An Introduction to Data Science

Gethin Williams

November 5, 2015

Contents

1 Introduction

The term data science is in vogue. Its definition, however, has proved somewhat elusive. Here's a little of what wikipedia has to say on the topic:

> Data science is, in general terms, the extraction of knowledge from data. It employs techniques and theories drawn from many fields within the broad areas of mathematics, statistics, and information technology, including signal processing, probability models, machine learning, statistical learning, computer programming, data engineering, pattern recognition and learning, visualization, uncertainty modeling, data warehousing, and high performance computing. Methods that scale to Big Data are of particular interest in data science, although the discipline is not generally considered to be restricted to such data...

This gives us a starting point but we can see that this definition encompasses a pretty broad range of activities. The term Big Data has also reared its head, but more of that later. In addition to verbose paragraphs, such as the above, we are offered graphical descriptions (see figure 1), as well as pithy one-liners:

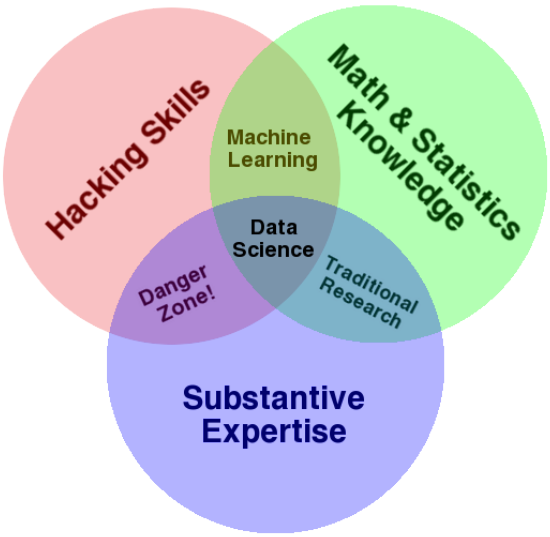


Figure 1: Drew Conway's Venn diagram of data science.

> Data Scientist (noun): Person who is better at statistics than any software engineer and better at software engineering than any statistician. –*Josh Wills*

Some people would prefer to relabel portions of Drew Conway's Venn diagram. For example, you may hear "traditional research" referred to as "data intensive research" and "machine learning" upgraded to "data science" itself.

Jeroen Janssens offers an acronym which usefully outlines the *process* of doing data science, stating that:

> Data science is OSEMN (pronounced as awesome). That is, it involves Obtaining, Scrubbing, Exploring, Modeling, and iNterpreting data.

It would not be wise if I attempted to cover every nook and cranny of this broad field. Rather, I will attempt to provide an introduction that is tailored to research activity ongoing here in the University of Bristol. A legitimate example of data science is to collect the browsing an shopping habits of an individual and to provide recommendations for further browsing or shopping. If you were a bookseller with a very large online presence, this sort of activity would be of great interest to you and your shareholders. However, that is not us. We are interested in doing academic research and so I will attempt to only cover topics, tools and methods that are relevant to that endeavour.

2 Overview

I will split the rest of the document into three main sections:

Data Wrangling Sourcing, preparing and storing your data.

Methods The statistical and computational foundations of the data analysis.

Tools The software which you investment your time into in order to get the job done.

I will not cover the set in Drew Conway's Venn diagram labelled, "Substantive Expertise". This is because you already have that and I, most likely, do not for your particular area of specialisation.

I've deliberately placed the data wrangling section first. This is becuase it highlights one of the distinctions that between data science and tradtional statistics. Namely that a data scientist will not typically be presented with a neatly packaged data set to analyse. Instead she will be responsible for obtaining the raw data, cleaning it and converting it into a format ammenable to any subsequent work or analysis, before communicating any outcomes and perhaps making the data available to others. Data wrangling typically involves a number of different tools. The Linux command line, with its offering of a pipeline of commands, is an environment well suited to data wrangling.

The methods section will have a great deal in common with material offered by traditional statistics textbooks, as well as tomes on machine learning. To that extent I will endeavour to provide links to other material, rather than rehash existing explanations or examples. I will however, try to place each method in an evolving context, relate them and outline their key attributes. Many of the tools available will provide ready access to these methods and so you should not expect to have to program any of these methods yourself (unless, of course you want to!) but, rather, to select judiciously from an available menu.

A great many good tools exist for data science and providing an exhaustive description and comparison would be a fools errand most likely superceded before completed. So, for the purpose of my examples, I will use R whenever possible. R is not a universal panacea, but I believe that it is a really good starting point. (Not least as it provides an interface to many of those other tools.) Whenever possible, I will also provide links to further reading to other tools which I think could be of interest.

3 Data Wrangling

In this first section we immediately engage with an examples of how working as a data scientist differs from what may have been traditionally thought of as statistical modelling, say. Being tasked with first sourcing and curating your data, before being able to embark on any analysis is a situation which, I think, will be familiar to those working in academic research.

I'm using the term data wrangling here in its broadest possible sense. Here, again, is a little of what wikipedia has on the topic:

> Data munging or data wrangling is loosely the process of manually converting or mapping data from one "raw" form into another format that allows for more convenient consumption of the data with the help of semi-automated tools. This may include further munging, data visualization, data aggregation, training a statistical model, as well as many other potential uses. Data munging as a process typically follows a set of general steps which begin with extracting the data in a raw form from the data source, "munging" the raw data using algorithms (e.g. sorting) or parsing the data into predefined data structures, and finally depositing the resulting content into a data sink for storage and future use. Given the rapid growth of the internet such techniques will become increasingly important in the organization of the growing amounts of data available...

In this section we'll consider how you might first source your data, transformations which might be advantageous (including file formats) as well as some possible 'data sinks'.

3.1 Data mining/scraping

.

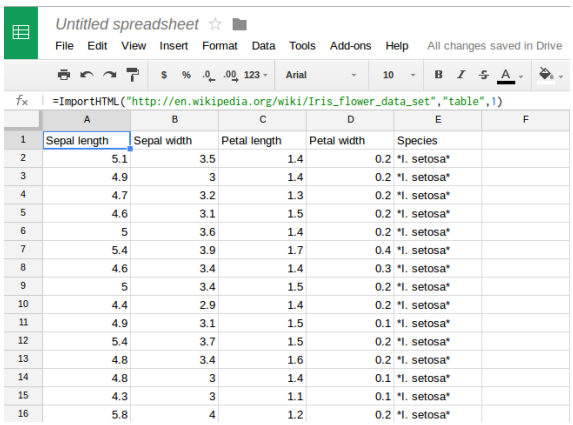
Let's imagine that some data you're interested in resides on the internet–it's not hard to do! But how are you going to access that data without resorting to some laborious manual process of cut-and-paste? Enter a data scraping tool.

First of all, let's identify some data on the web that we would be interested in using. http://en.wikipedia.org/wiki/Iris_flower_data_set describes 150 observations of three different species of Iris (50 each).

Now let's consider a simple yet effective approach to data scraping that requires no programming knowledge. Using Google Docs (available via your UoB account) you can open a spreadsheet and import tables and lists directly from a webpage. In the formula (*Fx)* cell type:

```
=ImportHTML("http://en.wikipedia.org/wiki/Iris_flower_data_set","table",1)
```

The first argument to the ImportHTML function is the URL of the page you are interested in. The second argument is either "table" or "list" and the last argument is the index of the item you are interested in, in the event that there are more than one table or list on the page. See figure 3.1 for the result.



	A	B	C	D	E	F
1	Sepal length	Sepal width	Petal length	Petal width	Species	
2	5.1	3.5	1.4	0.2	*I. setosa*	
3	4.9	3	1.4	0.2	*I. setosa*	
4	4.7	3.2	1.3	0.2	*I. setosa*	
5	4.6	3.1	1.5	0.2	*I. setosa*	
6	5	3.6	1.4	0.2	*I. setosa*	
7	5.4	3.9	1.7	0.4	*I. setosa*	
8	4.6	3.4	1.4	0.3	*I. setosa*	
9	5	3.4	1.5	0.2	*I. setosa*	
10	4.4	2.9	1.4	0.2	*I. setosa*	
11	4.9	3.1	1.5	0.1	*I. setosa*	
12	5.4	3.7	1.5	0.2	*I. setosa*	
13	4.8	3.4	1.6	0.2	*I. setosa*	
14	4.8	3	1.4	0.1	*I. setosa*	
15	4.3	3	1.1	0.1	*I. setosa*	
16	5.8	4	1.2	0.2	*I. setosa*	

Figure 2: Scraping a table from a web page using a Google Docs spreadsheet.

The data can then be saved as, e.g. a CSV (comma separated variable) file and subsequently used by other tools or applications (for example the resulting CSV file can be easily loaded into R to create a data frame: https://source.ggy.bris.ac.uk/wiki/R1#Reading_Data_from_File).

Nifty. Another, similar tool that you might try is the web-based: https://import.io.

An example of the category of programmable web scrapers is given by the python package, Beautiful Soup, http://www.crummy.com/software/BeautifulSoup/bs4/doc/. This package is available from various Linux package managers, including Ubuntu.

A programmable web scraper has a learning curve which is much steeper than the simpler Google Docs example given above. However, it offers significantly more in return. For example, imagine you had a long list of URLs to source some data from. It would be laborious to manually create spreadsheets to capture data from each one. If you were to use, for example, Beautiful Soup, you could write a loop which visited all the URLs in turn. Other things you could do include writing conditionals that controlled the behaviour of the script based on the content that you found at a URL; you could also recursively follow links embedded in a page. The sky's the limit and you can bring all of the power of the Python language to bear on your problem.

Below is a very simple example, showing how you can use Beautiful Soup to access the same tabular iris data that we using Google Docs. You can see that the HTML mark-up is stored in an object (called soup) and that you can search for HTML tags using the object's methods.

```
import urllib2
from BeautifulSoup import BeautifulSoup
page = urllib2.urlopen('http://en.wikipedia.org/wiki/Iris_flower_data_set')
soup = BeautifulSoup(page)
table = soup.find("table")
for row in table.findAll("tr"):
   cells = row.findAll("td")
   print cells
```

Some websites allow you to download data in a format other than the raw HTML code via Application Programmer Interfaces (APIs). This facility, which amounts to you requesting access to an appropriately formulated URL, often requires you to register with the website provider for your own key. Popular formats for this method of data download include the Extensible Markup Language (XML) and JavaScript Object Notation (JSON). An example of a data source that offers you access via an API is the New York Times (see: http://developer.nytimes.com/).

If you would like to try a quick and simple test of downloading data in JSON format, you can visit http://www.jsontest.com. The command line below will retrieve a small amount of data and store it in a file:

```
curl -s "http://headers.jsontest.com/" > test.json
```

Screen scraping is a subcategory of web scraping that involves the use of Optical Character Regognition (OCR) software. I'll not give an example of screen scraping here.

3.2 Data Cleaning and Transforms

Now that you have secured your data, either by scraping the web or via other means, it is likely that you'll need to undertake at least some data cleaning and format manipulation so that your data is in a suitable form to be consumed by subsequent steps of a process, working towards your goal.

As a very simple example, the rows of the table obtained using the Google Docs approach to web scraping look like:

```
5.1,3.5,1.4,0.2,*I. setosa*
```

The species labels have been decorated with some *s (perhaps because the text was italicised on the web page) which you probably don't want. How can we remove those asterisks in a non-labour intensive way?

There are a great many tools which can be brought to bear on the task of cleaning data stored in text files. Command line utilities such as sed, awk, grep, head and tail immediately spring to mind for those familiar with the Linux command line. For the example above, sed can be used to find all the *s and replace them with the empty string (effectively deleting them):

```
sed 's/\*//g' irises.csv > irises-without-asterisks.csv
```

Python provides many functions for manipulating strings and so the task of remove *s would also be easy if you were to access the data from a Python script.

Jeroen Janssens' blog post, gives some useful examples of converting, e.g. JSON to CSV format or XML to JSON. Jeroen has also written a book on the topic, http://datascienceatthecommandline.com/.

Languages such as Python and R also have many packages which facilitate similar format transformations.

All of the foregoing examples have used text format files. Binary format files require far less disk space than their text equivalents, however, processing these files may require you to create more bespoke tools to manipulate them. An exception to this, however, is the NetCDF format, for which many processing and visualisation tools exist, e.g. the NetCDF Operators (NCO, http://nco.sourceforge.net/); the Climate Data Operators (CDO, https://code.zmaw.de/projects/cdo) and the Panoply viewer (http://www.giss.nasa.gov/tools/panoply/).

3.3 Data Storage and Curation

We'll typically start exploring some data source using our PC or laptop. However, it's quite likely that this is not a good home for that data in the longer term. The reasons for this are many and varied. Let's start with some obvious ones. If all your important data is stored on your laptop and you lose it, or it goes phut, then you'll be recounting a tale a woe to your nearest and dearest. Perhaps all the full dataset is too large for your PC to hold it? So it seems likely that some form of remote data repository will provide a better home. However, not all repositories are created alike, and a number of other considerations will inform our decision. Different projects will have different demands. Perhaps you've completed all the data processing and analysis that you forsee and you would be happy with a low-cost archive facility. If, on the other hand, you plan to read and write to this dataset many times in the future, it would make sense to store the data on a readily available, better performing filesystem. This is particularly important for any High Performance Computing (HPC) based processing. If it is very important that the data is not lost if some computer hardware were to fail, then some way of managing multiple copies (preferable in geographically distant locations) will rise to the top of your wish list. What if the dataset is likely to evolve and you would like to track and access different versions? Version control system exist to address this need. How long do you want to store your data for? etc. etc.

The University of Bristol Research Data Service (http://data.bris.ac.uk/) offers support and advice on exactly these kinds of questions. They can help you formulate data management plan and also address requirements regarding data set down by Research Councils UK.

3.4 Databases

Part of finding a good home for you data is to consider whether it should be stored in a database. Wikipedia defines a database as:

> A database is an organized collection of data. The data is typically organized to model aspects of reality in a way that supports processes requiring information.. . .

This helps us to appreciate that selecting an appropriate database can greatly influence the ease, efficiency or success of subsequent steps towards your goal. A simple, yet often overlooked example of this is that data can be accessed much more quickly from a database than it can be from a number of small files.

Relational databases dominate the field and again wikipedia helps us to appreciate why:

> A relational database is a digital database whose organization is based on the relational model of data, as proposed by E.F. Codd in 1970. This model organizes data into one or more tables (or "relations") of rows and columns, with a unique key for each row. Generally, each entity type described in a database has its own table, the rows representing instances of that entity and the columns representing the attribute values describing each instance. Because each row in a table has its own unique key, rows in other tables that are related to it can be linked to it by storing the original row's unique key as an attribute of the secondary row (where it is known as a "foreign key"). Codd showed that data relationships of arbitrary complexity can be represented using this simple set of concepts.
>
> Prior to the advent of this model, databases were usually hierarchical, and each tended to be organized with a unique mix of indexes, chains, and pointers. The simplicity of the relational model led to its soon becoming the predominant type of database.

Thanks to standardisation and the creation of the Structured Query Language (SQL), accessing relational databases is very well supported and many options exist, including packages providing access from both Python and R.

Database technology hasn't stood still since the 1970s, however, and examples of a number of alternative database methodologies are readily available. These alternatives are often grouped together under the umbrella term, "NoSQL." Examples include; Apache CouchDB (`http://couchdb.apache.org/`), which a document based database (with a consequence that objects may differing sets of attributes) which makes much use of web standards and protocols, such as HTTP abd JSON; and Neo4J (`http://neo4j.com/`, which allows you to store data as a graph and so readily supports exploration of connectivity.

3.5 Big Data

Big data is a familiar term with a definition that proves elusive. Big is big, right? But size is relative. The reponses to questionnaires will not challenge the output of the Large Hadron Collider, when it comes to disk space usage. OK, but acknowledging that size is relative still doesn't offer us anything concrete. Perhaps the most pragmatic definition is that data becomes big when it can no longer be stored on a single computer. Sometimes working with big data is the nature of the beast (think google or facebook) and tools exist that are

designed to *scale*. However, it would be wise to acknowledge that systems designed to run across multiple computers are more complex by nature. Good software will shield a user from much of this complexity, but it is perhaps inevitable that a user will eventually have to confront at least some of this complexity and that attention will inevitably be diverted from your original goal. My advice is much the same as for parallel computing: if you absolutely *need* to use it then so be it, but don't be tempted just because it's a buzz-word.

4 Methods

In the previous section we considered the task of data wrangling and looked at some tools which help us acquire our data and prepare it for further analysis. This section focusses squarly on the analysis of data using well established statistical methods. We cannot expect that the blind application of tools alone will provide us with any worthwhile knowledge. Instead we must appreciate that understanding and correct use of these methods is key to gaining any meaningful insight from the data.

We have two sets of observations, represented by the variables x and y, respectively. Further, we believe that there is a relationship between the them, namely that y is a function of x, i.e. $y = f(x)$. But is y a continuous function of x, or do different values of x map to a disctete set of classes? The following sections will help us phrase and subsequently explore our hypotheses about the world.

Determining the relationships between observations is one aspect of this methods section. However, there are also other, more nuanced aspects to consider too. Sometimes we are interested in the exact form of a relationship, so we can learn something new about the world. Othertimes we may just want to find the most effective (in terms of speed or accuracy, or both) mapping between x and y, so that some application works as well as it can (and perhaps affords us the most profit).

We must also appreciate the limitations that we operate under. All data sets are finite and replete with biases and noise which distort our view of the truth. All methods are imbued with assumptions and procedural constraints which restrict the inferences that can be made. However, fear not. Forewarned is forearmed and good inferences can still be made. As our skills develop, we will design experiments which are robust to some biases and noise and we will temper our conclusions based upon our knowledge of the limitations of the processes employed in our work. We will also encounter other important topics such as overfitting, 'the curse of dimensionality' and feature selection and in turn knowledge of those issues will enrich our abilities.

4.1 Exploratory Data Analysis

Our first method is, the often overlooked, Exploratory Data Analysis (EDA). When we first gain access to some data, we can use summary statistics, such as means, medians etc. and graphs, such as scatter plots, to build an appreciation of the relationships which are present. EDA can be invaluable in buidling intuition before we throw a lot of cpu-cyles at a problem. This reduces the number of dead-ends that we encounter and helps us to draw conclusions faster.

R includes a number of datasets by default, which considerably simplifies the task of writing examples. Let's consider the `cars` dataset, which is a data frame consisting of 50 records in two columns–vehicle speed (miles per hour) and stopping distance (feet). Within

R, we can use the `summary` function to display some summary data (relevant to the object under consideration):

summary(cars)

gives:

```
     speed           dist
 Min.   : 4.0   Min.   :  2.00
 1st Qu.:12.0   1st Qu.: 26.00
 Median :15.0   Median : 36.00
 Mean   :15.4   Mean   : 42.98
 3rd Qu.:19.0   3rd Qu.: 56.00
 Max.   :25.0   Max.   :120.00
```

When we produce a scatter plot (figure 4.1):

plot(cars)

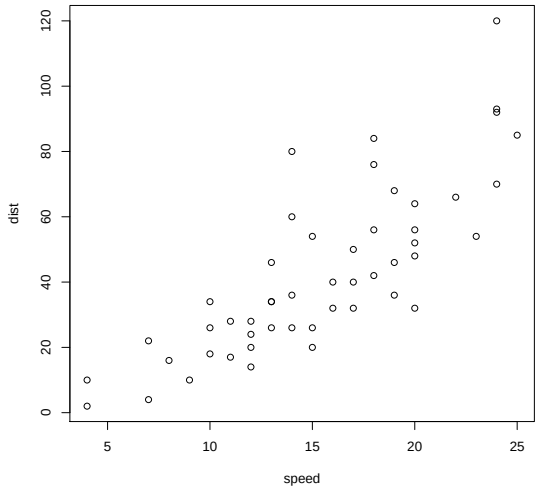


Figure 3: Scatter plot of the cars dataset.

we start to see the relationship emerging.

4.2 Linear Regression

Cast your mind back to your teenage years. Among many other life altering experiences, you will have likely encountered the equation of a straight line: $y = f(x) = mx + b$, where m is the gradient and b is the intercept with the y-axis. This is a linear model.

Fitting a linear model to your data amounts to testing the hypothesis that your observations can be explained by a linear relationship between x (the independent variable) and y (the dependent variable).

Once we have our model, we can use it to make predictions and answer questions. For example, we may observe a car travelling at 26 miles per hour (note this value is not in our training data) and may ask, how far will this car need to come to a halt if the driver steps on the brakes? We couldn't look up a corresponding speed from our training data and provide the observed stopping distance, but we can use the generalising property of our model of stopping distances (the line passes through all possible observed values) to make a prediction.

We can create our first linear model, using the least squares criterion for fitting the model very easily using the built-in R function, 'lm':

```
res.lm=lm(dist ~ speed, data=cars)
```

The ' ' notation indicates that we want to model of the relationship between the dependent (or response, or predicted) variable, to the left, and the independent (or stimulus, or observation) variable, to the right, given the built-in dataset called 'cars'.

The least squares criterion boils down to minimising the vertical distance between all the points in the dataset and the model, i.e a straight line. The result is the line of best fit *according to that criterion*.

We can plot the model (line) on top of the data (figure 4.2):

```
plot(cars)
abline(res.lm)
```

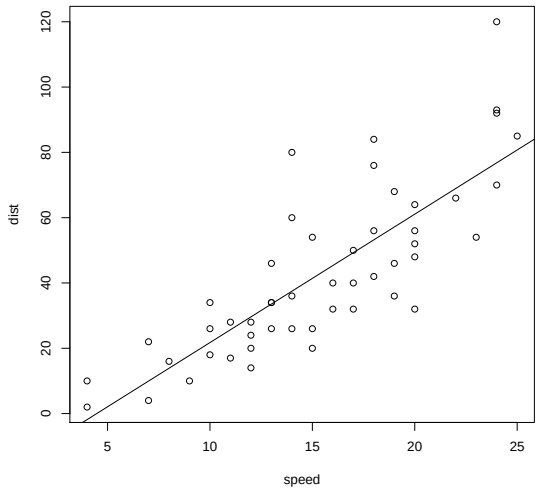


Figure 4: Scatter plot of the cars dataset overlaid with a line of best fit calculated using a least squares criterion.

It is not good practice to use any model blind, and we should always assess the quality of our fitted model. The 'summary' function helps us again here:

```
> summary(res.lm)
Error in summary(res.lm) : object 'res.lm' not found
> res.lm=lm(dist ~ speed, data=cars)
```

```
> summary(res.lm)

Call:
lm(formula = dist ~ speed, data = cars)

Residuals:
    Min      1Q  Median      3Q     Max
-29.069  -9.525  -2.272   9.215  43.201

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -17.5791     6.7584  -2.601   0.0123 *
speed         3.9324     0.4155   9.464 1.49e-12 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 15.38 on 48 degrees of freedom
Multiple R-squared:  0.6511,Adjusted R-squared:  0.6438
F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

The R-squared metric can be read as the proportion of the variance explained by our model. The model explains the majority of our data, but there are still some outliers. A low p-value is also indicative of a good model fit.

We could, of course, fit many more complex functional forms to the cars data, in the hope of improving our assessments of model fit and we would likely be successful in that regard. However, modelling every little lump and bump in our *training* set will likely reduce the quality of fit to previously unseen data–a *test* set. This makes intuitive sense, if we recall that our training set will ineviatably contain biases and noise. Therefore we should be sanguine about assessments of quality of fit and about the complexity of model that we employ. Even when our data does not exhibit a linear relationship *exactly*, there is often utility is assuming a simple model.

The above summary also tells us the values of the gradient ($m = 3.9324$) and the intercept ($b = -17.5791$). Using these values we can begin to make predictions with our model. For example, we can predict the stopping distance for a speed outside of the range covered by our training set, such as 30 mph:

```
> 3.9*30 - 17.6
[1] 99.4
```

Aside from the measures of model fit above, a quick sanity check shows that our model is not perfect. For example at a speed of zero, we should have a stopping distance of zero. However our intercept is -17.6.

Least squares is not the only criterion for fitting a linear model and R provides us with other functions, such as 'rlm' and 'lgs' which use other optimisation criteria. They are, again, very easy to use:

```
library(MASS)
res.rlm=rlm(dist ~ speed, data=cars)
res.lqs=lqs(dist ~ speed, data=cars)
```

To read more about these alternative linear regression functions (both from the MASS library of functions), see:

rlm - robust regression https://stat.ethz.ch/R-manual/R-devel/library/MASS/html/rlm.html

lqs - resistant regression https://stat.ethz.ch/R-manual/R-devel/library/MASS/html/lqs.html

Now that we have two more linear models we can plot them all on to of our data (figure 4.2):

```
plot(cars)
abline(res.lm, lty=1)
abline(res.rlm, lty=2)
abline(res.lqs, lty=3)
legend(x=5, y=100, legend=c("lm","rlm","lqs"), lty=c(1,2,3))
```

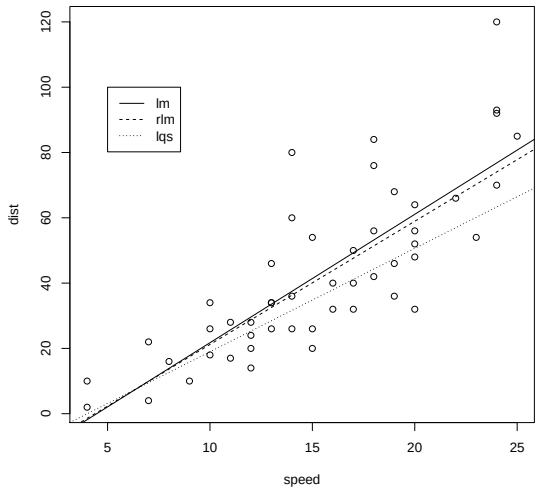


Figure 5: Scatter plot of the cars dataset overlaid with three different linear models.

Which one is the best? As a first approximation, we can say that they are all equally good. Philosophically, they are just expressions of differing opinions about what constitutes a good fit.

Examples of further reading on linear modelling in R, include:

- http://msenux.redwoods.edu/math/R/regression.php
- http://www.r-tutor.com/elementary-statistics
- http://www.r-tutor.com/elementary-statistics/simple-linear-regression

4.3 Logistic Regression

In the previous section, we were interested in building a model which could provide a continously valued prediction given some observation, such as the stopping distance required by a car travelling at some observed speed. In this section we turn our attention to the goal of being able to predict membership of a discrete class or category, againgiven some observation(s).

The logistic function is bounded in the range [0,1]. This is useful in the case where we want to predict membership of one of two classes, as the value of the logistic function can be interpreted as the probability of belonging to one of those two classes. A number of interesting problems can be framed as two category predictions. An example is classifing email as spam or not spam. Building a two category classifier using the logistic function is called logistic regression.

Another built-in data set, called 'mtcars', will provide useful data for us as we build our model. The data set contains, amongst other things, the recorded fuel efficiency and the kind of transmission–manual or automatic–for a number of cars. When we have fitted the logist function to this data set we'd like to be able to predict whether a car has a manual or automatic transmission, solely from the recorded fuel efficiency.

Let's make a scatter plot using the relevant columns from 'mtcars':

```
dat <- mtcars[c("mpg", "am")]
plot(dat)
```

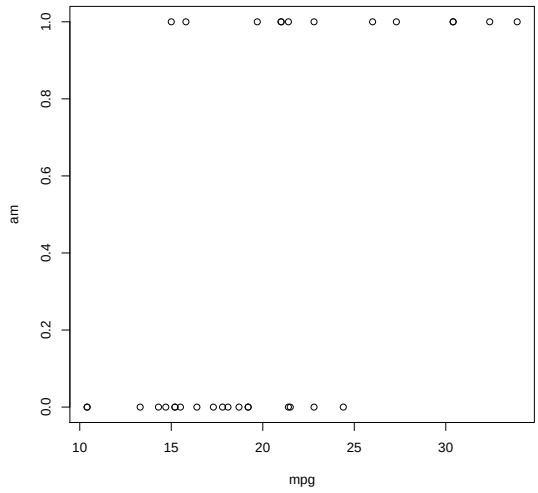


Figure 6: Scatter plot of mpg vs am from the built-in mtcars data set. A value of 1.0 on the am axis indicates an automatic transmission. A value of 0.0 indicates a manual transmission.

As with linear regression, fitting the model to the data requires only a simple call to the 'glm' function:

```
res.glm=glm(am~mpg, family=binomial, dat)
```

To learn more about the 'glm' function, see:

glm - generalized linear model https://stat.ethz.ch/R-manual/R-devel/library/stats/html/glm.html

A number of different functional forms can be used with a generalized linear model. Choosing 'binomial' as the family means that we will be performing logistic regression.

Let's plot the outcome of the model fitting:

```
plot(dat,xlab="Fuel efficiency (miles/gallon)", ylab="Probability of manual trans
curve(predict(res.glm,data.frame(mpg=x),type="resp"),add=TRUE)
points(as.vector(mtcars[["mpg"]]),fitted(res.glm),pch=20)
```

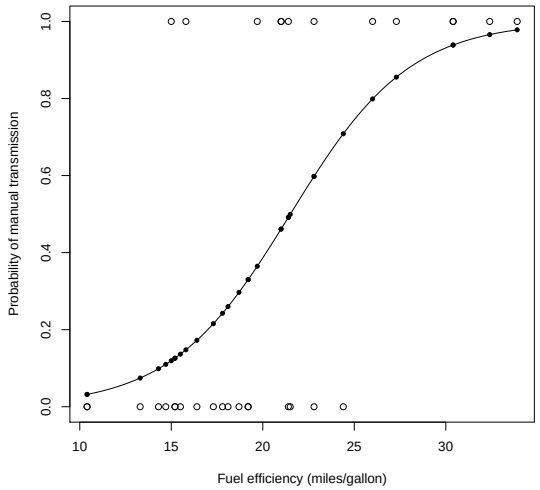


Figure 7: The logistic function as fitted to mpg vs am from the mtcars data set.

Slope is an intuitive means to assess classifier. If the training data is completely unseperable (i.e. class membership cannot be predicted from the observations) then the slope will be close to horizontal. This means that the probability of belonging to class X is pretty much each across all observational values (i.e. the observations provide no new information). On the other hand, if the slope is close to vertical, then the data is highly seperable based on the observations and classification accuracy will be high. We can see that fuel efficiency observations for the two categories of manual or automatic transmission overlap a good deal in our training data and so the slope is neither vertical nor horizontal and instead is somewhere between the two. We would expect classification accuracy to be neither good nor bad.

Again we can use the 'summary' function to tell us about the model fit:

```
Call:
glm(formula = am ~ mpg, family = binomial, data = dat)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5701  -0.7531  -0.4245   0.5866   2.0617
```

```
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -6.6035     2.3514  -2.808  0.00498 **
mpg           0.3070     0.1148   2.673  0.00751 **
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 43.230  on 31  degrees of freedom
Residual deviance: 29.675  on 30  degrees of freedom
AIC: 33.675

Number of Fisher Scoring iterations: 5
```

4.4 k-Nearest Neighbours

In the previous section we looked at a form of classifier that is suitable for a problem containing two classes. However, what if your problem involves more than two classes? In that case we'll need a model or algormithm which we can use to make prediction for an arbitrary number of classes. One such algorithm is the k-nearest neighbour (kNN) algorithm. Following the example on the R manual's kNN page:

k-nearest neighbours http://stat.ethz.ch/R-manual/R-devel/library/class/html/knn.html

The manual page gives us a concise description of the algorithm:

"k-nearest neighbour classification for test set from training set. For each row of the test set, the k nearest (in Euclidean distance) training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the kth nearest vector, all candidates are included in the vote."

We'll avail ourselves of another of R's built-in datasets. This time Edgar Anderson's observations of various aspects (petal and sepal length and width) of three species of Iris (https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/iris.html).

We can examine the relationship between these various observations and membership of species classes using the following scatter plot:

Now, let's divide the available data into training and test sets and then train and evaluate our kNN classifier using the relevant portions.

```
library(class)
train <- rbind(iris3[1:25,,1], iris3[1:25,,2], iris3[1:25,,3])
test <- rbind(iris3[26:50,,1], iris3[26:50,,2], iris3[26:50,,3])
cl <- factor(c(rep("s",25), rep("c",25), rep("v",25)))
iris3.knn <- knn(train, test, cl, k = 3, prob=TRUE)
```

The 'knn' function is in the 'class' library. Since we have cunningly split our data into two equally sized sets, we can re-use a vector of class lables ('cl') both when we train and test our model. The function factor is used to encode a vector as a factor–the terms category and

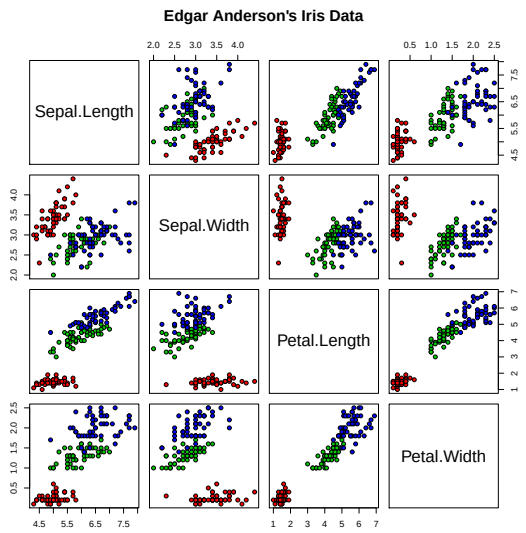


Figure 8: Pair-wise scatter plots of iris data set. Red, green and blue dots are used for examples of the three different Iris species.

enumerated type are also used for factors. Setting 'prob=true' means that the proportion of the votes for the winning class are returned as attribute 'prob'.

A clear way to inspect the performance of our model is to create a table of class predictions against the actual known classes for observations from the test set:

```
table(predicted=iris3.knn, actual=cl)
```

```
         actual
predicted  c  s  v
        c 23  0  4
        s  0 25  0
        v  2  0 21
```

We can see that all instances of the species *setosa* are correcly classified. There are 4 errors classifying instances of *virginica* and two errors for instances of *versicolor*–6 errors out of 25 in total.

kNN has problems if we use features along many axes for each data point. Also if the scale of these axes are radically different. From this we see the importance of the topics of feature selection and dimensionality reduction (see the next section). Other approaches to classification include decision trees, random forests and ANNs etc.

An interesting comparision of decision boundary shapes which can be formed using different forms of classification models is given on http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html.

4.5 Principle Component Analysis

In the previous sections we have seen a progression from linear regression, to logistic regression for simple classifiers and onto the k-nearest neighbour algorithm for classificaion tasks

involving more than two categories. In all these cases we used the observations without alteration. However the topic of 'feature selection' is a very important one and the features, or observations, that we construct can have a significant bearing on the predictive power of our models. For this reason I want to include a section of Principle Component Analysis, as an example of feature selection.

Wikipedia's entry on Principle Component Analysis (PCA) (https://en.wikipedia.org/wiki/Principal_component_analysis) tells us:

"Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set. The principal components are orthogonal because they are the eigenvectors of the covariance matrix, which is symmetric. PCA is sensitive to the relative scaling of the original variables."

Our goal in this section is to investigate classification accuracy again using the kNN algorithm on the Iris data, but this time we will transform (and subsample) our data using PCA. We'll first take the log of the values, as this is often recommended for all +ve observations, such as length measurements, (e.g. http://www.researchgate.net/post/What_is_the_best_way_to_scale_parameters_before_running_a_Principal_Component_Analysis_PCA) and then compute the principle components themselves using the 'prcomp' function:

```
log.ir <- log(iris[, 1:4])
ir.pca <- prcomp(log.ir, center = TRUE, scale. = TRUE)
```

Again, we can use 'summary' to inspect our 'model'. Indeed PCA can produce a valuable model in it's own right. For example, we may see an effect but are unsure which of many variables are responsibe. PCA can help us latent variables which 'explain' the effects.

```
summary(ir.pca)
```

```
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.7125 0.9524 0.36470 0.16568
Proportion of Variance 0.7331 0.2268 0.03325 0.00686
Cumulative Proportion  0.7331 0.9599 0.99314 1.00000
```

We can see also see how much of the variablity is explained by each of the components using a plot:

```
plot(ir.pca, type = "l", main = "Principle Components")
```

Now, we apply kNN to our transformed data. We'll use the training and test sets as defined previously and only the first two principle components:

```
log.train <- log(train)
train.pca <- prcomp(log.train, center = TRUE, scale. = TRUE)
log.test <- log(test)
```

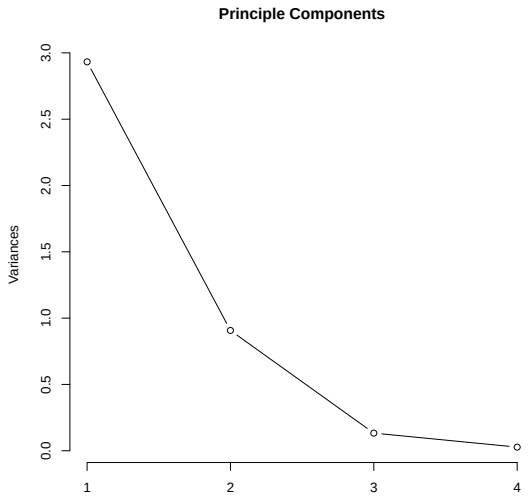


Figure 9: Principle componets which explain the variance in the iris data set.

```
test.pca <- prcomp(log.test, center = TRUE, scale. = TRUE)
iris3.pca.knn <- knn(train.pca$x[,1:2], test.pca$x[,1:2], cl, k = 3, prob=TRUE)
```

where, [,1:2] takes the first and second principle components of the projected data set, i.e.

```
> train.pca$x[,1:2]
               PC1         PC2
 [1,] -2.378951933 -0.29586612
 [2,] -2.199765498  0.70896486
...
```

rather than

```
> train.pca$x
               PC1         PC2          PC3          PC4
 [1,] -2.378951933 -0.29586612  0.214161332 -0.005070089
 [2,] -2.199765498  0.70896486  0.356350930  0.077571403
...
```

Our results this time are:

```
table(predicted=iris3.pca.knn, actual=cl)
```

```
         actual
predicted  c  s  v
        c 21  0  3
        s  0 25  0
        v  4  0 22
```

Again we see perfect classification for *setosa*. Three errors for *virginica* and four errors for instances of *versicolor*. 7 errors out of 25 is only marginally more than the 6 errors we had last time.

In general feature selection, such as dimensionality reduction, can offer use several things, including:

- It can aid the generalisation of a classifier on new data (i.e. guard against overfitting). (Approaches which enforce smoothness constraints, such as regularisation can also help avoid overfitting to the training data.)
- It can make a classification possible at all, for a task which would otherwise need too much training data.

For further reading on PCA in R see, e.g.:

- http://www.r-bloggers.com/computing-and-visualizing-pca-in-r/

5 Tools

Moving on from the sections on methods. We now have a short section on tools. There are many (often)tools available which you can use in your data science work. However, you'll come to appreciate that they implement similar methods. This is why I placed an emphasis on methods and why I have merely included a list of tools. Many of the tools come with their own, very good, documentation. Rather than reproduce that, I thought it was better to provide examples of using the methods, which you could, in turn recreate with your preferred tool.

R A great place to start (the basis for most of the examples in this document). https://www.r-project.org/

Python (scikit-learn) http://scikit-learn.org/stable/

Weka http://www.cs.waikato.ac.nz/ml/weka/ (and RWeka, https://cran.r-project.org/web/packages/RWeka/index.html.

Mahout http://mahout.apache.org/

various databases e.g. http://neo4j.com/, http://couchdb.apache.org/

Apache Spark http://en.wikipedia.org/wiki/Apache_Spark

6 Communicating Your Findings

Sooner or later you will need to communicate your results and (hopefully) impress someone. Two new initiatives that enable you to store document quality text alongside your code are:

R Markdown http://rmarkdown.rstudio.com/

IPython Notebook http://ipython.org/notebook.html

Attractive properties of both of the above include:

Reproducable The code can be directly run by someone else.

Versioning The combined documents can be kept under version control.

Rich Media Code can be embedded alongside explanatory text and graphics.

This document was written using LaTeX.