THE ESSENTIALS OF DATA ANALYTICS AND MACHINE LEARNING

DR. MICHAEL ASHCROFT

PERSONTYLE
DATA SCIENCE CENTRE OF EXCELLENCE

The Essentials of Data Analytics and Machine Learning

[A guide for anyone who wants to learn practical machining learning using R]

Author: Dr. Mike Ashcroft

Editor: Ali Syed

FEATURE TRANSFORMATION

Module 5

Feature transformation is the name given to replacing our original features with functions of these features. The functions can either be of individual features, or of groups of them. The result of these functions is itself a random variable – by definition the function of any random variable is itself a random variable. Utilizing feature transformations is equivalent to changing the bases of our feature space, and it is done for the same reason that we change bases in calculus: We hope that the new bases will be easier to work with.

5. Feature Transformations

In this module we will see that most of the advanced statistical modeling techniques utilize feature transformation / bases changes within their algorithms, and you should make sure you are comfortable with the concept.

Here we will look at some feature transformations that are common during preprocessing. They range from simple scaling and centering to some of the most complicated procedures in machine learning.

5.1. Scaling and Centering

Perhaps the most common feature transformation is one that is seldom thought of as a feature transformation at all. It is very common to scale and center the features you are working with.

Centering of a real valued variable is done by subtracting its sample mean from all values. The equation for calculating the sample mean is:

$$\bar{x} = \sum_{i=1}^{N} x_i$$

N is the number of values in the sample. Note that the sample mean is generally denoted by the bar over the variable, whereas the population mean is normally denoted by the Greek letter μ (mu).

Scaling of a real valued variable is done by dividing all its values by its sample standard deviation. The equation for calculating the sample standard deviation is:

$$SSD_X = \sqrt{\frac{1}{N-1}\sum_{1}^{N}(x_i - \bar{x})^2}$$

The reason for centering and scaling is that it places all variables on equal standing. Some statistical models, such as *k*-nearest neighbors, examine the distances between different data points. In such a method, variables with large absolute differences in values will be more important. Yet generally such absolute differences in values reflects nothing more than the metric chosen to measure the variable, and *a priori* it is unreasonable that one variable should

WHICH SAMPLE STANDARD DEVIATION?

You may be surprised by the negative one in our definition of sample standard deviation. The definition given is sometimes referred to as *corrected* sample standard deviation, and this subtraction of one from *N* is known as Bessel's correction after Friedrich Bessel.

The uncorrected sample standard deviation is given by:

$$SSD_X = \sqrt{\frac{1}{N}\sum_{1}^{N}(x_i - \bar{x})^2}$$

It is *biased*, as its expected value is *not* the true value of the population standard deviation. In fact its expected value will be a little less than the true value of the population standard deviation. What many people do not know is that the corrected sample standard deviation is *still* biased, albeit less so! Bessel's correction will remove the bias of the sample variance, but not the sample standard deviation, as the square root operator introduces additional bias not present in the variance case. It is also the case that the corrected sample standard deviation will often increase the mean squared error of the estimator.

It is interesting to ask ourselves which equation for sample standard deviation R uses in the scale functions. Since the documentation doesn't tell us, it is probably the correct version, but let's find out.

First, we see that scale uses the same definition of standard deviation as the sd function:

```
v = 1:10
all((v-mean(v))/sd(v) == scale(v))
```

We see that the documentation for sd is equally silent on the version used. So we test it:

```
> sd(c(2,4,6))
[1] 2
```

The mean of the vector <2,4,6> is 4, so we see that it is the corrected sample standard deviation that is being used:

$$\sqrt{\frac{1}{3-1}((2-4)^2 + (4-4)^2 + (6-4)^2)} = \sqrt{\frac{1}{2}(4+0+4)} = \sqrt{4} = 2$$

be more important than another merely for the scale of its metric. Scaling and centering removes such bias.

Original Features			Scaled and centered features		
X_1	X_2	X_3	X'_1	X'_2	X'_3
1	.093	14.8	-1.0139	-1.1985	1.0146
6	.289	12.0	1.3249	1.5814	-0.6088
4	.192	10.8	0.3900	0.2057	-1.3045
2	.177	12.1	-0.5459	-0.0071	-0.5508
1	.203	15.2	-1.0139	0.3617	1.2465
5	.111	13.4	0.8579	-0.9432	0.2029

$\bar{X}_1 = 3.1667$ $\bar{X}'_1 = 0$

$\bar{X}_2 = 0.1775$ $\bar{X}'_2 = 0$

$\bar{X}_3 = 13.05$ $\bar{X}'_3 = 0$

$SSD_{X1} = 2.1370$ $SSD_{X'1} = 1$

$SSD_{X2} = 0.0705$ $SSD_{X'2} = 1$

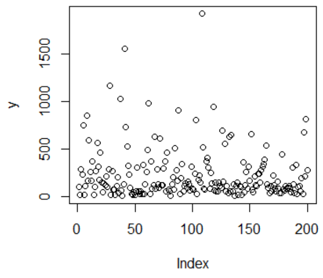
$SSD_{X3} = 1.7248$ $SSD_{X'3} = 1$

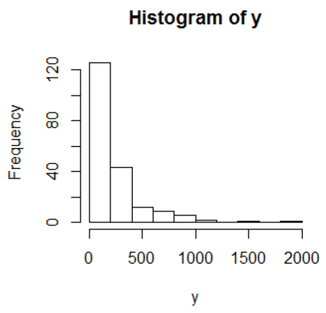
5.2. Expert Selected Changes of Bases

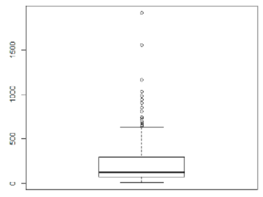
Traditional statistics sometimes relies on statisticians themselves selecting transformation of the variables they are dealing with to make them more tractable. Often this involves visual examination of plots of the data or functions of the data, as well as statistical tests. A common aim is the generation of a function that would make clearly non-normal data at least approximately normal. In this way, techniques that assumed the normality of data could be used.

Let's examine a simple example. We will generate some data, and examine some visual representations of it:

```
> y=exp(rnorm(200,5))
> plot(y)
> hist(y)
> boxplot(y)
```

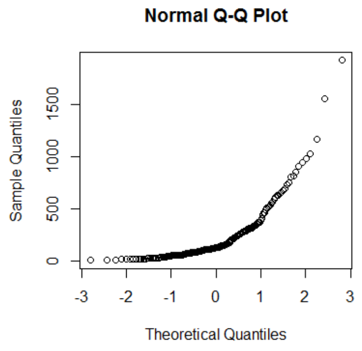






It is clear that the data is non-normal – the histogram is not approximately bell shaped, and there are many outliers on the boxplot. A more information visual test comes from plotting the quantiles against what would be expected for a normally distributed variable using the **qqnorm** function:

```
> qqnorm(y)
```



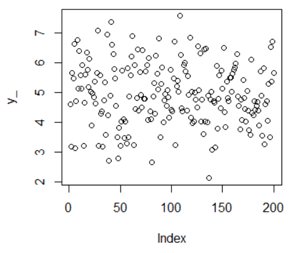
If y was approximately normal, the resulting line would be approximately straight, with slope 1. We can also apply statistical tests of normality, such as the *Shapiro-Wilk,* implemented in R in the **stats** package with the **shapiro.test** function:

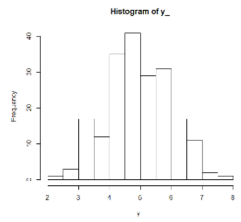
```
> shapiro.test(y)
        Shapiro-Wilk normality test
data:   y
W = 0.7079, p-value < 2.2e-16
```

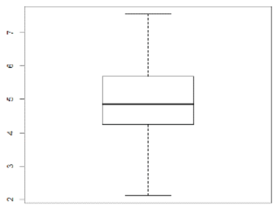
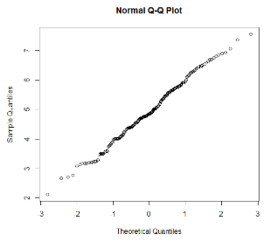
This is a statistical test of the type you will see repeatedly in this book. They work by generating a test statistic that is drawn from a known distribution under the null hypothesis. If the probability of obtaining the observed value of the test statistic under this hypothesis is low enough, we can reject it. Rejection is normally done only if the probability is below .05, .01 or even lower, and we talk about rejecting with the appropriate amount of confidence (95%, 99%, etc.). In this case the null hypothesis is that the data is drawn from a normal distribution. W is the test statistic, and most importantly the p-value is the probability that the data is drawn from a normal distribution. In this case, the probability is tiny.

An experienced statistician looking at these plots would have no trouble recognizing that a log transformation should transform this variable into one that is approximately normal. Let's try it:

```
> y_=log(y)
> plot(y_)
> hist(y_)
> boxplot(y_)
> qqnorm(y_)
```





The transformed data looks much more normal. The final test:

```
> shapiro.test(y_)
        Shapiro-Wilk normality test
data:  y_
W = 0.9943, p-value = 0.6394
```

We could now feel comfortable working with techniques that assume or require (near-)normality.

5.3. Principle Component Analysis

The principle components of a data set provide a projection onto a rotation of the data space that give 'directions of maximum variance'. Accordingly each succeeding component has the highest variance possible under the constraint that it is orthogonal to (i.e. uncorrelated with) the preceding components.

We can find the principle components for data *X* by finding the eigenvectors of a scaled and centered covariance matrix $\frac{1}{N-1}X^TX$ where *N* is the number of rows of *X*. Moreover, the eigenvalues give us a measure of the amount of variance explained by each principle component, and so are monotonically decreasing.

Eigenvectors and Eigenvalues

The eigenvectors of a square matrix are those vectors whose orientation is not changed by the linear map the matrix represents. Instead the map returns a scaled version of the eigenvector. This scalar is the eigenvector's associated eigenvalue.

So if A is a square matrix, v a vector and α a scaler, then claiming that v is an eigenvector with eigenvalue α is equalivalent to stating that $Av = \alpha v$.

It can be useful to project our input variable onto their first *n* principle components. This (hopefully) results in high information content per free parameter. We can analyze the eigenvalues to find good values of *n*.

As an example, let us look at the **Duncan** dataset from the **car** package. We load it and examine it with the **head** function:

```
> library(car)
> data(Duncan)
> head(Duncan)
           type income education prestige
accountant prof     62        86       82
pilot      prof     72        76       83
architect  prof     75        92       90
author     prof     55        90       76
chemist    prof     64        86       90
minister   prof     21        84       87
```

We would perform PCA on real X variables. So if prestige was our target variable, we would want to perform PCA only on income and education. We create a dummy matrix X consisting of only those columns:

```
> X=scale(Duncan[,2:3])
```

Now we obtain the eigenvalues and eigenvectors of X^TX:

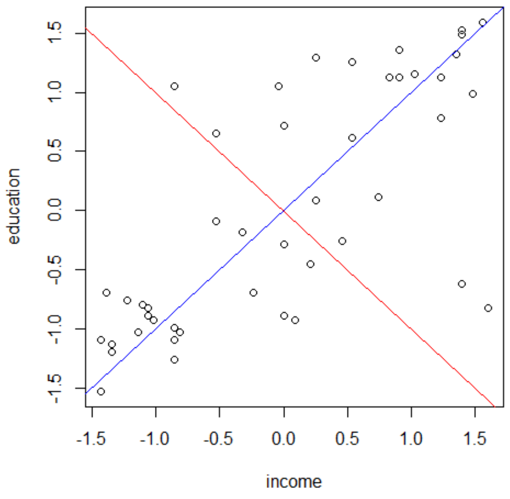
```
> eg=eigen(crossprod(X)/(nrow(X)-1))
> eg
```

```
$values
[1] 1.7245124 0.2754876

$vectors
          [,1]       [,2]
[1,] 0.7071068 -0.7071068
[2,] 0.7071068  0.7071068
```

We can plot the data and these two discovered principle components:

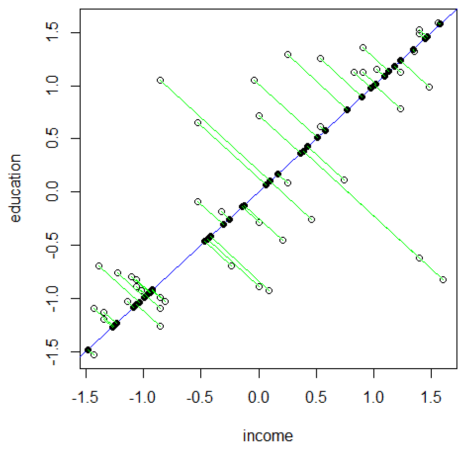
```
> plot(X)
> abline(0,eg$vectors[2,1]/eg$vectors[2,2],col
="blue")
> abline(0,eg$vectors[1,1]/eg$vectors[1,2],col
="red")
```



We see that the basis given by the first principle component is the blue line, and the basis given by the second principle component is the red line. Together they form a rotation matrix, and it is always the case that projecting onto all principle components will simply perform a rotation of the data within its existing ambient space.

What is interesting, though, is projecting our original data onto a *subset* of the principle components. Here there are only two principle components so the only interesting thing we can do is to project onto the first:

```
> pX=X%*%eg$vectors[,1]
> head(pX)
                [,1]
accountant 1.3772504
pilot      1.4290358
architect  1.8960047
author     1.2697216
chemist    1.4351268
minister   0.1432652
```



We can use pX to plot this projection from the original two dimensional space to the new one dimensional space:

```
> plot(X)
> abline(0,eg$vectors[2,1]/eg$vectors[2,2],col="blue")
> points(pX%*%t(eg$vectors[,1]),pch=16)
> segments(X[,1],X[,2],pX%*%t(eg$vectors[,1])[,1],pX%*%t(eg$vectors[,1])[,2],c
ol="green")
)
```

In this way we have created a new feature, pX, that is a linear combination of our original features *education* and *income*. By using it instead of the original two, we have reduced the number of features we are working with. Moreover, this linear combination has been chosen so as to be the directions of maximal variance in the original feature space, so we may hope that we retain *most* of the information that was present in the original feature set. PCA is also used for compression, and exactly as in that context we would like to minimize the information lost (visually represented in our example by the green lines of projection onto our chosen subspace) by the use of a subset of principle components.

In general, using PCA for feature transformation/selection in this way requires a clever selection of the number of first principle components to be used. It is common to use a *scree plot* for this, where this provides a plot of the eigenvalues (or normalized eigenvalues) associated with each of the principle components. Visual examination of this can assist in determining which eigenvectors to use.

To examine this, let us look at the **zprostate** data from the **bestglm** package.

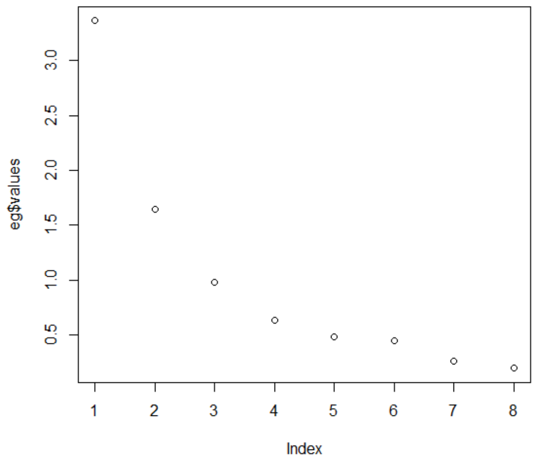
```
> library(bestglm)
Loading required package: leaps
> data(zprostate)
> head(zprostate)
      lcavol    lweight        age      lbph        svi        lcp    gleason      pgg45       lpsa train
1 -1.6373556 -2.0062118 -1.8624260 -1.024706 -0.5229409 -0.8631712 -1.0421573 -0.8644665 -0.4307829  TRUE
2 -1.9889805 -0.7220088 -0.7878962 -1.024706 -0.5229409 -0.8631712 -1.0421573 -0.8644665 -0.1625189  TRUE
3 -1.5788189 -2.1887840  1.3611634 -1.024706 -0.5229409 -0.8631712  0.3426271 -0.1553481 -0.1625189  TRUE
4 -2.1669171 -0.8079939 -0.7878962 -1.024706 -0.5229409 -0.8631712 -1.0421573 -0.8644665 -0.1625189  TRUE
5 -0.5078745 -0.4588340 -0.2506313 -1.024706 -0.5229409 -0.8631712 -1.0421573 -0.8644665  0.3715636  TRUE
6 -2.0361285 -0.9339546 -1.8624260 -1.024706 -0.5229409 -0.8631712 -1.0421573 -0.8644665  0.7654678  TRUE
```

The first eight columns of this dataset are the input variables. We can perform PCA on them as above:

```
> X=scale(zprostate[1:8])
> eg=eigen(crossprod(X)/(nrow(X)-1))
> eg
$values
[1] 3.3607452 1.6483063 0.9756610 0.6320111 0.4832909 0.4422130 0.2626238 0.1951486
$vectors
            [,1]        [,2]        [,3]       [,4]        [,5]        [,6]        [,7]        [,8]
[1,] -0.42224005  0.05369897  0.33161511  0.1006344  0.40591991  0.62501055 -0.17349703 -0.33648701
[2,] -0.18712415 -0.53877313  0.42249380 -0.1318237  0.43787931 -0.53236053  0.01046653  0.05978282
[3,] -0.22322795 -0.46863197 -0.24245432  0.7927534 -0.14212325  0.05859594  0.11361112  0.08228769
[4,] -0.08562866 -0.62886897 -0.08336125 -0.5105493 -0.43575619  0.36257288 -0.08275523 -0.03729392
[5,] -0.39020837  0.20742156  0.39519323  0.1203740 -0.58336085 -0.28266012 -0.46041071 -0.04314492
[6,] -0.46417852  0.19008255  0.18687032 -0.1360282 -0.12048345  0.12396510  0.63213860  0.51526685
[7,] -0.40572527  0.07198888 -0.53818216 -0.1580842  0.27860397 -0.04920224 -0.49109579  0.44086034
[8,] -0.44406941  0.08608338 -0.40627450 -0.1593499 -0.03101519 -0.30557578  0.31189904 -0.64290058
```

We can produce the scree plot from the eigenvalues:

```
> plot(eg$values)
```



Common advice is to look for an 'elbow' where the slope of descent levels off. The reality is that it is a very subjective procedure. Here I might suggest taking the first four or alternatively the first six principle components.

For those who go cold when reading the word 'eigen' R offers a number of functions that simplify performing PCA. One of these is **prcomp** in the **stats** package. This function uses a formula without response variable to specify the variables that we will perform PCA on, and for compatibility reasons requires you to manually set the *scale* parameter to TRUE:

```
> prcomp(~income+education,Duncan,scale=T)
Standard deviations:
[1] 1.3132069 0.5248692

Rotation:
                 PC1        PC2
income    -0.7071068 -0.7071068
education -0.7071068  0.7071068
```

You can see that the rotation matrix is exactly our eigenvectors. The standard deviations are the square root of the eigenvalues, which you will recall gave use the variance explained by each eigenvector/principle component.

If you would like the **prcomp** function to project your data onto the principle components for you, you can set the parameter *retX* to TRUE:

```
> resp=prcomp(~income+education,Duncan,scale=T,retX=T)
> head(resp$x)
                  PC1         PC2
accountant -1.3772504  0.21200587
pilot      -1.4290358 -0.31497249
architect  -1.8960047 -0.02163275
author     -1.2697216  0.50961179
chemist    -1.4351268  0.15412949
minister   -0.1432652  1.35095243
```

You can then select a subset of the principle components by selecting the appropriate columns:

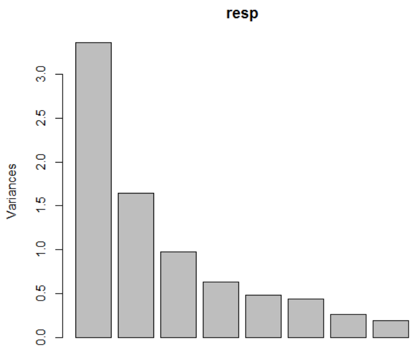
```
> head(resp$x[,1])
```

```
accountant      pilot  architect     author    chemist   minister
-1.3772504 -1.4290358 -1.8960047 -1.2697216 -1.4351268 -0.1432652
```

Finally, you can use the function **screeplot** on the returned object of **prcomp** to get a scree plot, Continuing the **zprostate** example:

```
> resp=prcomp(~lcavol+lweight+age+lbph+svi+lcp+gl
eason+pgg45,zprostate,scale=T,retX=T)
```

```
> screeplot(resp)
```



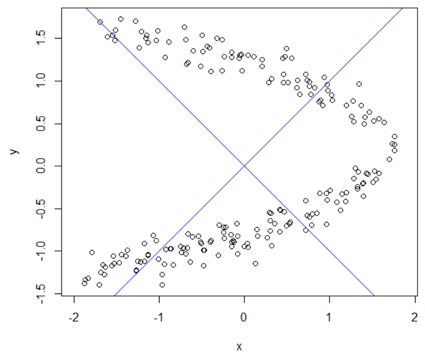
Note that what is plotted are the variances (eigenvalues) rather than the standard deviations and so match our manual example.

5.4.Non-linear Dimensionality Reduction Techniques

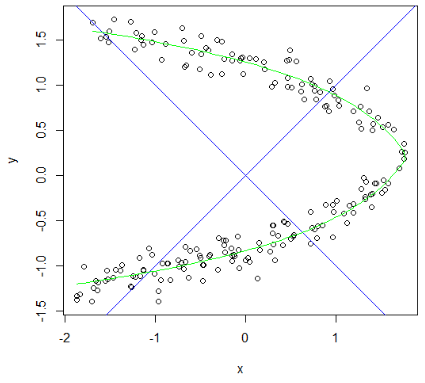
PCA is a linear transformation of the features. There are many techniques that allow non-linear transformations.

It is simple to see the attraction of non-linear feature transformation. Take the following crescent shaped dataset, which we will generate and perform PCA upon:

```
set.seed(0)
x=runif(200,10,20)
y_=5*((log(21-x)))*sample(c(-1,1),length(x),repla
ce=T)
y=y_+rnorm(100)
x=scale(x)
y=scale(y)
y_=scale(y_)
plot(x,y,asp=c(1,1))
pc=prcomp(cbind(x,y))
> pc
Standard deviations:
[1] 1.0902969 0.9006956
Rotation:
            PC1        PC2
[1,] -0.7071068 -0.7071068
[2,] -0.7071068  0.7071068
abline(0,pc$rotation[1,1]/pc$rotation[2,1],col="blue")
abline(0,pc$rotation[1,2]/pc$rotation[2,2],col="blue")
```



The principle components are in blue. If we think in terms of information compression, compressing our data to the first principle component (the down sloping line) would lead to a large amount of information loss. For this case PCA is a poor method for feature transformation.



But it is easy to see a curve which would be great to project our data onto. We approximate it in green:

```
> y_pos=y_[which(y_>0)][order(y_[which(y_>0)],dec
reasing=T)]
> x_pos=x[which(y_>0)][order(y_[which(y_>0)],decr
easing=T)]
> y_neg=y_[which(y_<=0)][order(y_[which(y_<=0)],decreasing=T)]
> x_neg=x[which(y_<=0)][order(y_[which(y_<=0)],decreasing=T)]
> points(c(x_pos,x_neg),c(y_pos,y_neg),type="l",col="green")
```

Projecting onto such a curve would be a non-linear transformation of the original data. Of course, finding such a useful non-linear transformation is not always a simple task.

There are a great many non-linear feature transformation techniques available, and this is an area of continual research. Some of the more common include:

- Kernel PCA
- Non-linear PCA
- Principle Curves and Surfaces
- Self-organizing maps
- Auto-encoders
- Deep Belief Networks
- Gaussian process latent variable models
- Laplacian eigenmaps
- Diffusion maps

A number of these are examined in later models: Kernel PCA is discussed in module 12, while auto-encoders and restricted Boltzmann machines are dealt with in module 16.

Many, if not most, sophisticated machine learning algorithms include non-linear feature transformation within the modelling algorithms themselves. This can be explicit, as in polynomial regression (module 7), splines (module 13), and neural networks (module 16), or implicit via the kernel trick (see module 12), such as kernel regression (module 12) and support vector machines (module 15). These mentioned methods are far from a complete list, and we will see many more examples in later modules.

5.5.Text Transformation: Adjusted Term-Document Matrices

As noted above, it may be that instead of the frequency counts for the given terms (here: concept-tokens) we would prefer to use functions of these counts as the features. We will look at two common examples of such functions.

The first is normalization. You may notice that post (i) has many more non-zero values in our example term-document matrix than the other posts. This is because it is longer. It may be that we would prefer to examine the proportion of each document's total number of terms that a given term makes up. In this case, we could simply normalize the rows:

	the	most	important	aspect	i	Think	be	do	it	nothing	will	if	you	so	believe	that
i.	$\frac{1}{33}$	$\frac{1}{33}$	$\frac{1}{33}$	$\frac{1}{33}$	$\frac{1}{33}$	$\frac{1}{33}$	$\frac{3}{33}$	$\frac{4}{33}$	$\frac{3}{33}$	$\frac{1}{33}$	$\frac{1}{33}$	$\frac{1}{33}$	$\frac{2}{33}$	$\frac{1}{33}$	$\frac{1}{33}$	$\frac{2}{33}$
ii.	$\frac{2}{24}$	0	0	0	$\frac{1}{24}$	$\frac{1}{24}$	$\frac{1}{24}$	$\frac{2}{24}$	$\frac{1}{24}$	0	0	0	$\frac{1}{24}$	0	0	0
iii.	$\frac{2}{15}$	0	0	0	0	0	0	$\frac{1}{15}$	$\frac{1}{15}$	$\frac{1}{15}$	0	0	0	0	0	0

	cannot	not	even	attempt	what	about	risk	should	remember	discretion	better	part	of
i.	$\frac{1}{33}$	$\frac{1}{33}$	$\frac{1}{33}$	$\frac{1}{33}$	0	0	0	0	0	0	0	0	0
ii.	0	$\frac{1}{24}$	0	0	$\frac{1}{24}$	$\frac{1}{24}$	$\frac{1}{24}$	$\frac{1}{24}$	$\frac{1}{24}$	$\frac{1}{24}$	$\frac{1}{24}$	$\frac{1}{24}$	$\frac{1}{24}$
iii.	0	0	0	0	0	0	0	0	0	0	0	0	0

	valor	right	on	take	to	streetz	forget	clown	,	.	!	☺
i.	0	0	0	0	0	0	0	0	$\frac{2}{33}$	$\frac{1}{33}$	$\frac{1}{33}$	0
ii.	$\frac{1}{24}$	0	0	0	0	0	0	0	0	$\frac{3}{24}$	0	$\frac{1}{24}$
iii.	0	$\frac{1}{15}$	$\frac{1}{15}$	$\frac{1}{15}$	$\frac{1}{15}$	$\frac{1}{15}$	$\frac{1}{15}$	$\frac{1}{15}$	0	0	$\frac{3}{15}$	0

The second is a weighting of the counts known as the *term frequency inverse document frequency* (tf-idf). It is often used when searching for documents using keywords, and seeks to provide a measure of how important a word is in distinguishing a document within a corpus. The idea is that a word will be good at distinguishing a document in relation to how often it occurs in the document (the term frequency) and inversely in relation to how often it occurs in the corpus (inverse document frequency). If a word occurs often in a document, and seldom elsewhere, the document should perform highly when the word is used in a keyword search. The tf-idf function is:

$$tfidf(t,d) = \left(.5 + \frac{.5f_{t,d}}{\max\limits_{t \in d}(f_{t,d})}\right)(\log \frac{N}{|\{d: t \in d\}|})$$

Where:

- t is a variable that ranges over terms
- d is a variable that ranges over documents (which are themselves simply sets of terms).
- $f_{t,d}$ is the frequency of term t in document d
- N is the number of documents
- $|\{d: t \in d\}|$ is the number of documents which contain t.

In our example, this would lead to:

	the	most	important	aspect	i	think	be	do	it	nothing	will	if	you	so	believe	that
i.	0	.687	.687	.687	.253	.253	.355	0	0	.253	.687	.687	.304	.687	.687	.824
ii.	0	.549	.549	.549	.270	.270	.270	0	0	.203	.549	.549	.270	.549	.549	.549
iii.	0	.549	.549	.549	.203	.203	.203	0	0	.270	.549	.549	.203	.549	.549	.549

	cannot	not	even	attempt	what	about	risk	should	remember	discretion	better	part	of
i.	.687	.253	.687	.687	.549	.549	.549	.549	.549	.549	.549	.549	.549
ii.	.549	.270	.549	.549	.732	.732	.732	.732	.732	.732	.732	.732	.732
iii.	.549	.203	.549	.549	.549	.549	.549	.549	.549	.549	.549	.549	.549

	valor	right	on	take	to	streetz	forget	clown	,	.	!	☺
i.	.549	.549	.549	.549	.549	.549	.549	.549	.824	.253	.253	.549
ii.	.732	.549	.549	.549	.549	.549	.549	.549	.549	.405	.203	.732
iii.	.549	.732	.732	.732	.732	.732	.732	.732	.549	.203	.405	.549

How do we find the tf-idf frequency in R?

```
# Make the term-document matrix
termDoc=rbind(
    c(1,1,1,1,1,1,3,4,3,1,1,1,2,1,1,2,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,2,1,1,0),
    c(2,0,0,0,1,1,1,2,1,0,0,0,1,0,0,0,1,0,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,
0,0,3,0,1),
    c(2,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,
1,0,0,3,0)
    )
colnames(termDoc)=c("the","most","important","aspect","i","think","be","do","i
t","nothing","will","if","you","so","believe","that",
                    "cannot","not","even","attempt","what","about","risk","sho
uld","remember","discretion","better","part","of",
                    "valor","right","on","take","to","streetz","forget","clown
",",",".","!",":)"
                    )
rownames(termDoc)=c("i","ii","iii")
#
# Calculate the tfidf
nDoc=nrow(termDoc)
docsWithTerm=apply(termDoc,2,function(col) length(which(col>0)))
tfidf=t(apply (termDoc , 1, function (row) {
    rMax=max(row)
    sapply(1:length(row),function(x) {
        (.5+(.5*row[x])/rMax)*log(nDoc/docsWithTerm[x])
    })
}))
colnames(tfidf)=colnames(termDoc)
rownames(tfidf)=rownames(termDoc)
#
# Display the tfidf
print(tfidf)
```

5.1.Exercises

1. Get the principle components of the income and education variables in the **Duncan** data set using the **prcomp** function?
2. Generate 100 random data values: X~unif(0,5). Project this data onto the coordinate systems X' specified by:
   a. X'=X^2
   b. X'=1/(1+e^(-x))
   c. X^'= 〖(|X-5|)〗 ^2 log⁡(|X-5|)
   d. X_1^'=(|X-5|)^2 log⁡(|X-5|), X_2^'=(|X+5|)^2 log⁡(|X+5|)