Aarhus University

Decision Support Systems

PROJECT REPORT

Authors:	Supervisor:
Lasse Lildholdt (201507170)	Christian Fischer Pedersen
Stinus Skovgaard (201507170)	
Daniel Tøttrup (201509520)	
Johan Vasegaard (201505215)	
Frederik Madsen (201504477)	

11. maj 2020

Indhold

Figurer

Kapitel 1

INTRODUCTION

In this project report, the reader will be presented with problem solutions for the course Decision support systems. Throughout the solution the reader will achieve knowledge on several different subject within the main area. Each topic will be presented with the theory along with solutions for appropiate exercise to validate the presented theory.

The main topics which will be handled in this report will be as follows:

- Simple linear regression / Multiple linear regression
- Logisitic regression / Linear discriminant analysis
- Cross validation / Bootstrap
- Subset selection
- Shrinkage methods / Dimension reduction methods
- Polynomial regression / Regression splines
- Support vector machines
- Clustering methods

Kapitel 2

SIMPLE LINEAR REGRESSION / MULTIPLE LINEAR REGRESSION

The simple Linear Regression approach is a quick and simple method for predicting a response Y based on X. The linear model is used to give an idea of the relationship between to dataset. For this to be true an assumption that the two variables have a linear relationship is needed. The mathematical representation of this can be seen below.

$$Y \approx \beta_0 + \beta_1 X \tag{2.1}$$

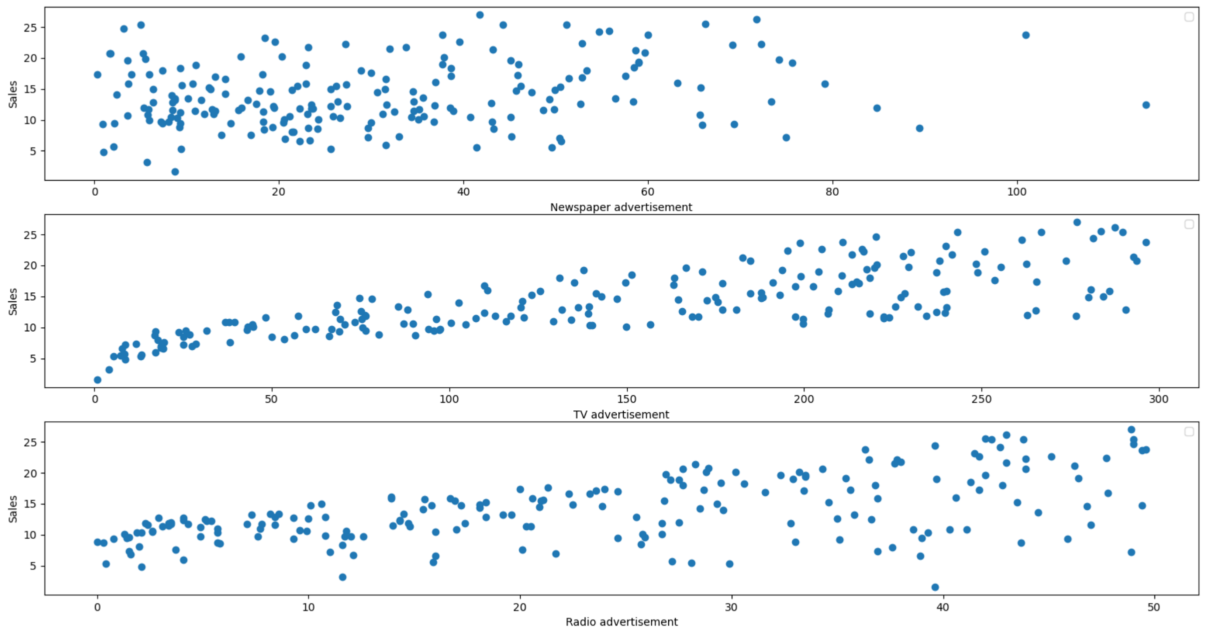
eq. (2.1) can also be seen as "Regressing Y onto X". As an example the dataset Advertising.csv contains sales on a certain product and advertisement money spent on certain media platforms. X represents TV advertising and Y represents sales. It is the possible to regress sales onto TV. This is expressed as:

$$sales \approx \beta_0 + \beta_1 TV \tag{2.2}$$

The two constants β_0 and β_1 represents the intercept (Where it intercepts the y-axis) and slope of the linear model. These constant needs to be predicted and when they have we will end out with a linear model that fits our data.

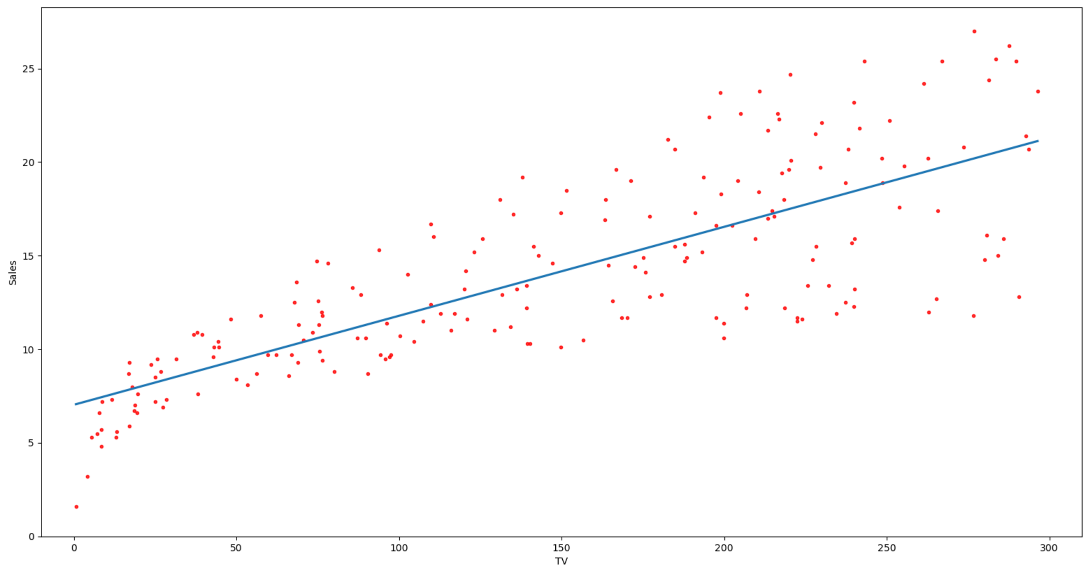
2.1 Predict β_0 and β_1

In the Advertising.csv dataset a number of observations (n = 200) have been made on amount of advertisements for tv, radio, newspaper and the corresponding sales. The advertisement data can be seen on fig. 2.1.



Figur 2.1: Advertisement data

To estimate β_0 and β_1 we need to introduce RSS which is the *residual sum of squares*. RSS is the difference or error between observed response value and predicted response value that is predicted by our linear model. On fig. 2.2 a linear model is put on top of the TV advertisement data. The RSS is the summed value of the difference between the red points to the blue line. We want this value to b as low as possible to get the most accurate model.



Figur 2.2: Linear regression on TV advertisement

We can define the RSS value as:

$$\begin{aligned} RSS = e_1^2 + e_2^2 ... e_n^2 \\ where \\ e = y_i - \hat{y} \end{aligned} \tag{2.3}$$

This means that to get to a final method we need to minimize RSS. Some calculus shows that these minimizers are:

$$\begin{aligned} \hat{\beta_1} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2} \\ \beta_0 = \overline{y} - \beta_1 \overline{x} \\ \textbf{where} \\ \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \\ \textbf{and} \\ \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \end{aligned} \tag{2.4}$$

This method of estimating the constants have been used on fig. 2.2. How this is done can be seen on the code snippet below.

```
x_mean = get_mean(x_data)
y_mean = get_mean(y_data)

num = 0
den = 0

for i in range(len(x_data)):
num += (x_data[i]-x_mean)*(y_data[i]-y_mean)
den += (x_data[i] - x_mean) ** 2

b1_hat = num/den
b0_hat = y_mean-b1_hat*x_mean

return b0_hat, b1_hat
```

β_0 and β_1 is estimated to 7.03 and 0.04 respectively for the TV advertisement data.

There will always be an error of some sort when estimating this linear model. Because of this we can add a error constant to eq. (2.1). The error constant (epsilon) is generated based on a gaussian distribution with a mean of 0. The equation will look as the following:

$$Y \approx \beta_0 + \beta_1 X + \epsilon \tag{2.5}$$

2.2 Multiple linear regression

Simple linear regression is a great to see the response of a single predictor variable. However there are times where it would be beneficial to have multiple predictor variables. Instead of having only a 1-dimensional regression line we can with multiple linear regression have a many-dimensional regression line (plane for 2D, cloud for 3D).

In simple linear regression we had an intercept constant, β_0 and a slope constant β_1. In this model we essentially have the same plus some extra slope constant and predictor variables. This can be seen on eq. (2.6).

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon \tag{2.6}$$

If we fit this equation to out advertisement example we would get:

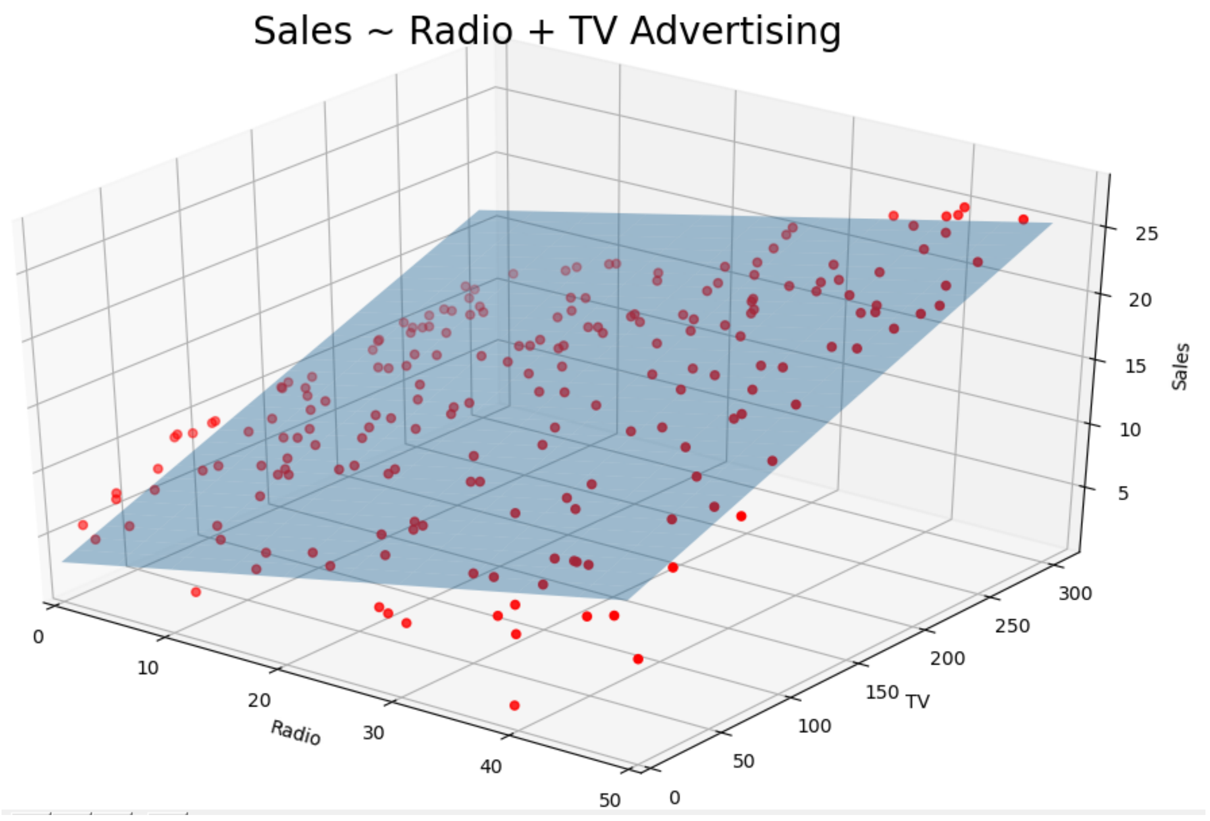
$$\mathbf{sales} = \beta_0 + \beta_1 \mathbf{TV} + \beta_2 \mathbf{Radio} + \beta_2 \mathbf{Newspaper} + \epsilon \tag{2.7}$$

2.2.1 Estimating the regression constants

To estimate the constants we do same as in the simple regression method with some changes. We still try to minimize the RSS value, but since there are more constants the RSS value is calculated with the equation eq. (2.8). However to estimate the constants itself the use of a python package has been used. Sklearn has a regression module that can estimate our constants. This can be seen in the codes snippet further down in this chapter.

$$\begin{aligned} RSS &= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - ... - \beta_p x_{ip})^2 \end{aligned} \tag{2.8}$$

When using two predictor variables the regression line becomes a plane. On fig. 2.3 the regression pane when using TV and Radio as predictor variables is shown. The plane is placed in a way such that the RSS value is as low as possible.



Figur 2.3: The regression pane with TV and Radio as predictor variables

The code for making the regression plane is listed below:

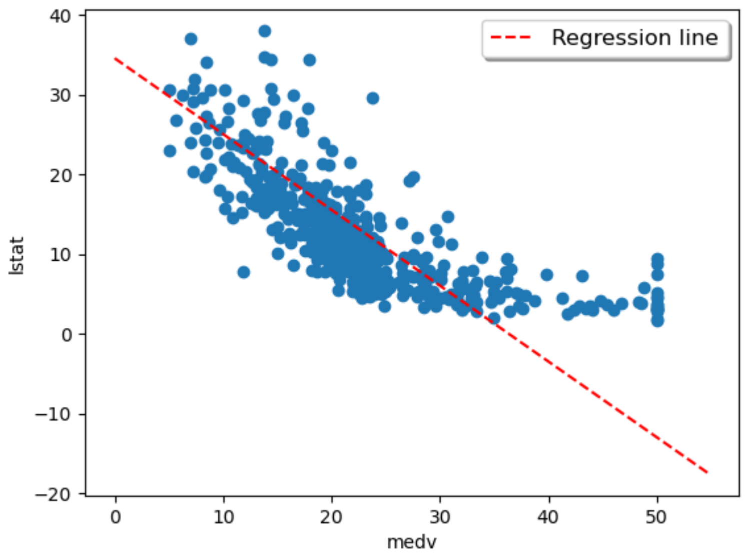
```
# Multiple Regression
regr = skl_lm.LinearRegression()

X = advertising[['Radio', 'TV']].values
y = advertising.Sales

# Fitting data to estimate constants.
regr.fit(X, y)

# Make it so that the plane fits all points
Radio = np.arange(0, 50)
TV = np.arange(0, 300)


B1, B2 = np.meshgrid(Radio, TV, indexing='xy')
Z = np.zeros((TV.size, Radio.size))

for (i, j), v in np.ndenumerate(Z):
# The response on TV and Radio
Z[i, j] = (regr.intercept_ + B1[i, j] * regr.coef_[0] + B2[i, j] * regr.coef_[1])
```

2.3 Lab results

2.3.1 Lab 3.6.2

The data used in this lab exercise is from the MASS library which contains housing data in the Boston area. First a linear regression model will be applied on the ”medv”and ”lstat. **medv** as response (y) and **lstat** as the predictor. The code snippet below shows how β_0 and β_1 is estimated. This linear model is plotted on the fig. 2.4.

```
x_mean = get_mean(x_data)
y_mean = get_mean(y_data)

num = 0
den = 0

for i in range(len(x_data)):
num += (x_data[i]-x_mean)*(y_data[i]-y_mean)
den += (x_data[i] - x_mean) ** 2

b1_hat = num/den
b0_hat = y_mean-b1_hat*x_mean

return b0_hat, b1_hat
```

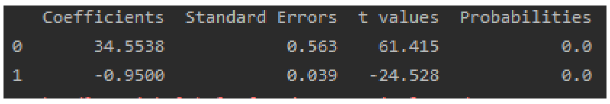


Figur 2.4: Plot of data and regression line

To get a more detailed look at the different statistical properties, the following code is executed and the use of sklearns LinearRegression module is used.

```
regr = skl_lm.LinearRegression()
regr.fit(X,y)

params = np.append(regr.intercept_, regr.coef_)
predictions = regr.predict(X)

newX = pd.DataFrame({"Constant": np.ones(len(X))}).join(pd.DataFrame(X))
MSE = (sum((y - predictions) ** 2)) / (len(newX) - len(newX.columns))

var_b = MSE * (np.linalg.inv(np.dot(newX.T, newX)).diagonal())
sd_b = np.sqrt(var_b)
ts_b = params / sd_b

p_values = [2 * (1 - stats.t.cdf(np.abs(i), (len(newX) - 1))) for i in ts_b]

sd_b = np.round(sd_b, 3)
ts_b = np.round(ts_b, 3)
p_values = np.round(p_values, 3)
params = np.round(params, 4)

myDF3 = pd.DataFrame()
myDF3["Coefficients"], myDF3["Standard Errors"], myDF3["t values"], myDF3["Probabilities"] = [params, sd_b, ts_b,
p_values]
print(myDF3)
```

This code will output the following:

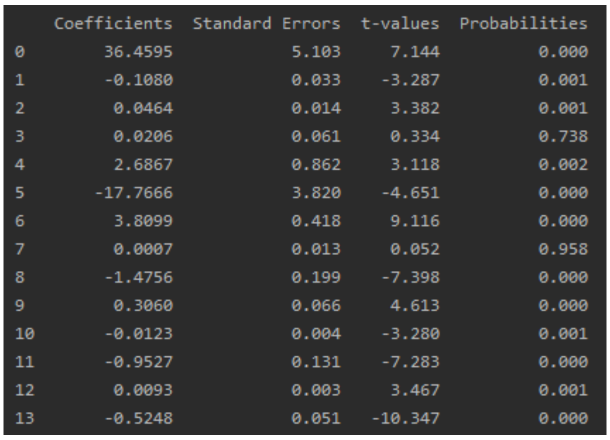


```
   Coefficients  Standard Errors  t values  Probabilities
0       34.5538            0.563    61.415            0.0
1       -0.9500            0.039   -24.528            0.0
```

Figur 2.5: Properties of regression line

2.3.2 Lab 3.6.3

Now we want to fit multiple predictor variables using the multiple linear regression method. This is also done using Sklearn LinearRegression module. The statistical properties when using all 13 predictor variables can be seen on fig. 2.6



```
    Coefficients  Standard Errors  t-values  Probabilities
0        36.4595            5.103     7.144          0.000
1        -0.1080            0.033    -3.287          0.001
2         0.0464            0.014     3.382          0.001
3         0.0206            0.061     0.334          0.738
4         2.6867            0.862     3.118          0.002
5       -17.7666            3.820    -4.651          0.000
6         3.8099            0.418     9.116          0.000
7         0.0007            0.013     0.052          0.958
8        -1.4756            0.199    -7.398          0.000
9         0.3060            0.066     4.613          0.000
10       -0.0123            0.004    -3.280          0.001
11       -0.9527            0.131    -7.283          0.000
12        0.0093            0.003     3.467          0.001
13       -0.5248            0.051   -10.347          0.000
```

Figur 2.6: Properties of multiple regression line

Kapitel 3

CLASSIFICATION

Classification refers to the practice of predicting qualitative responses. While there are many different classification techniques (also known as classifiers), In this chapter we will touch upon two of the most widely-used ones.

3.1 Logistic regression

Linear regression can be a very useful tool in field of supervised learning, however it requires assumptions that the responds variable Y is quantitative, which often isn't the case. In many contexts, like when discussing eye color, the variable is qualitative, taking on values such blue, green, or brown.

3.1.1 Linear vs. Logistic regression

Why is linearly regression ill suited for cases involving a qualitative responds?

For an example of this, let us consider the case that we are trying to predict which disease a hospital patient has, based on his symptoms. One (simplified) way of encoding this could be

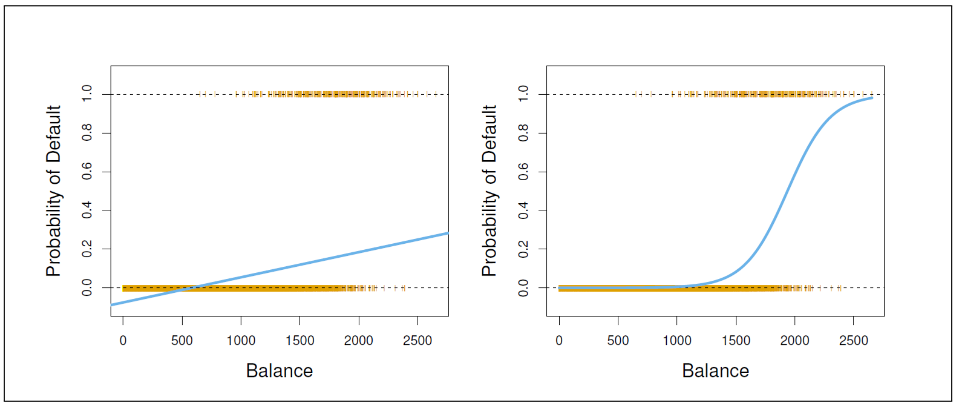
$$\{Rabies,\ Flu,\ Common\ cold,\ Ebola\}$$

However doing this, implies an ordering of the conditions, as well as implying that the the difference between Flu and $Commoncold$ is the same as the difference between $Commoncold$ and $Ebola$.
Furthermore simply changing the ordering of this coding, such as to

$$\{Flu,\ Rabies,\ Ebola,\ Common\ cold\}$$

Implies a completely different relationship between the conditions. Each separate coding would produce different linear models, each leading to a different set of predictions.

Another problem with using Linear regression, cab be seen by applying it to the data set *Default*, the result of which can be seen on fig. 3.1



Figur 3.1: Classification of Default data. Left using linear regression to estimate probability, Right using logistic regression (Source: ISLR book [1] used in DSS course)

Here we see that this model estimates some probabilities to be less than 0, and if we were to attempt to predict for a very large balance, we would get a probability greater then 1, neither of these predictions are sensible. Instead we can see the Logistic regression is a much better fit, offering only values between 0 and 1, as well as simply fitting the data more accurately.

3.1.2 The logistic model

In linear regression, we can use the model shown on eq. (3.1) to represent probabilities.

$$p(X) = \beta_0 + \beta_1 X \tag{3.1}$$

However taking this approach to predict the data on defaulting, ie: $Pr(default = Yes|balance)$, simply yields the model illustrated on the left-hand side of fig. 3.1. Here we see the problems previously discussed, namely that predictions can be less than 0 and more than 1.

In order to prevent this problem, another model for $p(X)$ must be used, one that guarantees an output between 0 and 1 for all values of X. There are numerous functions that can do this, in Logistic regression, the **logistic function** is used , which can be seen on eq. (3.2).

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} \tag{3.2}$$

After some rearranging, the following transformation equation is found (see eq. (3.3)). This transformation is also called the *log odds* or *logit* transformation of $p(X)$.

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X \tag{3.3}$$

We now have a linear model, that is able to model probabilities on a non-linear scale.
The next step is to estimate the coefficients β_0 and β_1, this is done based on available training data.

In order to do this, we use maximum likelihood. This likelihood gives us the probability of observed zeros and ones in our data. We then pick β_0 and β_1 to maximize the likelihood of the observed data. The formula for this *likelihood function* can be seen on fig. 3.2

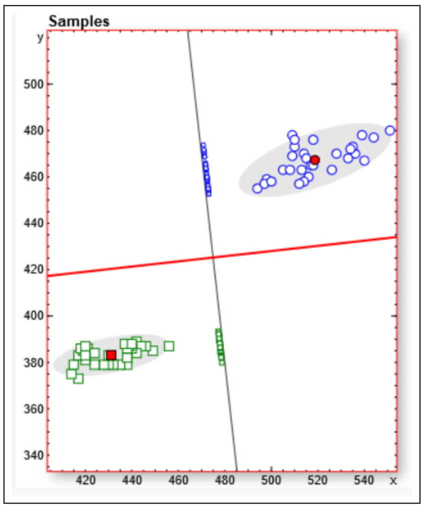
$$\boxed{\ell(\beta_0, \beta) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i)).}$$

Figur 3.2: Likelihood function

Once the coefficients have been estimated, it is a simple to calculate the probability of *default* for any given credit card balance.

3.2 Linear discriminant analysis

Linear Discriminant analysis is like PCA, but It focuses on maximizing the separability among known categories. This means that LDA makes a line that projects the samples onto it in such a way that it maximizes the separation of these two categories. LDA is used when classes are well separated, and it is also popular when dealing with more than two classes.



Figur 3.3: LDA illustration

On fig. 3.3 this concept is clearly illustrated and it is easy to see that the samples are well separated onto the (almost) vertical line.

This is done by maximizing the distance between each mean vector of each class and minimizing the variance of each class. On the picture above the red dots are the mean vectors. This is not limited to 2-dimensionel problems and work very well on many-dimensional problems and separates the classes well.

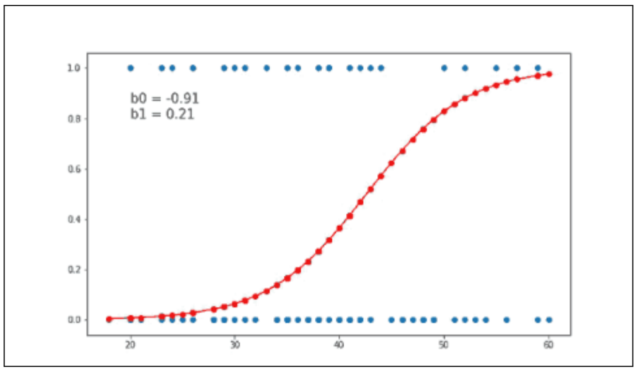
Kapitel 4

CROSS VALIDATION / BOOTSTRAP

When we are working with different machine learning models, we need two different data sets. The first set is often referred to as the training data set, and is used to create the model and define the curve seen in 4.1. The other dataset is referred to as the test set and is used to check if the model performs well. The data between the two sets can not be identical because we want to evaluate how the model performs on new data sets.

Because we often dont have well defined test sets available, we need some methods to gain test data. Here we have several diffenret options. Some methods will not provide new data, but instead use the training data to estimate the test error (the performance). Another approach, which is the one discussed here, is a method where you hold out some part of the data, in the training process and use the hold out data as a test data set.

When this is done with several different splits (different hold out sets) and we evaluate our model using the approach, we are achieving cross validation. The size of the hold our set can vary. If we choose the heaviest computational split, we only contains one sample in the hold out set. This is called leave-one-out-cross-validation LOOCV.



Figur 4.1: Logistic regression model illustration

When we have gained several different splits from using cross validation, we need to evaluate how our model performs in these different testing scenarios. We often use mean least square to evaluate our model. The formula for calculating this for our model with cross validation can be seen in 4.1.

$$CV_{(k)} = \sum_{k=1}^{K} \frac{n_k}{n} MSE_k \tag{4.1}$$

The need for cross validation comes from the fact that we easily can achieve overfitting if we

Kapitel 5

SUBSET SELECTION

This chapter will address the subject Subset selection. Subset selection concerns with selecting or shrinking the coefficients of features to make the model more interpretable and in some cases to predict better.

Linear models are simple and can be interpreted because it usually has a small number of coefficients. In cases where the number of predictors is bigger than the number of samples, we can't use the full least squares, because the solutions is not even defined. In such cases we must reduce the number of features to be able to obtain a solution. It is also very important to not fit your data too hard which regularize, or selection of features also helps with. Along the same lines, when we have a small number of features the model becomes more interpretable.

There exist many different methods to perform subset selection. One of the methods to select the most important features regarding a specific response is called best subset selection. This is where we identify a subset of the predictors that is most related to the response.

5.1 Best subset selection

Best subset selection algorithm is a simple algorithm used to understand which predictors are mostly linked to the responds. To perform best subset selection, we fit a separate least squares regression for each possible combination. It starts out by fitting all p models that contain exactly one predictor, then fitting all p models that contain exactly two predictors and so forth. The number of models to fit can be calculated like this:

$$(\frac{p}{k}) = \frac{p!}{k!(p-k)!} \tag{5.1}$$

Where p is the number of all predictors, and k is the number of predictors in the subset. After all possible combinations of p models has been fitted, we then look at the resulting model, with the goal of identifying the best one.
Best subset selection algorithm can be divided into three steps:

1. Let M_0 denote the *null model*, which contain no predictors. This model simply predicts the sample mean for each observation.

2. For k=1,2,...p:

   (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.

   (b) Pick the best among these $\binom{p}{k}$ models, and call it M_k. Here *best* is defined as having the smallest RSS, or equivalently largest R^2

3. Select a single best model from among M_0,...,M_p using crass-validated prediction error, AIC, BIC or adjusted R^2

The task of to selecting the best subset model, must be performed with care, because RSS decreases monotonically and the R^2 increases monotonically, as the number of features included in the models increases. So, if we use these statistics, we will always end up med the model involving all the variables. Another problem with a low RSS or a high R^2 is that it only indicates a model with a low training error. And a low training error does not equal a low test error. So, for those reasons we need to use cross-validation, AIC, BIC or adjusted R^2.

5.2 Stepwise Selection

If the number of predictors p gets too large, best subset selection algorithm cannot be applied for computational reasons. Another reason why best subset selection is not always the best algorithm to select at subset, is for statistical reasons if the p is too large. Then there is a higher chance of finding models that look good on the training data, even though they might not have any predictive power on future data. For these reasons, stepwise methods, which explore a far more restricted set of models, are attractive alternatives to best subset selection.

In this section we'll touch upon two different stepwise selection methods; the forward stepwise selection method and the backwards stepwise selection method.

5.2.1 Forward stepwise selection

Forward stepwise selection method is very similar to the best subset selection method. Like the best subset selection method, the forward stepwise selection also begins with a model containing no predictors, and adds one predictor at a time until all predictors are in the model. But unlike the best subset selection method the forward stepwise selection doesn't look at all possible models that contains k predictors at each step. Instead, we are just looking at the models that contain the k-1 predictors that already were chosen in the previous step, plus one more. This means that at the k-th step, we are looking at a much more restricted set of models compared to the best subset selection method.
Forward stepwise selection method can be divided into three steps:

1. Let M_0 denote the *null model*, which contain no predictors.

2. For k=0,...,p-1:

   (a) Consider all p-k models that augment the predictors in M_k with one additional predictor.

   (b) Chose the best among these p-k models, and call it $M_k + 1$. Here *best* is defined as having the smallest RSS, or highest R^2

3. Select a single best model from among M_0,...,M_p using crass-validated prediction error, AIC, BIC or adjusted R^2

Compared to best subset selection forward stepwise selection has a computational advantage, as we consider 2^p models in best subset selection and only p^2 models in forward stepwise selection method. This relates to the fact that forward stepwise selection is not guaranteed to find the best possible model out of all 2^p models containing subset of p predictors.

5.2.2 Backward stepwise selection

Backward stepwise selection is exactly the opposite of forward stepwise selection. In contrast backward stepwise selection begins with the full least squares model containing all p predictors, and then removes

the least useful predictor, one at a time.
Forward stepwise selection method can be divided into three steps:

1. Let M_p denote the *full model*, which contain all p predictors.

2. For k=p,p-1,...,1:

   (a) Consider all k models that contain all but one of the predictors in M_k, for a total of k-1 predictors.

   (b) Chose the best among these k models, and call it $M_k - 11$. Here *best* is defined as having the smallest RSS, or highest R^2

3. Select a single best model from among M_0,...,M_p using crass-validated prediction error, AIC, BIC or adjusted R^2

Just like forward stepwise selection, backwards stepwise selection is not guaranteed to gives us the best model containing a particular subset of p predictors.

The major difference between forward and backwards stepwise selection is that the number of samples needs to be larger than the number of variables to perform the backward stepwise selection method (so its possible to fir a least squares model). This is not case for forward stepwise selection, it can be applied when n<p and p>n.

5.3 Choosing the optimal model

As already explained RSS and R^2 are not suitable for selecting the best model among a collection of models, because these quantities are related to the training error and not the test error. So, in order to select the best model with respect to the test-error there are two common approaches:

1. Indirectly estimate the test error by making an adjustment to the training error to account for the bias due to overfitting.

2. Directly estimate the test error by either using a validation set approach or a cross-validation approach.

In this section we'll only talk about AIC, BIC and adjusted R^2 which all are indirectly estimate of the test error.

AIC

AIC stand for Akaike information criterion and deals with the trade-off between goodness of fit of the model and simplicity of the model or it deals with the risk of overfitting and underfitting.

$$AIC = \frac{1}{n\hat{\sigma}^2}(RSS + 2d\hat{\sigma}^2) \tag{5.2}$$

The best model according to the AIC is where the AIC is smallest.

BIC

BIC stands for Bayesian information criterion. This is closely related to the AIC. Both the BIC and the AIC introduces a penalty term for number of parameters in the model to resolve the problem of overfitting. The penalty term is larger in the BIC than in the AIC.

$$BIC = \frac{1}{n}(RSS + log(n)d\hat{\sigma}^2) \tag{5.3}$$

The best model according to the BIC is where the BIC is smallest.

Adjusted R^2

Another very popular approach for selecting among a set of models that contain a different number of variables. R^2 is defined as 1-RSS/TSS, where TSS is the total sum of squares for the response. But as already mentioned R^2 keeps increasing as more variables are added to the model. The adjusted R^2 is calculated as:

$$AdjustedR^2 = 1 - \frac{RSS/(n-d-1)}{TSS/(n-1)} \tag{5.4}$$

Unlike the AIC and BIC, a large value for adjusted R^2 indicates a model with a small test error.

5.4 Lab 6.5.1

In Lab 6.5.1 we have implemented best subset selection and performed it on the “Hitters”-dataset, to predict a baseball players salary based on various statistics. This has been implemented in python and will be discussed briefly in this section, for a more detailed look into the implementation see the source-code in the appendix.

In this section only the heart of the implemented algorithm will be presented. The function getBestModel returns the model with the lowest RSS, based on all combinations of k number of features.

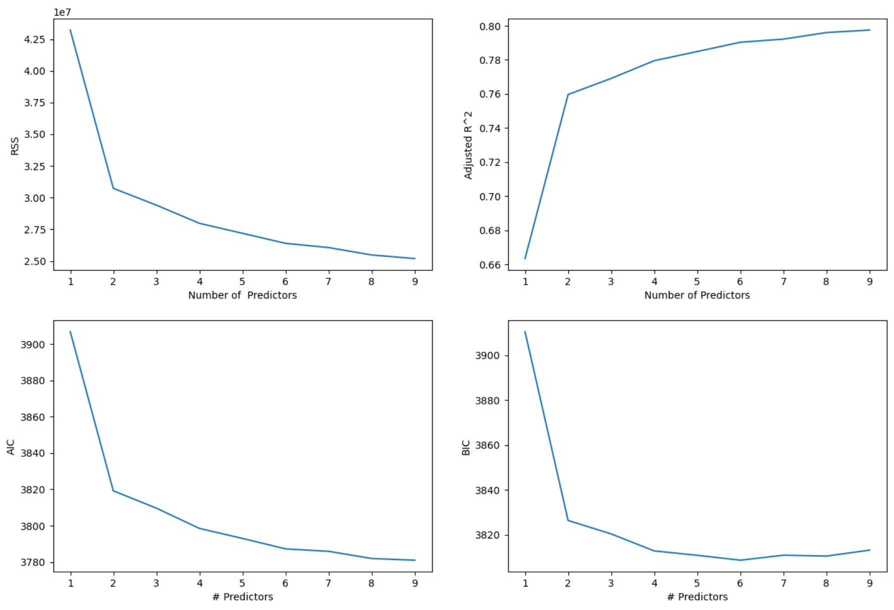
```
def getBestModel(k, X, y):
        results = []
        # Goes through all combinations of k number of features
        for combination in itertools.combinations(X.columns, k):
        results.append(subset_Process(combination, X, y))

        # Stores all combinations i a single dataframe
        models = pd.DataFrame(results)

        #Select the model with the lowest RSS
        best_model = models.loc[models['RSS'].argmin()]

        return best_model
```

The function subset_Process is called by the getBestModel function described above. This function performs linear regression on a model and calculates the RSS the given model.

```
def subset_Process(predictor, X, y):
        # sm.OLS is an estimator that performs linear regression on a model
        temp_model = sm.OLS(y, X[list(predictor)])
        model = temp_model.fit()

        #Then calculate the RSS for the chosen model, and return the model and its RSS together
        RSS = ((model.predict(X[list(predictor)]) - y) ** 2).sum()
        return {"model": model, "RSS": RSS}
```

These two functions are repeated a lot of times to estimate all possible combinations of k predictors.

On fig. 5.1 are the results plotted. Because the number of combinations increases dramatically as k increases so does the computation time. For this reason, k is equal to nine. We can see that according to the BIC, the model with 6 variables performs the best. But according to Adjusted R^2 and AIC a model with more variables than six might be better. Again none of these measures gives us an entirely accurate picture, but they all agree that a model with fewer than five predictors is insufficient.



Figur 5.1: Lab 6.5.1 plotted results

Kapitel 6

SHRINKAGE AND DIMENSION REDUCTION METHODS

Contrary to subset selection methods, which uses least squares to fit a linear model containing a subset with n of all p predictors, shrinkage methods fits a model containing all p predictors. This is done by a penalty that regularizes the coefficient estimates and thereby shrinks them towards zero. By shrinking the coefficient estimates their variance can be significantly reduced. In this section the two shrinkage methods Ridge regression and The Lasso will be covered.
Both shrinking methods tries to control variance by either using a subset of the original variables or by shrinking their coefficients towards zero and uses all of the original predictors. Another class of approaches, dimension reduction, is one that transforms the predictors first and then fits a least squares model using the transformed predictors. The reduction comes from the fact that the methods reduces the problem of estimating $p + 1$ coefficients to estimating $M + 1$ where $M < p$. In this section the two dimension reduction methods Principal Component Regression and Partial Least Squares will be covered.

6.1 Ridge regression

Ridge regression is very similar to least square fitting in that it seeks to minimize RSS, but it adds a second term called the shrinkage penalty.
The equation 6.1 shows the full equation for ridge regression. Here the first term is the RSS where $\beta_0, ..., \beta_p$ is to be estimated such that it is minimized, but the second term introduces a penalty to β_j which effectively shrinks it towards zero. This penalty is scaled with λ such that as it moves towards zero, the penalty moves towards zero and the equation produces the least squares estimates. Moving the tuning parameter towards infinity will in turn drive the coefficient estimates towards zero.
Thus ridge regression will produce a different set of coefficient estimates, $\hat{\beta}^R_\lambda$, for each value of λ hence making it a tuning parameter. It is then critical to choose a good value for λ, which can be done e.g. by the cross-validation method.

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_i j)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 \tag{6.1}$$

The advantage of ridge regression over standard least squares comes from the bias-variance trade-off. By increasing the value of λ and thereby the effect of the penalty term, it is possible to decrease the variance of the predictor however the bias increases. As the test MSE (mean squared error) is a function of the variance plus the squared bias, finding a λ value which decreases the variance more than it increases the bias can result in a lower test MSE. Thus ridge regression will be superior whenever the least squares estimates have high variance.

6.2 The Lasso

The Lasso improves upon a disadvantage of ridge regression, namely that it includes all p predictors in the final model, because it only reduces the magnitudes of the coefficients but never actually excludes any of them. Where ridge regression uses the ℓ_2-norm in its regularization term (equation 6.1) , the Lasso uses ℓ_1-norm (equation 6.2). This has the effect of forcing some of the coefficients to be zero, depending on te value of λ, instead of only driving them towards zero. Therefore the Lasso acts like subset selection, as it effectively performs variable selection yielding a sparse model which exactly is its improvement over ridge regression.
Selecting a good value for λ is critical just as it is for ridge regression and can be done by cross-validation.

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_i j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{6.2}$$

6.3 Principal Component Regression

As with all dimension reduction methods Principal Component Regression (PCR) works in two steps: first transformed predictors are obtained and then a model is fit using the transformed predictors.
The underlying method for obtaining the transformed predictors is in this case PCA (Principal Component Analysis). PCA derives a low-dimensional set of features from a large set of variables by acquiring basis vectors, *principal components*, that forms an orthogonal basis. This is done by finding vectors with values that minimizes the sum of squared perpendicular distances between each point and the vector. Finding the next principal component is done as a linear combination of the variables that is uncorrelated with the principal component before it and has the largest variance. Up to $M <= p$ principal components can be constructed this way, with p being the number of predictors. By this construction the first principal component will contain the most information of the data-set with each following principal component containing less and less information.
Fitting a least square model to $Z_1, ..., Z_M$ principal components instead of $X_1, ..., X_p$ data points with $p << M$, can in theory lead to better results as it can mitigate overfitting. This stems from the notion that most or all of the information in the data relating to the response if contained in the $Z_1, ..., Z_M$ principal component. If however $p == M$ PCR will perform the same as doing least square fitting on all of the original predictors.
In PCR the number of principal components M used for least square fitting becomes a hyper parameter that needs to be chosen carefully. This is typically done using the cross-validation method.

6.4 Partial Least Squares

In PCR the M principal components is guaranteed to best explain the predictors, however they are not guaranteed to best explain the response. This is due to the unsupervised nature of PCR, where the response does not supervise the identification of the transformed features (principal components).
Unlike PCR, Partial Least Squares (PLS) is supervised in the sense that the response is used to identify the transformed features that not only approximates the original predictors but also relates to the response.
PLS first identifies a new set of features $Z_1, ..., Z_M$ that represents $M < p$ linear combinations of the original p predictors. This is done by first standardizing the p predictors. Then the first translated feature Z_1 is calculated as in equation 6.3, with the difference that each θ_{j1} is set equal to the coefficient from the simple linear regression of the response Y onto X_j. In doing this tweaked version PLS places a higher

weight on variables that are strongly related to the response. To compute Z_2 the residuals from regressing each variable on Z_1 are then used following the same calculations as for determining Z_1. Lastly least squares as in equation 11.1 is used to fit a linear model to predict Y using $Z_1, ..., Z_M$ translated feature instead of the original p features just like in PCR.

$$Z_m = \sum_{j=1}^{p} \phi_{jm} X_j \tag{6.3}$$

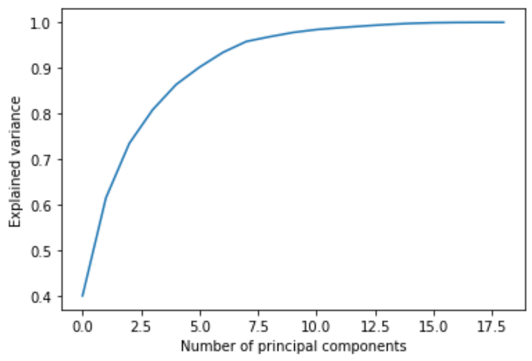
$$y_i = \theta_0 + \sum_{m=1}^{M} \theta_m Z_{im} + \epsilon_i \tag{6.4}$$

6.5 Lab results

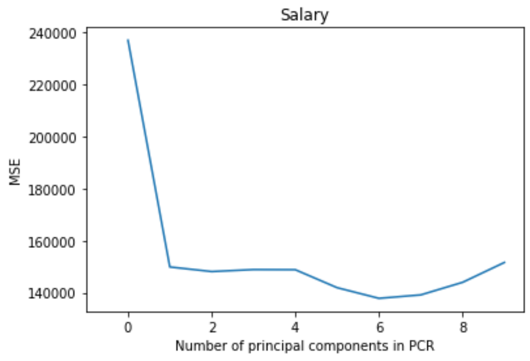
Principal Component Regression(PCR) and Partial Least Squares(PLS) has been applied on the data set ”Hitters”to predict salary as in exercise 6.7.1 and 6.7.2 in the course book. This has been done by finding the number of components that gives the smallest MSE on the training set.
Tenfold cross validation has been used to find the MSE as a function of components as seen on code listing 6.5 showing the implementation for PCR. A similar implementation has been used for PLS. The cross validation has been done for 10 components, as PCA shows an explained variance of over 90% with a flattened curve beyond 10 components, as seen on figure 6.1.
The results for PCR can be seen on figure 6.2, where six principal components gives the lowest MSE. For PLS the lowest MSE value is achieved using only two components. Applying PCR with seven components on the test set gives a MSE of 114098.06, while applying PLS with two components on the test set gives a MSE of 104838.51. Thus for the data set used in this example PLS clearly performs better with both a lower component count and a lower MSE value on the test set.

```
def subset_Process(predictor, X, y):
        # Scale the data
        pca = PCA()
        X_reduced = pca.fit_transform(scale(X_train))
        # Setup folds and linear regression
        mse = []
        regr = linear_model.LinearRegression()
        kf_10 = KFold(n_splits=10, shuffle=True, random_state=1)
        # Single CV to get MSE for the intercept
        score = cross_val_score(regr, np.ones((len(X_reduced),1)), y.ravel(), cv=kf_10, scoring='neg_mean_squared_error').mean()
        mse.append(-score)
        # CV for pc principal components
        for i in np.arange(1, pc):
                score = cross_val_score(regr, X_reduced[:,:i], y, cv=kf_10, scoring='neg_mean_squared_error').mean()
                mse.append(-score)
        # Plot MSE as function of principal components after linear regression
        ....
```

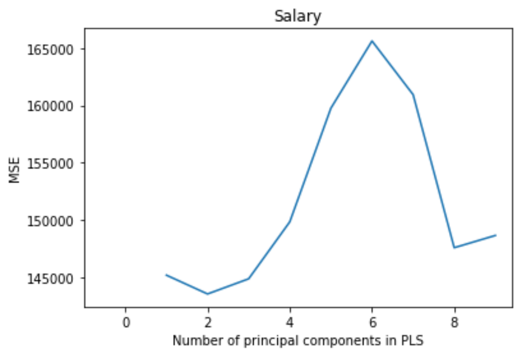
Code Listing 6.1: Code snippet showing tenfold cross validation of PCR



Figur 6.1: PCA on the dataset ”Hitters”



Figur 6.2: MSE as a function of components in PCR



Figur 6.3: MSE as a function of components in PLS

Kapitel 7

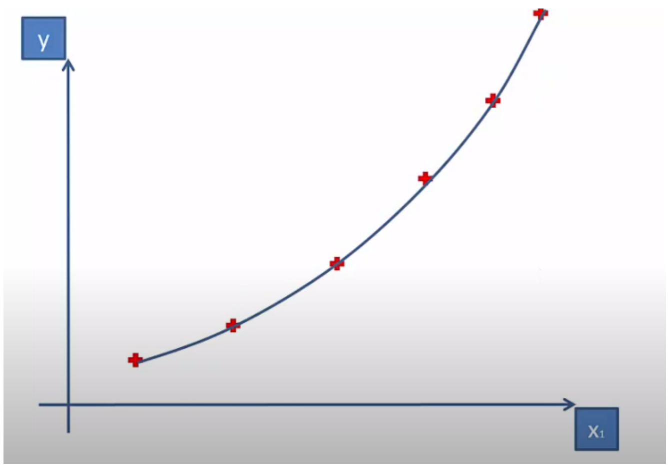
POLYNOMIAL REGRESSION / REGRESSION SPLINES

Polynomial regression is used in situations where our data does not fit into a linear model. A linear model is the simplest one but is often not enough to describe a data series. Other methods can also be used to model data which does not fit into a linear context, such multiple linear regression.

The polynomial regression takes the form:

$$y = \beta_0 + \beta_1 * x_i + \beta_2 * x_i^2 + \beta_3 * x_i^3 + \beta_d * x_i^d \tag{7.1}$$

We see that it is very similar to the multiple linear regression. The difference lies in the x variables which is no longer different in each part, but instead is squared, cubed etc. With this type of function, we can make our model fit more complex datasets as the one seen on fig. 7.1 :



Figur 7.1: Polynomial regression

However, the data is not always as good of a fit, for polynomial regression, as the case seen in the picture above. The data might be presented in a way where the logistic context changes over different segments of the dataset. In this case we can try to model the data using a very complex polynomial, but there is a way of doing this more accurate.

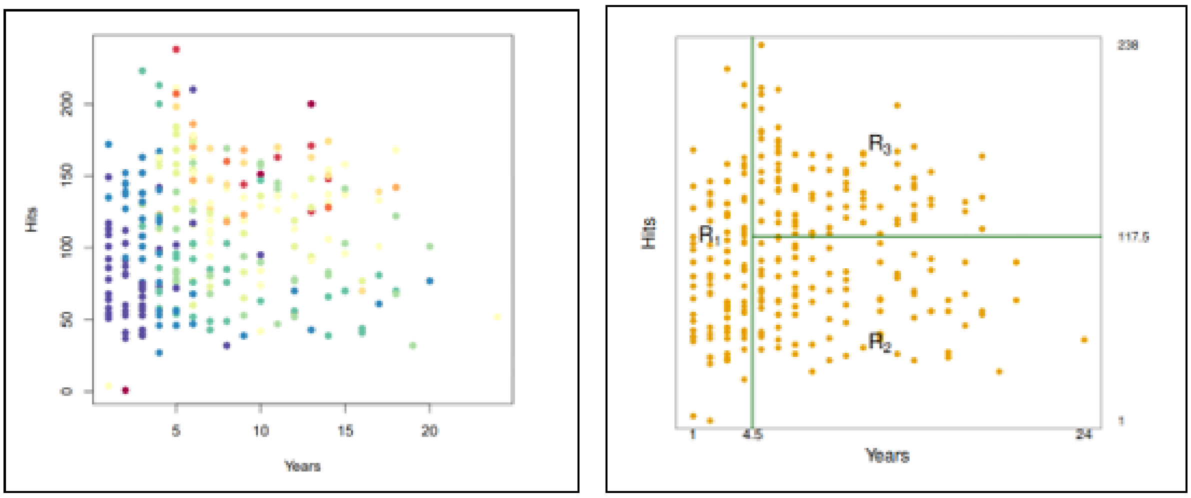
We split our dataset sequence into segments and try to describe the data in each segment using their own polynomial. This way of modeling our data is called splines. It can be done with both linear models and polynomial models. Splines most ensure that the polynomials for each segment are continuous. This makes sure that the complete model of the entire dataset is very smooth.

Kapitel 8

TREE BASED METHODS

Decision trees are a separate method used for both regression and classification problems. The method works by splitting data into a tree like structure, hence the name of the method.

When creating a decision tree, we need to divide our feature space into smaller "boxes" in order to create the tree structure. We use an example for explanation:

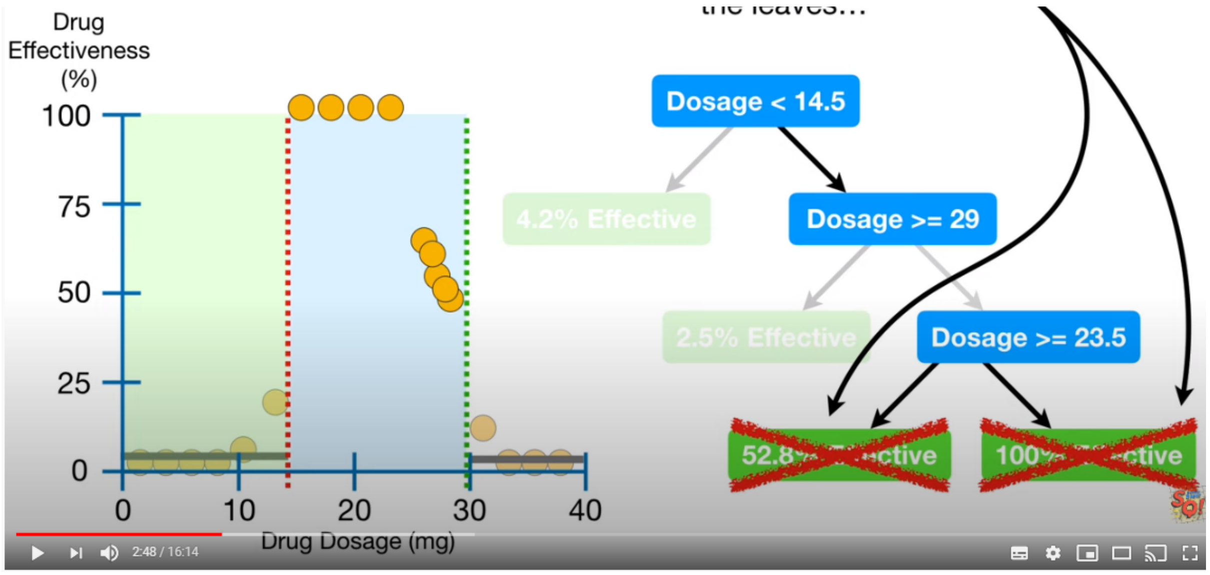


Figur 8.1: Example of tree based method for classification

Here we have baseball salary data, which is depended on years of experience and hits made the previous year. We see from the division that the first internal node comes from 4.5 years of experience. If the player has less than that amount of experience, he will fall into the first segment. If he has more, we have a second division in 117.5 hits. This yields additional two segments as seen from the figure.

In order to create the segments, we use a method called recursive binary splitting. This is done by starting at the top of the tree for splitting the feature space. It is called a greedy approach because we only consider the current split in each decision, and therefore not taking the reminding part of the tree into consideration when making the splits. Predictions can be made by using the average of all training data points within a region as a measure for that region.

When creating the trees, me must ensure not to overfit. If we create a very large tree, the regions will be too biased toward the training data. This can be fixed using pruning. If we see that our tree is overfitted with some test data, we can remove one of the leaves from the tree revealing a larger segment. The average of this segment will now have larger residuals towards the training data but perform better with the test data.



Figur 8.2: Example of pruning a decision tree

But how do we decide which of the leaves to remove, if pruning is necessary? We start by creating all the trees possible by removing one leaf at the time. Then we create a tree score for each tree which is defined by the following formula:

$$Treescore = RSS + \alpha * T \tag{8.1}$$

RSS is the sum of the residuals squared. This will of course be lowest for the most complex tree, so in order to compensate for that, we add a penalty for complexity. Alpha is a constant found by cross validation and T is the number of terminal nodes on the tree.

Decision trees are often not as competitive in terms of prediction errors as other methods. However, we have certain things we can do to improve the quality of the tree. One of them is called bootstrap aggregation or bagging. Bagging is a way of reducing the variance of a learning method. It is logical that if we had several different trainings sets available and were able to aggregate our predictions, based on all these training sets, we would be able to get a lower prediction error. However, these training sets are often not available. Therefore, we instead create bootstrap datasets, which as previously described, are subsets drawn from the original set. As we draw these subsets with replacement, we can create different bootstrap sets and use them to average out the result from the datasets.

Kapitel 9

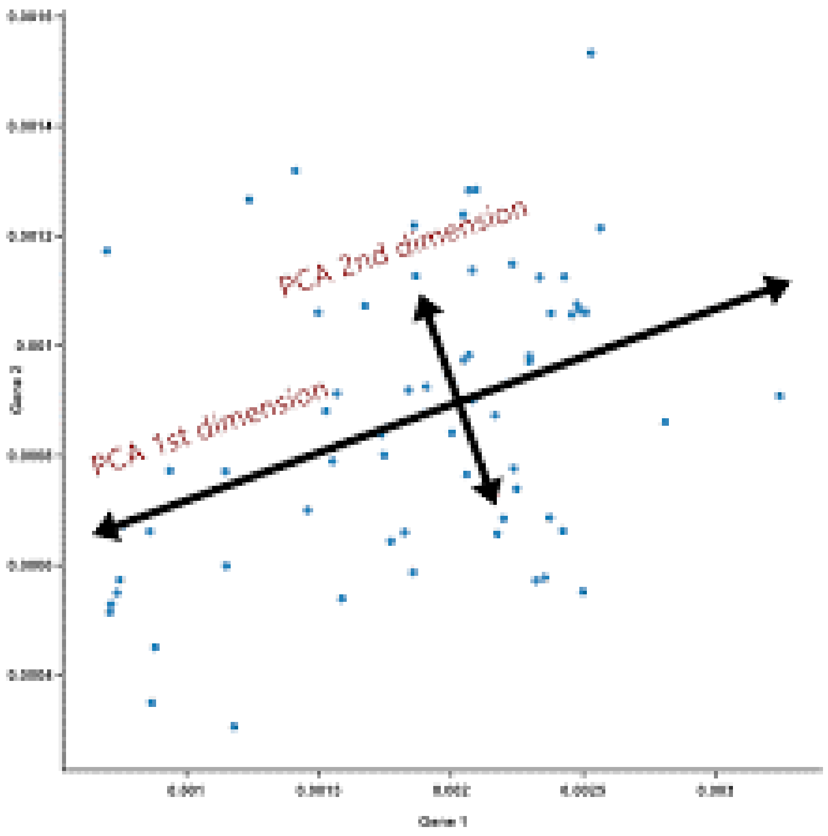
UNSUPERVISED LEARNING AND PRINCIPAL COMPONENTS

When we are talking about unsupervised learning, we are talking about the problem of creating machine learning, without access to labeled data. With unlabeled data we can't make predictions of a response which often is the case with supervised learning. Instead we can try to group data based on features of the dataset at try to make assumptions based on this.

PCA is a way of doing just that, as well as other things. Principal component analysis let us investigate data even with a very high dimensional feature set. Under normal circumstances we would not be able to illustrate the more than two or maybe three features of a data set. So how do we cluster the data points together if can't evaluate more than two or three features. We use Principal component analysis.

PCA works by trying to investigate how much information is stored within each feature about the datapoints. With PCA we create different principal components, based on the number of features. The first principal component would be the linear combination of the features which maximizes the datapoints distance to the origin of the plot. This value could also be the variance. When this linear combination is found we find the next one by determining the linear combination that goes through the origin and is perpendicular to the first one. This is done for every feature in the dataset.

When all the principal components are calculated and maybe illustrated, we can calculate which of the principal components, contributes most to the diversion between the samples. The two components which has the highest contribution is chosen for the 2D PCA plot. We calculate the contribution by looking at the sum of the squared distances from the datapoints to the principal component. The sum of squared distances from a single component, held against the entire dataset would yield how much contribution to the variation in the set that each component has.



Figur 9.1: PCA plot of two principal components

Therefore, when we are choosing two different components for the PCA plot, the first component will have a higher say in the difference between the data points in that direction. From the picture seen above at figure 9.1 we can conclude that the datapoints that are separated in the X direction is more different than the datapoints that are divided in the Y direction. The higher the contribution from both the chosen components are, the better a representation of the dataset, the plot will present.

Kapitel 10

SUPPORT VECTOR MACHINES

In this chapter we'll discuss the subject Support Vector Machines (SVM). SVM performs well in a variety of settings and are often considered one of the best "out of the box "classifiers. In this chapter we'll touch upon two different variants of SVM. First, the simple an intuitive classifier called maximal margin classifier, which the support vector machine is a generalization of. Although the *Maximal Margin Classifier* is elegant and simple, there is a lot of datasets where the classifier can't be applied, because it requires the classes to be separated by a linear boundary. So, we'll also touch upon an extension of the *Maximal Margin Classifier* called the *Support Vector Classifier*, which can be applied to a broader range of datasets.

10.1 Maximal Margin Classifier

Maximal Margin Classifier is the simplest of different type of SVM. It tries to find a plane that separates the classes in the feature space. When we talk about a plane to separate the classes, we talk about a hyperplane.

A hyperplane in p dimensions is a flat affine subspace of dimensions p-1. If we'll look at two dimensions, a hyperplane would be a one-dimensional subspace. If we look at $p > 3$ dimensions, it can be hard to visualize the hyperplane.
A hyper plane has the form in p-dimensional settings [1]:

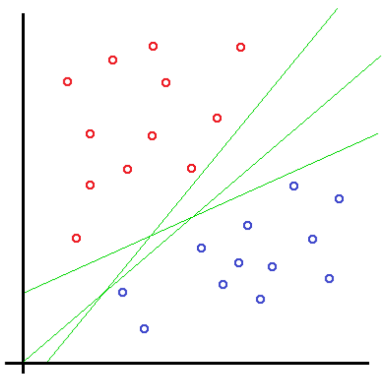
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = 0 \tag{10.1}$$

If a point $X = (X_1, X_2, ..., X_p)^T$ satisfies eq. (10.1) then X lies on the hyperplane. If on the other hand X does not satisfy the eq. (10.1) like:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p < 0 \tag{10.2}$$

It tells us that X lies to one side of the hyperplane, and if its greater than zero, then it lies on the other side of the hyperplane. So, one can easily determine on which side of the hyperplane a point lies. This fact makes it possible to use a hyperplane to classify variables. Suppose we have a n x p matrix X that consists of n training observations in p-dimensional space, and these observations can fall into two classes depending on the outcome of eq. (10.1), which we represent as {-1,1} where -1 represent the one class and 1 represents the other class. Like other classifiers our goal is to develop a classifier that based on the training data can correctly classify the test data.

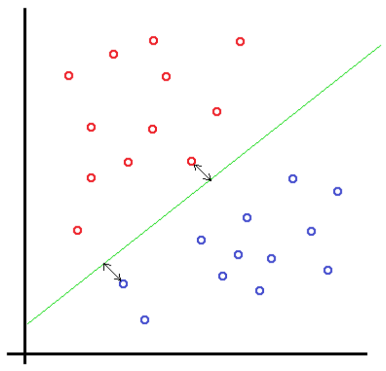
On fig. 10.1 is illustrated three different hyperplanes (the green lines) that separate the training data correctly.



Figur 10.1: Illustrates different hyperplanes to separate the classes

If separating hyperplanes like these exist, it is possible to a very natural classifier, where a test observation is at assigned to a class depending on which side of the hyperplane its located. Then we classify the test observation based on the sign of $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + ... + \beta_p x_p^*$. If its positive then its assigned to class 1, and if its negative, then assigned to class -1. The magnitude of $f(x^*)$ tells us how far form the hyperplane the test variable is, which also tells us how certain we are about a class assignment.

If there exist a hyperplane that perfectly separate out training data, that means there exist an infinite number of hyperplanes that perfectly separates the training data. And how to chose between this infinite number of hyperplanes to use as the classifier? This is where the Maximal Margin Classifier comes into the picture. This is selecting the hyperplane that is the farthest from the training data. This is done by calculating the perpendicular distance from each training observation. The smallest distance from the training observation to the hyperplane is called the margin.
So, the aim is to generate a hyperplane that has the largest margin on the training set, and hop it also has a large margin on the test set. On fig. 10.2 is illustrated the hyperplane using the Maximal Margin Classifier.



Figur 10.2: Hyperplane maximum margin

The maximum hyperplane can be solved as follows [1]:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{Maximize}}\ \text{M}$$

$$subject\,to \sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_i 1 + \beta_2 x_i 2 + ... + \beta_p x_i p) \geq M \,\forall\, i = 1, ..., n$$

10.2 Support Vector Classifier

The maximal margin classifier is a natural way to perform classification, but it has its drawback. As already explained often no separation hyperplane exists, and so there is no maximal margin classifier. This leads us an extension of maximal margin classifier also called Support Vector Classifier. The support vector classifier introduces something called "soft margin". A soft margin, does not seek the largest possible margin so every observation is on the correct side of the margin and hyperplane. Instead it allows some observations to be on the incorrect side of the margin, or even incorrect side of the hyperplane. This is also very useful when you have a lot of noisy data, because then one observation can change the separating hyperplane drastically and often not for the better. This actually has to be the case when there doesn't exist a hyperplane that perfectly separates the observations.

The Support Vector Classifier classifies a test observation like the Maximum Margin Classifier, on which side of the hyperplane it lies. But instead, of perfectly separate all training data, the Support Vector Classifier classifies most of the training data into the correct classes and may misclassify a few of the training data observations. The solution to this problem can be written as follows:

$$\underset{\beta_0, \beta_1, ..., \beta_p, \epsilon_1, ..., \epsilon_n}{\text{Maximize M}}$$

$$subject\,to \sum_{j=1}^{p} \beta_j^2 = 1$$

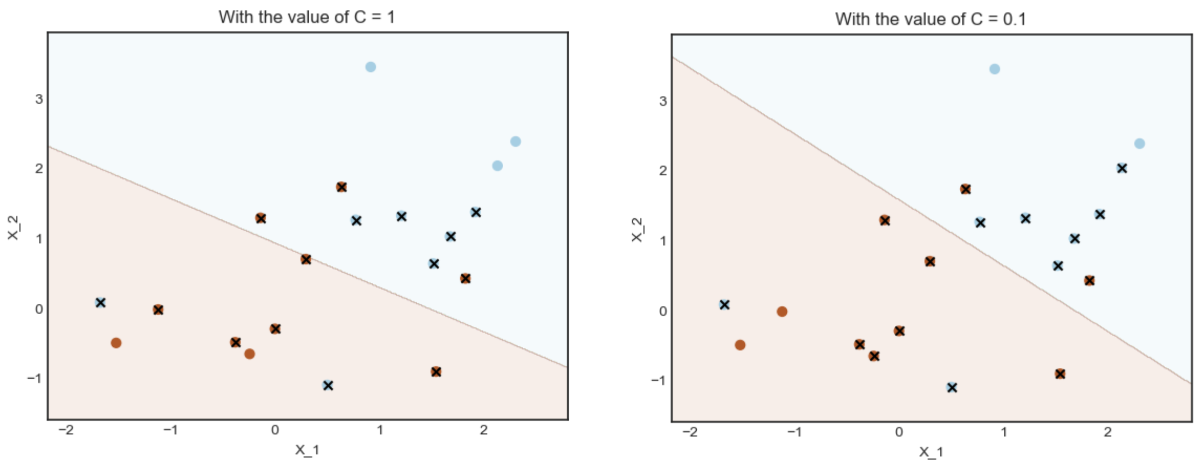
$$y_i(\beta_0 + \beta_1 x_i 1 + \beta_2 x_i 2 + ... + \beta_p x_i p) \geq M(1 - \epsilon_i)$$

$$\epsilon_i \geq, \sum_{i=1}^{n} \epsilon_i \leq C$$

Where C is a nonnegative tuning parameter. As before M is the width of the margin. The new parameter ϵ are a slack variable, that allow individual observations to be on the wrong side of the hyperplane. We still classify test observations based on the sign of $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + ... + \beta_p x_p^*$.
The slack variable ϵ_i tells where the ith observation is located relative to the hyperplane and margin. If $\epsilon_i = 0$ then the observation is on the correct side of the margin. If $\epsilon_i > 0$ then its on the wrong side of the margin, and if $\epsilon_i > 1$ then its on the wrong side of the hyperplane.

C is treated as a tuning parameter, that controls the bias-variance trade-off. When C is small, we seek narrow margins that are rarely violated, which may have low bias but high variance. If C is large, the margin is wider and allow more violations; this amount to fitting the data less hard which potentially gives more bias but may have lower variance.

An example of different C values can be seen from Lab 9.6.1 (The source code can be seen in the appendix). In this Lab exercise 20 random training values has been generated and 10 of them was assigned with the value 1 and 10 with the value -1. We then applied the support vector classifier on the data with the C value of 1 and 0.1, and then plotted the results. On fig. 10.3 it is clear that the hyperplane and the margin changes with the value of C. As explained the smaller the C value is the wider is the margin.



Figur 10.3: Lab 9.6.1 result of different C values

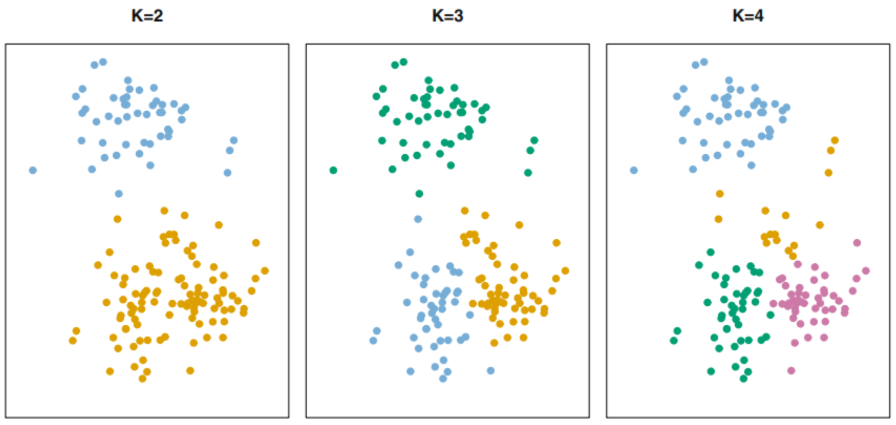
Kapitel 11

CLUSTERING METHODS

Clustering refers to grouping data into different clusters or classes. We do this to get observations or samples clustered into groups where they share some similarities. This is used extensively in the field of machine learning. An example of clusters or classes could be handwritten numbers (0-9). Each number has distinct features which can be divided into clusters. An unsupervised problem would be to divide the numbers into subclasses. In a supervised problem we would already have made the clustering and then divide new samples into the already known clusters, or in other words, predict some outcome based on the sample.

11.1 K-means clustering

K-means clustering is a simple method of clustering data. It works by having a number (K) mean-vectors and assigning a sample to the mean-vector that it's closest to. On fig. 11.1 150 sample have been generated and three different values for K. It is easy to see that the K-means method tries to cluster samples that are close to each other in the same class.



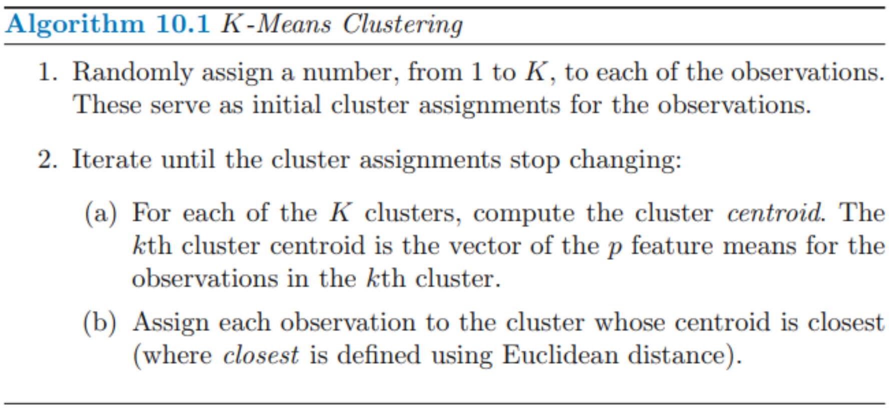
Figur 11.1: K-means clustering example

This works by minimizing the squared euclidean distance to each mean-vector.

$$minimize_{C1 \ldots C_K}\{\sum_{k=1}^{K}\frac{1}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{p}(x_{ij}-x_{i'j})^2\} \tag{11.1}$$

$|C_k|$ denotes the number of samples in the kth cluster. Or the within variation of the kth cluster which is the sum of all the pairwise squared euclidean distances between the samples in the kth cluster, divided by the total number of samples in kth cluster.

To implement this as an algorithm the book [1] covers how this should be done.

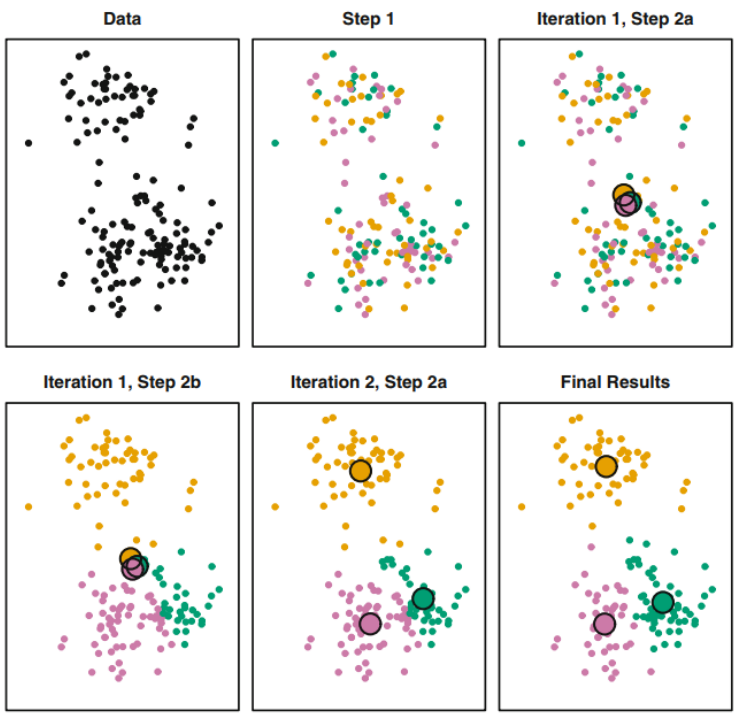


Algorithm 10.1 *K-Means Clustering*

1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.
2. Iterate until the cluster assignments stop changing:
   (a) For each of the K clusters, compute the cluster *centroid*. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.
   (b) Assign each observation to the cluster whose centroid is closest (where *closest* is defined using Euclidean distance).

Figur 11.2: K-means clustering pseudo code

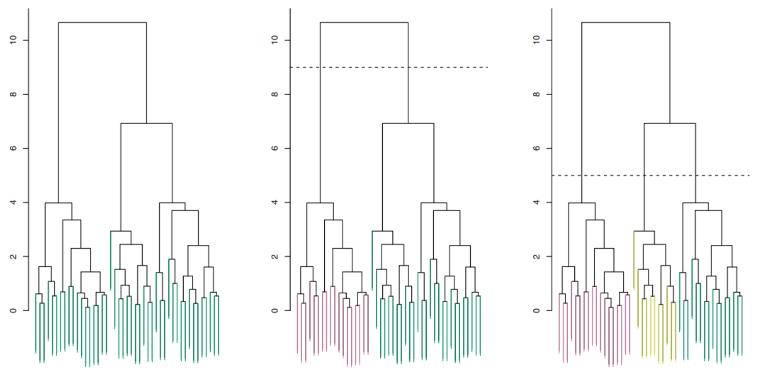
In action this algorithm performs well and can be seen on **??**



Figur 11.3: K-means clustering in action

11.1.1 Hierarchical Clustering

Hierarchical Clustering makes use of tree-based representation of the samples. This tree-based representation is called a Dendrogram and looks like an upside-down tree. An example of a dendrogram can be seen on fig. 11.4 [1]



Figur 11.4: Dendrogram example

Its easy to see how the clustering works. It starts from the bottom works its way up. To see if two samples are similar we can compare how far up on the vertical axis two samples fuses. If they fuse at the bottom they are very similar and at to top not very similar.

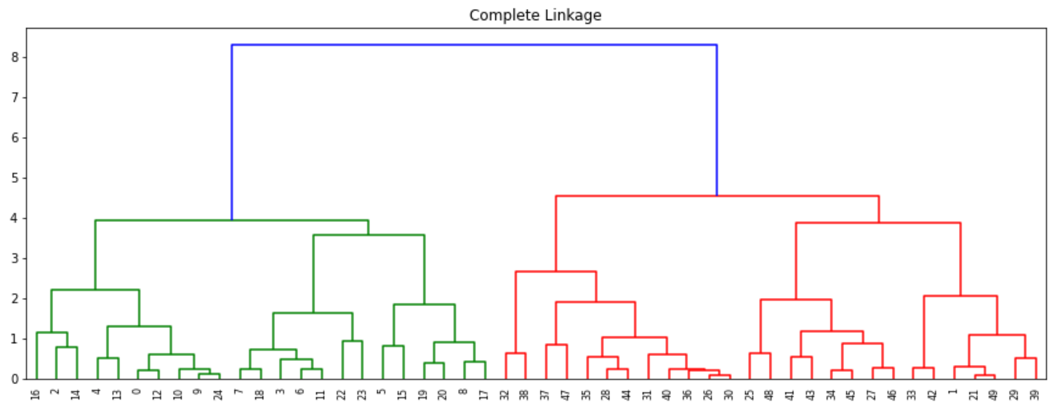
A hierarchical clustering algorithm will work like the dendrogram. That means that if we have n sample, it will also starts at the bottom and look at the two samples that are most similar and fuses them together so we have n-1 clusters. Next it will again fuse the two most similar samples and will be left with n-2 clusters. The algorithm will keep going until all samples belong to a cluster.

11.2 Lab results

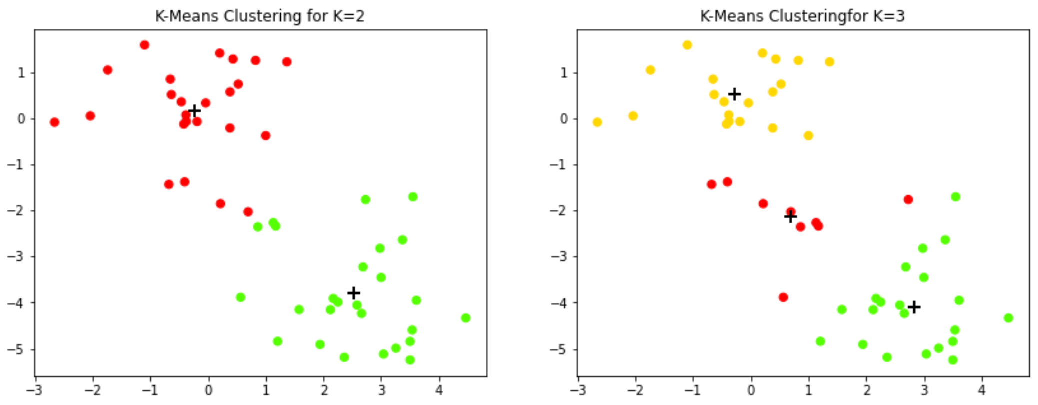
The data for the two lab exercises 10.5.1 and 10.5.2 are generated with the code listed in 11.2. The data is made so that it contains two natural clusters.
Performing KMeans on the generated data with k = 2 and k = 3 both with 20 random initial assignments results in the data clustering on figure 11.6. For k = 2 the data is clearly divided into two clusters (red and green), while for k = 3 it is divided into three clusters (red, green and yellow). Adding additional clusters also results in different cluster means as is evident by the black +, which denotes the given clusters mean.
Figure 11.5 shows the ”complete” hierarchical clustering of the data, which is generated as ”bottom up” clustering by comparing the largest pairwise distance between points in possible clusters. The colors red and green marks the two clusters that is known to be in the data, while the blue color marks the ”single cluster”being all the data.
Using ”single” hierarchical clustering of the data, the graph would instead show clustering by comparing the smallest pairwise distance between points in possible clusters. Lastly also ”average” hierarchical clustering could be used, which of course would use the average distance instead.

```
# Generate data
np.random.seed(2)
X = np.random.standard_normal((50,2))
X[:25,0] = X[:25,0]+3
X[:25,1] = X[:25,1]-4
```

Code Listing 11.1: Code for generating the data



Figur 11.5: Complete histogram of clusters in the data set



Figur 11.6: Clusters generated by KMeans with k = 2 and k = 3

Litteratur

[1] Gareth-James-Daniela-Witten-Trevor-Hastie-Robert-Tibshirani. *An-Introduction-to-Statistical-Learning-with-Applications-in-R*. 2015.