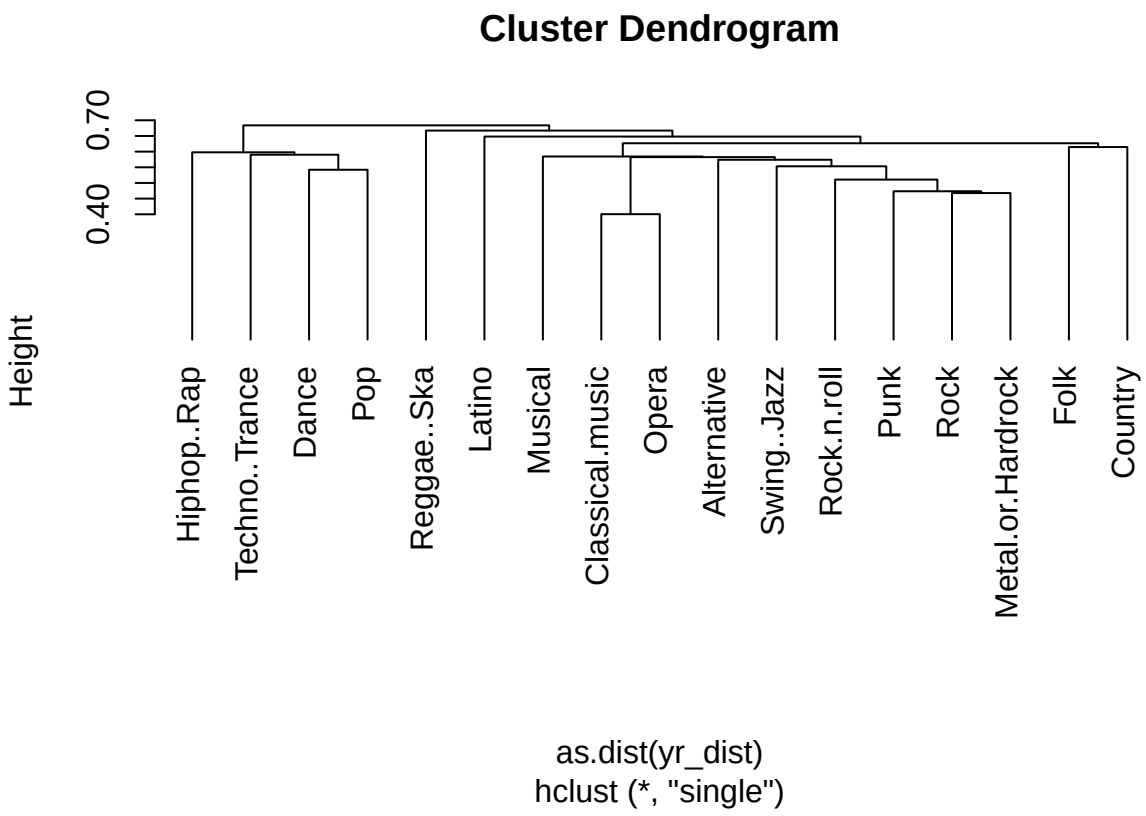
- K-Means is an unsupervised method. You don't need a response variable or pre-labeled data.
- K-Means requires the number of clusters as input. Total error is a non-increasing function of number of clusters.
- K-Means is not suitable for every application.
- As K-Means uses sum of squared errors, it is advised to scale the components. If component X1 ranges from 0 to 1 and component X2 ranges from -1000 to 1000, the algorithm will weigh considerably on X2.
- Differing density, non-globular shapes, differing sizes all affect K-Means performance.

Hierarchical Clustering

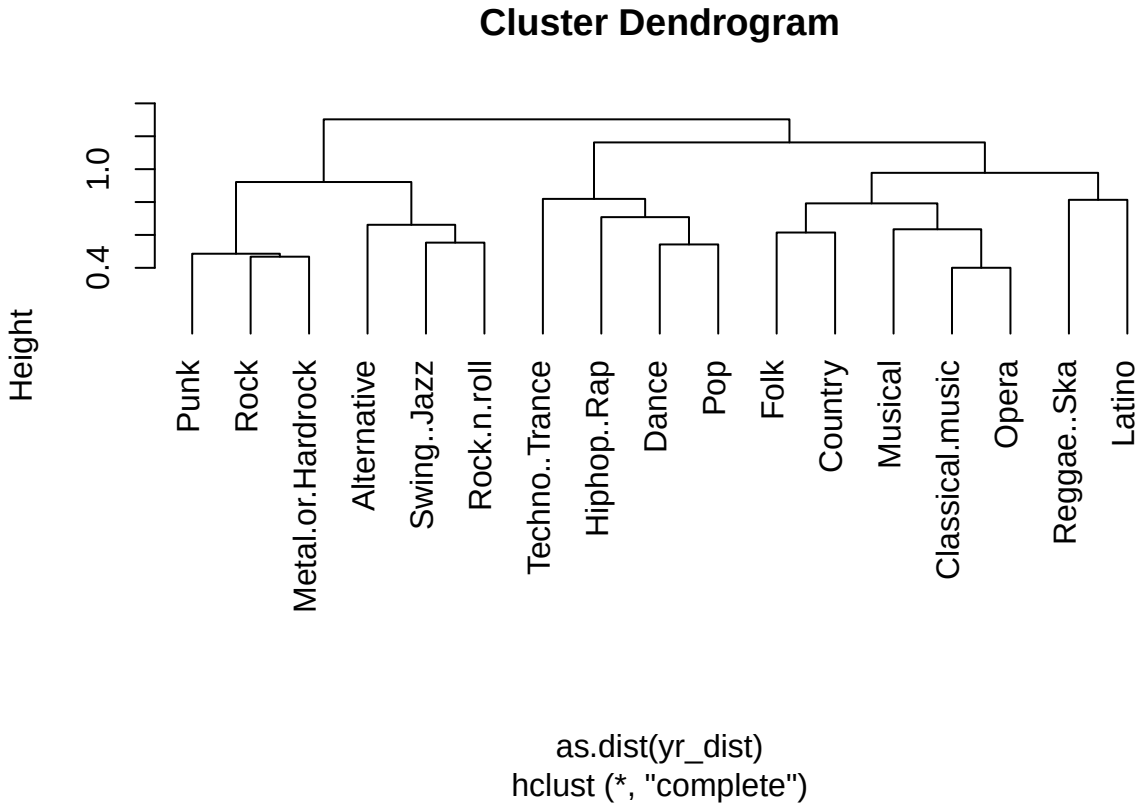
Algorithms like K-Means are categorized as Partitioning Clustering, which means each node can only reside in one cluster. In Hierarchical Clustering there are clusters within larger clusters, resulting in a tree like structure. There are also two types of hierarchical clustering methods, divisive ("top-down") or agglomerative ("bottom-up"). The former starts with all the nodes in a cluster and splits the clusters on the way down and the latter starts merging nodes up to a single cluster. Examples here belong to agglomerative clusters.

Let's use MDS output of Young People Survey for Hierarchical Clustering. Single link (or MIN) is known to minimize the closest dissimilarity between the nodes in pairwise clusters. Complete link (or MAX) is the opposite, it wants to minimize the largest dissimilarity. Average method aims to minimize the average distance between nodes in each cluster. Centroid only checks cluster centers. Ward method

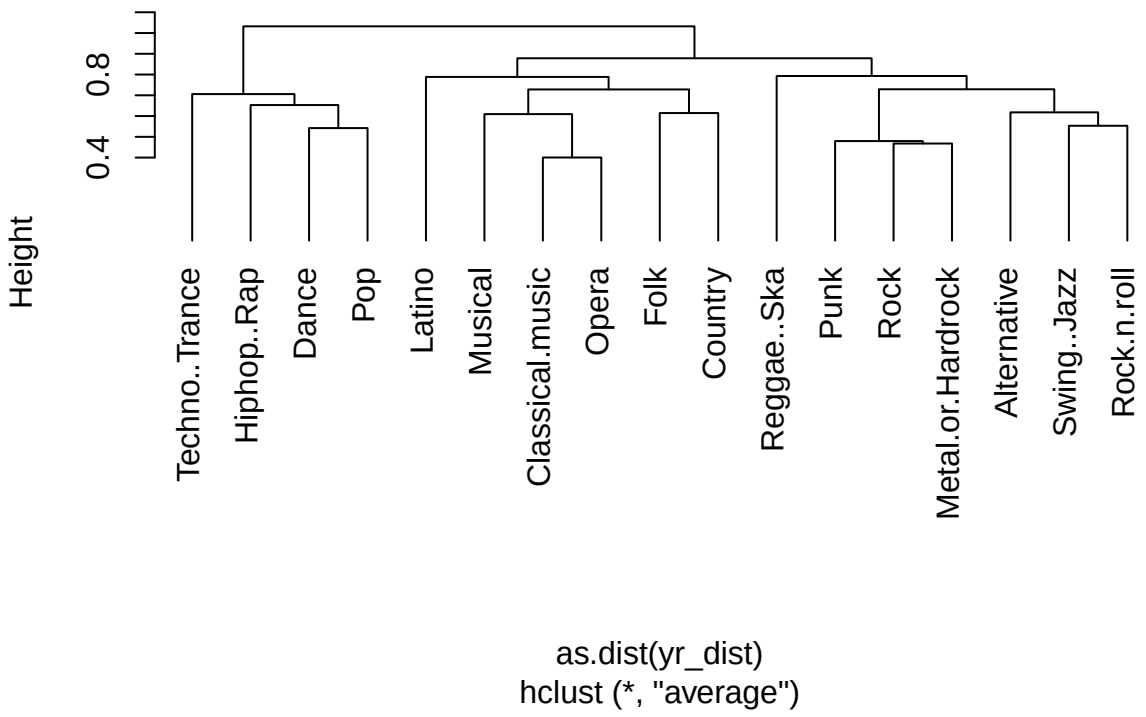
```
### Single link (MIN) method
yr_hc<-hclust(as.dist(yr_dist),method="single")
plot(yr_hc,hang=-1)
```



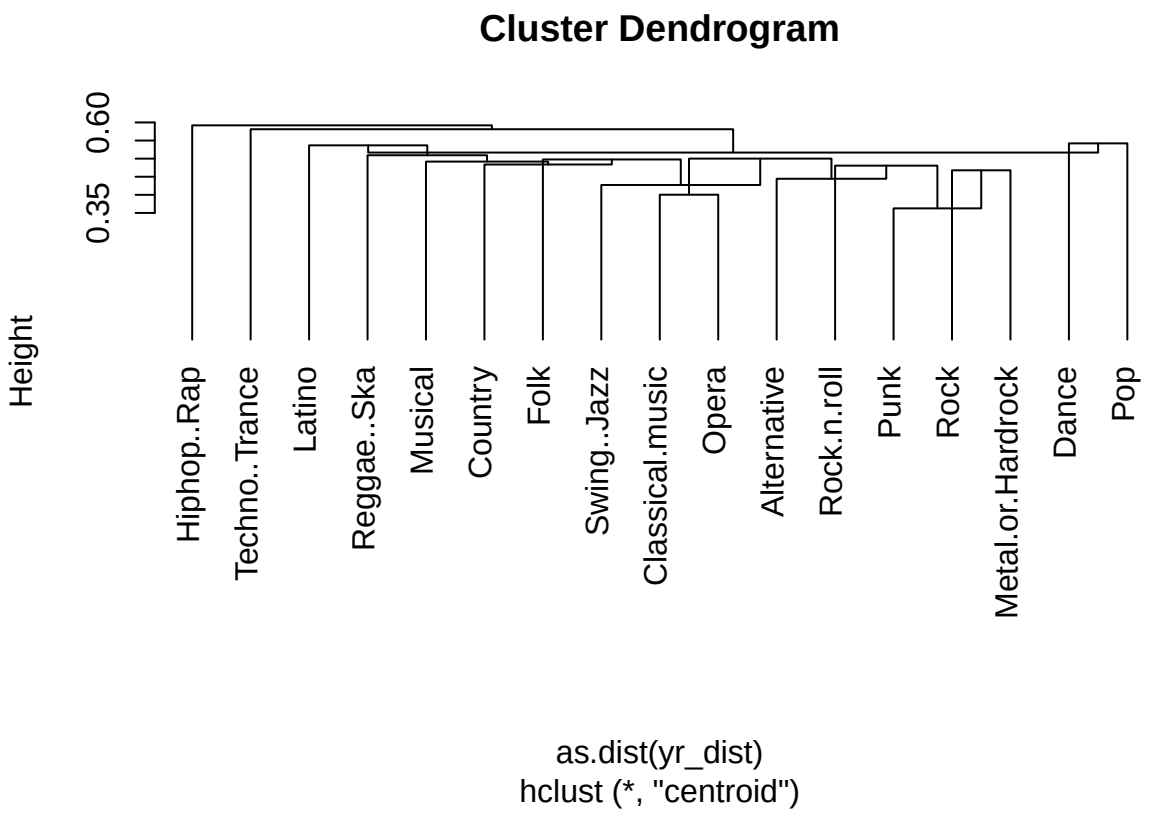

```
### Complete link (MAX) method
yr_hc<-hclust(as.dist(yr_dist),method="complete")
plot(yr_hc,hang=-1)
```



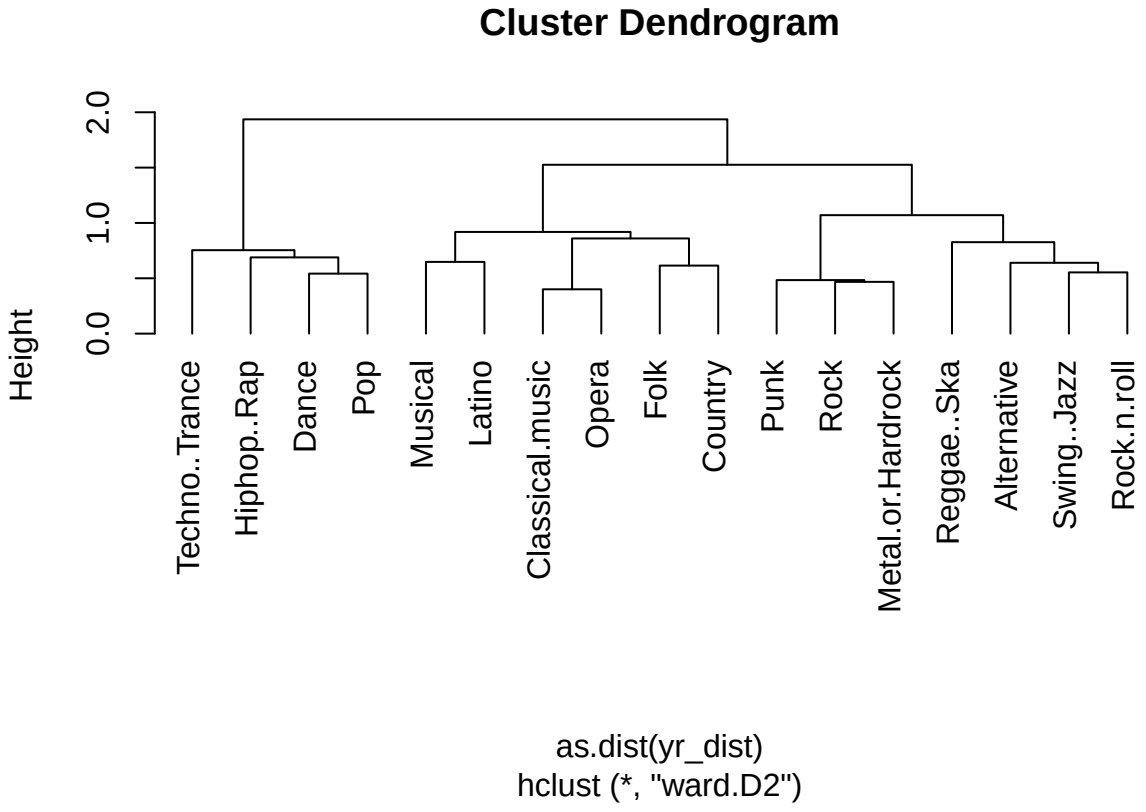

```
### Average method
yr_hc<-hclust(as.dist(yr_dist),method="average")
plot(yr_hc,hang=-1)
```




```
### Centroid method
yr_hc<-hclust(as.dist(yr_dist),method="centroid")
plot(yr_hc,hang=-1)
```




```
### Ward method
yr_hc<-hclust(as.dist(yr_dist),method="ward.D2")
plot(yr_hc,hang=-1)
```



Properties

- Hierarchical clustering not only provides insight about the proximity of nodes, but also the proximity of sub-groups.
- Depending on the agglomeration method, cluster performance might vary significantly. MIN can handle non eliptical shapes but it is sensitive to noise and outliers. MAX handles outliers/noise better but it has a tendency to break large clusters and it is biased towards globular shapes. Average is a midway between MIN and MAX. Ward's method is also better with noisy data and biased towards globular shape. It is also known to provide good starting centroids for K-Means.
- Once agglomeration is done, it cannot be changed.
- We don't have a comprehensive objective function to optimize.
- Might not be suitable if data is too noisy, sparse or asymmetrical in size.

References

- http://www3.nd.edu/~rjohns15/cse40647.sp14/www/content/lectures/13%20-%20Hierarchical%20Clustering.pdf

Supervised Learning (Regression and Classification)

Main distinction between supervised and unsupervised (e.g. clustering) learning is that supervised learning methods have a response variable consisting of values (regression) or labels (classification).

Linear Regression

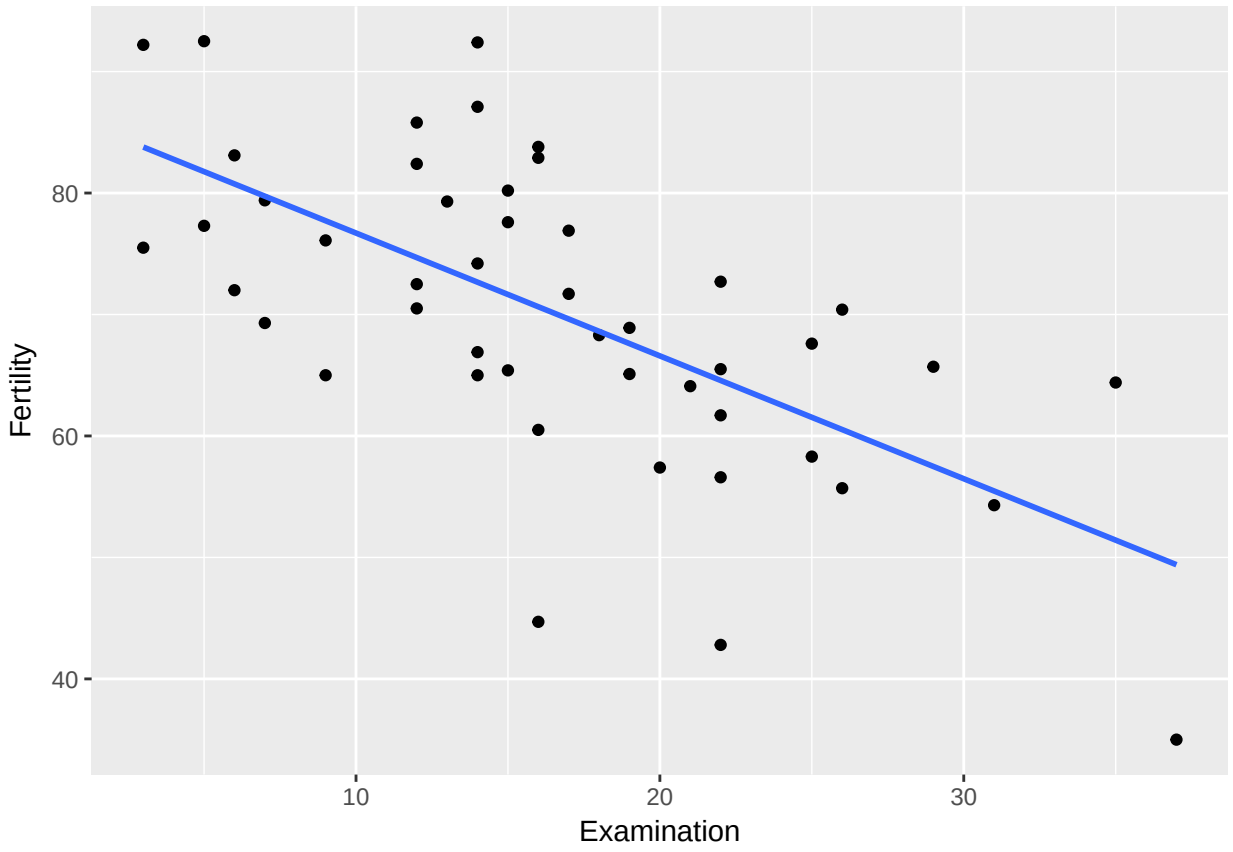
Linear regression is the most basic kind of regression. The structure is quite easy to comprehend ($Y = \beta_0 + \beta_1 * X_1 + \cdots + \beta_k * X_k$).

Following example uses `swiss` data (included in base R). Data consists of fertility measure and socio-economic indicators of 47 French-speaking provinces of Switzerland at 1888. These indicators are agriculture (percentage of males work in agriculture), examination (percentage of highest marks in army examination), education (percentage of people who received education beyond primary school), catholic (percentage of catholic people) and infant mortality (as percentage of babies who live less than 1 year).

```
#First check the data
head(swiss)
```

```
##              Fertility Agriculture Examination Education Catholic
## Courtelary        80.2        17.0          15        12     9.96
## Delemont          83.1        45.1           6         9    84.84
## Franches-Mnt      92.5        39.7           5         5    93.40
## Moutier           85.8        36.5          12         7    33.77
## Neuveville        76.9        43.5          17        15     5.16
## Porrentruy        76.1        35.3           9         7    90.57
##              Infant.Mortality
## Courtelary               22.2
## Delemont                 22.2
## Franches-Mnt             20.2
## Moutier                  20.3
## Neuveville               20.6
## Porrentruy               26.6
```

```
#Run a simple linear regression
ggplot(data=swiss) + geom_point(aes(x=Examination,y=Fertility)) +
 geom_smooth(method='lm',aes(x=Examination,y=Fertility),se=FALSE)
```




```
fert_vs_exam<-lm(Fertility ~ Examination, data=swiss)
summary(fert_vs_exam)
```

```
##
## Call:
## lm(formula = Fertility ~ Examination, data = swiss)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -25.9375  -6.0044  -0.3393   7.9239  19.7399
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  86.8185     3.2576  26.651  < 2e-16 ***
## Examination  -1.0113     0.1782  -5.675 9.45e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.642 on 45 degrees of freedom
## Multiple R-squared:  0.4172, Adjusted R-squared:  0.4042
## F-statistic: 32.21 on 1 and 45 DF,  p-value: 9.45e-07
```

```
#Now Fertility vs all
fert_vs_all<-lm(Fertility ~ ., data=swiss)
summary(fert_vs_all)
```

```
##
## Call:
```

```
## lm(formula = Fertility ~ ., data = swiss)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.2743  -5.2617   0.5032   4.1198  15.3213
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      66.91518   10.70604   6.250 1.91e-07 ***
## Agriculture      -0.17211    0.07030  -2.448  0.01873 *
## Examination      -0.25801    0.25388  -1.016  0.31546
## Education        -0.87094    0.18303  -4.758 2.43e-05 ***
## Catholic          0.10412    0.03526   2.953  0.00519 **
## Infant.Mortality  1.07705    0.38172   2.822  0.00734 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.165 on 41 degrees of freedom
## Multiple R-squared:  0.7067, Adjusted R-squared:  0.671
## F-statistic: 19.76 on 5 and 41 DF,  p-value: 5.594e-10
```

Fertility can seemingly be explained by those covariates. But what about classification problems? Linear regression is not very well suited in categorical response variables.

Logistic regression

Logistic regression is commonly used in binary response variables such as high/low, accept/reject, yes/no type situations. In the following example we will build a credit scoring model which will "tell" us if a demanded loan is good or bad. We are going to use the German Credit data (download from here). It consists of credit applications and classified as accepted/rejected. The other 20 columns are explanatory variables such as account balance, occupation, gender and marital status, savings, purpose of the credit, payment status. For our example we will use only three covariates (account balance, occupation sex/marital status).

Covariates are score variables (can also be considered as categorical). **Account.Balance** (1: no running account, 2: no balance or debit, 3: 0 <= ... < 200 DM, 4: ... >= 200 DM or checking account for at least 1 year) and **Occupation** (1: unemployed / unskilled with no permanent residence, 2: unskilled with permanent residence, 3: skilled worker / skilled employee / minor civil servant, 4: executive / self-employed / higher civil servant).

```
set.seed(58)
##Get the data
credit_data<-read.csv("data/german_credit.csv") %>%
select(Creditability, Account.Balance, Occupation) %>%
tbl_df()

print(head(credit_data))
```

```
## # A tibble: 6 x 3
##   Creditability Account.Balance Occupation
##           <int>           <int>      <int>
## 1             1               1          3
## 2             1               1          3
## 3             1               2          2
## 4             1               1          2
## 5             1               1          2
```

```
## 6             1               1          2
```

```
##Use 50% as training set
train_sample<-sample(1:nrow(credit_data),round(nrow(credit_data)/2),replace=FALSE)
credit_train <- credit_data %>% slice(train_sample)
credit_test <- credit_data %>% slice(-train_sample)

#Assume covariates are scores
credit_model<-glm(Creditability ~ Account.Balance + Occupation,
    family=binomial(link="logit"),data=credit_train)
#See the summary of the model
summary(credit_model)
```

```
##
## Call:
## glm(formula = Creditability ~ Account.Balance + Occupation, family = binomial(link = "logit"),
##     data = credit_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2588  -1.1574   0.4764   0.9073   1.3505
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)       0.2832     0.5214   0.543    0.587
## Account.Balance   0.7221     0.0946   7.633  2.3e-14 ***
## Occupation       -0.3509     0.1655  -2.120    0.034 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 591.05  on 499  degrees of freedom
## Residual deviance: 519.66  on 497  degrees of freedom
## AIC: 525.66
##
## Number of Fisher Scoring iterations: 4
```

```
#Check the predictions for both in sample and out of sample data
credit_model_in_sample<-
data.frame(actual=credit_train$Creditability,
    fitted=predict(credit_model,type="response"))
credit_model_in_sample %>%
    group_by(actual) %>%
    summarise(mean_pred=mean(fitted),sd_pred=sd(fitted))
```

```
## # A tibble: 2 x 3
##   actual mean_pred   sd_pred
##    <int>     <dbl>     <dbl>
## 1      0 0.6244155 0.1450404
## 2      1 0.7595741 0.1592563
```

```
credit_model_out_of_sample<-
data.frame(actual=credit_test$Creditability,
    fitted=predict(credit_model,newdata=credit_test,type="response"))
credit_model_out_of_sample %>%
```

```
    group_by(actual) %>%
    summarise(mean_pred=mean(fitted),sd_pred=sd(fitted))
```

```
## # A tibble: 2 x 3
##   actual mean_pred   sd_pred
##    <int>     <dbl>     <dbl>
## 1      0 0.6344927 0.1533762
## 2      1 0.7555894 0.1638322
```

```
#Model as covariates are assumed as ordered factors
credit_model_fctr<-
    glm(Creditability ~ ordered(Account.Balance,levels=1:4) +
    ordered(Occupation,levels=1:4),
     family=binomial,data=credit_train)
#See the summary of the model
summary(credit_model_fctr)
```

```
##
## Call:
## glm(formula = Creditability ~ ordered(Account.Balance, levels = 1:4) +
##     ordered(Occupation, levels = 1:4), family = binomial, data = credit_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4421  -1.1930   0.4408   0.9038   1.2520
##
## Coefficients:
##                                       Estimate Std. Error z value
## (Intercept)                             0.9975     0.2236   4.461
## ordered(Account.Balance, levels = 1:4).L   1.6032     0.2276   7.045
## ordered(Account.Balance, levels = 1:4).Q   0.5324     0.2660   2.001
## ordered(Account.Balance, levels = 1:4).C   0.2121     0.3011   0.704
## ordered(Occupation, levels = 1:4).L       -0.2720     0.5437  -0.500
## ordered(Occupation, levels = 1:4).Q       -0.4392     0.4271  -1.028
## ordered(Occupation, levels = 1:4).C        0.3921     0.2680   1.463
##                                       Pr(>|z|)
## (Intercept)                           8.17e-06 ***
## ordered(Account.Balance, levels = 1:4).L 1.85e-12 ***
## ordered(Account.Balance, levels = 1:4).Q   0.0454 *
## ordered(Account.Balance, levels = 1:4).C   0.4812
## ordered(Occupation, levels = 1:4).L        0.6169
## ordered(Occupation, levels = 1:4).Q        0.3037
## ordered(Occupation, levels = 1:4).C        0.1434
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 591.05  on 499  degrees of freedom
## Residual deviance: 513.16  on 493  degrees of freedom
## AIC: 527.16
##
## Number of Fisher Scoring iterations: 5
```

```
#Check the predictions for both in sample and out of sample data
credit_model_in_sample<-
    data.frame(actual=credit_train$Creditability,
        fitted=predict(credit_model_fctr,type="response"))
credit_model_in_sample %>% group_by(actual) %>%
    summarise(mean_pred=mean(fitted),sd_pred=sd(fitted))
```

```
## # A tibble: 2 x 3
##   actual mean_pred   sd_pred
##    <int>     <dbl>     <dbl>
## 1      0 0.6173277 0.1368243
## 2      1 0.7623032 0.1648146
```

```
credit_model_out_of_sample<-
    data.frame(actual=credit_test$Creditability,
        fitted=predict(credit_model_fctr,newdata=credit_test,type="response"))

credit_model_out_of_sample %>%
    group_by(actual) %>%
    summarise(mean_pred=mean(fitted),sd_pred=sd(fitted))
```

```
## # A tibble: 2 x 3
##   actual mean_pred   sd_pred
##    <int>     <dbl>     <dbl>
## 1      0 0.6365251 0.1494191
## 2      1 0.7528284 0.1697649
```

K Nearest Neighbors (KNN)

The name explains everything. For each node, the algorithm checks the k nearest nodes (by euclidean distance of covariates). Based on their "votes", the class of the node is estimated. If there is a tie,

```
##We are going to use knn function of the class package
## If the class package is not loaded use install.packages("class")
##Below example is from iris data set
## train is the covariates of the training set
train <- rbind(iris3[1:25,,1], iris3[1:25,,2], iris3[1:25,,3])
## cl is the class of the training set
## First 25 nodes are (s)etosa, second 25 are versi(c)olor, third 25 are (v)irginica
cl <- factor(c(rep("s",25), rep("c",25), rep("v",25)))
## test is the test set we want to predict its classes
test <- rbind(iris3[26:50,,1], iris3[26:50,,2], iris3[26:50,,3])
##The following function uses the train set and classes to predict
#the class of the test set's classes
knn_cl<-class::knn(train, test, cl, k = 3, prob=TRUE)
knn_cl
```

```
##  [1] s s s s s s s s s s s s s s s s s s s s s s s s s c c v c c c c c v c
## [36] c c c c c c c c c c c c c c c v c c v v v v v v v v v v c v v v v v v
## [71] v v v v v
## attr(,"prob")
##  [1] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [8] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [15] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [22] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.6666667
```

```
## [29] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.6666667 1.0000000
## [36] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [43] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [50] 1.0000000 1.0000000 0.6666667 0.7500000 1.0000000 1.0000000 1.0000000
## [57] 1.0000000 1.0000000 0.5000000 1.0000000 1.0000000 1.0000000 1.0000000
## [64] 0.6666667 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [71] 1.0000000 0.6666667 1.0000000 1.0000000 0.6666667
## Levels: c s v
```

```
##Let's also compare knn estimates vs actual test data classes (same order as train data)
table(data.frame(actual=cl,estimate=knn_cl))
```

```
##       estimate
## actual  c  s  v
##      c 23  0  2
##      s  0 25  0
##      v  3  0 22
```

References

- These lecture notes were initially used at Bogazici University Engineering Management Master Program.
- https://onlinecourses.science.psu.edu/stat857/node/215