Machine Learning A-Z

Obrusník Vít

CTU - Faculty of electrical engineering

Abstract. This document summarizes Machine Learning A-Z course available at Udemy. `https://www.udemy.com/machinelearning/learn/v4/overview`. Authors are Kirill Eremenko and Hadelin de Ponteves.
All concepts are shown in Python and R. Course provides the students with code templates which are included in this document.
Every section has a dataset to train and test the model.

Table of Contents

1 Preprocessing

First of all we need to prepare our dataset. We can encounter several problems. Then we want to split the dataset.

1.1 Problems with datasets

1. Missing data
   There might be some columns missing in the dataset. In that case our models won't work. We can either delete the incomplete rows or we can replace the missing data with average values.
2. Categorical data
   Some parameters (e.g. City['Madrid', 'Barcelano', 'Sevilla']) need to be replaced with integer values (e.g. City[1, 2, 3]). We need to focus on not to fall into **Dummy Variable Trap**. More information: `http://www.algosome.com/articles/dummy-variable-trap-regression.html`
3. Splitting the dataset into training set and test set
   We usually train the model on 80% of the dataset and test it on 20%.
4. Feature scaling
   We might want to scale the data to have normalized magnitudes. This is not always necessary and depends on the method/library. All the methods that use Euclidian distance are dependent on Feature scaled dataset.

1.2 Code template in Python

Note that some parts of the code might be omitted.

```python
# Data Preprocessing Template

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values

# Taking care of missing data
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values = 'NaN', strategy = 'mean', axis = 0)
imputer = imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])

# Encoding categorical data
# Encoding the Independent Variable
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
onehotencoder = OneHotEncoder(categorical_features = [0])
X = onehotencoder.fit_transform(X).toarray()
# Encoding the Dependent Variable
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)

# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)
```

1.3 Code template in R

Note that some parts of the code might be omitted.

```r
# Data Preprocessing Template

# Importing the dataset
dataset = read.csv('Data.csv')

# Taking care of missing data
dataset$Age = ifelse(is.na(dataset$Age),
                     ave(dataset$Age, FUN = function(x) mean(x, na.rm = TRUE)),
                     dataset$Age)
dataset$Salary = ifelse(is.na(dataset$Salary),
                        ave(dataset$Salary, FUN = function(x) mean(x, na.rm = TRUE)),
                        dataset$Salary)

# Encoding categorical data
dataset$Country = factor(dataset$Country,
                         levels = c('France', 'Spain', 'Germany'),
                         labels = c(1, 2, 3))
dataset$Purchased = factor(dataset$Purchased,
                           levels = c('No', 'Yes'),
                           labels = c(0, 1))

# Splitting the dataset into the Training set and Test set
# install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$DependentVariable, SplitRatio = 0.8)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)

# Feature Scaling
training_set = scale(training_set)
test_set = scale(test_set)
```

2 Regression

Regression is a statistical method of predicting the value of **dependent variable (DV)** based on values of **independent variable(s) (IVs)**. More information here `https://en.wikipedia.org/wiki/Regression_analysis`.

2.1 Regression code template in Python

```python
# Regression

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)

# Feature Scaling
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""

# Fitting  Regression to the Training set
# Creating Regressor HERE

# Predicting the Test set results
y_pred = regressor.predict(X_test)

# Visualising the Training set results
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('(Training set)')
plt.xlabel('IV')
plt.ylabel('DV')
plt.show()

# Visualising the Test set results
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('(Test set)')
plt.xlabel('IV')
plt.ylabel('DV')
plt.show()
```

2.2 Regression code template in R

```r
# Simple Linear Regression

# Importing the dataset
dataset = read.csv('Data.csv')

# Splitting the dataset into the Training set and Test set
# install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$Salary, SplitRatio = 2/3)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)

# Feature Scaling
# training_set = scale(training_set)
# test_set = scale(test_set)

# Fitting Simple Linear Regression to the Training set
# Creating Regressor HERE

# Predicting the Test set results
y_pred = predict(regressor, newdata = test_set)

# Visualising the Training set results
library(ggplot2)
ggplot() +
  geom_point(aes(x = training_set$YearsExperience, y = training_set$Salary),
             colour = 'red') +
  geom_line(aes(x = training_set$YearsExperience, y = predict(regressor, newdata = training_set)),
            colour = 'blue') +
  ggtitle('(Training set)') +
  xlab('IV') +
  ylab('DV')

# Visualising the Test set results
library(ggplot2)
ggplot() +
  geom_point(aes(x = test_set$YearsExperience, y = test_set$Salary),
             colour = 'red') +
  geom_line(aes(x = training_set$YearsExperience, y = predict(regressor, newdata = training_set)),
            colour = 'blue') +
  ggtitle('(Test set)') +
  xlab('IV') +
  ylab('DV')
```

2.3 Simple Linear Regression

Problem: Our dataset contains years of experience and salary. We want to fit the model to recommend the salary based on years of experience.

YearsExperience	Salary
1.1	39343.00
1.3	46205.00
1.5	37731.00
2.0	43525.00
2.2	39891.00
2.9	56642.00
3.0	60150.00
3.2	54445.00
3.2	64445.00
3.7	57189.00
3.9	63218.00
4.0	55794.00
4.0	56957.00
4.1	57081.00
4.5	61111.00
4.9	67938.00
5.1	66029.00
5.3	83088.00
5.9	81363.00
6.0	93940.00
6.8	91738.00
7.1	98273.00
7.9	101302.00
8.2	113812.00
8.7	109431.00
9.0	105582.00
9.5	116969.00
9.6	112635.00
10.3	122391.00
10.5	121872.00

Simple Linear Regression is the easiest model following the equation

$$y = b_1 * x + b_0 \tag{1}$$

where y is the dependent variable which we are interested in (salary) and x is the independent variable (years of experience).

In order to use Linear Regression we simply replace comment from the template with the following code in Python

```python
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

And in R we use the following code to fit the Simple Linear Model

```
# Fitting Simple Linear Regression to the Training set
regressor = lm(formula = Salary ~ YearsExperience,
               data = training_set)
```

Results are following.

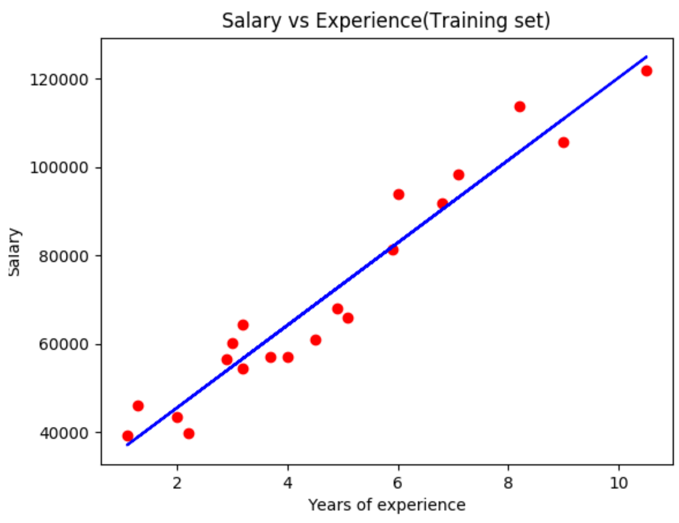


Fig. 1. Simple Linear regression in Python

2.4 Multiple Linear Regression

Problem: We want to predict how profitable will the company be when we know its spending in R&D, Advertisement and Administration. We have a dataset containing 50 startups, their country and their spending to train and test the model. This is the sample of the dataset:

Multiple Linear Regression follows this equation

$$y = b_1 * x_1 + b_2 * x_2 + \cdots + b_n * x_n + b_0 \qquad (2)$$

where y is the dependent variable which we are interested in (profit) and x_i is the independent variable (spending).

Assumptions of Linear Regression

There are several assumptions of Linear Regression:

- Linearity

- Homoscedasticity
  In statistics, a sequence or a vector of random variables is homoscedastic if all random variables in the sequence or vector have the same finite variance.
- Multivariable normality
  More information: `https://en.wikipedia.org/wiki/Multivariate_normal_distribution`
- Independence of errors
  Error values are statistically independent.
- Lack of multicollinearity
  Multicollinearity in regression occurs when predictor variables (independent variables) in the regression model are more highly correlated with other predictor variables than with the dependent variable. More information: `http://www.researchconsultation.com/multicollinearity-multiple-regression.asp`

5 methods of building models

There are some methods to create predicting model. I am listing a few of them.

1. All-in (fit the model with all the independent variable)
2. Backward Elimination
3. Forward Selection
4. Bidirectional Elimination
5. Score comparison (fit the models with all possible combinations of independent variables and compare them)

Backward elimination

While building a model we might want to get rid of some IVs. Some IVs usually don't have significant impact on the results. Backward elimination is an algorithm that can help us choose only significant variables.

1. Select a significance level to stay in the model (e.g. SL = 0.05)
2. Fit the full model with all possible predictors
3. Consider the predictor with the highest P-value. If $P > SL$, go to step 4, otherwise **Model is ready**
4. Remove the predictor
5. Fit the model without this variable

Forward Selection

1. Select a significance level to stay in the model (e.g. SL = 0.05)
2. Fil all simple regression models $y\ x_i$ Select the one with the lowest P-value
3. Keep this variable and fit all possible models with one extra predictor added to the one(s) you already have
4. Consider the predictor with the lowest P-value. If $P < SL$ go to step 3. Otherwise **Your model is ready**

Bidirectional elimination

1. Select a significance level to enter and to stay in the model (e.g. SLEnter = 0.05, SLStay = 0.05)
2. Perform the next step of Forward Selection (new variables must have P < SLEnter to enter)
3. Perform ALL steps of Backward Elimination (old variables must have P < SLStay to stay).
4. No new variables can enter and no old variables can exit. **Your model is ready**

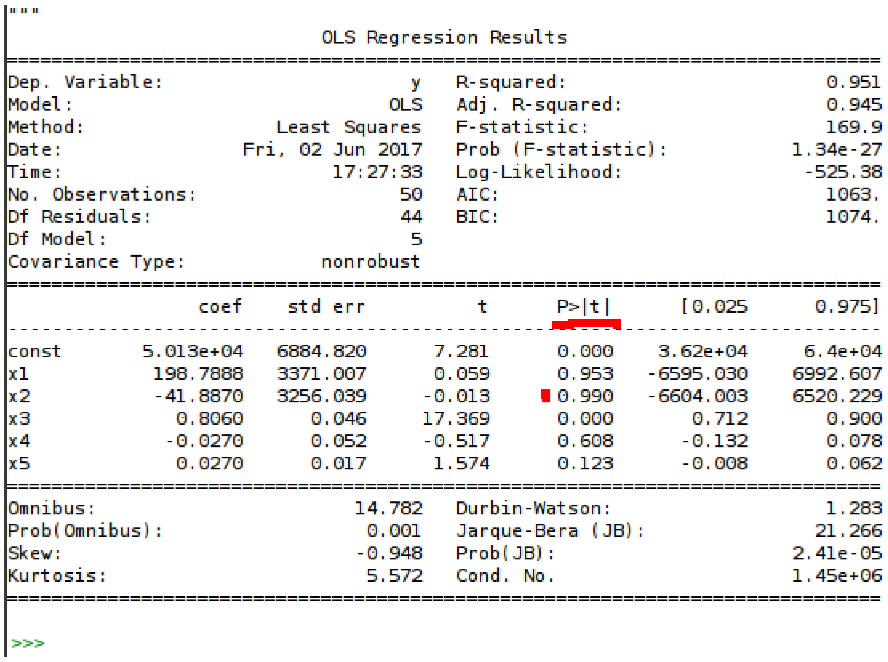
Code

The following code implements Multiple Linear Regression in Python.

```python
# Fitting Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

The following code implements Backward elimination in Python. The method called `summary()` is used to display all the P-values of Independent Variables. Then we choose the one with the biggest P-value, remove it and fit the modle again.

```python
# Building the optimal model using Backward Elimination
import statsmodels.formula.api as sm
X = np.append(arr = np.ones((50, 1)).astype(int), values = X, axis = 1)
X_opt = X[:, [0, 1, 2, 3, 4, 5]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
X_opt = X[:, [0, 1, 3, 4, 5]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
X_opt = X[:, [0, 3, 4, 5]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
X_opt = X[:, [0, 3, 5]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
X_opt = X[:, [0, 3]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
```



```
"""
                              OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.951
Model:                            OLS   Adj. R-squared:                  0.945
Method:                 Least Squares   F-statistic:                     169.9
Date:                Fri, 02 Jun 2017   Prob (F-statistic):           1.34e-27
Time:                        17:27:33   Log-Likelihood:                -525.38
No. Observations:                  50   AIC:                             1063.
Df Residuals:                      44   BIC:                             1074.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       5.013e+04   6884.820      7.281      0.000    3.62e+04     6.4e+04
x1           198.7888   3371.007      0.059      0.953   -6595.030    6992.607
x2           -41.8870   3256.039     -0.013      0.990   -6604.003    6520.229
x3             0.8060      0.046     17.369      0.000       0.712       0.900
x4            -0.0270      0.052     -0.517      0.608      -0.132       0.078
x5             0.0270      0.017      1.574      0.123      -0.008       0.062
==============================================================================
Omnibus:                       14.782   Durbin-Watson:                   1.283
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               21.266
Skew:                          -0.948   Prob(JB):                     2.41e-05
Kurtosis:                       5.572   Cond. No.                     1.45e+06
==============================================================================

>>>
```

Fig. 2. Calling summary function in Python. We choose the variable with biggest P-value and remove it.

The following code implements Multiple Linear Regression in R.

```r
# Fitting Multiple Linear Regression to the Training set
regressor = lm(formula = Profit ~ .,
               data = training_set)
```

The following code implements Backward elimination in R.

```r
# Building the optimal model using Backward Elimination
regressor = lm(formula = Profit ~ R.D.Spend + Administration + Marketing.Spend + State,
               data = dataset)
summary(regressor)
# Optional Step: Remove State2 only (as opposed to removing State directly)
# regressor = lm(formula = Profit ~ R.D.Spend + Administration + Marketing.Spend + factor(State, exclude = 2),
#                data = dataset)
# summary(regressor)
regressor = lm(formula = Profit ~ R.D.Spend + Administration + Marketing.Spend,
               data = dataset)
summary(regressor)
regressor = lm(formula = Profit ~ R.D.Spend + Marketing.Spend,
               data = dataset)
summary(regressor)
regressor = lm(formula = Profit ~ R.D.Spend,
               data = dataset)
summary(regressor)
```

2.5 Polynomial Regression

Problem: We have a set of job positions and their according salaries. We want to train the model to propose the best salary for new empleyees.

Polynomial regression is the first **nonlinear** model.

Position	Level	Salary
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

Polynomial Regression follows this equation

$$y = b_1 * x_1^1 + b_2 * x_1^2 + \cdots + b_n * x_n^n + b_0 \tag{3}$$

The following code implements the Polynomial regressor in Python

```
# Fitting Polynomial Regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 5)
X_poly = poly_reg.fit_transform(X)
poly_reg.fit(X_poly, y)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
```

The following code implements the Polynomial regressor in R

```
# Fitting Polynomial Regression to the dataset
dataset$Level2 = dataset$Level^2
dataset$Level3 = dataset$Level^3
dataset$Level4 = dataset$Level^4
poly_reg = lm(formula = Salary ~ .,
              data = dataset)
```

The following picture shows visualisation of train model and real data (dots). Model is fitted by polynomial of 5th degree.

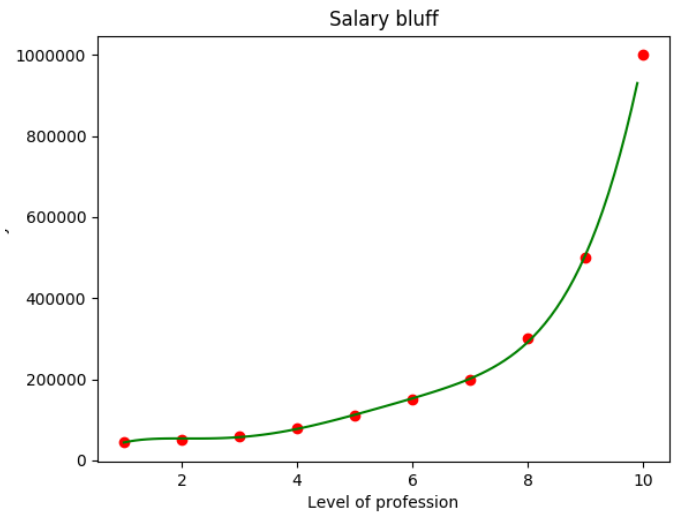


Fig. 3. Polynomialregression in Python

2.6 Support Vector Regression (SVR)

Problem: The same as in Polynomial regression.

SVR is the second **nonlinear** model. SVR model from `sklearn` Python library requires the dataset to be scaled in preprocessing.

The following code implements Support Vector Regressor in Python

```python
# Fitting SVR to the dataset
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X, y)
```

Kernel is the method used by the predictor. *RBF* means *Radial Basis function kernel*. More information here: `https://en.wikipedia.org/wiki/Kernel_method`

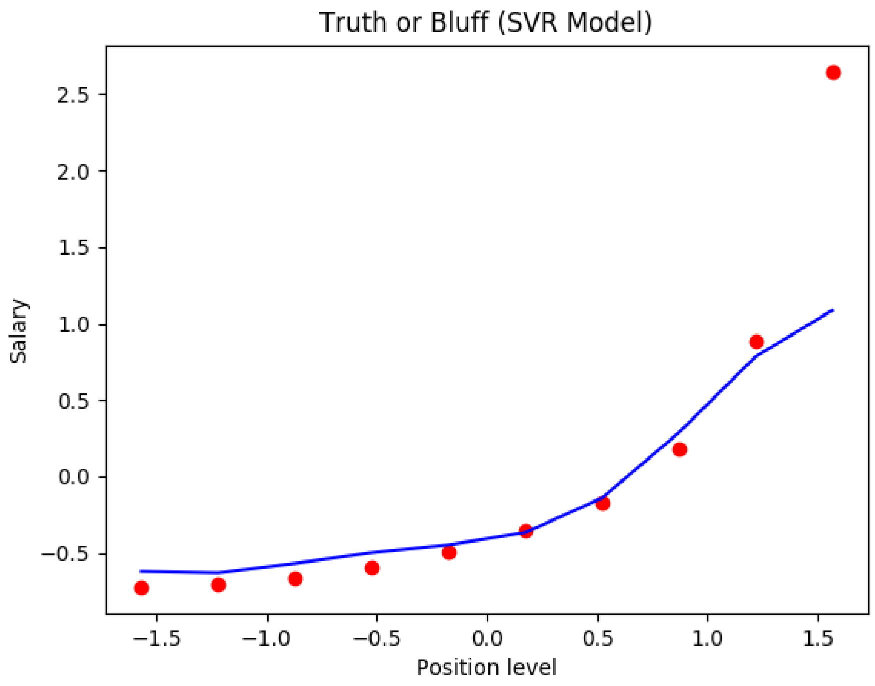


Fig. 4. Support Vector regression in Python

2.7 Decision Tree Regression

Problem: The same as in Polynomial regression.

The following code implements Decision Tree Regressor in Python

```
# Fitting Decision Tree Regression to the dataset
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X, y)
```

The following code implements Decision Tree Regressor in R

```
# Fitting Decision Tree Regression to the dataset
# install.packages('rpart')
library(rpart)
```

```
regressor = rpart(formula = Salary ~ .,
                  data = dataset,
                  control = rpart.control(minsplit = 1))
```

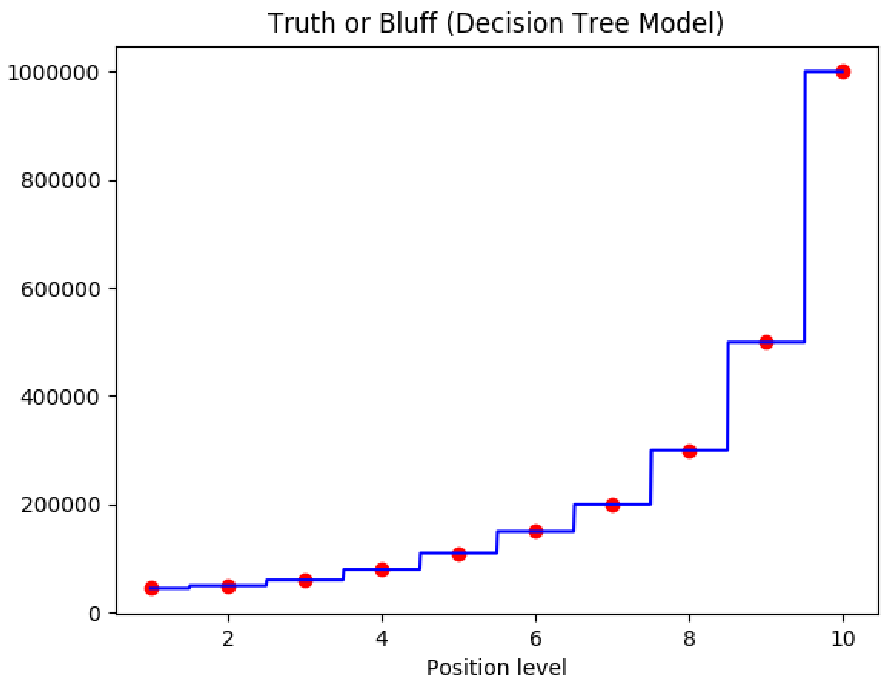


Fig. 5. Decision Tree regression in Python

2.8 Random Forest Regression

Problem: The same as in Polynomial regression.

Random Forest algorithm uses multiple Decision Trees and then average the results. Number of Decision Trees determines the number of splits and the value level of prediction.

The following code implements Random Forest Regressor in Python

```
# Fitting Random Forest Regression to the dataset
from sklearn.ensemble import RandomForestRegressor
```

```
regressor = RandomForestRegressor(n_estimators = 10, random_state
regressor.fit(X, y)
```

The following code implements Random Forest Regressor in R

```
# Fitting Random Forest Regression to the dataset
# install.packages('randomForest')
library(randomForest)
set.seed(1234)
regressor = randomForest(x = dataset[-2],
                         y = dataset$Salary,
                         ntree = 500)
```

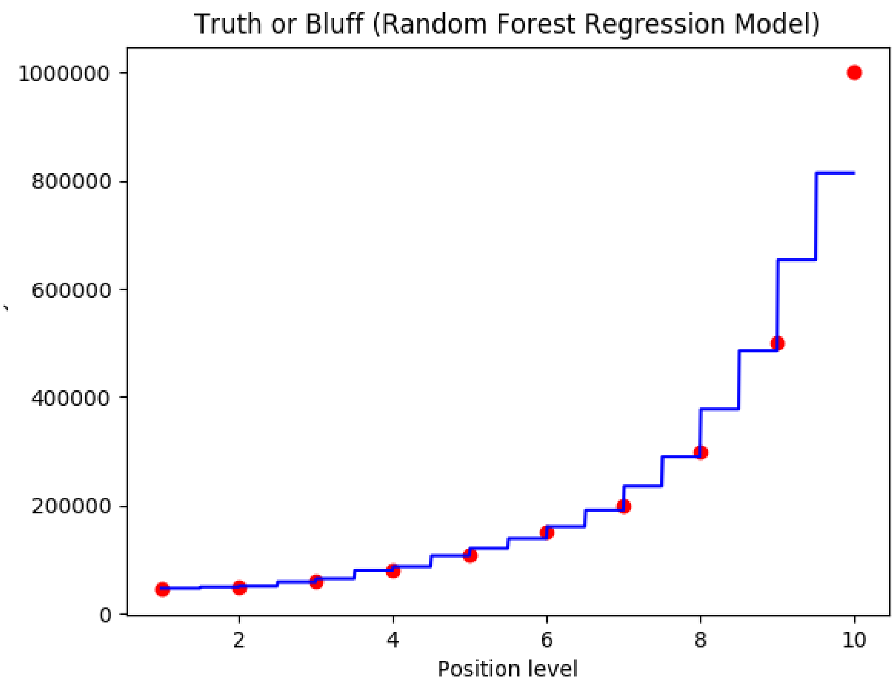


Fig. 6. Random Forest regression in Python

2.9 Evaluating model performance

R-squared

R^2 - The Goodness of fit

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{4}$$

$$SS_{res} = \sum_{0}^{n} (y_{i_{actual}} - y_{i_{predicted}})^2$$

$$SS_{tot} = \sum_{0}^{n} (y_{i_{actual}} - y_{average})^2$$

SS_{res} means sum of squares residual and SS_{tot} means sum of squares total, n is the number of observations.

The closer R^2 is to 1, the better.

The biggest disadvantage is that R^2 never decrease if we introduce new independent variable to the model. That's why we use Adjusted R^2.

Adjusted R-squared

$$R^2_{adj} = 1 - (1 - R^2) \cdot \frac{n-1}{n-p-1} \tag{5}$$

p... number of regressors (independent variables), n... sample size

3 Classification

Classification is a statistical method which identifies a new observation into one of the several categories. More information here: `https://en.wikipedia.org/wiki/Statistical_classification`.

Problem: We will try to deal with the same problem for all listed methods below. We have a dataset of 400 users of some social network. The dataset tells us whether the user bought or didn't buy the product. The dataset contains user ID, salary and age. This is the sample of the dataset and the visualisation of the training set. Note that the axis shows values already scaled

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15569641	Female	58	95000	1
15815236	Female	45	131000	1
15811177	Female	35	77000	0
15680587	Male	36	144000	1

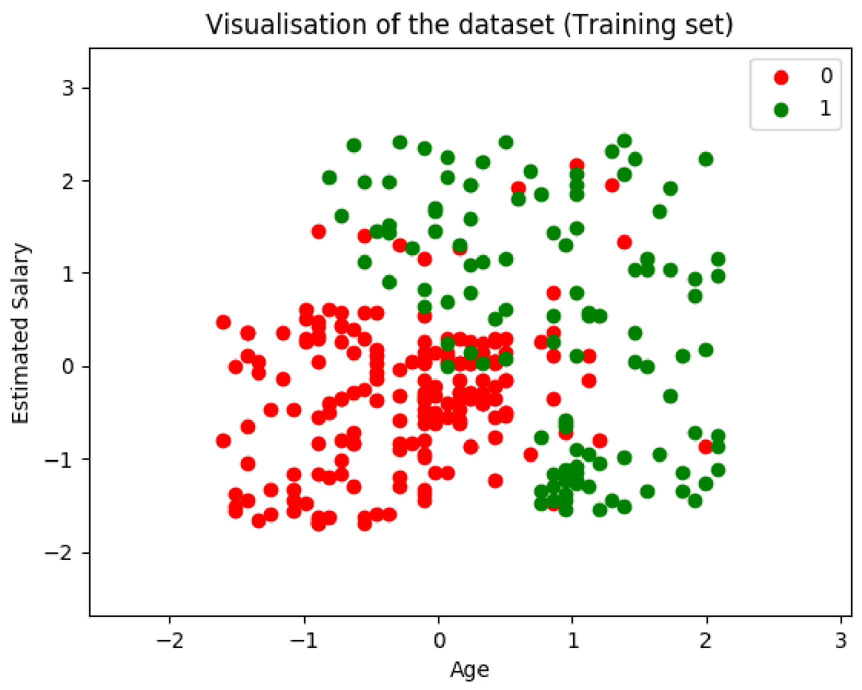


Fig. 7. Visualisation of the dataset for Classification problems. 1 means purchased, 0 means otherwise. Python

3.1 Classification code template in Python

```python
# Classification template

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting classifier to the Training set
# Create your classifier here

# Predicting the Test set results
```

```python
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Classifier (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

3.2 Classification code template in R

```r
# Classification template

# Importing the dataset
dataset = read.csv('Social_Network_Ads.csv')
dataset = dataset[3:5]

# Encoding the target feature as factor
dataset$Purchased = factor(dataset$Purchased, levels = c(0, 1))

# Splitting the dataset into the Training set and Test set
# install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$Purchased, SplitRatio = 0.75)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)

# Feature Scaling
training_set[-3] = scale(training_set[-3])
test_set[-3] = scale(test_set[-3])

# Fitting classifier to the Training set
# Create your classifier here

# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set[-3])

# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred)

# Visualising the Training set results
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
     main = 'Classifier (Training set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
```

```
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))

# Visualising the Test set results
library(ElemStatLearn)
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3], main = 'Classifier (Test set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

3.3 Logistic regression

Logistic regression deals with the situations when DV is in binary form (0/1, bought/didn't bought, etc.). The basic theory of Logistic regression is as follows: We take the Simple Linear Regression

$$y = b_0 + b_1 * x_1, \tag{6}$$

apply the *Sigmoid function*

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

and solve

$$ln(\frac{p}{1-p}) = b_0 + b_1 * x_1. \tag{8}$$

So basically this is what happens:

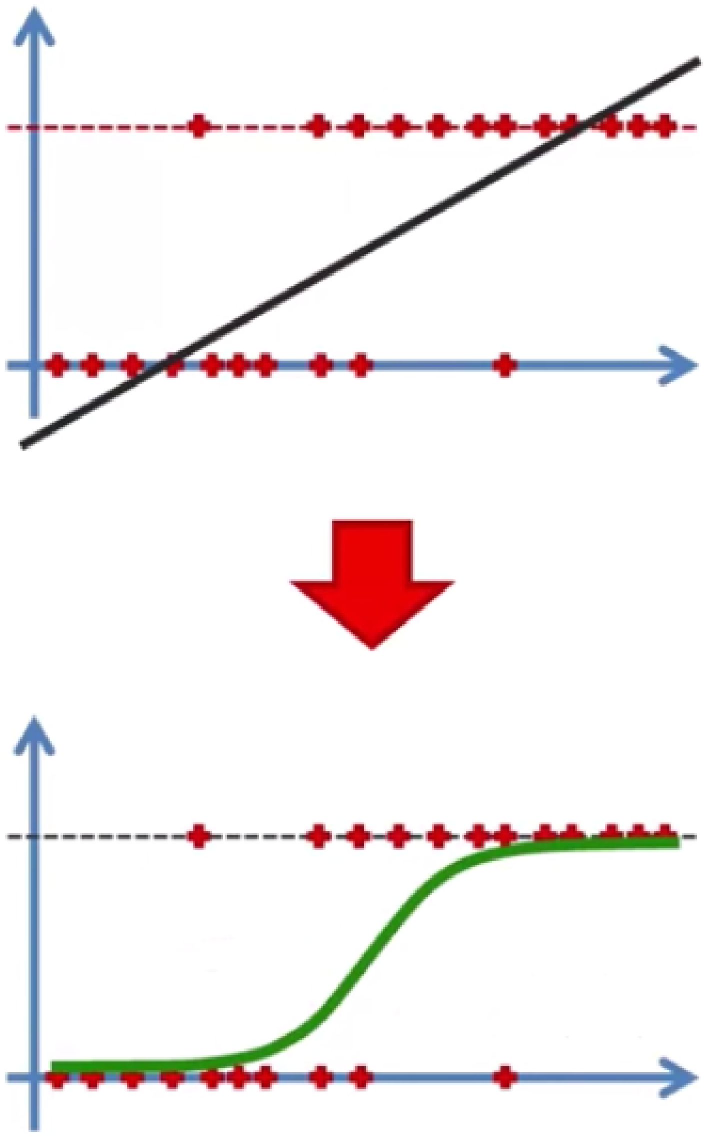


Fig. 8. Top - Simple Linear Regression, Bottom - Logistic Regression

This code implements Logistic regression in Python.

```
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

This code implements Logistic regression in R.

```
# Fitting Logistic Regression to the Training set
classifier = glm(formula = Purchased ~ .,
                 family = binomial,
                 data = training_set)
```

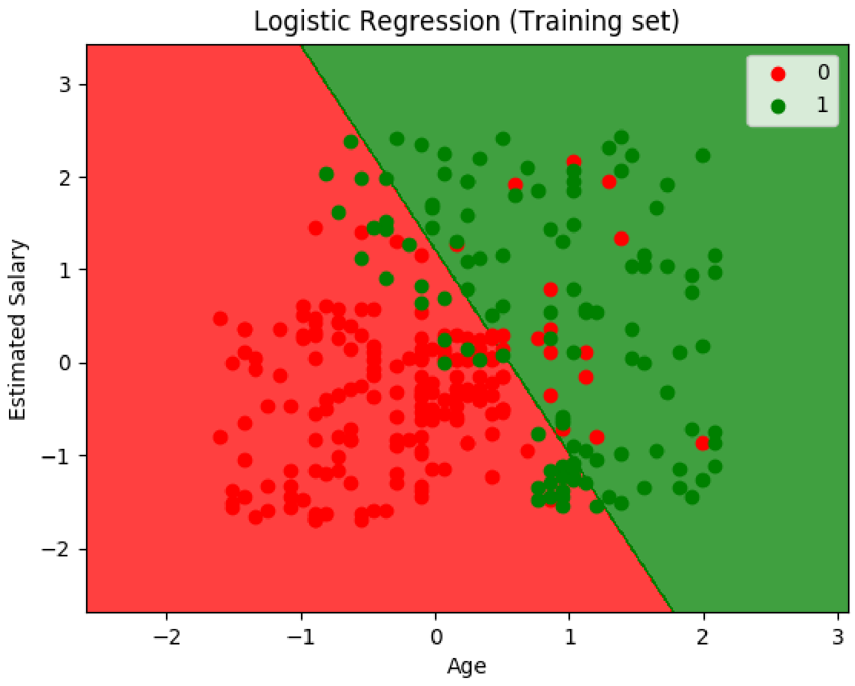


Fig. 9. Logistic regression in Python

3.4 K-Nearest neighbours

K-Nearest neigbours works as follows: we take the new observation which we want to classify. Then we take a look at k nearest points from the training set. Based on the nearest neighbours we decide the class of the new observation. We need to choose k first.

This code implements the K-NN classifier in Python.

```python
# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5,
      metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

This code implements the K-NN classifier in R.

```r
# Fitting K-NN to the Training set and Predicting the Test set res
library(class)
y_pred = knn(train = training_set[, -3],
             test = test_set[, -3],
             cl = training_set[, 3],
             k = 5,
             prob = TRUE)
```

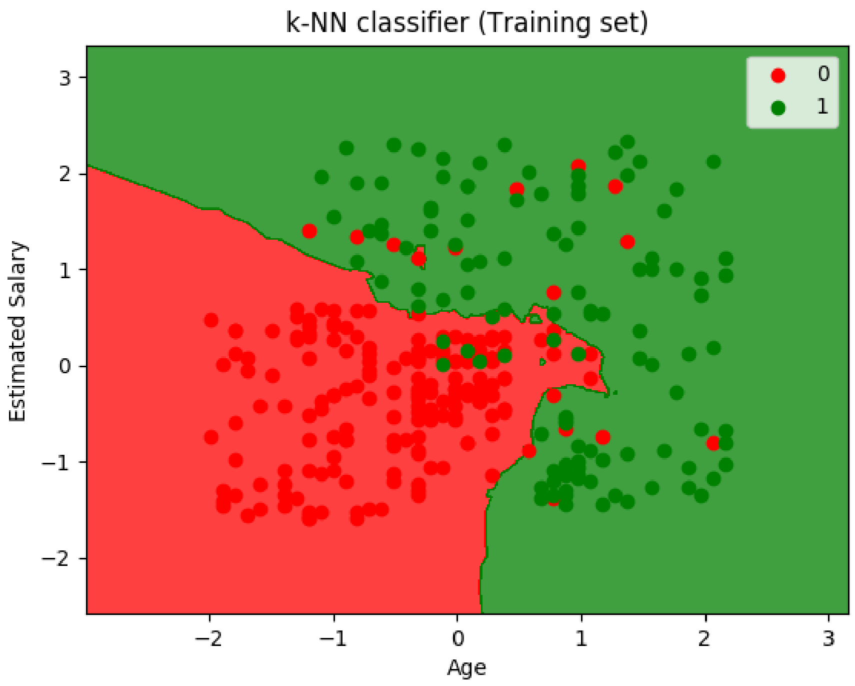


Fig. 10. k-NN algorithm with $k = 5$ in Python

3.5 SVM (Support Vector Machine)

SVM classifier can be linear with linear kernel. SVM is **nonlinear** when using different kernel using the *kernel trick* which maps the data to higher dimension to be linearly separable. More about SVM: `https://en.wikipedia.org/wiki/Support_vector_machine`.

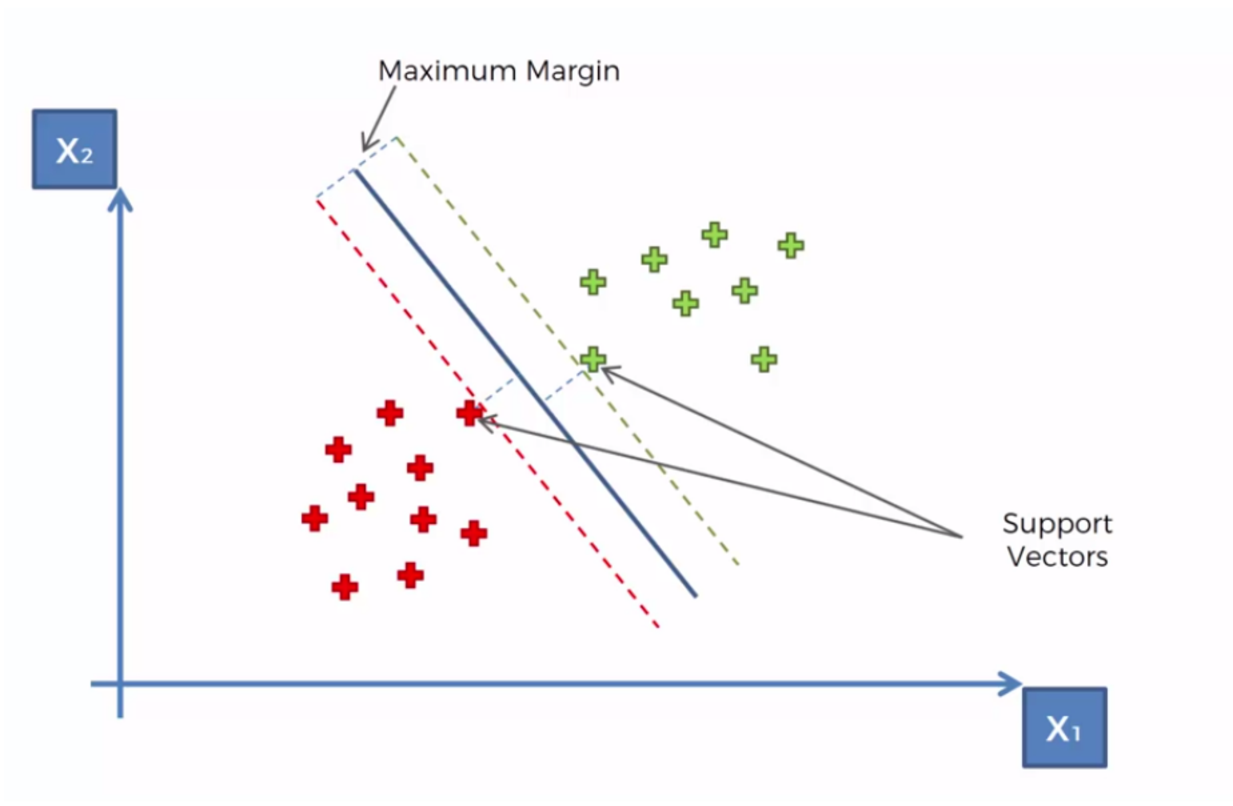


Fig. 11. Support Vector Machine basic

This code implements the SVM classifier in Python.

```
# Fitting SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
```

This code implements the SVM classifier in R.

```
# Fitting SVM to the Training set
# install.packages('e1071')
library(e1071)
classifier = svm(formula = Purchased ~ .,
                    data = training_set,
                    type = 'C-classification',
                    kernel = 'linear')
```

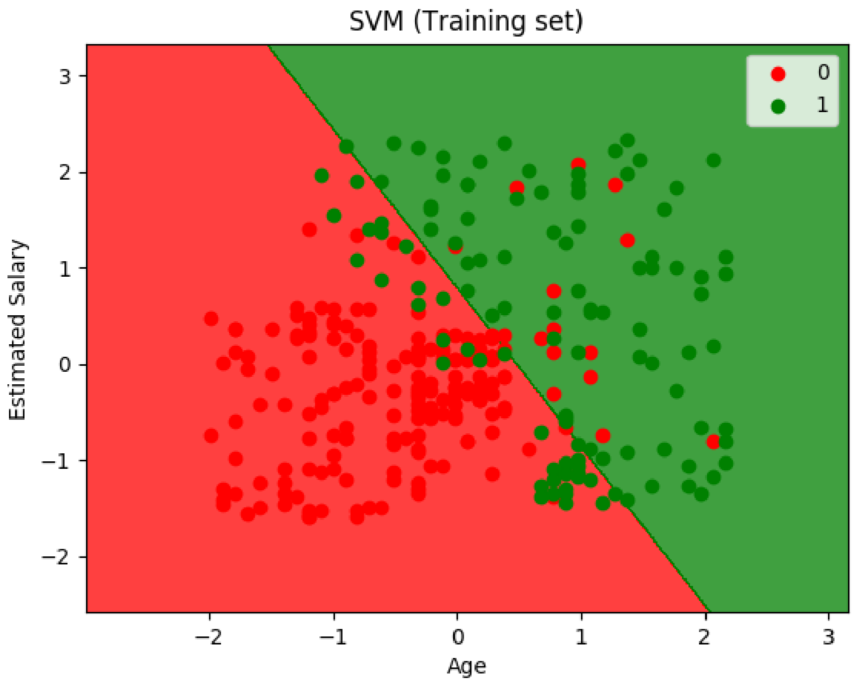


Fig. 12. Support Vector Machine with **linear** kernel in Python

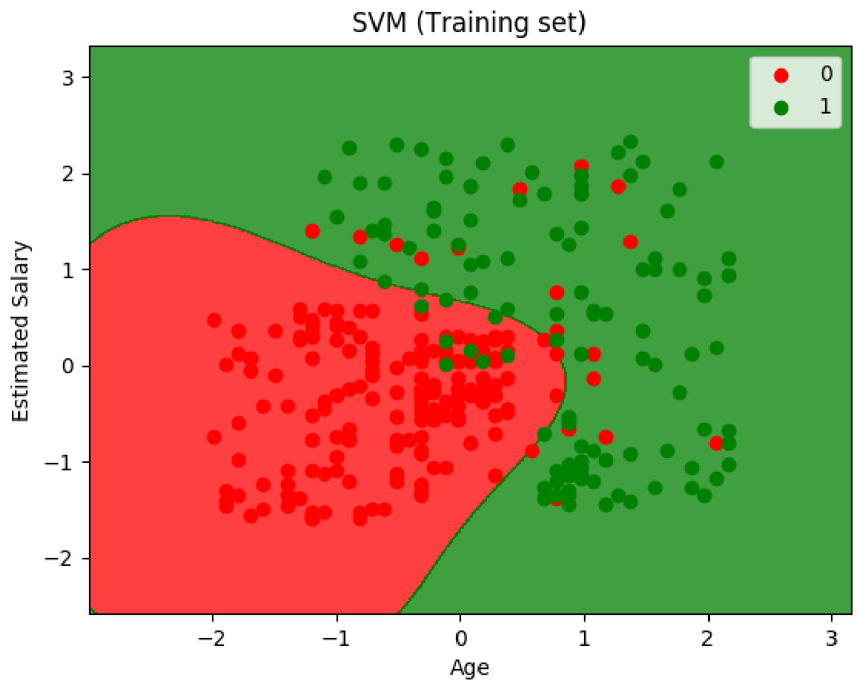


Fig. 13. Support Vector Machine with **RBF** kernel in Python

3.6 Kernel SVM

More about Kernel SVM in the Udemy course.

3.7 Naive Bayes

The Bayes theorem is the known equation:

$$P(A|X) = \frac{P(X|A) \cdot P(A)}{P(X)}. \tag{9}$$

A is the DV and can have two values $a_1; a_2$ and X is the feature vector.

In Naive Bayes Classification method we calculate the variables in the equation in this order:

1. $P(A) = \frac{\text{number of observations with property-}a_i}{\text{number of total observations}}$ ... prior probability.
2. $P(X) = \frac{\text{number of similar observations}}{\text{number of total observations}}$ ... marginal likelihood. Probabilty of the occurence of the particular combination of feature vector X. Number of similar observations depends on the are we define.

3. $P(X|A) = \frac{\text{number of similar observations with property-}a_i}{\text{number of total observations with property-}a_i}$ ... likelihood.

4. $P(A|X)$ ... finally we get posterior probabily which we were looking for.

This code implements the Naive Bayes Classifier in Python.

```python
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

This code implements the Naive Bayes Classifier in R.

```r
# Fitting Naive Bayes to the Training set
# install.packages('e1071')
library(e1071)
classifier = naiveBayes(x = training_set[-3],
                        y = training_set$Purchased)
```

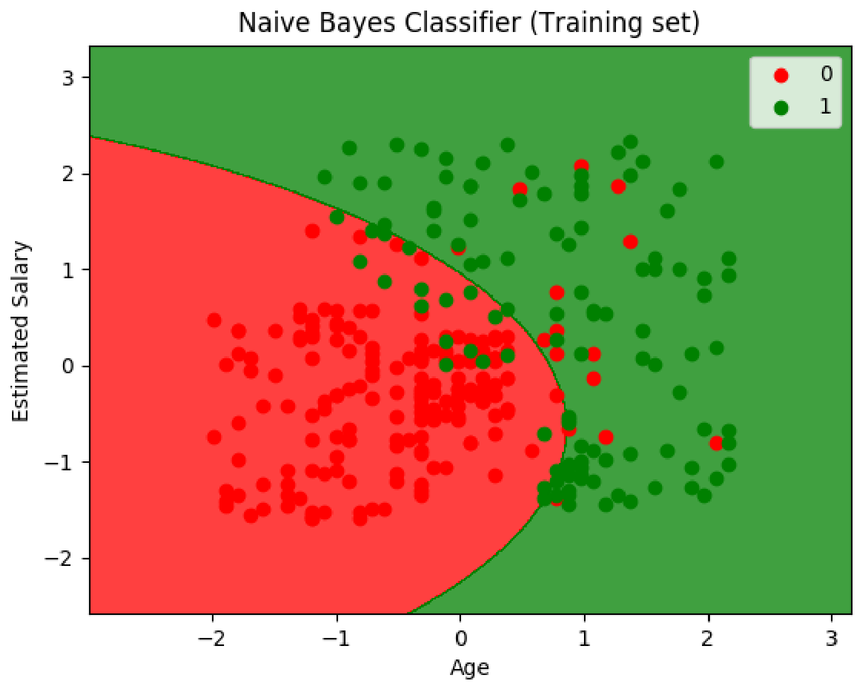


Fig. 14. Naive Bayes Classifier in Python

3.8 Decision Tree Classification

Term **CART** - Classification and Regression Trees.
Similar to Decision Tree Regression.
This code implements the Decision Tree Classifier in Python.

```python
# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy')
classifier.fit(X_train, y_train)
```

This code implements the Decision Tree Classifier in R.

```r
# Fitting Decision Tree Classification to the Training set
# install.packages('rpart')
library(rpart)
```

```
classifier = rpart(formula = Purchased ~ .,
                   data = training_set)
```

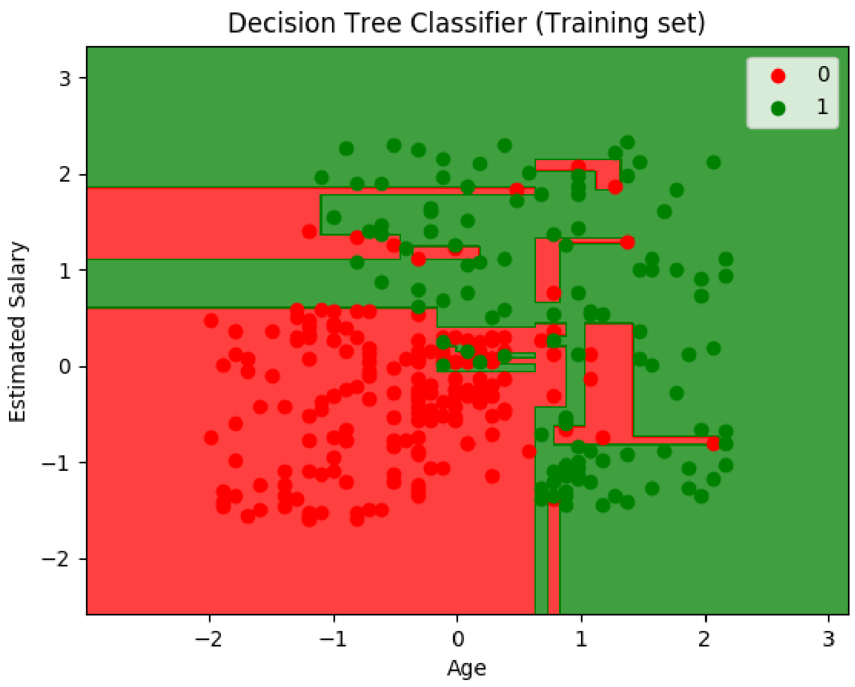


Fig. 15. Decision Tree in Python

In R we can also see the Decision Tree itself simply by using `plot(classifier)`.

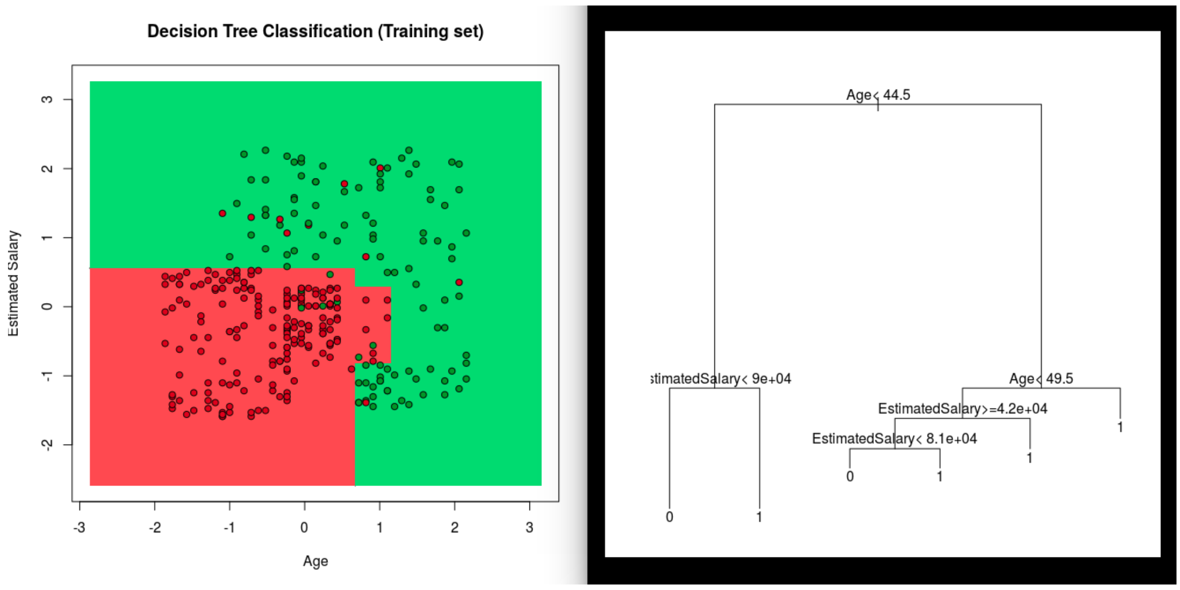


Fig. 16. Decision Tree in R. Visualisation and Decision Tree itself

3.9 Random Forest Classification

Similarly to Random Forest Regression, the Random Forest Classification uses many Decision Trees and then average their results. Random Forest Classification is the algorithm of choice for the Microsoft Kinect developers.

This code implements the Decision Tree Classifier in Python.

```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10,
                                    criterion = 'entropy')
classifier.fit(X_train, y_train)
```

This code implements the Decision Tree Classifier in R.

```
# Fitting Random Forest Classification to the Training set
# install.packages('randomForest')
library(randomForest)
set.seed(123)
classifier = randomForest(x = training_set[-3],
                                y = training_set$Purchased,
                                ntree = 500)
```

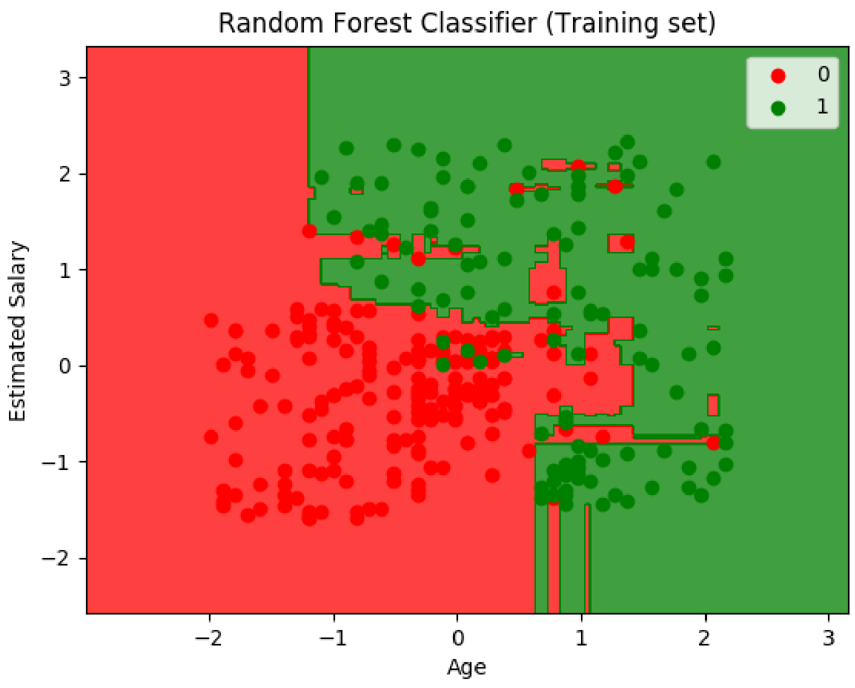


Fig. 17. Random Forest Classifier with 10 Decision Trees in Python

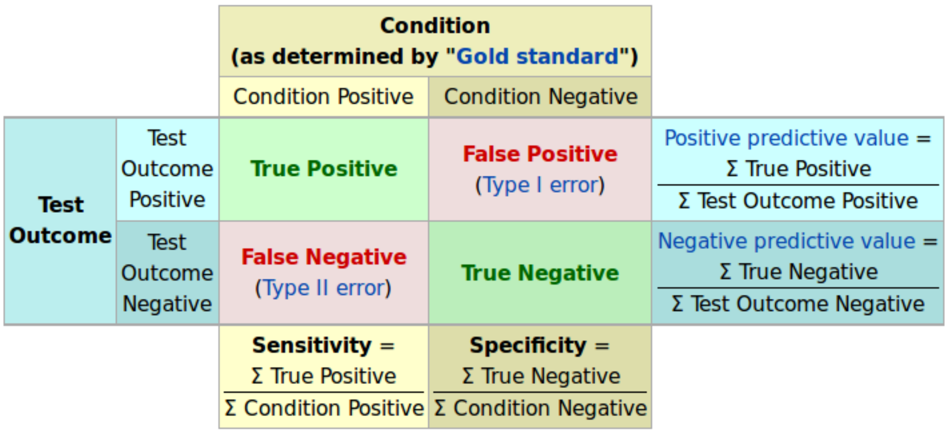
3.10 Evaluating Classification model performance

The following approaches are used to pick the best model.

Confusion matrix

Confusion matrix tells us number of Correct predictions (sum of diagonal) and number of wrong predictions. It also distinguish

- Type error 1 (False positive)
- Type error 2 (False negative).



		Condition (as determined by "Gold standard")		
		Condition Positive	Condition Negative	
Test Outcome	Test Outcome Positive	**True Positive**	**False Positive** (Type I error)	Positive predictive value = $\frac{\Sigma \text{ True Positive}}{\Sigma \text{ Test Outcome Positive}}$
	Test Outcome Negative	**False Negative** (Type II error)	**True Negative**	Negative predictive value = $\frac{\Sigma \text{ True Negative}}{\Sigma \text{ Test Outcome Negative}}$
		Sensitivity = $\frac{\Sigma \text{ True Positive}}{\Sigma \text{ Condition Positive}}$	**Specificity** = $\frac{\Sigma \text{ True Negative}}{\Sigma \text{ Condition Negative}}$	

Fig. 18. Confusion matrix

Accuracy paradox

Unfortunately the confusion matrix has one big withdrawal. Let's define Accuracy rate.

$$AR = \frac{n_{correct}}{n_{total}} \tag{10}$$

It should not be considered the ultimate measure for the performance of the models. Let's consider the following scenario. We build a model with the following confusion matrix:

Table 1. Situation before

	Predicted DV		
Actual DV		0	1
	0	9700	150
	1	50	100

We simply calculate that the Accuracy rate is $AR = 9800/10000 = 98\%$. Then we abandon the model and always predict negative (0). We come out with the following confusion matrix.

Table 2. Situation without model but with better AR

		Predicted DV	
		0	1
Actual DV	0	9850	0
	1	150	0

We calculate that the Accuracy rate is $AR = 9850/10000 = 98.5\%$. We have three times more Type 2 errors but the AR is better. That's why **Accuracy rate is not the best evaluating technique!**

CAP curve

CAP stands for Cumulative Accuracy Profile.

Problem: We have a dataset of 100000 customers and 10% of them will purchase the product. We want to build a model which finds the customers with the highest probability of purchasing the product. Then we contact them. We want our model to have the highest possible hit rate of contacted-purchased. The "Crystal ball" will have 100% hit rate, that means first 10000 contacted equeals 10000 purchases.

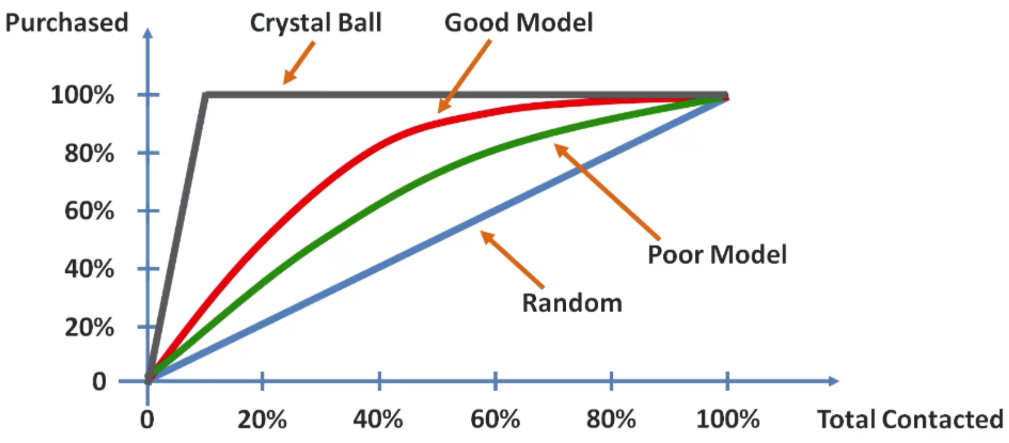


Fig. 19. CAP curve explained

4 Clustering

Clustering is the form of unsupervised learning. That means we do not have a training set but we have to find the clusters ourselves. We will show two approches. The first step in both of them is to establish the number of clusters k. Each approach use different method.

Problem: We are data analysts hired by mall company. They provide us with the dataset containing information about their customers. They want us to divide their customers into several (not specified k) classes. This is the sample of the dataset. We will use only *salary* and *spending score*.

CustomerID	Genre	Age	Annual Income (k)	Spending Score (1-100)
0001	Male	19	15	39
0002	Male	21	15	81
0003	Female	20	16	6
0004	Female	23	16	77
0005	Female	31	17	40
0006	Female	22	17	76

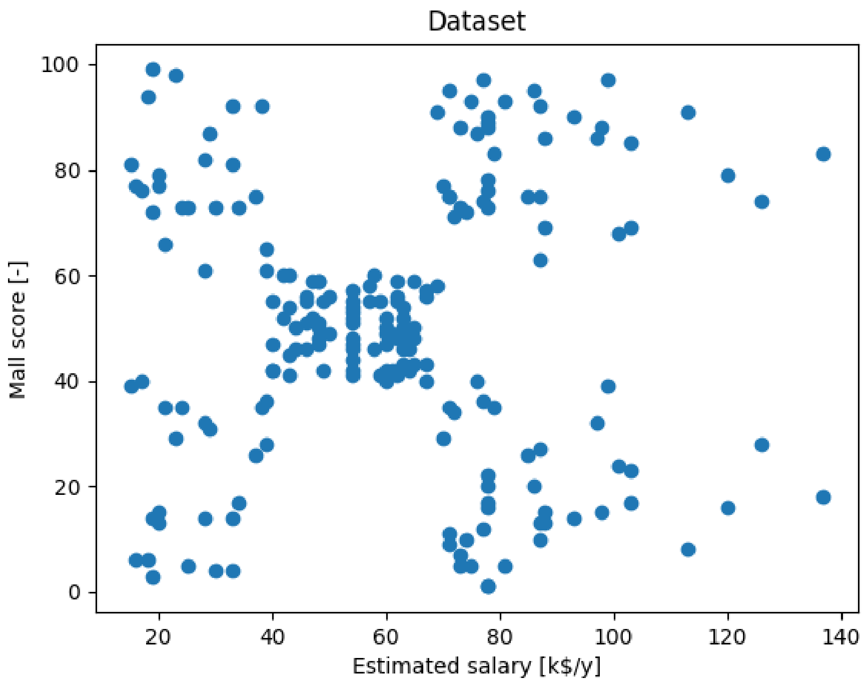


Fig. 20. Visualisation of the dataset for clustering problem in Python

4.1 K-means clustering

1. Choose the number k of clusters.
2. Select at random k points. The *centroids.* (not necessarily from the dataset)
3. Assign each datapoint to the closest *centroid*, that forms k clusters.
4. Compute and place the new *centroids* for each cluster.
5. Reassing each point according to the new *centroids* and go to step 4. If no reassignment took place. **Your model is ready**.

Selecting the random points for *centroids* might be problematic and can lead to faulty results. K-means++ algorithm deals with it.

Elbow method to find optimal K

First of all we need to establish the number of clustes k. In K-means we use the "Elbow method".

We place a *Centroid* in the dataset and calculate the distances between all the points and the centroid. Then we place another *centroid* and calculate and sum the distances of all points to the closest *centrois.* We will call this value *WCSS* We continue to increase the number of *centroids* k. It is clear that as k closes to the number of the points in the dataset, the *WCSS* gets closer to zero.

The equation can be written as

$$WCSS = \sum_{1}^{k}\Big(\sum_{P_i \text{in Cluster i}} dist(P_i, C_1)^2\Big) \tag{11}$$

where k is the number of clusters. We can plot the values in the graph and choose the k from the "elbow".

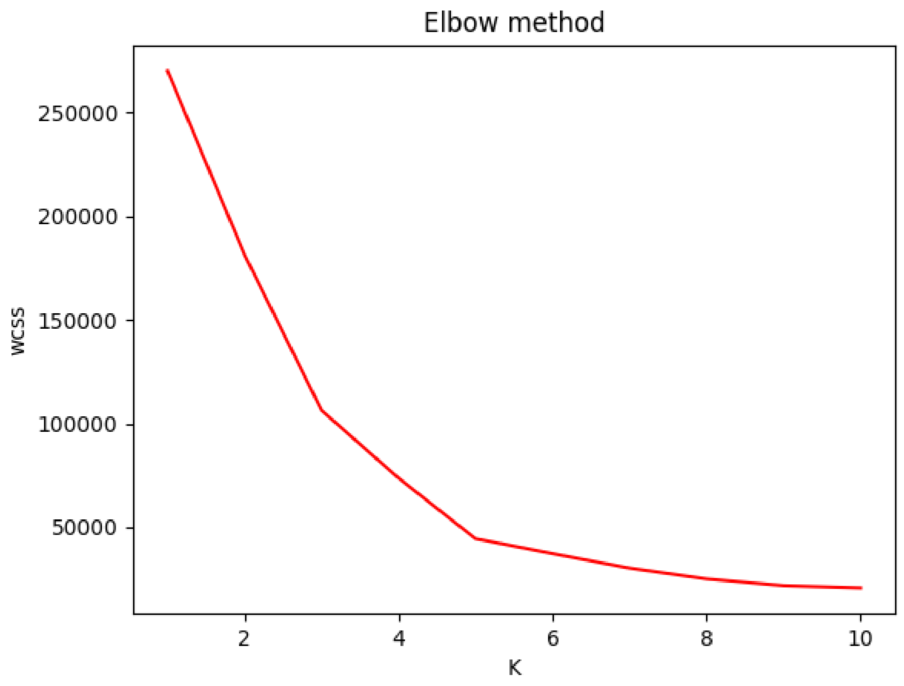


Fig. 21. Elbow method to determine the optimal K in Python

From the picture we see that the optimal k is 5.

Implementation

This code implements the K-means algorithm in Python with the elbow method and visualisation.

```python
# K-Means Clustering

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values

# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)

# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
             s = 300, c = 'yellow', label = 'Centroids')
plt.title("Dataset clustered by HC")
plt.ylabel("Mall score [-]")
plt.xlabel("Estimated salary [k$/y]")
plt.legend()
plt.show()
```

This code implements the K-means algorithm in R with the elbow method and visualisation.

```
# K-Means Clustering

# Importing the dataset
dataset = read.csv('Mall_Customers.csv')
dataset = dataset[4:5]

# Using the elbow method to find the optimal number of clusters
set.seed(6)
wcss = vector()
for (i in 1:10) wcss[i] = sum(kmeans(dataset, i)$withinss)
plot(1:10,
     wcss,
     type = 'b',
     main = paste('The Elbow Method'),
     xlab = 'Number of clusters',
     ylab = 'WCSS')

# Fitting K-Means to the dataset
set.seed(29)
kmeans = kmeans(x = dataset, centers = 5)
y_kmeans = kmeans$cluster

# Visualising the clusters
library(cluster)
clusplot(dataset,
         y_kmeans,
         lines = 0,
         shade = TRUE,
         color = TRUE,
         labels = 2,
         plotchar = FALSE,
         span = TRUE,
         main = paste('Clusters of customers'),
         xlab = 'Annual Income',
         ylab = 'Spending Score')
```

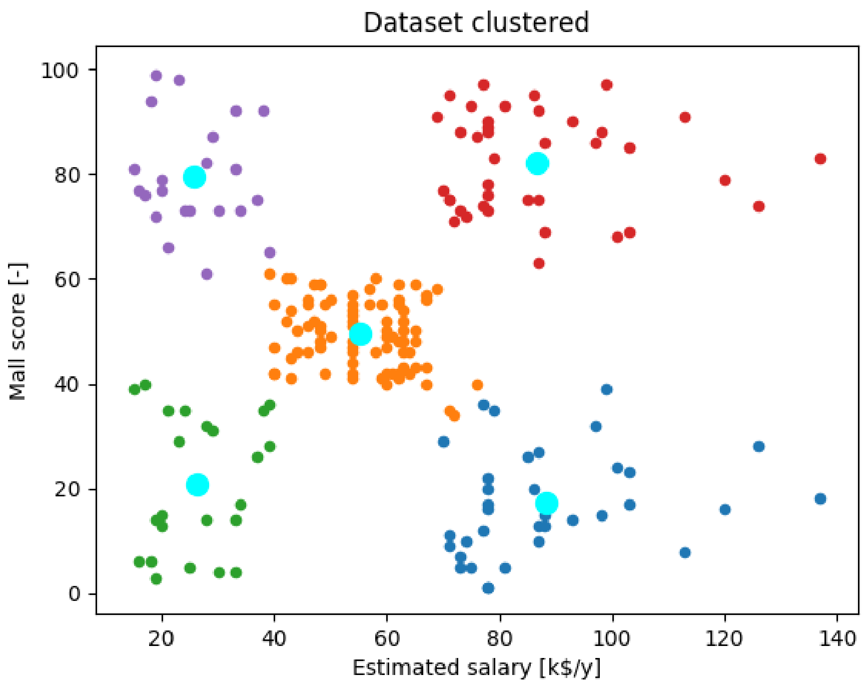


Fig. 22. Result of K-mean algorithm in Python, blue points are centroids.

4.2 Hierarchical clustering

Agglomerative HC:

1. Make each data point a single cluster (that makes N clusters)
2. Take the two closest points and make them 1 cluster (N-1 clusters)
3. Take the two **closest clusters** and make them 1 cluster (N-2 clusters)
4. Repeat step 3 until there is only 1 cluster

What is the **distance between clusters**? We have several valid answers. It depends on the usecase.

- Distance of closest points
- Distance of furthest points
- Average distance
- Distance of centroids

Dendrograms

To choose the optimal k - the number of clusters, we use dendrograms. Dendrograms trace the history of points clustering and their distances (or disimilarities). We choose the longest vertical line that can be split by horizontal line without interruption.

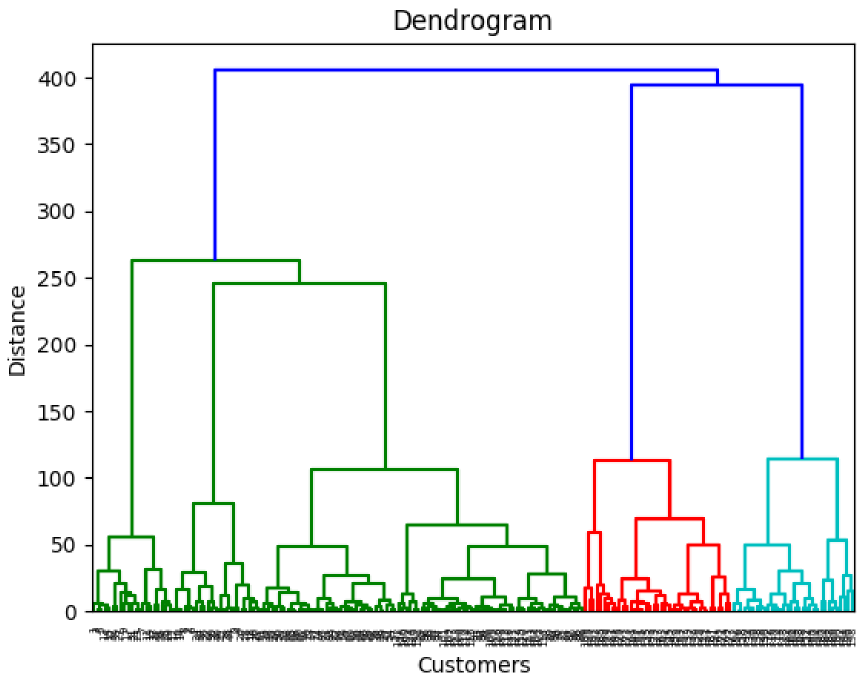


Fig. 23. Dendrogram - scipy library in Python

Code

linkage is the method that is trying to minimize the variance in each cluser.

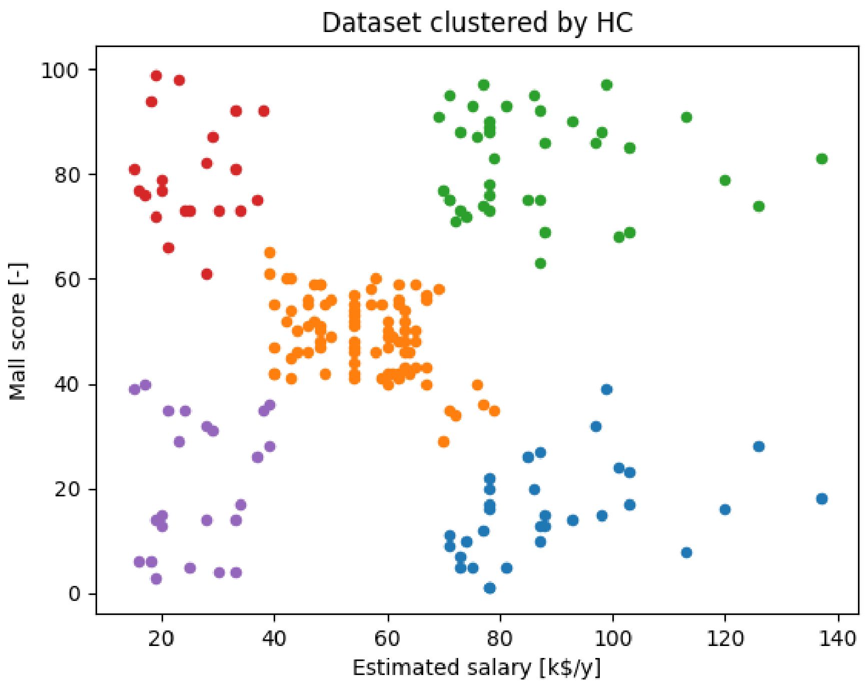


Fig. 24. Result of hierarchical clustering in Python

5 Association rule learning

"People who bought ... also bought ..."

Problem: We have a dataset containing transactions in a mall. We want to find associative rules. Here is the sample of the dataset. One row is one basket bought in the mall.

shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage
cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral

5.1 Apriori

The important variables. The examples are from the problem.

- Support (M1) $= \frac{transactions\ containing\ M1}{total\ transactions}$
- Confidence (M1 $\rightarrow$ M2) $= \frac{transactions\ containing\ M1\ and\ M2}{transactions\ containing\ M1}$
- Lift (M1 $\rightarrow$ M2) $= \frac{Confidence(M1 \rightarrow M2)}{Support(M2)}$

The algorithm has following steps:

1. Select a minimum support and confidence.
2. Take all the subsets in transactions having higher support than minimum support.
3. Take all the rules of these subsets having higher confidence than minimum confidence.
4. Sort the rules by decreasing lift.

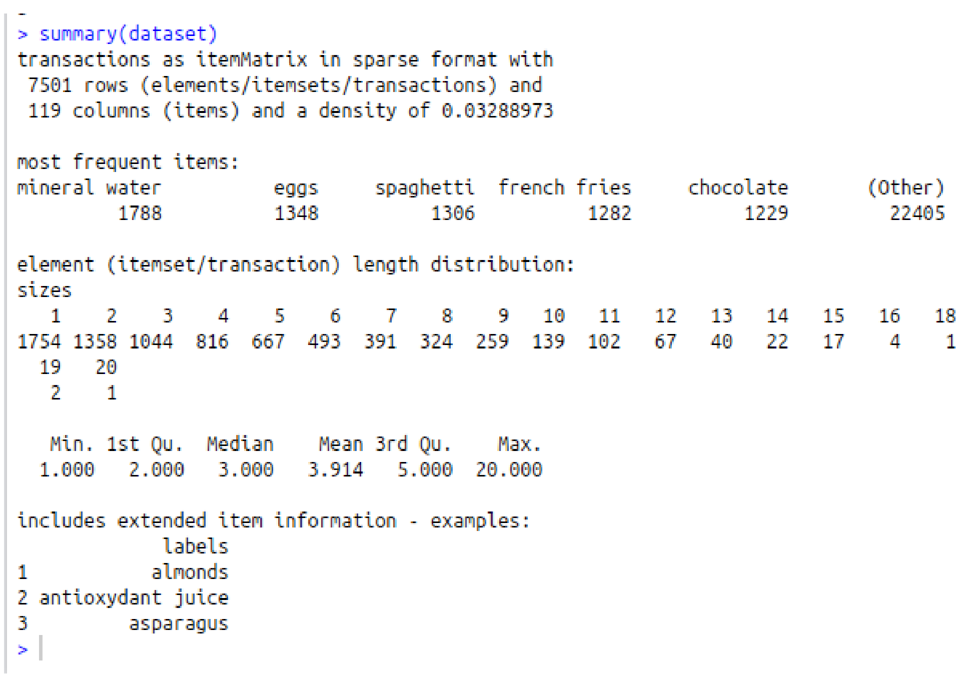
This code implements the Apriori algorithm in R.

```r
# Apriori

# Data Preprocessing
#install.packages('arules')
library(arules)
dataset = read.csv('Market_Basket_Optimisation.csv', header = FALSE)
dataset = read.transactions('Market_Basket_Optimisation.csv', sep = ',', rm.duplicates = TRUE)
summary(dataset)
itemFrequencyPlot(dataset, topN = 10)

# Training Apriori on the dataset
# FIRST we set wanted support. In our case we want to aim for products
#   which are bought at least 3 or 4 times a day. That means support = (4*7)/7501
# SECOND we set the confidence to default value (0.8) and see how many rules we find.
#   if we don't find enough rules then we lessen the confidence
rules = apriori(data = dataset, parameter = list(support = 0.003, confidence = 0.2))

# Visualising the results
inspect(sort(rules, by = 'lift')[1:10])
```



```
-
> summary(dataset)
transactions as itemMatrix in sparse format with
 7501 rows (elements/itemsets/transactions) and
 119 columns (items) and a density of 0.03288973

most frequent items:
mineral water          eggs     spaghetti  french fries     chocolate       (Other)
         1788          1348          1306          1282          1229         22405

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   18
1754 1358 1044  816  667  493  391  324  259  139  102   67   40   22   17    4    1
  19   20
   2    1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   3.000   3.914   5.000  20.000

includes extended item information - examples:
             labels
1           almonds
2 antioxydant juice
3         asparagus
>
```

Fig. 25. Summary of the R object. It is the matrix with a small amount of "ones" and a lot of "zeros". The number of rows equals the number of transactions and number of columns equals the cardinality of bought items set

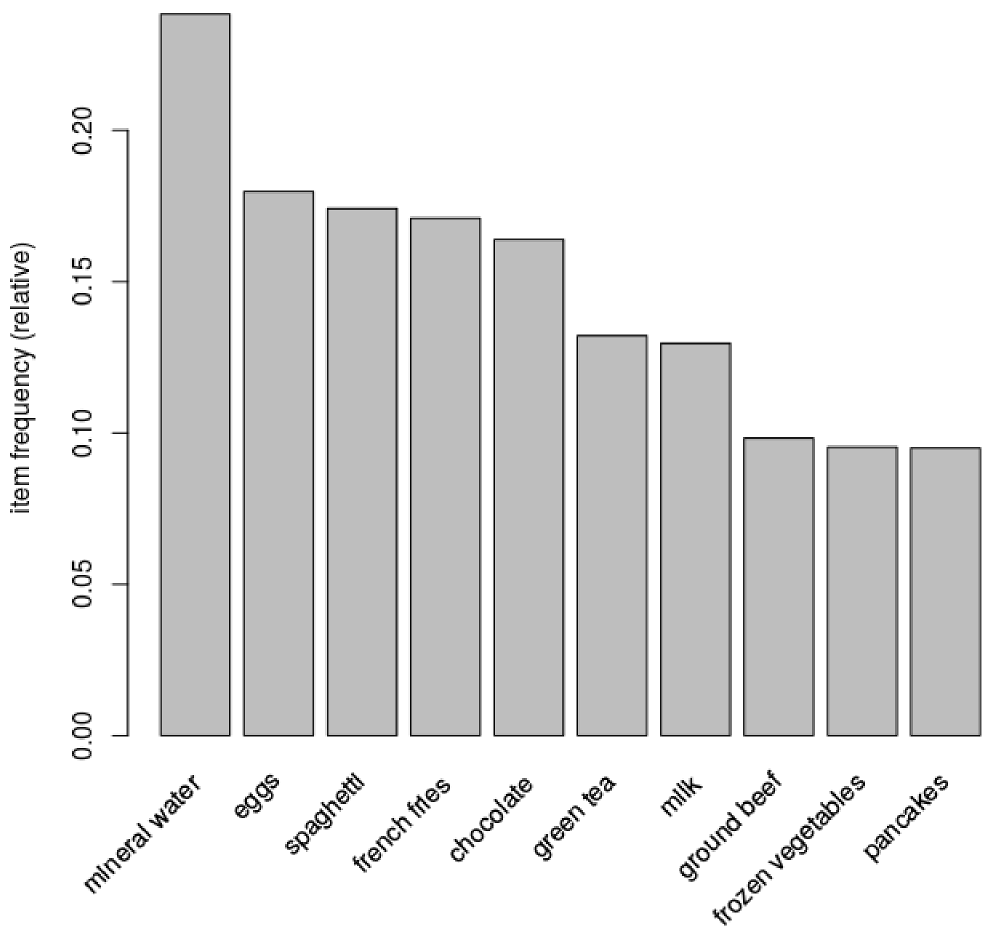


Fig. 26. Visualisation of the most bought products. That means it shows their support

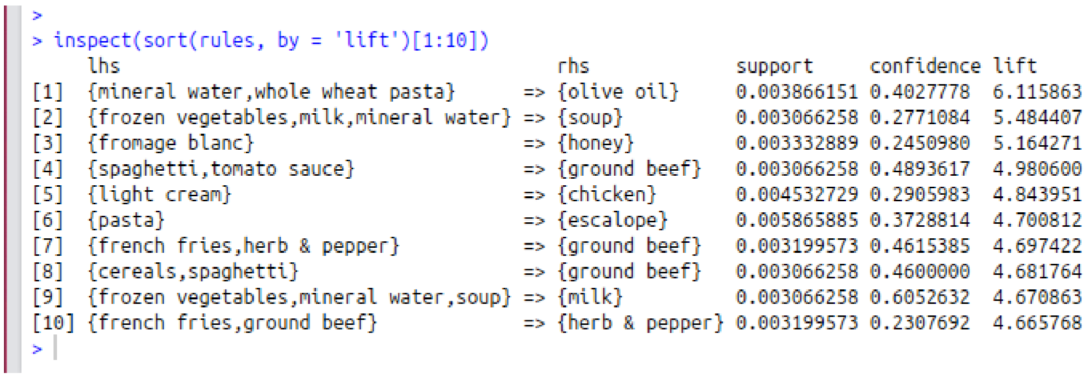

```
>
> inspect(sort(rules, by = 'lift')[1:10])
     lhs                                       rhs             support     confidence lift
[1]  {mineral water,whole wheat pasta}      => {olive oil}     0.003866151 0.4027778  6.115863
[2]  {frozen vegetables,milk,mineral water} => {soup}          0.003066258 0.2771084  5.484407
[3]  {fromage blanc}                        => {honey}         0.003332889 0.2450980  5.164271
[4]  {spaghetti,tomato sauce}               => {ground beef}   0.003066258 0.4893617  4.980600
[5]  {light cream}                          => {chicken}       0.004532729 0.2905983  4.843951
[6]  {pasta}                                => {escalope}      0.005865885 0.3728814  4.700812
[7]  {french fries,herb & pepper}           => {ground beef}   0.003199573 0.4615385  4.697422
[8]  {cereals,spaghetti}                    => {ground beef}   0.003066258 0.4600000  4.681764
[9]  {frozen vegetables,mineral water,soup} => {milk}          0.003066258 0.6052632  4.670863
[10] {french fries,ground beef}             => {herb & pepper} 0.003199573 0.2307692  4.665768
>
```

Fig. 27. Top 10 found rules in the dataset of bascet. Sorted by lift

5.2 Eclat

Eclat is similar to Apriori but much easier. We use only support.

The algorithm has following steps:

1. Select a minimum support.
2. Take all the subsets in transactions having higher support than minimum support.
3. Sort these subsets by decreasing support.

This code implements the Eclat algorithm in R. The only differences between this and Apriori is the function call and only support parameter.

```

# Training Eclat on the dataset
rules = eclat(data = dataset,
        parameter = list(support = 0.003, minlen = 2))
```

6 Reinforcement learning

Reinforcement Learning is a branch of Machine Learning, also called Online Learning. It is used to solve interacting problems where the data observed up to time t is considered to decide which action to take at time $t+1$. It is also used for Artificial Intelligence when training machines to perform tasks such as walking. Desired outcomes provide the AI with reward, undesired with punishment. Machines learn through trial and error.

6.1 Upper Confidence Bound (UCB)

6.2 Thompson sampling

7 Natural language processing

Natural Language Processing (or NLP) is applying Machine Learning models to text and language. Teaching machines to understand what is said in spoken and written word is the focus of Natural Language Processing. Whenever you dictate something into your iPhone / Android device that is then converted to text, that's an NLP algorithm in action.

You can also use NLP on a text review to predict if the review is a good one or a bad one. You can use NLP on an article to predict some categories of the articles you are trying to segment. You can use NLP on a book to predict the genre of the book. And it can go further, you can use NLP to build a machine translator or a speech recognition system, and in that last example you use classification algorithms to classify language. Speaking of classification algorithms, most of NLP algorithms are classification models, and they include Logistic Regression, Naive Bayes, CART which is a model based on decision trees, Maximum Entropy again related to Decision Trees, Hidden Markov Models which are models based on Markov processes.

A very well-known model in NLP is the Bag of Words model. It is a model used to preprocess the texts to classify before fitting the classification algorithms on the observations containing the texts.

In this part, you will understand and learn how to:

- Clean texts to prepare them for the Machine Learning models,
- Create a Bag of Words model,
- Apply Machine Learning models onto this Bag of Worlds model.

8 Deep learning

Deep Learning is the most exciting and powerful branch of Machine Learning. Deep Learning models can be used for a variety of complex tasks:

- Artificial Neural Networks for Regression and Classification
- Convolutional Neural Networks for Computer Vision
- Recurrent Neural Networks for Time Series Analysis
- Self Organizing Maps for Feature Extraction
- Deep Boltzmann Machines for Recommendation Systems
- Auto Encoders for Recommendation Systems

In this part, you will understand and learn how to implement the following Deep Learning models:

1. Artificial Neural Networks for a Business Problem
2. Convolutional Neural Networks for a Computer Vision task

8.1 Artificial Neural Network

8.2 Convolutional Neural Network

9 Dimensionality reduction

9.1 Principal Component Analysis (PCA)

9.2 Linear Discriminant Analysis (LDA)

9.3 Kernel PCA

10 Model selection and boosting

10.1 Model selection

10.2 XGBoost