Twitter & Sentiment Analysis (Text analytics)

Tool: Bag of words model
Logistic regression, CART, Random forest

The Analytics Edge:

Sentiment analysis from massive amounts of unstructured data can be automated using tools of analytics by replacing the more common method of polling users on their opinions. New mediums of communication such as Twitter, Facebook give companies and organizations a chance to understand sentiments using tools of analytics.

Overview:

Twitter was set up in 2006 and is a social networking and communication website.
The service enables user to send and read short 140 character messages called "tweets".
The site generates large number of tweets each with a fairly short length. The service as of May 2015 has more than 500 million users. Twitter valuation is over 20 billion dollars today and is ranked as one of the top 10 most visited websites by web traffic

alysts.

Microblogging

Effect of Twitter

Twitter is used by celebrities to reach out to their fans and followers.

Twitter is used by companies to communicate with their customers, hear their thoughts and understand trends.

Twitter (and more generally social networking sites) play an important role in modern day activism such as the Arab Spring.

"Twitter" mood has been shown to have a predictive power on stock market prices.

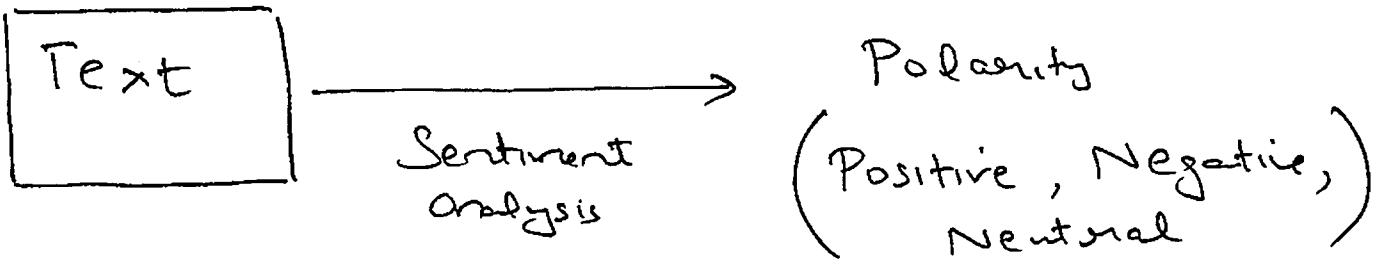
Twitter has been used to predict box office performance after movies have been released based upon how many times films are mentioned in tweets

Websites: www.sentiment140.com

R provides a package "twitteR" that can be used to read in tweets directly from Twitter into R for text analytics.

Sentiment analysis refers to the use of text analytics, natural language processing, computational linguistics to identify and extract subjective information in source materials.

The basic task in sentiment analysis is to classify the polarity (sentiment) of the text



Challenges in text analytics (tweets)

Tweets are textual data, typically with poor spellings (short-forms), use of non traditional grammar and also multilingual

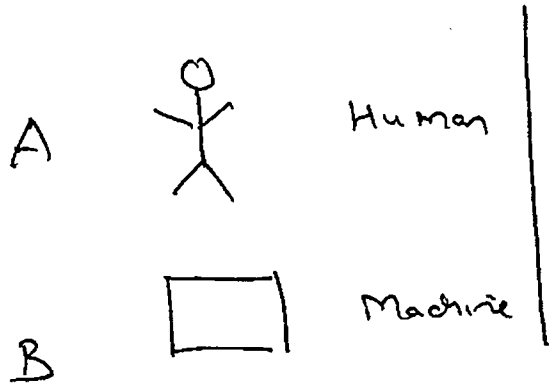
For example:

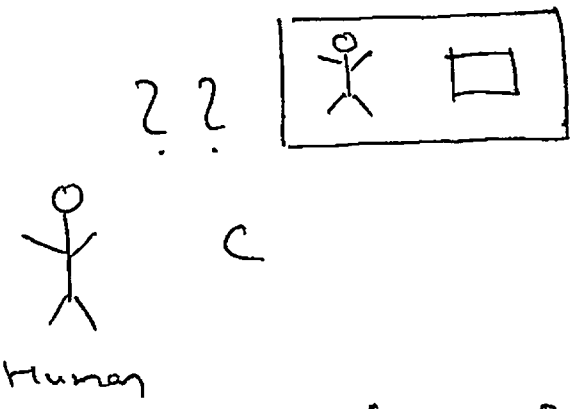
U say dat iphone 5s didnt bring anything new 2?

Turing Test

The Turing test was introduced by Alan Turing in his 1950 paper "Computing Machinery & Intelligence".

This is a test of a machine's ability to exhibit intelligent behavior that is indistinguished from a human. Turing proposed that the human evaluator would judge between natural language conversations with a human and a machine. If the evaluator cannot reliably tell the machine from the human using a text only channel then the machine passes the test





To determine if A or B is human

Why is text analytics hard?

Ambiguity in the language of English that sometimes even humans can't decipher sentences completely.

Examples:

1) John saw the man on the mountain with a telescope.

Who has the telescope? John, Man on the mountain, mountain?

2) John and Mary took two trips around France. They were both wonderful.

They refers to? John and Mary
Two trips

3) Medicine helps dog bite victim.

Does the medicine help the dog to bite the victim or does it help the victim who is bitten by the dog?

Working with data from Twitter

Twitter data can be collected using an interface called API.

To predict the sentiment of tweets, it is important to have the tweets in a training set to be categorized as per the sentiment

There are a few ways this can be done:

1) One carefully goes through every tweet in the training set and provides a sentiment polarity for the tweet.

2) One can use centralized work places such as Amazon Mechanical Turk where small tasks can be assigned to individuals who work remotely and do the sentiment categorization for a few tweets at a small price.

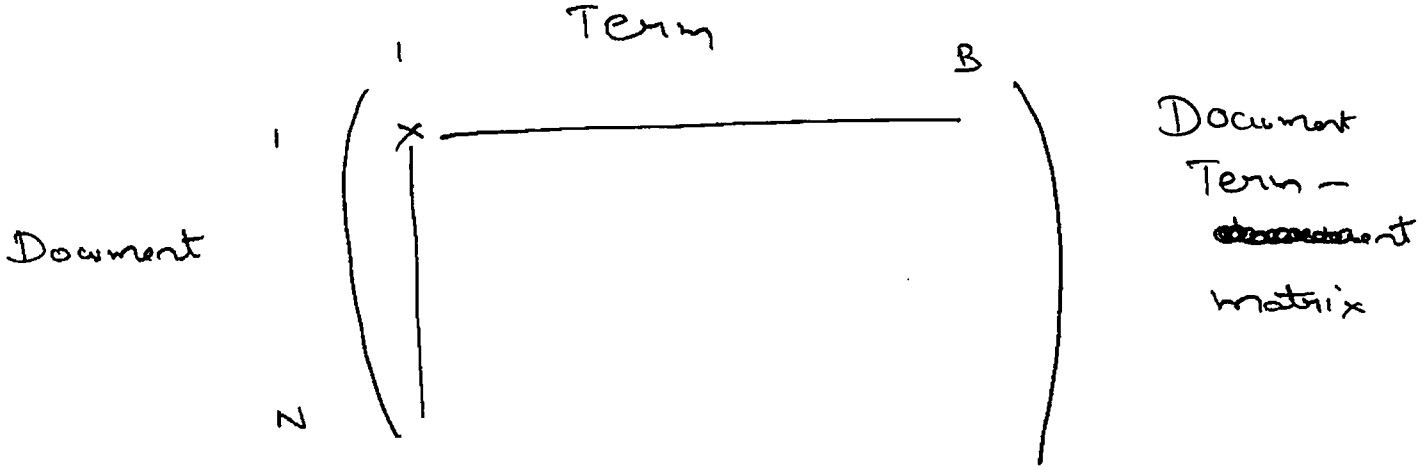
3) One can use in tweet if emoticons are present such as :) or :(: to categorize tweets.

Bag of words model

Bag of words is a simple approach to represent text.

Terms (Words)	1	. . .	B
Documents	1	. . .	N

Each document contains sentences. In the bag of words model the text is represented as a bag of words, disregarding the grammar and word order but keeping multiplicity. (Order is not maintained in this model).



document-

Each element in the term – document matrix represents a measure of the frequency of occurrence of the term (word) in the document.

Example: John likes to watch movies. Mary likes movies too.
John also likes to watch football.

	John	likes	to	watch	movies	also	Mary	football	too
1	1	2	1	1	2	0	1	0	1
2	1	1	1	1	0	0	0	1	0

Pre-processing

Data is often unstructured and hence preprocessing needs to be done.

For example:

1) Stopwords are words that are filtered out in processing text.

These refer to the most common words in the language. For example "the".

Example: Dharma and Greg rocks

Here in a bag of words model, the stopword "and" would be removed. However in context it might talk about the show "Dharma and Greg" where and is part of the show name

It is possible to use ngrams (bigram, 3-grams etc: see Google Ngram Viewer) in such cases.

We will not use these features in this work though.

2) Removing punctuations, converting Upper case to lower case. These are other types of preprocessing commonly done.

3) Stemming : The Porter stemming algorithm is used to remove inflected words to their word stem, base or root form.

Example : "cats" (and possibly "catty") should be identified with the root "cat"

Martin Porter in 1980 invented the Porter Stemmer, one of the most common algorithms for stemming English.

"revive", "revival" would be stemmed to

"reviv"

The stem itself might not be a valid word but it is closely related.

In R, the preprocessing steps can be done using packages such as the "tm" (text mining package)

Analytics on dataset

```
twitter <- read.csv("twitter.csv",
                    stringsAsFactors = FALSE)
```

This helps read in the text data and ensures that the string is read in as a character vector. We need to use this whenever we are doing text analytics

```
str(twitter)
head(twitter)
summary(twitter)
```

This consists of 2664 observations of 2 variables

$Sentiment : Takes values 1, 2, 4, 5 where

5 = very positive
4 = slightly positive
2 = slightly negative
1 = very negative

The tweets were classified by the contributors.

$tweet : Text of tweet

The data comes from the website www.crowdflower.com/data-for-everyone

CrowdFlower is a crowdsourcing data management company where it allows users to access online workforce of millions of people to clean, label & enrich data.

```
twitter$Negative <- as.factor(twitter$Sentiment <= 2)
table(twitter$Negative)
```

FALSE	TRUE
1800	775

} 29.09 % of the tweets were labeled as negative

Install.packages("tm")
library(tm)

Load a text mining package in R.

Preprocessing textual tweets data

Corpus ← Corpus(VectorSource(twitter$tweet))

Corpus represents a collection of documents (tweets in this example). The VectorSource helps interpret each element of the twitter$tweet object as a document while the Corpus command helps represent it as a corpus.

Corpus

Here the corpus consists of 2664 documents (tweets)

as.character(Corpus[[1]])

"Two places I'd invest all my money if I could: 3D printing and Self-driving Cars!!!"

as.characters(corpus[[2664]])

"I'd never trust a self-driving car to take my kid to school."

Note that this is a developmental package & there could be some variations based on the computers you run the code on.

corpus ← tm_map(corpus, to lower)

The function tm_map(·) applies mapping (transformation) to the corpus. Here the goal is to convert all the text to lower case.

as.character(corpus[[1]])

"two places i'd invest all my money if i could: 3d printing and self-driving cars!!!"

Stopwords("english")

The list of stopwords in english in the tm package.

Examples: "i", "me", "our", "themselves" "too" "i'd" (A total of 174)

Corpus ← tm_map(corpus, removeWords, stopwords("english"))

This remove stop words in english from corpus.

as.character(corpus[[1]])

"two places invest money: 3d printing self-driving cars!!!"

as.character(corpus[[26647]])

"never trust self-driving car take kid school"

```
corpus <- tm_map(corpus, removeWords,
                 c("drive", "driving", "driver",
                   "self-driving", "car", "cars"))
```

Removes words such as drive, car since many of the tweets possibly contain these words and it is most possibly not going to be a predictor of the polarity of tweets

```
as.character(corpus[[1]])
```

"two places invest money : 3d printing !!!"

```
as.character(corpus[[2664]])
```

"never trust take kid school."

```
corpus <- tm_map(corpus, removePunctuation)
```

Removes punctuation marks

```
as.character(corpus[[1]])
```

"two places invest money 3d printing"

```
as.character(corpus[[2664]])
```

"never trust take kid school"

Corpus ← tm_map (Corpus, StemDocument)

This stems the words in text using Porter's stemming algorithm. Note that You might need to load the Snowball C package to use this in some cases.

as.character (corpus [[1]])

"two place invest money 3d print"

as.character (corpus [[3]])

"autonom vehicl reduc traffic fatal 90"

freq ← DocumentTermMatrix (corpus)

freq

This creates a document term matrix from the text corpus. We have 2664 documents (rows) & 5620 terms (columns). Out of the 2664 x 5620 terms, 22170 terms are nonzero while the remaining are zero. (sparsity is close to 100%.)

Inspect (freq[1,])

This inspects information in document-term matrix for first document. Entries of 1 for terms invest, money, place, print, two

findFreqTerms (freq, lowfreq = 50)

This finds terms with a frequency of 50 (occurs in 50 places atleast). There are

freq[, "accid"]

There are 63 nonzero entries & 2601 zero entries for accid.

freq ← removeSparseTerms(freq, 0.995)

This removes sparse terms from the document term matrix, In this example, the terms from the matrix are removed which have atleast 99.5% empty entries.

freq is now a document-term matrix with 2664 documents and 265 terms.

twitterSparse ← as.data.frame(as.matrix(freq))

Converts the Document Term matrix to a matrix and then to a dataframe

Colnames(twitterSparse) ← make.names(Colnames(twitterSparse))

This helps to ensure that valid names are there for columns. For example to change variable names starting with numbers is modified.

twitterSparse$Neg ← twitter$Neg

Adds the output variable (polarity of tweet) to new dataframe

```
library(caTools)
set.seed(123)
```

Create training and test set

Ensure caTools is installed

```
spl ← sample.split(twittersparse$Neg, SplitRatio = 0.7)
train ← subset(twittersparse, spl == TRUE)
test ← subset(twittersparse, spl == FALSE)
```

Running different predictive algorithms

Baseline model (Predict most occuring case in training set on test set)

```
table(train$Neg)
FALSE  TRUE
 1322   542
```

Majority in training set are tweets with positive polarity (sentiment).

Accuracy = 0.709.

```
table(test$Neg)
FALSE  TRUE
  567   233
```

Accuracy of baseline model

$$= \frac{567}{567+233} = 0.708$$

Note the accuracy in test set of the baseline model is not surprising since we created the traning and test set to be balanced.

Logistic regression

```
model1 <- glm(Neg ~ ., data = train,
              family = binomial)
```

Summary(model1) AIC = 2006.4

A quick check on sign of coefficients

accid	1.161	(Occurence of accident related words in tweets ⇒ greater chance of negative sentiment)
amaz	-1.748×10	(Occurence of amaze related words in tweets ⇒ lesser chance of negative sentiment)
awesom	-3.144	(Occurence of awesome related words in tweets ⇒ lesser chance of negative sentiment)
dont	5.192	(occurence of dont related words in tweets ⇒ greater chance of negative sentiment)

These seem to be reasonable. Note the warning message that some of the fitted probabilities are close to 0 or 1. This can be an issue in estimating parameters but we can still use the model

```
predict1 <- predict(model1, newdata = test,
                    type = "response")

table(predict1 >= 0.5, test$Neg)
```

Predict	Actual: False	Actual: True
False	429	123
True	88	110

$$\text{Accuracy} = \frac{429+110}{429+110+123+88} = \underline{0.736}$$

Note that though we have removed words such as drive, driving, driver in developing the corpus. Using the tm_map(·) with removeWords command, in the final corpus this word such as "drive" and "driver" still exists

A closer look at these tweets reveal:

which (twitterSparse[,"drive"] != 0)

This provides indices of tweets that still have drive stem in it

twitter[11,2]

" ....."

twitter[134,2]

" ..."

"drives"

"drives"

} We can also remove drives in applying tm_map

which (twitterSparse[,"driver"] != 0)

twitter[83,2]

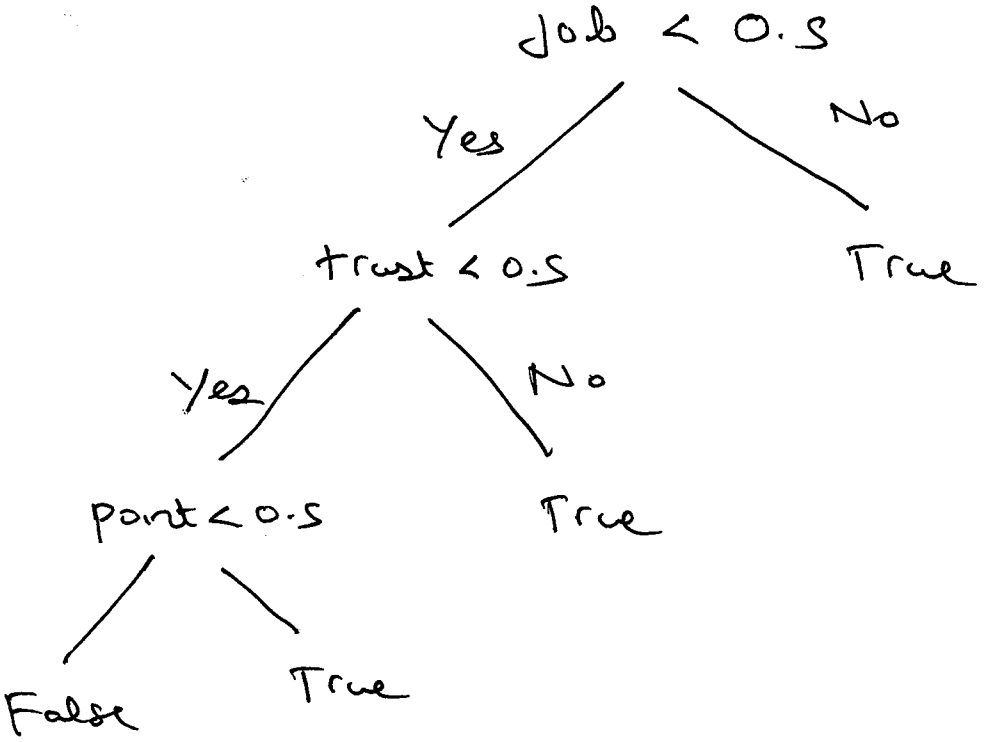
" driverless"

twitter[209,2]

" .... drivers"

CART

```
library (rpart)

Set.seed (111)

model 2 <- rpart (Neg ~ . , data = train)

Summary (model 2)

library (rpart.plot)

prp (model 2)
```



Here True indicate
Neg = 1 and
False indicates Neg = 0

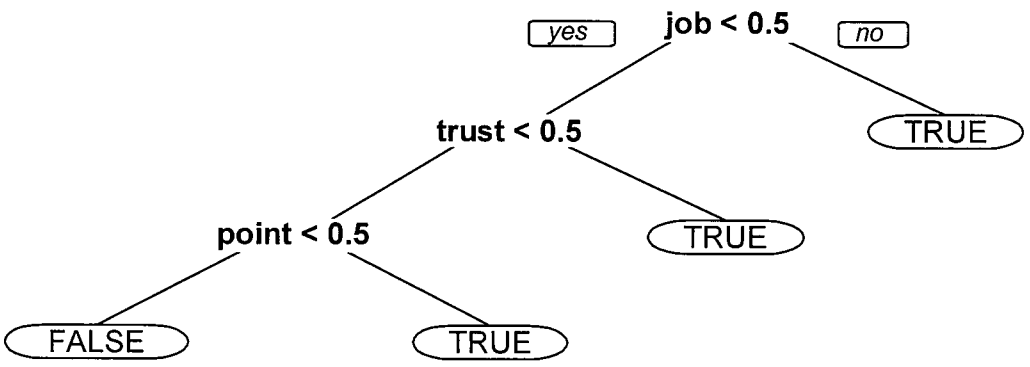
Note that in logistic regression, Job has a +ve coefficient (highly Significant)

```
predict 2 <- predict (model2, newdata = test, type = "class")

table (predict 2, test $ Neg)
```

	False	True
False	560	217
True	7	16

Accuracy = 0.72

You can also try & reduce cp with


yes
job < 0.5
no
trust < 0.5
TRUE
point < 0.5
TRUE
FALSE
TRUE

Set.seed(123)

model2a ← rpart(Neg ~ ., data = train, cp = 10^{-6})

Uses cp = 10-6 minimum value to do cross-validation

prp(model2a)

This grows a much deeper tree.

printcp(model2a)

We use the smallest value of xerror to choose where to prune the tree.

A reasonable value here is 0.0009 (you can change a bit this value)

model2b ← prune.rpart(model2a, cp=0.0009)

prp(model2b)

This is a smaller tree than model2a

predict2b ← predict(model2b, newdata = test, type = "class")

table(predict2b, test$Neg)

	False	True
False	520	180
True	47	53

Accuracy = 0.71

Random forest

```
library(randomForest)
```

Ensure randomForest package is installed

```
set.seed(112)
model3 <- randomForest(Neg ~ ., data = train)
```

We use default parameters to fit the randomForest model (500 trees). This will take more time to solve since our dataset is larger in this example.

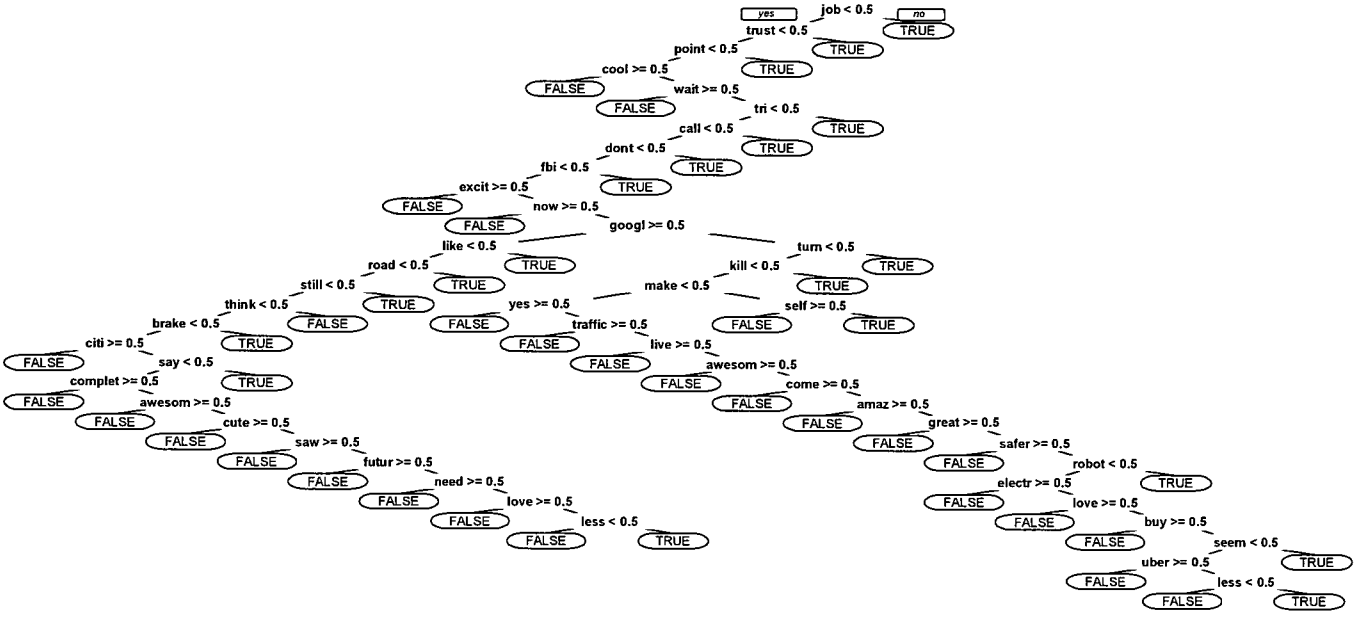
```
Summary(model3)
```

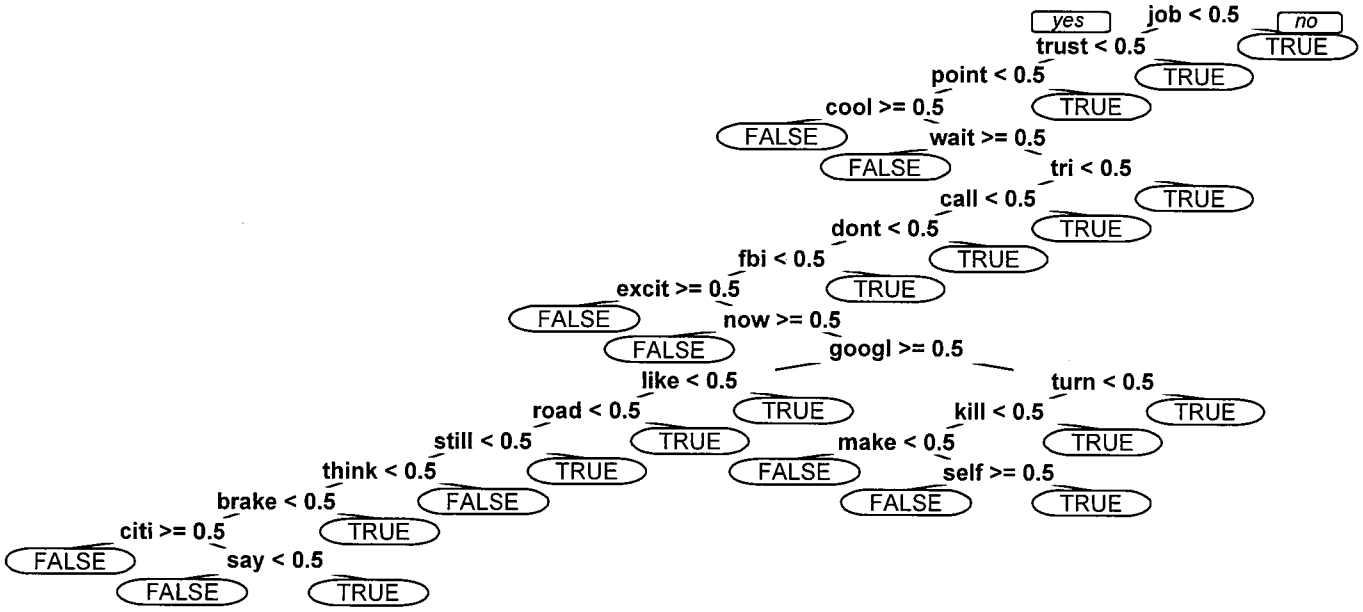
In the training set the accuracy is 0.732

```
predict3 <- predict(model3, newdata = test,
                    type = "class")
table(predict3, test$Neg)
```

	False	True
False	498	133
True	69	100

Accuracy = 0.7475


yes
job < 0.5
no
trust < 0.5
TRUE
TRUE
point < 0.5
TRUE
cool >= 0.5
FALSE
wait >= 0.5
FALSE
tri < 0.5
TRUE
call < 0.5
TRUE
dont < 0.5
TRUE
fbi < 0.5
TRUE
excit >= 0.5
FALSE
now >= 0.5
FALSE
googl >= 0.5
like < 0.5
TRUE
turn < 0.5
TRUE
road < 0.5
TRUE
kill < 0.5
TRUE
still < 0.5
TRUE
make < 0.5
TRUE
think < 0.5
FALSE
yes >= 0.5
FALSE
self >= 0.5
TRUE
brake < 0.5
TRUE
traffic >= 0.5
FALSE
FALSE
live >= 0.5
citi >= 0.5
FALSE
FALSE
awesom >= 0.5
say < 0.5
FALSE
come >= 0.5
complet >= 0.5
TRUE
FALSE
amaz >= 0.5
awesom >= 0.5
FALSE
FALSE
great >= 0.5
cute >= 0.5
FALSE
FALSE
safer >= 0.5
saw >= 0.5
FALSE
FALSE
robot < 0.5
futur >= 0.5
FALSE
TRUE
electr >= 0.5
need >= 0.5
FALSE
FALSE
love >= 0.5
love >= 0.5
FALSE
buy >= 0.5
less < 0.5
FALSE
FALSE
seem < 0.5
TRUE
uber >= 0.5
FALSE
less < 0.5
FALSE
TRUE


yes
job < 0.5
no
trust < 0.5
TRUE
point < 0.5
TRUE
cool >= 0.5
TRUE
FALSE
wait >= 0.5
FALSE
tri < 0.5
call < 0.5
TRUE
dont < 0.5
TRUE
fbi < 0.5
TRUE
excit >= 0.5
TRUE
FALSE
now >= 0.5
FALSE
googl >= 0.5
like < 0.5
TRUE
turn < 0.5
road < 0.5
TRUE
kill < 0.5
TRUE
still < 0.5
TRUE
make < 0.5
TRUE
FALSE
think < 0.5
TRUE
FALSE
self >= 0.5
FALSE
TRUE
brake < 0.5
TRUE
FALSE
citi >= 0.5
FALSE
say < 0.5
TRUE
FALSE