Statistical inference open-sheet

Data Science '17

Regularisation techniques

Why are we doing all of this, anyway?

MLE and OLS try to minimise the *Residual Sum of Squares*, $RSS = \sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2$. However, they do not minimise the *Mean Square Error*, $MSE = \frac{RSS}{n}$. This matters because we can decompose the MSE to show the *bias-variance trade-off*:

$$MSE = bias^2 + variance$$

- What this tells us is that if we want to minimise the Mean Square Error, we have to minimise both bias and variance (or find a combination of both which minimises MSE). The problem with MLE and OLS is that they ensure there is a low bias, but not necessarily a low variance.
- One way of interpreting this is to say that OLS and MLE are very good at predicting the current sample. And for a high sample size $n >> p$, it will also be a good predictor for new data (see that $MSE = \frac{RSS}{n}$), since the estimate is less likely to be influenced by individual observations.
- However, when the number of observations is not much higher than the number of variables, our estimates will have high variance: that is to say, if you were to change our sample for another (or introduce new observations), our estimates would be quite different, and consequently, so would our predictions.
- Another way of looking at it is to say that while MLE and OLS are good at predicting our training set output variables, it will not be good at predicting output variables in a new test set.
- Also, reducing the number of variables can make the model easier to interpret.
- Finally, when $p >> n$, then we simply cannot compute MLE or OLS.

Subset selection

Best subset selection

- Given an output variable, n observations, and p input variables that could be included in your model:
- Let M_0 denote the *null model*, which contains no predictors. This simply predicts the sample mean for each observation
- Fit all the $\binom{p}{k}$ models that contain exactly k predictors.
- Pick the best among these $\binom{p}{k}$ models, call it M_k. Here *best* is defined as having the smallest RSS, or equivalently, the highest R^2.
- Select the single best model using one of the following techniques: cross-validation, C_p, AIC, BIC or adjusted R^2.

Problem: Computationally difficult, since you need to estimate $\binom{p}{k}$ model, and altogether there are 2^p combinations of estimators to include (for example, if the number of variables $p = 10$, then there are 1,000 possible models, and if $p = 20$, there are over a million possible models).

Forward Stepwise Selection

- Given an output variable, n observations, and p input variables that could be included in your model:
- Let M_0 denote the *null model*, which contains no predictors. This simply predicts the sample mean for each observation.

- For $k = 0, 1, ..., p-1$, Consider all $p-k$ models that augment the predictors in M_k with one additional predictors.
- Choose the *best* among those $p-k$ models, and call it M_{k+1}. Here *best* is defined as having the smallest RSS, or equivalently, the highest R^2.
- Select the single best model using one of the following techniques: cross-validation, C_p, AIC, BIC or adjusted R^2.

This is better than the basic *Best Subset Selection* since it doesn't consider all possible combinations of p possible variables: here it considers $1 + \sum_{k=0}^{p-1}(p-k) = 1 + p(p+1)/2$ models. The intuition here is that, at each step, we add the variable which gives the greatest *additional* improvement to the fit of the model. Another advantage to this method is that you can apply it to high-dimensional problems, where $p > n$, since you start with 0 variables and build up your model.

Problem: It does not guarantee that we will pick the best possible model among the 2^p possible models: imagine if we have $p = 3$ possible variables. Imagine if the best 1-variable model should contain X_1, but the best 2-variable model should include X_2 and X_3. Forward stepwise selection will necessarily have X_1 in the 2-variable model, with either X_2 or X_3, and as such it will not be the best model.

Backward Stepwise Selection

- Given an output variable, n observations, and p input variables that could be included in your model:
- Let M_p denote the *full* model, which contains all of the p predictors.
- For $k = p, p-1, ..., 1$, consider all k models that contain all but one of the predictors in M_k, for a total of $k-1$ predictors.
- Choose the *best* among these k models, and call it M_{k-1}. Here *best* is defined as having the smallest RSS (highest R^2).
- Select the single best model using one of the following techniques: cross-validation, C_p, AIC, BIC or adjusted R^2.

Again, this method searches only through $1 + p(p+1)/2$ models, so it can be applied to cases with a large p.

Problem: For the same reason as for *forward stepwise selection*, this method is not guaranteed to select the best model. Another disadvantage is that you cannot apply it to cases where $p > n$, since you start with p variables in your model.

Ridge Regression

Choose the β which minimises:

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = RSS + \lambda\sum_{j=1}^{p}\beta_j^2$$

Note that in the above, we apply the shrinkage penalty to all β parameters *except* β_0, the intercept: this is because β_0 is the mean value of the output variable when all $x_j = 0$. Typically though, you would want to standardise and center the variables around 0 ($\frac{x_{ij}-\bar{x}_j}{\sigma^2}$), so the β_0 disappears.

When centering and standardising our variables, Ridge becomes selecting β which minimises:

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = RSS + \lambda\sum_{j=1}^{p}\beta_j^2$$

For low values of λ, our Ridge regression constrains very little, and at $\lambda = 0$, we just run MLE, where we have low bias, but have higher variance. At λ at very high values, we drive all our β estimates to 0, with higher bias, but low variance. We want to find the λ which minimises MSE (not just RSS or high R^2, as

these measure how well our model fits the training data, but do not guarantee a good fit for our test data, so it may no predict well new observations).

Note that the Ridge regression constraint is circular (or spherical, or hyperspherical in higher dimensions) combination of the various β_js, so the contour function for the RSS is very likely to include a combination of our variables, but the estimated β_js will be small.

Lasso

The lasso is like the Ridge Regression, but corresponds to finding the β which minimises:

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j|$$

Once again, we try to find the λ which minimises MSE.

Note here that the Lasso constraint is a square (polyhedron, polytope in higher dimensions), meaning here we have a linear tradeoff between the β_j estimates imposed by our constraints. This means our RSS contour function is likely to first hit corners of our constraint, where some $\beta_j = 0$. The lasso enforces sparsity.

When to use lasso, when to use ridge?

Basically it depends on the data: if all p variables really matter in predicting the output variable, then getting rid of some variables is probably not a very good idea. In that case, the increase in $bias^2$ will offset the reduction in variance, so our MSE will be higher for Lasso than for Ridge.
However, if only a few variables out of the p possible variables matter in predicting the output variable, getting rid of some variables is a good idea: the increase in $bias^2$ from using Lasso will not be very big, because it will not have missed any important variables, and the variance will be lower, so the MSE for Lasso will be lower than for Ridge.
Also note that when we have many p potential variables to include, the Lasso results are easier to interpret, since we only keep a few variables at the end.

Principal Component Analysis (PCA)

What's the point?

Imagine you have a p variables $X_1, X_2, ..., X_p$, with n observations (as usual). When conducting exploratory data analysis, you could, for instance, plot each variable against each other to try to find interesting patterns. But this is pretty complicated and hard to analyse, as you will have to make $\binom{p}{2}$ plots.

Instead, you turn to *Principal Component Analysis (PCA)*, which is a type of *unsupervised learning* tool which allows for dimensionality reduction. Basically, it tries to capture the most amount of variance across your X variables in just a few dimensions (instead of the full p dimensions). It allows you to find some potentially *interesting* patterns in your data, where *interesting* is measured by the amount by which our observations vary along these few dimensions. We call these dimensions *Principal Components*.

Note that it is not like regression, in that you don't look at the effect of X on Y - there is no Y here.

PCA for (Data Science) dummies

- So, as mentioned above, we have our matrix of variables X which is an $n \times p$ matrix. We make sure all our variables are normalised here.

- A *principal component* is a linear combination of our set of features $X1, X_2, ..., X_p$. For instance, our *first principal component* Z_1 could be written:

$$Z_1 = \alpha_{11}X_1 + \alpha_{21}X_2 + ... + \alpha_{p1}X_p$$

  Where $\alpha_{11}, \alpha_{21}, ..., \alpha p1$ are called *loadings*. Z_1 is a column vector containing elements $z_{11}, z_{21}, ..., z_{n1}$ (so there is one z_{i1} per observation in the data). Each z_{i1} element is called a *score*, and is as follows:

$$z_{i1} = \alpha_{11}x_{i1} + \alpha_{21}x_{i2} + ... + \alpha_{p1}x_{ip}$$

- The *first principal component* is the principal component which has the largest sample variance, subject to the constraint that $\sum_{j=1}^{p} \alpha_{j1}^2 = 1$. In short, it solves the following optimisation problem:
  maximise $\frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{p} \alpha_{j1}x_{ij})^2$ subject to $\sum_{j=1}^{p} \alpha_{j1}^2 = 1$
- This is basically equivalent to finding the values for α_{j1} such that:
  $\alpha_1 = argmax_{\alpha_1}\{var(Z1)\}$ subject to $\sum_{j=1}^{p} \alpha_{j1}^2 = 1$
- Note: it is necessary to add a constraint on our α_{j1} parameters since otherwise we could just increase them to infinity in order to maximise variance.
- Then, the *second principal component* is the Z_2 with the highest variance, subject to the added constraint that Z_2 is orthogonal to Z_1 (so they are uncorrelated).
- So basically Z_1 tries to capture the maximum amount of variance among our X variables. Then Z_2 tries to capture the maximum amount of *additional variance* (i.e which has not already been captured by Z_1). We could carry on with Z_3, Z_4... all the way up to a maximum of Z_p if we wanted (but with p principal components, we don't have dimensionality reduction anymore).

A couple of examples

Example n°1

For example, imagine we only have $p = 2$ potential variables, so we only have X_1 and X_2 variables. Consider this graph which plots X_2 vs X_1, and also shows the first two principal components:

As you can see, the first principal component is the dimension over which there is the highest variance over both variables. Then the second principal component, which is orthogonal to the first, captures the rest of the variance. In this case, all the variance is captured, since we only had $p = 2$ dimensions to start off with anyway.

Example n°2

In this example, we now have $p = 3$ variables. You can see from below that they computed the first 2 principal components, in the dimensions which captured the most amount of variance in the data (left hand-side). On the right, they then the first vs the second principal component, thus reducing dimensionality:

How many principal components to use?

There is no perfect way of determining this. One measure is to look at how much of the *total variance* of our variables is captured by our principal components. The *total variance* in a dataset (assuming all variables are centered at mean 0) is defined as:

$$\sum_{j=1}^{p} Var(X_j) = \sum_{j=1}^{p} \frac{1}{n}\sum_{i=1}^{n} x_{ij}^2$$

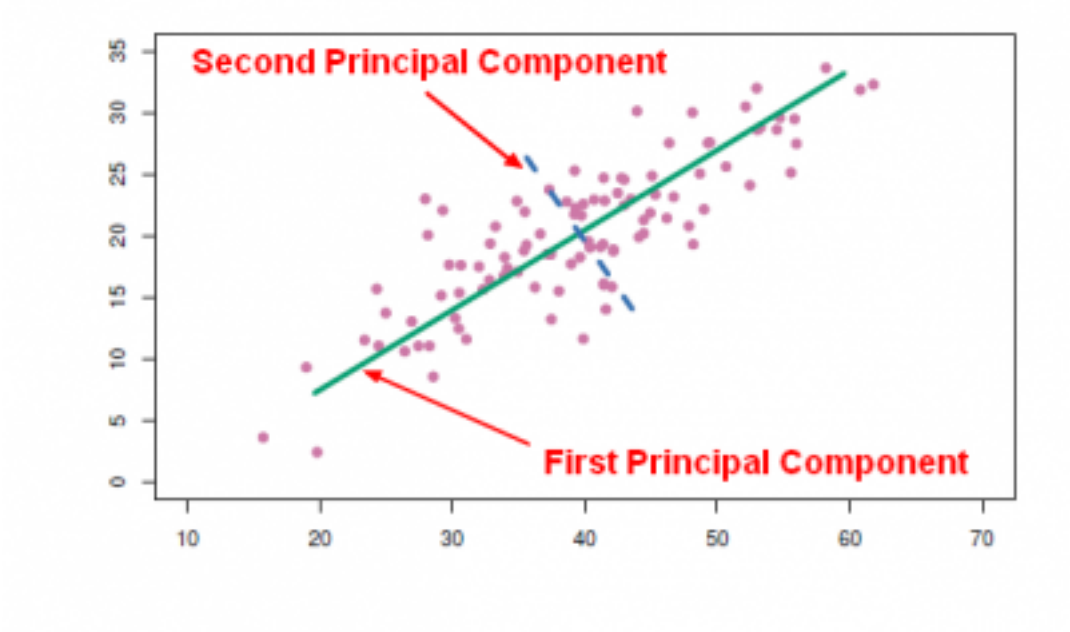


Figure 1: Principal components in two dimensions

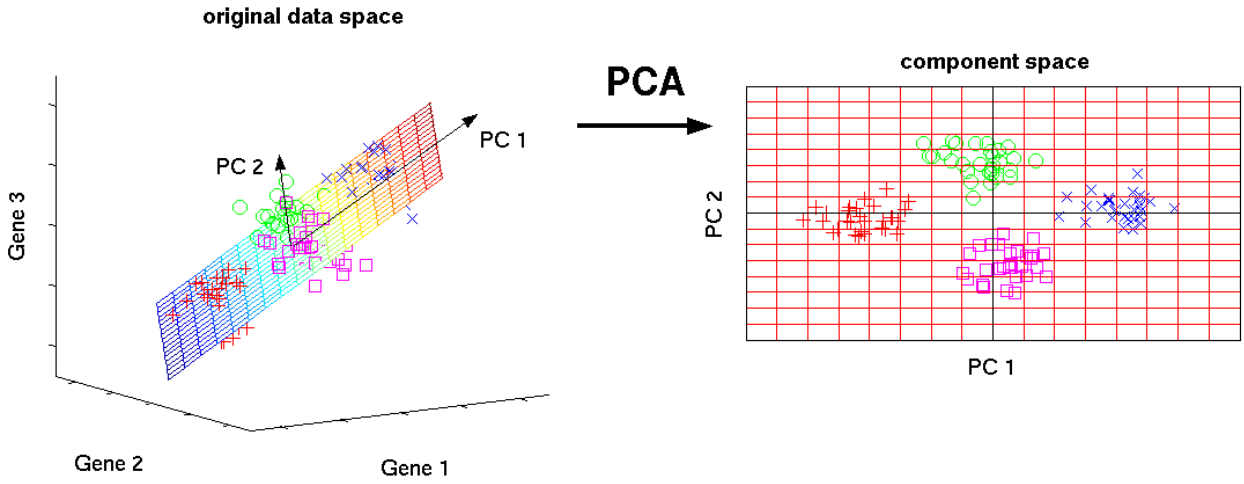


Figure 2: Principal components in 3D reduced to 2D

The variance explained by the m^{th} principal component is:

$$\frac{1}{n}\sum_{i=1}^{n} z_{im}^2 = \frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{p}\alpha_{jm}x_{ij})^2$$

So we can calculate the *proportion of variance explained (PVE)*:

$$PVE = \frac{\sum_{i=1}^{n}(\sum_{j=1}^{p}\alpha_{jm}x_{ij})^2}{\sum_{j=1}^{p}\sum_{i=1}^{n}x_{ij}^2}$$

We can then plot the PVE against the number of principal components (called a *scree plot*) and just decide when we think extra principal components stop capturing much more extra variance.

Why are principal components so cool?

Well, they are very much linked to eigenvectors and SVD. To see why, we reformulate trying to find the first principal component as trying to find the vector $\mathbf{u}_1$ (here $\mathbf{u}_1$ is just α_1 but I change to adapt to Bishop's notation) such that we maximise $\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1$ subject to $\mathbf{u}_1^T\mathbf{u}_1 = 1$, where $\mathbf{S}$ is the data covariance matrix.
We set up our lagrangian:

$$\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 + \lambda_1(1 - \mathbf{u}_1^T\mathbf{u}_1)$$

Which we maximise by differentiating with respect to $\mathbf{u}_1$, and setting equal to 0. We solve to find:

$$\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1$$

From above, we can see that $\mathbf{u}_1$ is an eigenvector of S, with corresponding eigenvector λ_1. We reorganise to find:

$$\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 = \lambda_1$$

So the left-hand-side will be maximised for the highest λ_1, in other words the highest eigenvalue. So we can say that the first principal component is for the eigenvector with the highest associated eigenvalue! Which also means we can calculate principal components using SVD.

Implementation in R

Here we use the USArrests data set which comes with R, which contains data on murders, assaults, and rape, as well as the proportion of urban population, by US State, so it is a 50×4 dataframe.

```
states <- row.names(USArrests)
states
```

```
##  [1] "Alabama"        "Alaska"         "Arizona"        "Arkansas"
##  [5] "California"     "Colorado"       "Connecticut"    "Delaware"
##  [9] "Florida"        "Georgia"        "Hawaii"         "Idaho"
## [13] "Illinois"       "Indiana"        "Iowa"           "Kansas"
## [17] "Kentucky"       "Louisiana"      "Maine"          "Maryland"
## [21] "Massachusetts"  "Michigan"       "Minnesota"      "Mississippi"
## [25] "Missouri"       "Montana"        "Nebraska"       "Nevada"
## [29] "New Hampshire"  "New Jersey"     "New Mexico"     "New York"
## [33] "North Carolina" "North Dakota"   "Ohio"           "Oklahoma"
## [37] "Oregon"         "Pennsylvania"   "Rhode Island"   "South Carolina"
## [41] "South Dakota"   "Tennessee"      "Texas"          "Utah"
## [45] "Vermont"        "Virginia"       "Washington"     "West Virginia"
## [49] "Wisconsin"      "Wyoming"
```

We look at the mean and variance of the data, and note that they vary greatly - partly because the scales are different:

```
apply(USArrests, 2, mean)
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

```
apply(USArrests, 2, var)
```

```
##     Murder    Assault   UrbanPop       Rape
##   18.97047 6945.16571  209.51878   87.72916
```

We now perform PCA on the dataset using *prcomp*, which automatically centers the data to 0, and setting $scale = TRUE$ also sets the standard deviations to 1:

```
pr.out <- prcomp(USArrests, scale = TRUE)
```

The *rotation* matrix provides the principal component *loadings* for each principal component:

```
pr.out$rotation
```

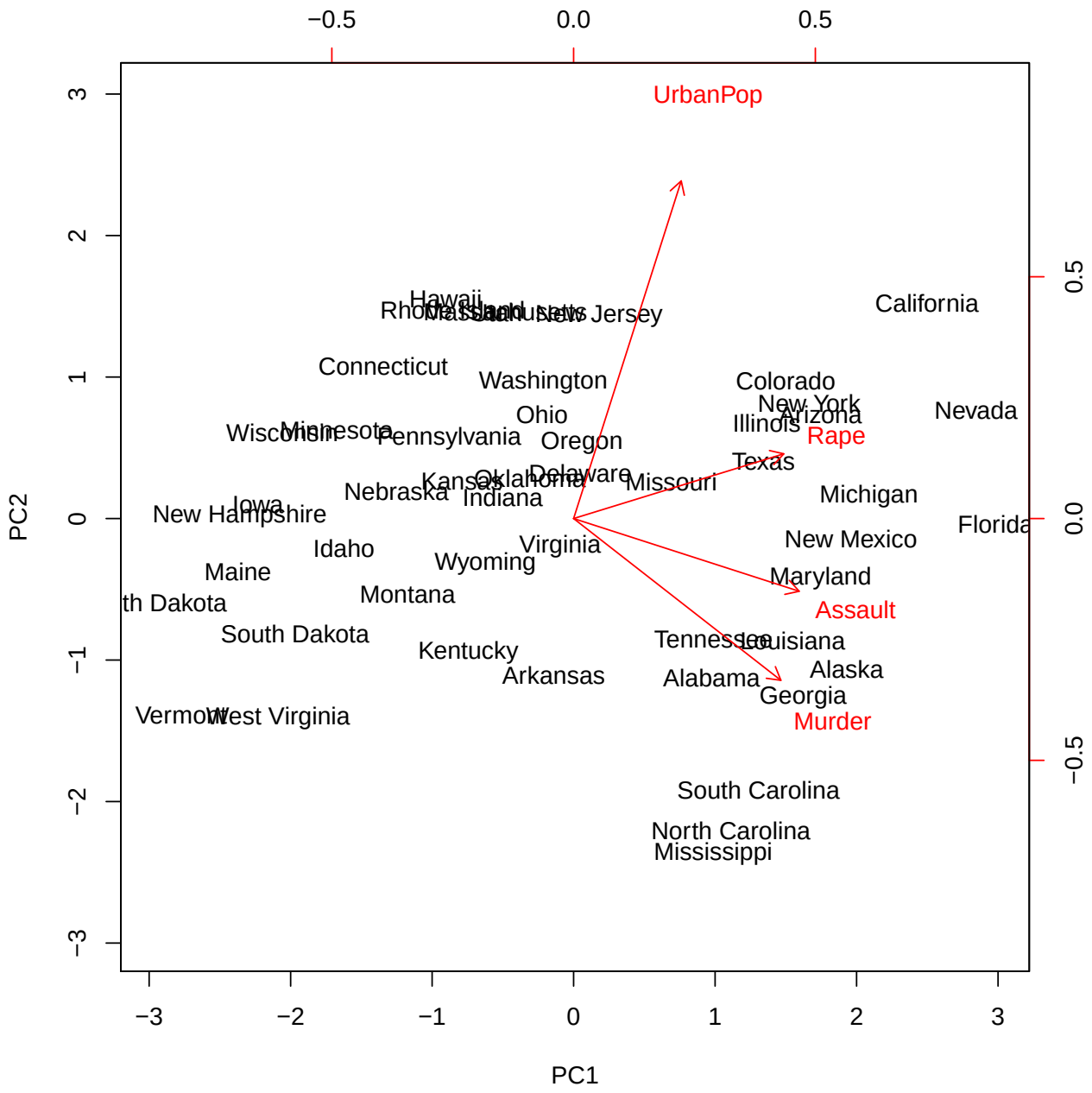
```
##                 PC1        PC2        PC3         PC4
## Murder   -0.5358995  0.4181809 -0.3412327  0.64922780
## Assault  -0.5831836  0.1879856 -0.2681484 -0.74340748
## UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
## Rape     -0.5434321 -0.1673186  0.8177779  0.08902432
```

(for the purposes of the plotting, we change the signs of our score vectors and loadings. This changes nothing to our results, but in the specific case of this example makes interpreting the plot a lot easier):

```
pr.out$rotation <- -pr.out$rotation
pr.out$x <- -pr.out$x
```

We can plot the first two principal components using this simple function:

```
biplot(pr.out, scale = 0)
```



Interpreting the biplot: in the biplot above, we can see that the first principal component has captured largely the variance of the variables rape, assault and murder, giving much less weight to UrbanPop. This first PC can be interpreted as a measure of serious crime.
The second principal component, however, puts a lot of weight on Urban pop and less so on the other variables. So the second PC can be interpreted as a measure of urbanisation.
What does this graph tell us? We can see that the 3 crime variables are pretty closely correlated, while UrbanPop is less correlated with them. The states to the right have higher crime rates, and the states further above are more urbanised.

Next we try to see how many principal components to select. First we calculate the variance of the principal components:

```
pr.var <- pr.out$sdev^2
pr.var
```

```
## [1] 2.4802416 0.9897652 0.3565632 0.1734301
```

We can calculate the proportion of the variance explained by each PC by dividing their variance by the total variance explained by all four PCs, which is simply the sum of their variance:

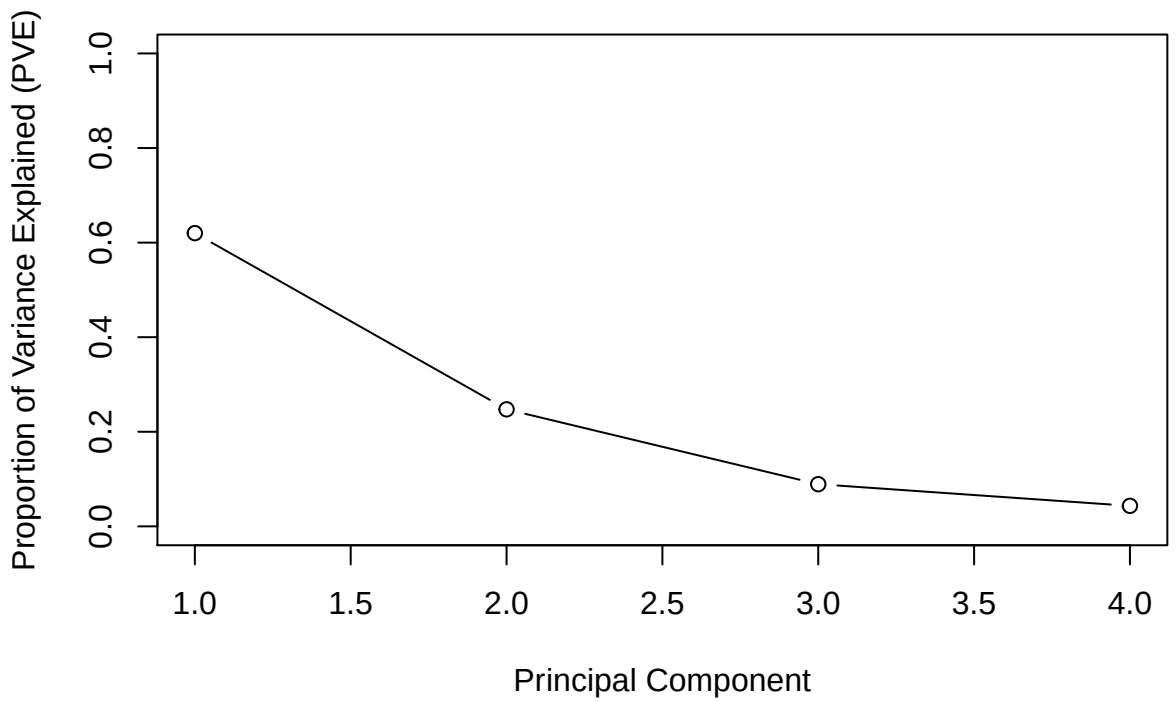
```
pve <- pr.var/sum(pr.var)
pve
```

```
## [1] 0.62006039 0.24744129 0.08914080 0.04335752
```

We see from the above that the first principal component explains 62% of the total variance, the second 25%, etc.

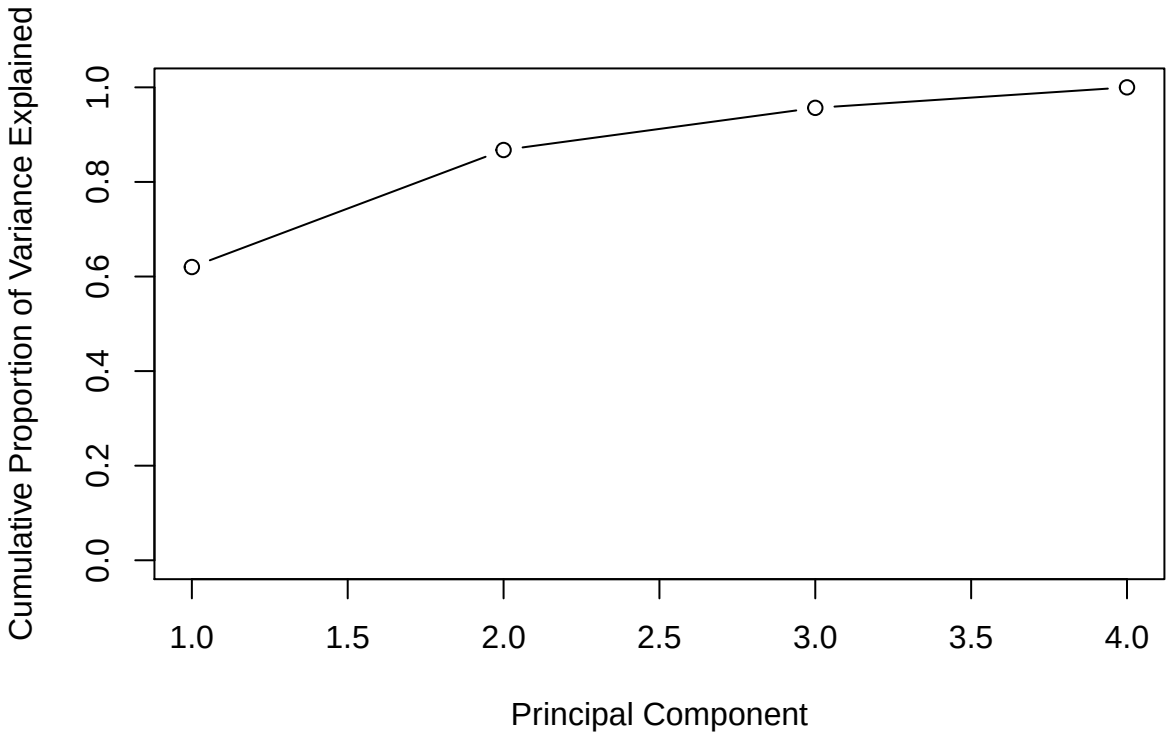
We can make a *scree plot* of the PVE as follows:

```
plot(pve, xlab="Principal Component", ylab="Proportion of Variance Explained (PVE)",
     ylim=c(0,1), type='b')
```



We can also plot the cumulative PVE:

```
plot(cumsum(pve), xlab="Principal Component", ylab="Cumulative Proportion of Variance Explained",
     ylim=c(0,1), type='b')
```



These plots suggest that we could just use 2 principal components, as adding the third and fourth don't explain much more of the variance.

Outliers detection

Leverage

The basic tool to examine the fit are the residuals, which are not independent and sum to 0 if there is the intercept. Their covariance matrix is $var(e) = \sigma^2[I - H]$. The leverage of an observation point i is h_{ii} of the $H = X(X^TX)^{-1}H$ matrix. Since the trace of H is p (the dimension of the model space), we say that h_{ii} is large if it is three times greater than the average. So if $h_{ii} \geq \frac{3(M+1)}{N}$.

Plotting the standarized residuals vs leverage or fitted values coloring the ones with high leverage gives a good insight. In order to do that we need a model that may fit well the data, otherwise we may not see the outliers.

Graphical models

Graphical model: Were graph models and statistics intersect.

In a graphical model, each variables is inside a node (some grouped together). A box with nodes means that the nodes inside are replicated many times.

Directed graphical models (DAG) - (Bayesian Networks): All the edges have a direction.

The double circle means that the variable is a deterministic function of his parents.

The grey colour means what will be observed.

More than 2 variables inside a box implies they have the same probability distribution.

If a variable x is generated by other variable y we put a directed edge y->x.

A graph and its factorization are linked (the graph describes the dependency).

(x)->(y)->(z)
->(w)

p(x,y,z,w)=p(x)p(y|x)p(z|y)p(w|y)

Latent variables

Classification

Supervised classification: is the case when the output t belongs to one of m classes. Then, we want to predict the class given inputs x (using the features $\Phi(x)$).

When we have two classes sometimes we code them as $0/1$ and other times as $-1/1$.

Discriminative models

Model $p(t|x)$ meaning that they will learn the probability of $t = C_i$ given an input x. The idea behind a discriminative model is to find the boundary between classes in the input space, so it can be decide to each classe belongs given an input. The boundaries will be linear on the features space, but since the features can be a non-linear transformation of the inputs x, the boundaries may be non-linear. Usually discriminative models give better performance in classification tasks if we have a lot of data or our class-conditional assumptions give a poor approximation to the true distributions.

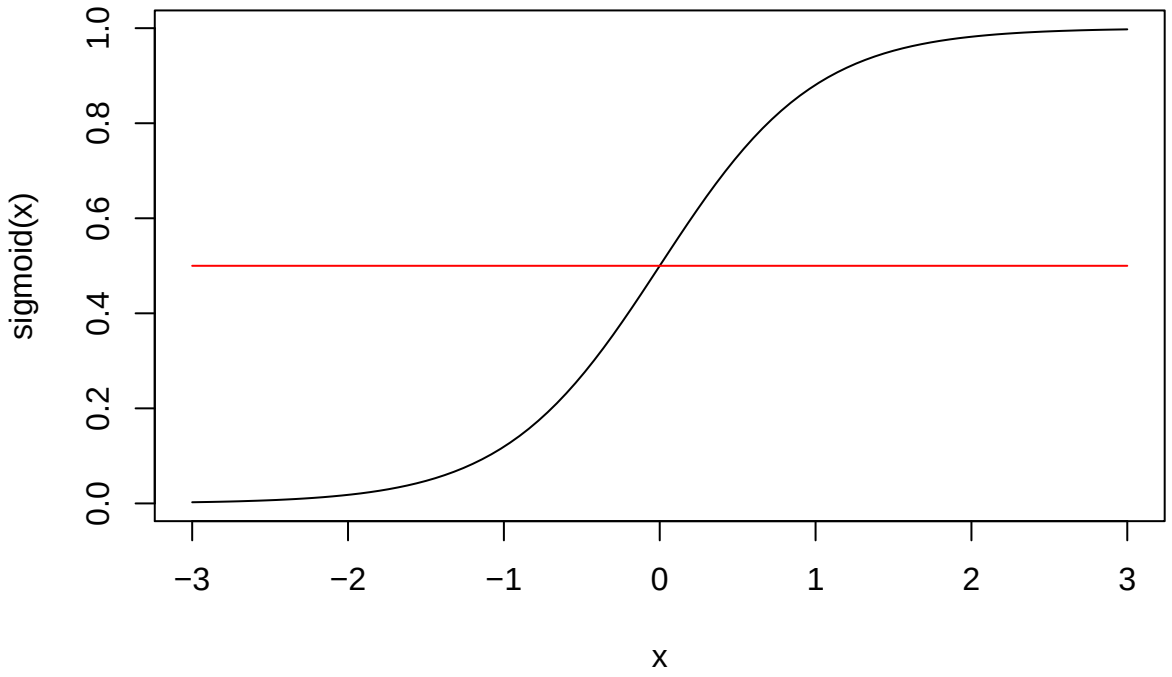
Logistic regression

In logistic regression we supose that the outcome t is either 1 or 0. In order to predict we want to estimate the logit function of the observations $w^T\Phi(x) = \text{logit}(P(t = 1|x)) = \ln \frac{P(t=1|x)}{1-P(t=1|x)}$, where the inverse is the sigmoid function and it is a function that has a result between 0 and 1 $P(t = 1|x) = \frac{e^{\text{logit}((t=1|x))}}{1+e^{\text{logit}((t=1|x))}} = \frac{1}{1+e^{-(w^T\Phi(x)+w_0)}} = \sigma(w^T\Phi(x)+w_0)$ ($P(t = 1|x) = \sigma(w^T\Phi(x)+w_0)$ is the probability, or likelihood, of being in the class 1). The final outcome that we want is a stochastic variable, that will be 1 if the likelihood is greater than 0.5 and 0 if it is less than 0.5. In order to get this final outcome, we need to estimate the coefficients $(w_0, \ldots, w_M)$ that will define a boundary in form of a criteria to decide whether a new input belongs to the class 1 or 0. For a $\sigma(w^T\Phi(x) + w_0)$ this will happen on $w^T\Phi(x) + w_0 = 0$, so the decision boundary will be in $w^T\Phi(x) = -w_0$. Observe that since $w^T\Phi(x)$ is a linear function, the boundary is linear in the space of features, but it could be non-linear in the space of x if the features are non-linear functions of x.

The sigmoid function is a good one to differentiate between two classes:

```
library(ggplot2)
sigmoid = function(x) {
  1 / (1 + exp(-2*x))
}

curve(sigmoid,from=-3,to=3)
curve(0.5+0*x,from=-3,to=3,col="red",add=T)
```



Example:

Imagine we have a spam detector. Then for given inputs like the source of the mail, content of the mail, subject of the mail and previous actions, the computer has to decide whether it is spam or not. The inputs that we have may be continuous or discrete and we treat them as any other regression function ($y(x) = w^T\Phi(x)$). The idea is to find a y_0 such that when $y > y_0$ the probability of being spam is greater than 0.5 and if $y < y_0$ the probability of being spam is less than 0.5 (the sign of the inequalities can be the other way around). So, we would train the model with previous data and find the point y_0 where the likelihood is maximum, let's say we find that this point is 0. Then for $w^T\Phi(x) > 0$ with any new data point we will catalog x as spam.

The likelihood of logistic regression is

$$p(\mathbf{t}|w) = \prod_{n=1}^{N} P(t_n = 1|x)^{t_n}(1 - P(t_n = 1|x))^{1-t_n},$$

the deviance (taken from the log-likelihood) is

$$-2\mathcal{L}(t, w, x) = -2\sum_{n=1}^{N} \left(t_n \ln P(t_n = 1|x) + (1 - t_n)\ln(1 - P(t_n = 1|x))\right).$$

We get rid of the constant because will not be important when minimizing and the gradient is

$$-\nabla\mathcal{L}(t, w, x) = 2\sum_{n=1}^{N}(P(t_n = 1|x) - t_n)\Phi_n = \sum_{n=1}^{N}(\sigma(w^T\Phi_n) - t_n)\Phi_n = \Phi^T(y - t),$$

where y is the vector of $\sigma(w^T\Phi_n)$. This has no closed solution, but since the log-likelihood is concave (and so the deviance convex), we can get to the w that maximizes the log-likelihood using gradient descend. In which

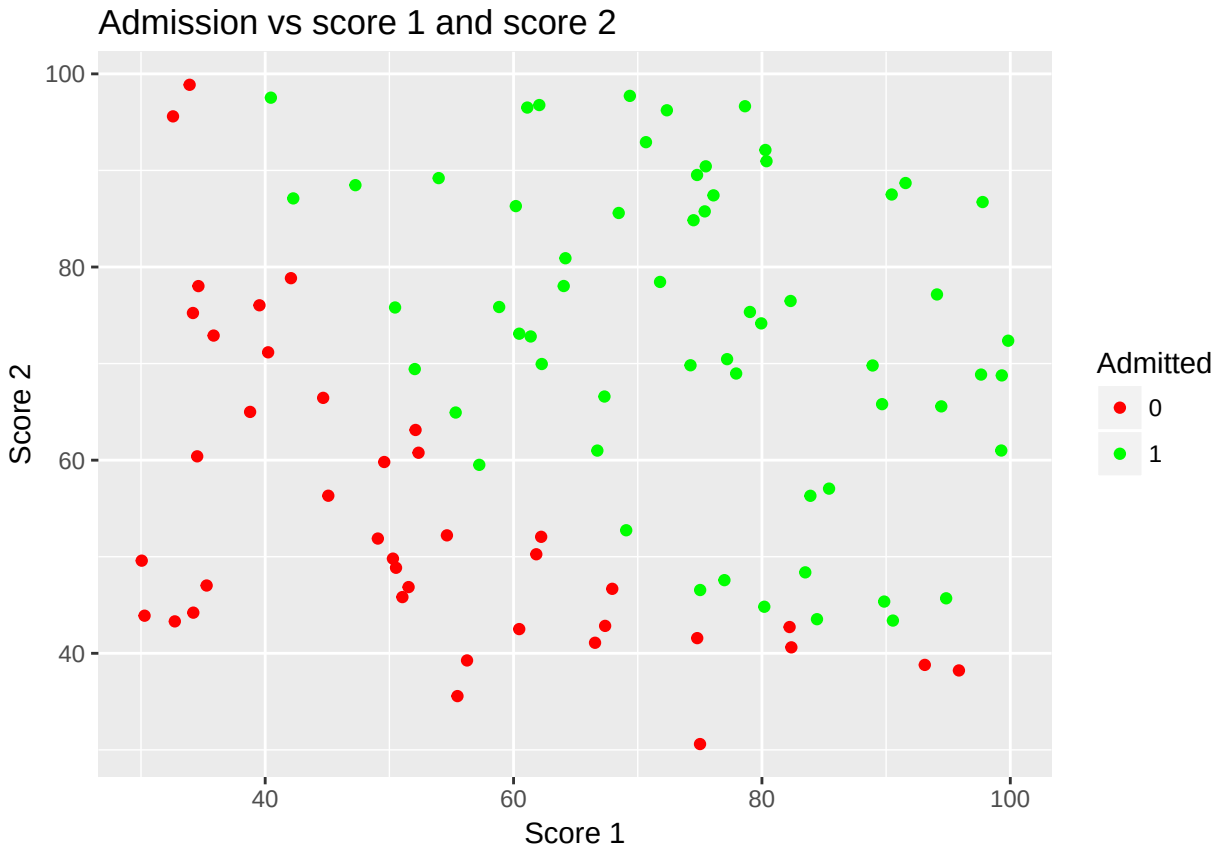
$$w^{\text{new}} = w^{\text{old}} - \frac{\alpha}{N}\Phi(x)^T(\sigma(w^T\Phi(x)) - t)$$

Implementation:

We use data of the applicants for a course. In this data we have two exam scores and the admission (if they were admitted 1 or not 0). Let's visualize the data first

```r
#Load data
data <- read.csv("dataexams.csv")

#Create plot
ggplot(data=data,aes(x=score.1,y=score.2))+
  geom_point(aes(colour=as.factor(label)))+
  scale_color_manual(values=c("red","green"))+
  labs(colour="Admitted")+
  xlab("Score 1")+
  ylab("Score 2")+
  ggtitle("Admission vs score 1 and score 2")
```




```r
#Predictor variables. We add an intercept (that will be the one that will give
#us the correspondant hyperplane, in this case a line w1x1+w2x2=-w0)
X <- as.matrix(data[,c(1,2)])
X <- cbind(rep(1,nrow(X)),X)
#Response variable
Y <- as.matrix(data$label)
```

We define the sigmoid function and the cost function (deviance)

```r
#Sigmoid function
sigmoid <- function(z)
{
  g <- 1/(1+exp(-z))
  return(g)
}
```

```
#Cost Function
cost <- function(theta)
{
  N <- nrow(X)
  g <- sigmoid(X%*%theta)
  J <- (1/N)*sum((-Y*log(g)) - ((1-Y)*log(1-g)))
  return(J)
}
```

Then, we apply the R function optim() that uses gradient descend optimization. We put as initial parameter $w = 0$

```
#Intial theta and its cost
initial_theta <- rep(0,ncol(X))
cost(initial_theta)
```

```
## [1] 0.6931472
```

```
# Derive theta using gradient descent using optim function
theta_optim <- optim(par=initial_theta,fn=cost)
theta <- theta_optim$par
```

From this we find that the parameters are $w = (-25.165, 0.2063, 0.2015)$, so the line $0.2063x_1 + 0.2015x_2 = 25.165$ determines the boundary that determines if a student will be accepted or not. This is how the sample would be predicted

```
#Define the line function and compute a sequence to plot it.
fy<-function(x,w0,w1,w2){
  return((-w0-w1*x)/w2)
}
x<-seq(25,101,by=1)
y<-fy(x,theta[1],theta[2],theta[3])
l<-as.data.frame(cbind(x,y))
#Plot it
ggplot()+
  geom_point(data=data,aes(x=score.1,y=score.2,colour=as.factor(label)))+
  scale_color_manual(values=c("red","green"))+
  geom_line(data=l,aes(x=x,y=y))+
  labs(colour="Admitted")+
  xlab("Score 1")+
  ylab("Score 2")+
  ggtitle("Admission vs score 1 and score 2")
```

Extra: See appendix

Generative models

Model the probability distribution of the inputs and the outputs $p(x, y)$ and then from there it recovers the conditional probability $p(y|x)$ to do the classification. Generative models can have more broad uses such as generating new data similar to the one modelled.

Appendix

Logistic regression

Firstly, it does not need a linear relationship between the dependent and independent variables. Logistic regression can handle all sorts of relationships, because it applies a non-linear log transformation to the predicted odds ratio. Secondly, the independent variables do not need to be multivariate normal – although multivariate normality yields a more stable solution. Also the error terms (the residuals) do not need to be multivariate normally distributed. Thirdly, homoscedasticity is not needed. Logistic regression does not need variances to be heteroscedastic for each level of the independent variables. Lastly, it can handle ordinal and nominal data as independent variables. The independent variables do not need to be metric (interval or ratio scaled).

However some other assumptions still apply.

Binary logistic regression requires the dependent variable to be binary and ordinal logistic regression requires the dependent variable to be ordinal. Reducing an ordinal or even metric variable to dichotomous level loses

a lot of information, which makes this test inferior compared to ordinal logistic regression in these cases.

Secondly, since logistic regression assumes that P(Y=1) is the probability of the event occurring, it is necessary that the dependent variable is coded accordingly. That is, for a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.

Thirdly, the model should be fitted correctly. Neither over fitting nor under fitting should occur. That is only the meaningful variables should be included, but also all meaningful variables should be included. A good approach to ensure this is to use a stepwise method to estimate the logistic regression.

Fourthly, the error terms need to be independent. Logistic regression requires each observation to be independent. That is that the data-points should not be from any dependent samples design, e.g., before-after measurements, or matched pairings. Also the model should have little or no multicollinearity. That is that the independent variables should be independent from each other. However, there is the option to include interaction effects of categorical variables in the analysis and the model. If multicollinearity is present centering the variables might resolve the issue, i.e. deducting the mean of each variable. If this does not lower the multicollinearity, a factor analysis with orthogonally rotated factors should be done before the logistic regression is estimated.

Fifthly, logistic regression assumes linearity of independent variables and log odds. Whilst it does not require the dependent and independent variables to be related linearly, it requires that the independent variables are linearly related to the log odds. Otherwise the test underestimates the strength of the relationship and rejects the relationship too easily, that is being not significant (not rejecting the null hypothesis) where it should be significant. A solution to this problem is the categorization of the independent variables. That is transforming metric variables to ordinal level and then including them in the model. Another approach would be to use discriminant analysis, if the assumptions of homoscedasticity, multivariate normality, and absence of multicollinearity are met.

Lastly, it requires quite large sample sizes. Because maximum likelihood estimates are less powerful than ordinary least squares (e.g., simple linear regression, multiple linear regression); whilst OLS needs 5 cases per independent variable in the analysis, ML needs at least 10 cases per independent variable, some statisticians recommend at least 30 cases for each parameter to be estimated.

Some selection tools

Model Selection

A few important things

Residual Sum of Squares:

$$RSS = \sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2$$

Total Sum of Squares:

$$TSS = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

Mean Square Error:

$$MSE = \frac{RSS}{n}$$

Bias-variance decomposition of MSE:

$$MSE = bias^2 + variance$$

Standard Errors for our estimators:

$$SE(\hat{\beta}_j)^2 = \frac{\sigma^2}{\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}$$

95% confidence interval for β_j:

$$\hat{\beta}_j \pm 2.SE(\hat{\beta}_j)$$

Correlation between variables X and Y:

$$cor(X, Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Formula for R^2:

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}$$