Social Media Mining

Archit Gupta

September 17, 2016

Contents

Social Media Mining

Social media mining can and should have broad interpretation. It is not the intent to confine social media mining to sentiments or opinions, but rather we suggest that a sentiment or opinion is a useful tool for many research pursuits. Until recently, sentiment was understood as a ubiquitous and constant part of the human experience, with variations in sentiments changing only slightly up or down. Klaus Scherer (2000) developed a working definition as follows:

"Emotions (sentiments) are episodes of coordinated changes in several components in response to external and internal events of major significance to the organism."

It is our intent to understand, measure, and interrelate these changes in a sentiment. Scherer's typology of emotions is a useful grounding point for the understanding of sentiments, and as a jumping-off point for a discussion of the difficulty in measuring sentiment-laden text.

Scherer's typology of emotions

Scherer's typology of emotions is briefly explained as follows:

- Emotion: This is a brief, organically synchronized evaluation of a major event, for example, being angry, sad, joyful, ashamed, proud, or elated
- Mood: This is a diffused, non-caused, low-intensity, long-duration change in subjective feeling, for example, being cheerful, gloomy, irritable, listless, depressed, or buoyant
- Interpersonal stance: This is an affective stance towards another person in a specific interaction, for example, being friendly, flirtatious, distant, cold, warm, supportive, or contemptuous
- Attitude: This is enduring, affectively colored beliefs or dispositions towards objects or persons, for example, being liking, loving, hating, valuing, or desiring
- Personality traits: These are stable personality dispositions and typical behavior tendencies, for example, being nervous, anxious, reckless, morose, hostile, or jealous

Generally, when we try to measure a sentiment, we talk about Scherer's emotions; though, in some situations, we might try to capture longer-term phenomena such as moods.

Sentiment polarity - data and classification

Social media mining primarily involves the following two steps:

1. Identifying and retrieving content related to the topic of interest.
2. Measuring the polarity of each datum.

The first step, message retrieval, requires some a priori insight into the topic of interest. The goal of message retrieval is to seek out only the messages or pieces of text that contain sentiment-laden content related to a particular topic. This topic could be almost anything of interest, subject to the constraint that information exists about it on public social media.

To gather data, we generally collect content that contains a manually specified (set of) keyword(s). This is called the target. For example, the target for US presidential approval would use the topic keyword obama. We may wish to add context to analyses done on particular keywords by adding additional opposing or specifying keywords. For example, in addition to Trump, we could add Clinton to provide a counterpoint if we were studying the 2016 presidential election campaigns. Depending on the purpose of our analysis, we could jointly search for, say, Clinton and economy to target more specific subjects.

Topic models represent a second, more sophisticated, and potentially more thorough way of capturing bits of text that are relevant to a particular analysis. These models take very large sets of documents as their inputs and group them probabilistically into estimated topics. That is, each document is proclaimed to be a mixture of one or more topics that are themselves estimated from the data. This allows users to find texts that are related to a topic, though they may not explicitly use a particular keyword.

Social data mining is the detection of attitudes, and the easiest way to understand it is through the following structure:

sentiment = {data source, source, target, sentiment, polarity}}

The parameters are explained in detail as follows:

- Data source: This relates to understanding the source of the data; that is, is the source a sentence or an entire document? Twitter or a blog?
- Source or holder: This is the one that expresses a sentiment or an opinion,
- Target or aspect: The target or aspect is what or to whom the sentiment is directed toward.
- Type of sentiment: This is the type(s) of emotion(s) expressed, that is, like, love, hate, value, desire, and so on.
- Polarity: These are juxtapositional sentiments on a dimension, that is, positive or negative.

Supervised social media mining - lexicon-based sentiment

Lexicon-based sentiment classification is perhaps the most basic technique for measuring the polarity of the sentiment of a group of documents (that is, a corpus). Lexicon-based sentiment measurement requires a dictionary of words (a lexicon) and each word's associated polarity score. For example, a lexicon may contain the word excellent, which might have a score of positive two. Similarly, the word crummy may score negative one and a half. In the simplest implementation of lexicon-based sentiment analysis, all of the words in a document are compared to the words in the lexicon. Every time a word is used that is in the lexicon, the associated score is added to that text's overall sentiment score.

A lexicon-based sentiment often entails merely counting the opinion words from a subset of data from a particular source. This approach certainly has errors, as does perhaps all natural language processing; however, in aggregate, the lexicon-based approach has proven to be fairly robust, even when only used on subsets. Additionally, there are many possible ways to aggregate the sentiment scores of each word, but most commonly, they are simply summed up to form an overall score for a document.

Dictionary-based and corpus-based approaches augment preassembled lexicons in one of the following two ways:

- Using a dictionary (that is, synonyms and antonyms) to add keywords external to our corpus to enhance our preassembled lexicon(s)
- Using the corpus directly to add words already internal to our corpus that are keywords but are not accounted for by preassembled lexicons

Merging preassembled lexicons, dictionary-based lexicons, and corpus-based lexicons offers the best chance to successfully estimate sentiment.

Supervised social media mining - Naive Bayes classifiers

Methods to extract sentiments from documents can be broadly classified into supervised and unsupervised approaches.

Supervised methods are those that utilize data that has been tagged or labeled. In the parlance of statistics, these approaches utilize observations with both independent and dependent variables. For instance, the following Naive Bayes classifier approach involves a training dataset of documents that have already been scored as having positive or negative sentiment; a statistical model based on these forms the basis of scoring further documents.

In contrast, unsupervised learning algorithms do not require a dependent variable to be provided. For instance, the IRT-based method scales documents along a continuum of sentiments with no need to provide a labeled training set. Additionally, lexicon-based approaches mentioned earlier can also function without prelabeled observations.

The Naive Bayes classifier, in spite of its unfortunate name, turns out to be a highly useful tool for sentiment analysis. At the most general level, the Naive Bayes classifier is exactly that: a classifier. Classifiers are statistical tools that are used for, among other things, predicting which of two or more classes a new observation belongs to. In our case, we want to train our classifier to be able to distinguish documents featuring positive sentiment from those featuring negative sentiment (the two types or classes of interest). To do so, we feed the algorithm a large set of documents that arealready coded as containing positive or negative sentiments about a particular topic. Then, if all goes as planned, we can pass new documents to the model and have it predict the direction of their sentiment, or valence, for us. The downside to this and other supervised techniques is having to handcode a sufficient set of initial training data.

So, where did the naive part of the name come from, and why is this method useful in spite of its self-assumed simplicity? The goal of any classifier is to determine which class or type a new observation belongs to based on its characteristics and previous examples from both types that we have seen before (that is, from existing data). Some types of classifiers can account for the fact that the characteristics we use for this prediction may be correlated. That is, if we are trying to predict whether an e-mail is spam or not by looking at what words and phrases the e-mail contains, the words easy and money are likely to co-occur (and are likely to be indicative of spam messages). The Naive Bayes classifier does not try to account for correlations between characteristics. It just uses each characteristic separately to try to determine each new observation's class membership.

The naive assumption that all of the characteristics of an observation are unrelated is always wrong. In predicting whether or not to extend a loan to an individual, a bank may look at their credit score, whether or not they own a home, their income, and their current debt level. Obviously, all of these things are likely to be correlated. However, ignoring the correlations between predictive characteristics allows us to do two things that would otherwise be problematic. First, it allows us to include a huge number of characteristics, which becomes important. This is because in text analysis, individual words often have predictive characteristics, and documents often contain thousands of unique words. Other models have a difficult time accommodating this number of predictors. Secondly, the Naive Bayes classifier is fast, thus allowing us to use large training sets to train a model and to generate results quickly.

Unsupervised social media mining - Item Response Theory for text scaling

The techniques set out earlier for scaling or classifying sentiments in texts are fairly robust; that is, they tend to work well under a wide variety of conditions such as heterogeneous text lengths and topic breadths. However, each of these methods requires substantial analyst input, such as labeling training data or creating a lexicon. Item Response Theory (IRT) is a theory, but will be used in this text to refer to a class of statistical models that rely on that theory, providing a way to scale texts according to sentiment in the absence of labeled training data. That is, IRT models are unsupervised learning models.

Process of social media mining

Any data mining activity follows some generic steps to gain some useful insights from the data. Since social media is the central theme of this book, let's discuss these steps by taking example data from Twitter:

- Getting authentication from the social website
- Data visualization
- Cleaning and preprocessing
- Data modeling using standard algorithms such as opinion mining, clustering, anomaly/spam detection, correlations and segmentations, recommendations
- Result visualization

Getting authentication from the social website - OAuth 2.0

Most social media websites provide API access to their data. To do the mining, we (as a third-party) would need some mechanism to get access to users' data, available on these websites. But the problem is that a user will not share their credentials with anyone due to obvious security reasons. This is where OAuth comes in the picture. According to its home page (http://oauth.net/), OAuth can be defined as follows:

An open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications.

Let's look at the various steps involved:

1. The client accesses the web app with the button Login via Twitter (or Login via LinkedIn or Login via Facebook).
2. This takes the client to an app, which will authenticate it. The client app then asks the user to allow it the access to his/her resources, that is, the profile data. The user needs to accept it to go the next step.
3. The client is then redirected to a redirect link via the authenticating app, which the client app has provided to the authenticating app. Usually, the redirect link is delivered by registering the client app with the authenticating app. The user of the client app also registers the redirect link and at the same time authenticating app also gives the client app with client credentials.
4. Using the redirect link, the client contacts the website in the client app. During this step, a connection between authenticating app and client app is made and the authentication code received in the redirect request parameters. So, an access token is returned by the authenticating app.

Differences between OAuth and OAuth 2.0

Here are some of the major differences:

- More flows in OAuth 2.0 to permit improved support for non-browser based apps
- OAuth 2.0 does not need the client app to have cryptography
- OAuth 2.0 offers much less complicated signatures
- OAuth 2.0 generates short-lived access tokens, hence it is more secure

- OAuth 2.0 has a clearer segregation of roles concerning the server responsible for handling user authorization and the server handling OAuth requests

Data Visualization

A number of visualization R packages for text data are available as R package. These libraries, based on available data and objective, provide various options varying from simple clusters of words to the one inline with semantic analysis or topic modeling of the corpus. These libraries provide means to better understand text data.

- wordcloud : One of the simplest and most frequently used visualization libraries is the simple word cloud. The basic intent to using word cloud is to visualize the weights of the words present. The "wordcloud" R library helps the user get an understanding of weights of a word/term with respect to the tf-idf matrix.
- sentiment analysis wordcloud : the user get an understanding of what kind of sentiments are present and why the particular documents (collection of tweets) are of a particular nature (joy, sadness, disgust, love, and so on.). Timothy Jurka developed one such package.The two main functions of this package are *Classify emotion* and *Classify polarity*

The most commonly used visualization library for Facebook data is *Gephi*.The key difference between Facebook and Twitter is the richness of the profile of a user and the social connections one shares on Facebook. Gephi helps users visualize both of the distinctions in a very pleasant way. It enables a user to understand the impact one Facebook profile has, or could have, over the network. Gephi is highly customizable and user-friendly library.

Preprocessing and cleaning in R

Preprocessing and cleaning are the very basic and first steps in any data-mining problem. A learning algorithm on a unified and cleaned dataset cannot only run very fast, but can also produce more accurate results. The first steps involve the annotation of target data, in the case of classification problems and understating the feature vector space, to apply an appropriate distance measure for clustering problems. Identification of noise samples and their clean up is a very tricky task but the better it's done, the more accuracy one can expect in the results. As mentioned previously, you need to be careful in cleaning tasks as this can lead to a rejection of good samples. Furthermore, the preprocessing steps need to be a reversible process because at the end of the exercise, the results need to be processed back to the original sample space for it to make sense.

An example of social media mining

Twitter

- What are people talking about right now?
- Mining entities from user's tweets
- Sentiment analysis

Facebook

- Gender analysis of Facebook post likes
- Analysis of Facebook friends network
- Inferring community behavior dynamically
- Fraud prevention
- Questions such as "Who influences whom?"
- Getting peoples' interest based on their chat history, such as with whom they are chatting, what they are chatting, where they are chatting, and so on.

CaseStudy : Twitter sentiment analysis on Taxi Aggregators of India

Depending on the objective, and based on the functionality to search any type of tweets from the public timeline, one can always collect the required corpus. For example, you may want to learn about customer satisfaction levels with various cab services, which are up and coming in the Indian market. These start-ups are offering various discounts and coupons to attract customers, but at the end of the day, the service quality determines the business of any organization. These start-ups are constantly promoting themselves on various social media websites. Customers are showing various sentiments on the same platform. Let's target the following:

- Meru Cabs: A radio cabs service based in Mumbai, India, launched in 2007
- Ola Cabs: A taxi aggregator company based in Bangalore, India, launched in 2011
- TaxiForSure: A taxi aggregator company based in Bangalore, India, launched in 2011
- Uber India: A taxi aggregator company headquartered in San Francisco, California, launched in India in 2014

Let's make it our goal to get the general sentiments about each of the preceding service providers based on the customer sentiments present in the tweets on Twitter.

performing twitter authentication

```
source(file.path(path2file,"TwitterAuthentication.R"))
```

```
## [1] "Using direct authentication"
```

Collecting tweets as a corpus

```
Meru_tweets = searchTwitter("MeruCabs", n=2000, lang="en")
Ola_tweets = searchTwitter("OlaCabs", n=2000, lang="en")
TaxiForSure_tweets = searchTwitter("TaxiForSure", n=2000, lang="en")
Uber_tweets = searchTwitter("Uber_Delhi", n=2000, lang="en")
```

We always never get the tweets we have asked for. Following are the number of tweet returned:

```
length(Meru_tweets)
```

```
## [1] 242
```

```
length(Ola_tweets)
```

```
## [1] 2000
```

```
length(TaxiForSure_tweets)
```

```
## [1] 42
```

```
length(Uber_tweets)
```

```
## [1] 478
```

Since these tweets are only from the last week's tweets on Twitter, they suggest there is more discussion about these taxi services in the following order:

- OlaCabs
- Uber_Delhi
- MeruCabs
- TaxiForSure

Cleaning the corpus

Before applying any intelligent algorithms to gather more insights from the tweets collected so far, let's first clean the corpus. In order to clean up, we need to understand what the list of tweets looks like:

```
head(Ola_tweets, 2)
```

```
## [[1]]
## [1] "sandipanmallick: @Olacabs True https://t.co/6l7cRumEJ7"
##
## [[2]]
## [1] "raaghav_rathi: @Olacabs @Uber charge 3-5x of the actual fare. Other states hv barred this surcha
```

We see the following:

- Lots of meta information such as @people, URLs and #hashtags
- Punctuation marks, numbers, and unnecessary spaces
- Some of these tweets are retweets from other users; for the given application, we would not like to consider retweets (RTs) in sentiment analysis.

We clean all these data by using the following code block:

```
MeruTweets <- sapply(Meru_tweets, function(x) x$getText())
OlaTweets = sapply(Ola_tweets, function(x) x$getText())
TaxiForSureTweets = sapply(TaxiForSure_tweets, function(x) x$getText())
UberTweets = sapply(Uber_tweets, function(x) x$getText())

catch.error = function(x)
{
  # let us create a missing value for test purpose
  y = NA

  # Try to catch that error (NA) we just created
  catch_error = tryCatch(tolower(x), error=function(e) e)

  # if not an error
  if (!inherits(catch_error, "error"))
  y = tolower(x)

  # check result if error exists, otherwise the function works fine.
  return(y)
}

cleanTweets <- function(tweet){
  # Clean the tweet for sentiment analysis
  # remove html links, which are not required for sentiment analysis
  tweet = gsub("(f|ht)(tp)(s?)(://)(.*)[.|/](.*)", " ", tweet)

  # First we will remove retweet entities from the stored tweets (text)
  tweet = gsub("(RT|via)((?:\\b\\W*@\\w+)+)", " ", tweet)

  # Then remove all "#Hashtag"
  tweet = gsub("#\\w+", " ", tweet)

  # Then remove all "@people"
  tweet = gsub("@\\w+", " ", tweet)
```

```
  # Then remove all the punctuation
  tweet = gsub("[[:punct:]]", " ", tweet)

  # Then remove numbers, we need only text for analytics
  tweet = gsub("[[:digit:]]", " ", tweet)

  # finally, we remove unnecessary spaces (white spaces, tabs etc)
  tweet = gsub("[ \t]{2,}", " ", tweet)
  tweet = gsub("^\\s+|\\s+$", "", tweet)

  # if anything else, you feel, should be removed, you can
  # For example "slang words" etc using the above function and methods.
  # Next we'll convert all the word in lower case.This makes uniform pattern.
  tweet = catch.error(tweet)
  tweet
}

cleanTweetsAndRemoveNAs<- function(Tweets) {
  TweetsCleaned = sapply(Tweets, cleanTweets)
  # Remove the "NA" tweets from this tweet list
  TweetsCleaned = TweetsCleaned[!is.na(TweetsCleaned)]
  names(TweetsCleaned) = NULL
  # Remove the repetitive tweets from this tweet list
  TweetsCleaned = unique(TweetsCleaned)
  TweetsCleaned
}

MeruTweetsCleaned <- cleanTweetsAndRemoveNAs(MeruTweets)
OlaTweetsCleaned <- cleanTweetsAndRemoveNAs(OlaTweets)
TaxiForSureTweetsCleaned <- cleanTweetsAndRemoveNAs(TaxiForSureTweets)
UberTweetsCleaned <- cleanTweetsAndRemoveNAs(UberTweets)
```

Here's the size of each of the cleaned tweet lists:

```
length(MeruTweetsCleaned)
```

```
## [1] 176
```

```
length(OlaTweetsCleaned)
```

```
## [1] 1538
```

```
length(TaxiForSureTweetsCleaned)
```

```
## [1] 23
```

```
length(UberTweetsCleaned)
```

```
## [1] 340
```

Estimating sentiment (A)

It's worth mentioning here that not all the tweets represent sentiments. A few tweets can be just information/facts, while others can be customer care responses. Ideally, they should not be used to assess the customer sentiment about a particular organization.

As a first step, we'll use a Naïve algorithm, which gives a score based on the number of times a positive or a negative word occurred in the given sentence (and, in our case, in a tweet).

```
opinion.lexicon.pos = scan(file.path(path2file
                                     ,'twitter-sentiment-analysis-tutorial-201107/data/opinion-lexicon-
                           , what='character'
                           , comment.char=';')
opinion.lexicon.neg = scan(file.path(path2file
                                     ,'twitter-sentiment-analysis-tutorial-201107/data/opinion-lexicon-
                           , what='character'
                           , comment.char=';')
head(opinion.lexicon.neg)
```

```
## [1] "2-faced"    "2-faces"    "abnormal"   "abolish"    "abominable" "abominably"
```

```
head(opinion.lexicon.pos)
```

```
## [1] "a+"         "abound"     "abounds"    "abundance"  "abundant"   "accessable"
```

We'll add a few industry-specific and/or especially emphatic terms based on our requirements:

```
pos.words = c(opinion.lexicon.pos,'upgrade')
neg.words = c(opinion.lexicon.neg,'wait','waiting', 'wtf', 'cancellation')
```

Now, we create a function, score.sentiment(), which computes the raw sentiment based on the simple matching algorithm:

```
getSentimentScore <- function(sentences, words.positive,words.negative, .progress='none')
{
  require(plyr)
  require(stringr)
  scores = laply(sentences,  function(sentence, words.positive, words.negative) {
    # Let first remove the Digit, Punctuation character and Control characters:
    sentence = gsub('[[:cntrl:]]', '', gsub('[[:punct:]]', '',gsub('\\d+', '', sentence)))

    # Then lets convert all to lower sentence case:
    sentence = tolower(sentence)

    # Now lets split each sentence by the space delimiter
    words = unlist(str_split(sentence, '\\s+'))

    # Get the boolean match of each words with the positive & negative opinion-lexicon
    pos.matches = !is.na(match(words, words.positive))
    neg.matches = !is.na(match(words, words.negative))

    # Now get the score as total positive sentiment minus the total negatives
    score = sum(pos.matches) - sum(neg.matches)
    return(score)
  }, words.positive, words.negative, .progress=.progress )

  # Return a data frame with respective sentence and the score
  return(data.frame(text=sentences, score=scores))
}
```

Now, we apply the preceding function to the corpus of tweets collected and cleaned so far:

```
MeruResult = getSentimentScore(MeruTweetsCleaned, pos.words ,neg.words)
```

```
## Loading required package: plyr
```

```
##
## Attaching package: 'plyr'
```

```
## The following object is masked from 'package:twitteR':
##
##     id
```

```
## Loading required package: stringr
```

```
OlaResult = getSentimentScore(OlaTweetsCleaned, pos.words ,neg.words)
TaxiForSureResult = getSentimentScore(TaxiForSureTweetsCleaned,pos.words ,neg.words)
UberResult = getSentimentScore(UberTweetsCleaned, pos.words ,neg.words)
```

Here are some sample results:

```
head(MeruResult)
```

```
##
## 1
## 2
## 3
## 4
## 5 whatever the competitions i go with the <U+653C><U+3E64><U+613C><U+3E30><U+623C><U+3E64><U+653C><U+
## 6
```

```
head(OlaResult)
```

```
##                                                                                text score
## 1                                                                              true     0
## 2 charge x of the actual fare other states hv barred this surcharge then y not wb     0
## 3                                                                              cont     0
## 4                          visit oko for a course meal through presents powered by     0
## 5                               the dining experience at chi ni at through https <U+0085>     0
## 6                                   visit beyond indus through presents powered by     0
```

```
head(TaxiForSureResult)
```

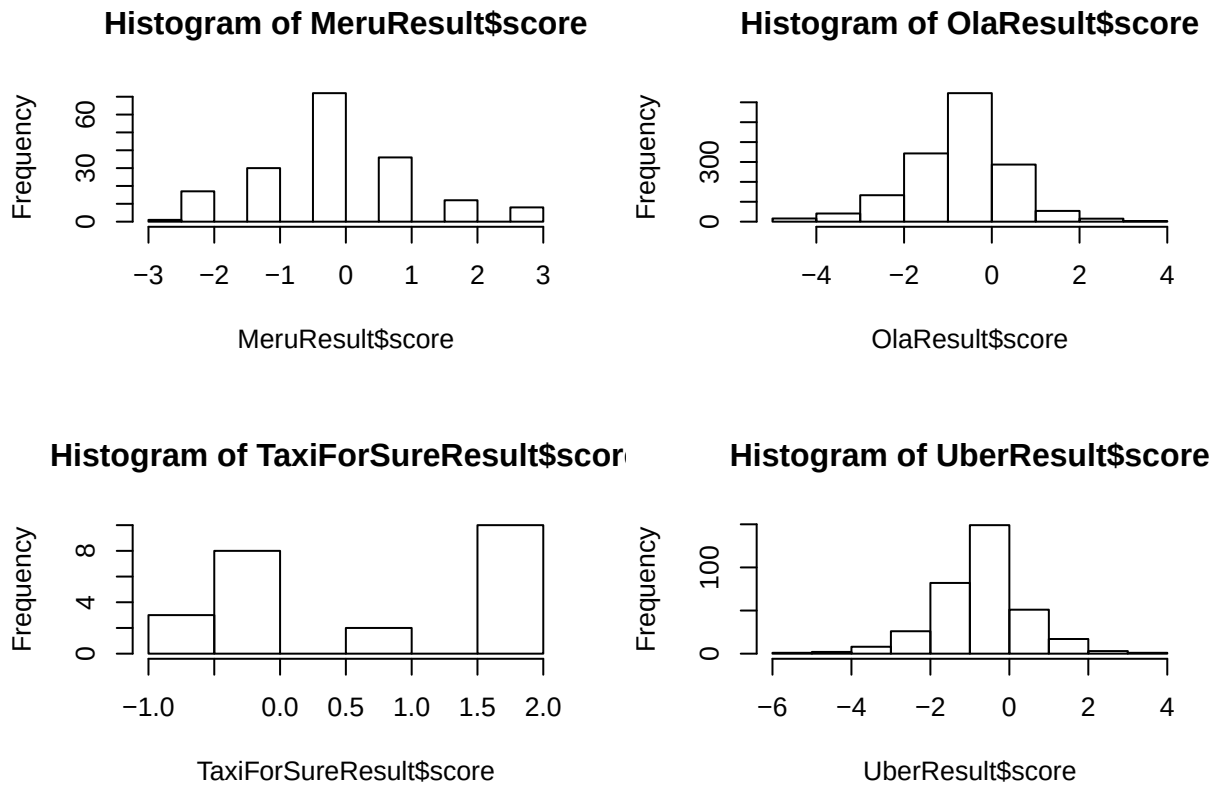
```
##
## 1 how to deal with exciting ideas to how to craft the perfect team lessons from taxiforsure journey
## 2                              how to deal with exciting ideas to how to craft the perfect team <U+0
## 3                                          how to deal with exciting ideas to how to craft the pe
## 4            how to deal with exciting ideas to how to craft the perfect team lessons from taxiforsu
## 5     rt yourstoryco how to deal with exciting ideas to how to craft the perfect team <U+0096> from
## 6  new ideas how to deal with exciting ideas to how to craft the perfect team lessons from taxiforsu
```

```
head(UberResult)
```

```
##
## 1
## 2
## 3
## 4 hate you uber from first day and now shamelessness at its peak now looting customers chup chup ke
## 5
## 6
```

creating histograms of the results

```
par(mfrow = c(2, 2))
hist(MeruResult$score)
hist(OlaResult$score)
hist(TaxiForSureResult$score)
hist(UberResult$score)
```



Let's not compute a point statistic for the scores we have computed so far. Since the numbers of tweets are not equal for each of the four organizations, we compute a mean and standard deviation for each.

```
pointStat <- data.frame("Organization" = c('Meru Cabs','Ola Cabs', 'TaxiForSure', 'Uber India')
                        ,"Mean Sentiment Score" = c(mean(MeruResult$score)
                                                    , mean(OlaResult$score)
                                                    , mean(TaxiForSureResult$score)
                                                    , mean(UberResult$score))
                        ,"Standard Deviation" = c(sd(MeruResult$score)
                                                  , sd(OlaResult$score)
                                                  , sd(TaxiForSureResult$score)
                                                  , sd(UberResult$score))
                        )
pointStat
```

```
##   Organization Mean.Sentiment.Score Standard.Deviation
## 1    Meru Cabs           0.09659091           1.222078
## 2     Ola Cabs          -0.22431730           1.173778
## 3  TaxiForSure           0.82608696           1.154130
## 4   Uber India          -0.21764706           1.179838
```

Estimating sentiment (B)

Instead of using simple matching of opinion lexicon, we'll use something called Naive Bayes to decide on the emotion present in any tweet. We will require packages called Rstem and sentiment to assist with this.

```
#install.packages("Rstem", repos = "http://www.omegahat.org/R", type="source")
library(Rstem)
require(sentiment)
```

building the bayes classifier for sentiment analysis:

```
library(sentiment)

# classify_emotion function returns an object of class data frame
# with seven columns (anger, disgust, fear, joy, sadness, surprise, best_fit)
#and one row for each document:

MeruTweetsClassEmo = classify_emotion(MeruTweetsCleaned
                                      , algorithm="bayes", prior=1.0)
OlaTweetsClassEmo = classify_emotion(OlaTweetsCleaned
                                     , algorithm="bayes", prior=1.0)
TaxiForSureTweetsClassEmo = classify_emotion(TaxiForSureTweetsCleaned
                                             , algorithm="bayes",prior=1.0)
UberTweetsClassEmo = classify_emotion(UberTweetsCleaned
                                      , algorithm="bayes", prior=1.0)
```

The function classify_emotion() generates results belonging to one of the following six emotions: anger, disgust, fear, joy, sadness, and surprise. When the system is not able to classify the overall emotion as any of the six, NA is returned:

```
head(MeruTweetsClassEmo, 15)
```

```
##       ANGER              DISGUST            FEAR               JOY                SADNESS
##  [1,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "1.02547755260094" "1.7277074477352"
##  [2,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "1.02547755260094" "1.7277074477352"
##  [3,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "1.02547755260094" "1.7277074477352"
##  [4,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "1.02547755260094" "1.7277074477352"
##  [5,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "1.02547755260094" "1.7277074477352"
##  [6,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "1.02547755260094" "1.7277074477352"
##  [7,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "7.34083555412328" "7.34083555412328"
##  [8,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "1.02547755260094" "7.34083555412328"
##  [9,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "1.02547755260094" "1.7277074477352"
## [10,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "1.02547755260094" "1.7277074477352"
## [11,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "1.02547755260094" "1.7277074477352"
## [12,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "1.02547755260094" "1.7277074477352"
## [13,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "1.02547755260094" "1.7277074477352"
## [14,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "7.34083555412328" "1.7277074477352"
## [15,] "1.46871776464786" "3.09234031207392" "2.06783599555953" "1.02547755260094" "1.7277074477352"
```

Let's substitute these NA values with the word unknown to make further analysis easier:

```
# we will fetch emotion category best_fit for our analysis purposes.
MeruEmotion = MeruTweetsClassEmo[,7]
OlaEmotion = OlaTweetsClassEmo[,7]
TaxiForSureEmotion = TaxiForSureTweetsClassEmo[,7]
UberEmotion = UberTweetsClassEmo[,7]
```

```
#replacing NA with unknown
MeruEmotion[is.na(MeruEmotion)] = "unknown"
OlaEmotion[is.na(OlaEmotion)] = "unknown"
TaxiForSureEmotion[is.na(TaxiForSureEmotion)] = "unknown"
UberEmotion[is.na(UberEmotion)] = "unknown"
```

Further, we'll use another function, classify_polarity(), provided by the sentiment package, to classify the tweets into two classes, pos (positive sentiment) and neg (negative sentiment). The idea is to compute the log likelihood of a tweet, assuming it belongs to either of the two classes. Once these likelihoods are calculated, a ratio of the pos-likelihood to neg-likelihood is calculated, and, based on this ratio, the tweets are classified as belonging to a particular class. It's important to note that if this ratio turns out to be 1, then the overall sentiment of the tweet is assumed to be "neutral". The code is as follows:

```
MeruTweetsClassPol = classify_polarity(MeruTweetsCleaned,algorithm="bayes")
OlaTweetsClassPol = classify_polarity(OlaTweetsCleaned,algorithm="bayes")
TaxiForSureTweetsClassPol =classify_polarity(TaxiForSureTweetsCleaned, algorithm="bayes")
UberTweetsClassPol = classify_polarity(UberTweetsCleaned,algorithm="bayes")
```

we get the following output

```
head(MeruTweetsClassPol, 15)
```

```
##       POS                 NEG                  POS/NEG             BEST_FIT
##  [1,] "9.47547003995745" "9.47547003995745"  "1"                 "neutral"
##  [2,] "1.03127774142571" "0.445453222112551" "2.31512017476245"  "positive"
##  [3,] "33.4217525744328" "0.445453222112551" "75.0286470393703"  "positive"
##  [4,] "1.03127774142571" "8.78232285939751"  "0.117426534862834" "negative"
##  [5,] "1.03127774142571" "8.78232285939751"  "0.117426534862834" "negative"
##  [6,] "8.78232285939751" "9.47547003995745"  "0.926848253686942" "negative"
##  [7,] "9.47547003995745" "26.8423564950873"  "0.353004403383572" "negative"
##  [8,] "17.2265151579293" "17.8123396772424"  "0.967111309916146" "negative"
##  [9,] "1.03127774142571" "9.47547003995745"  "0.108836578774127" "negative"
## [10,] "9.47547003995745" "18.5054868578024"  "0.512035706640291" "negative"
## [11,] "24.9775602759011" "0.445453222112551" "56.0722406663613"  "positive"
## [12,] "1.03127774142571" "0.445453222112551" "2.31512017476245"  "positive"
## [13,] "1.03127774142571" "9.47547003995745"  "0.108836578774127" "negative"
## [14,] "9.47547003995745" "8.78232285939751"  "1.07892526745566"  "neutral"
## [15,] "1.03127774142571" "0.445453222112551" "2.31512017476245"  "positive"
```

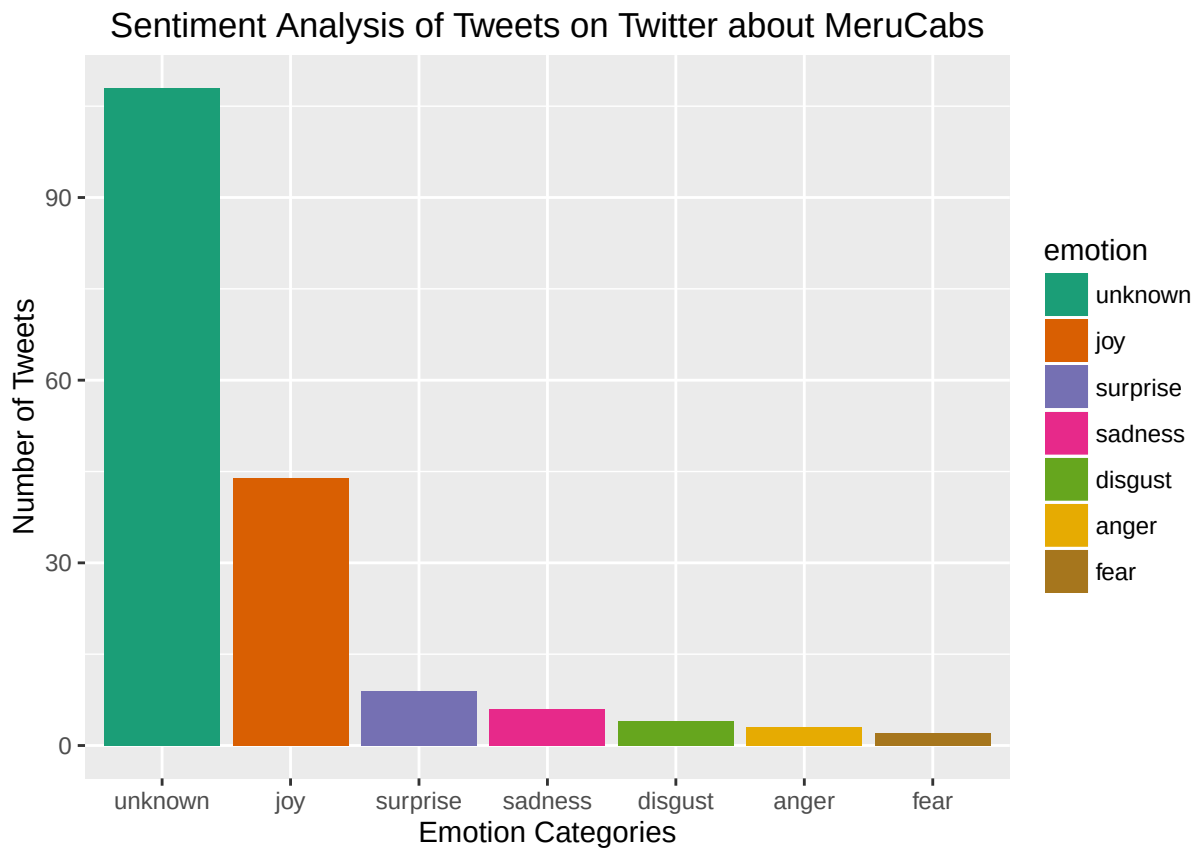
The preceding figure shows a few results obtained by using the classify_polarity() function of the sentiment package for Meru Cabs tweets. We'll now generate consolidated results from the two functions in a data frame for each cab service for plotting purposes:

```
# we will fetch polarity category best_fit for our analysis purposes,
MeruPol = MeruTweetsClassPol[,4]
OlaPol = OlaTweetsClassPol[,4]
TaxiForSurePol = TaxiForSureTweetsClassPol[,4]
UberPol = UberTweetsClassPol[,4]

# Let us now create a data frame with the above results
MeruSentimentDataFrame = data.frame(text=MeruTweetsCleaned
                                    , emotion=MeruEmotion
                                    , polarity=MeruPol
                                    , stringsAsFactors=FALSE)
OlaSentimentDataFrame = data.frame(text=OlaTweetsCleaned
                                   , emotion=OlaEmotion
```

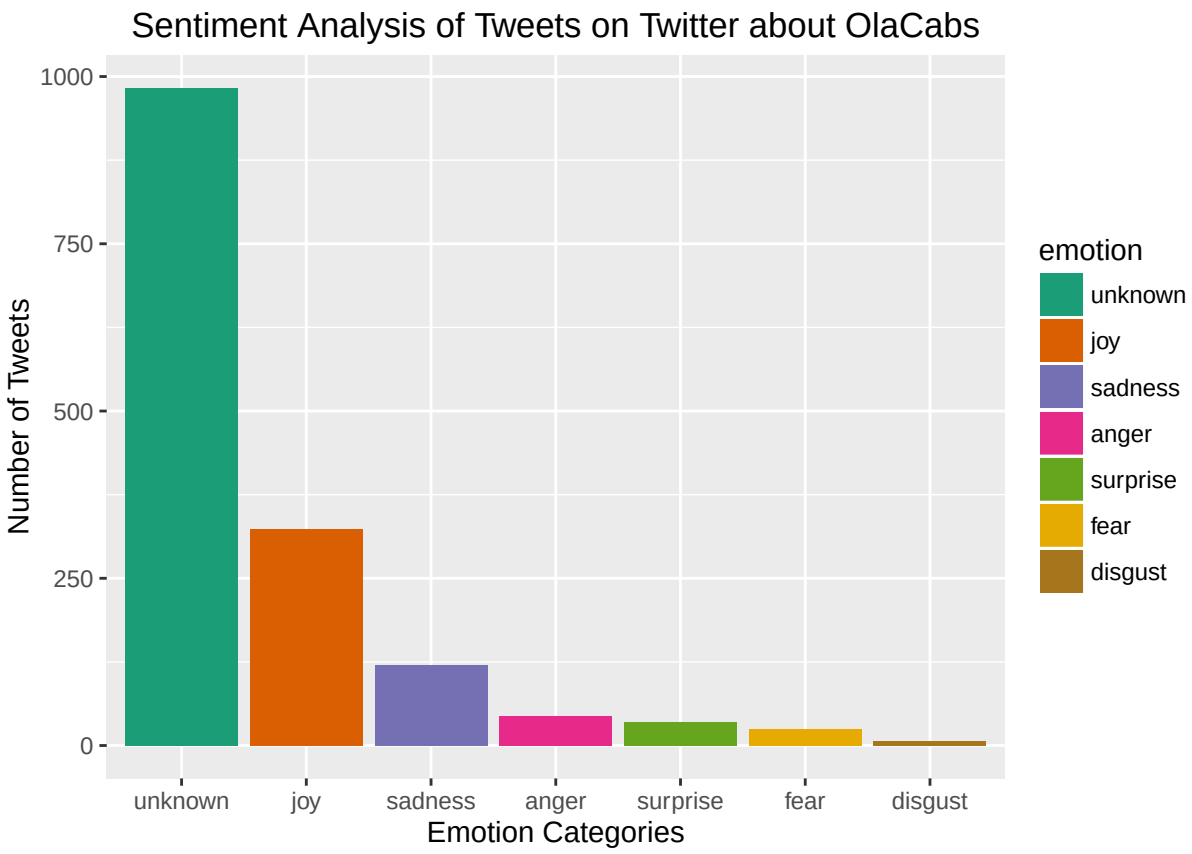
```
                                    , polarity=OlaPol
                                    , stringsAsFactors=FALSE)
TaxiForSureSentimentDataFrame =data.frame(text=TaxiForSureTweetsCleaned
                                          , emotion=TaxiForSureEmotion
                                          , polarity=TaxiForSurePol
                                          , stringsAsFactors=FALSE)
UberSentimentDataFrame = data.frame(text=UberTweetsCleaned
                                    , emotion=UberEmotion
                                    , polarity=UberPol
                                    , stringsAsFactors=FALSE)

# rearrange data inside the frame by sorting it
MeruSentimentDataFrame = within(MeruSentimentDataFrame
                                , emotion <-factor(emotion
                                                   ,levels=names(sort(table(emotion)
                                                                      ,decreasing=TRUE))))
OlaSentimentDataFrame = within(OlaSentimentDataFrame
                               , emotion <- factor(emotion
                                                   , levels=names(sort(table(emotion)
                                                                       ,decreasing=TRUE))))
TaxiForSureSentimentDataFrame =within(TaxiForSureSentimentDataFrame
                                      , emotion <- factor(emotion
                                                          ,levels=names(sort(table(emotion)
                                                                             ,decreasing=TRUE))))
UberSentimentDataFrame = within(UberSentimentDataFrame
                                , emotion <- factor(emotion
                                                    , levels=names(sort(table(emotion)
                                                                        ,decreasing=TRUE))))

plotSentiments1 <- function (sentiment_dataframe,title) {
  library(ggplot2)
  ggplot(sentiment_dataframe, aes(x=emotion)) +
  geom_bar(aes(y=..count.., fill=emotion)) +
  scale_fill_brewer(palette="Dark2") +
  ggtitle(title) +
  theme(legend.position='right') + ylab('Number of Tweets') +
  xlab('Emotion Categories')
}


plotSentiments1(MeruSentimentDataFrame
                , 'Sentiment Analysis of Tweets on Twitter about MeruCabs')
```

```
##
## Attaching package: 'ggplot2'
```

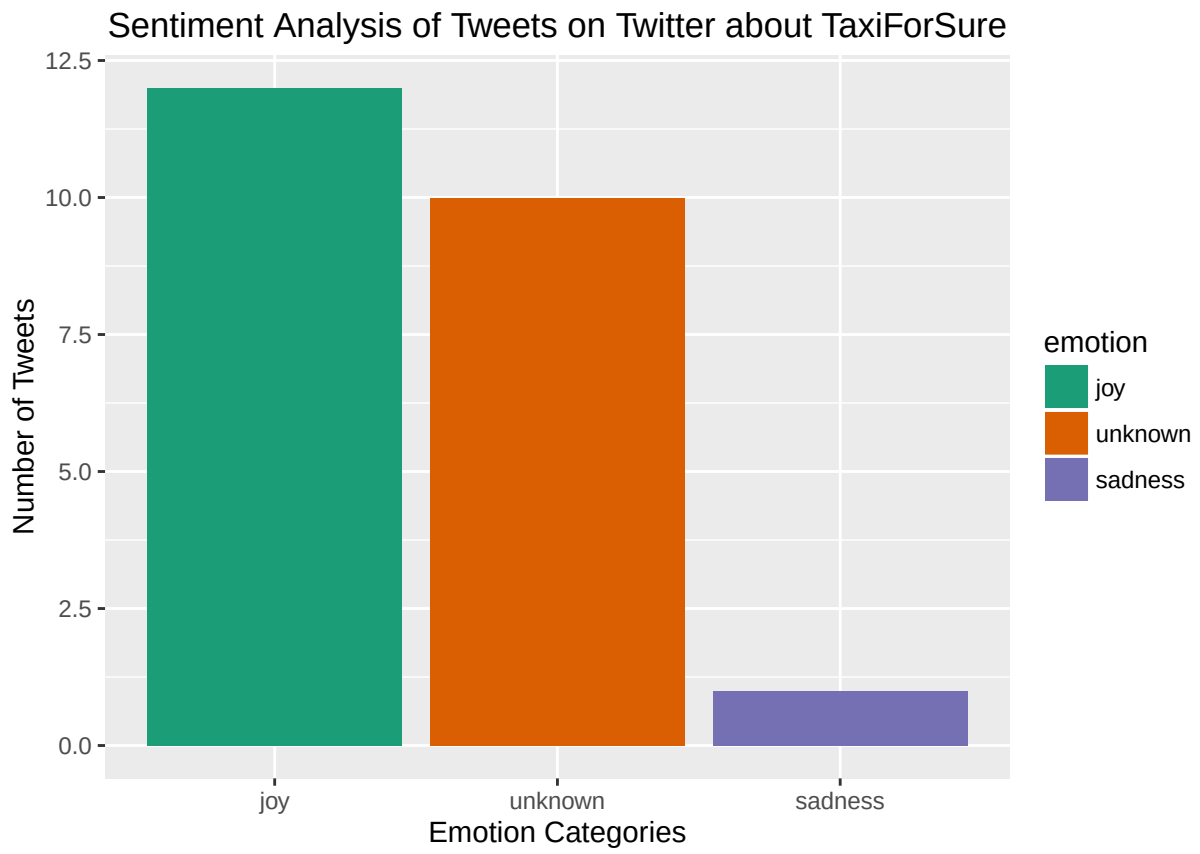
```
## The following object is masked from 'package:NLP':
##
##     annotate
```



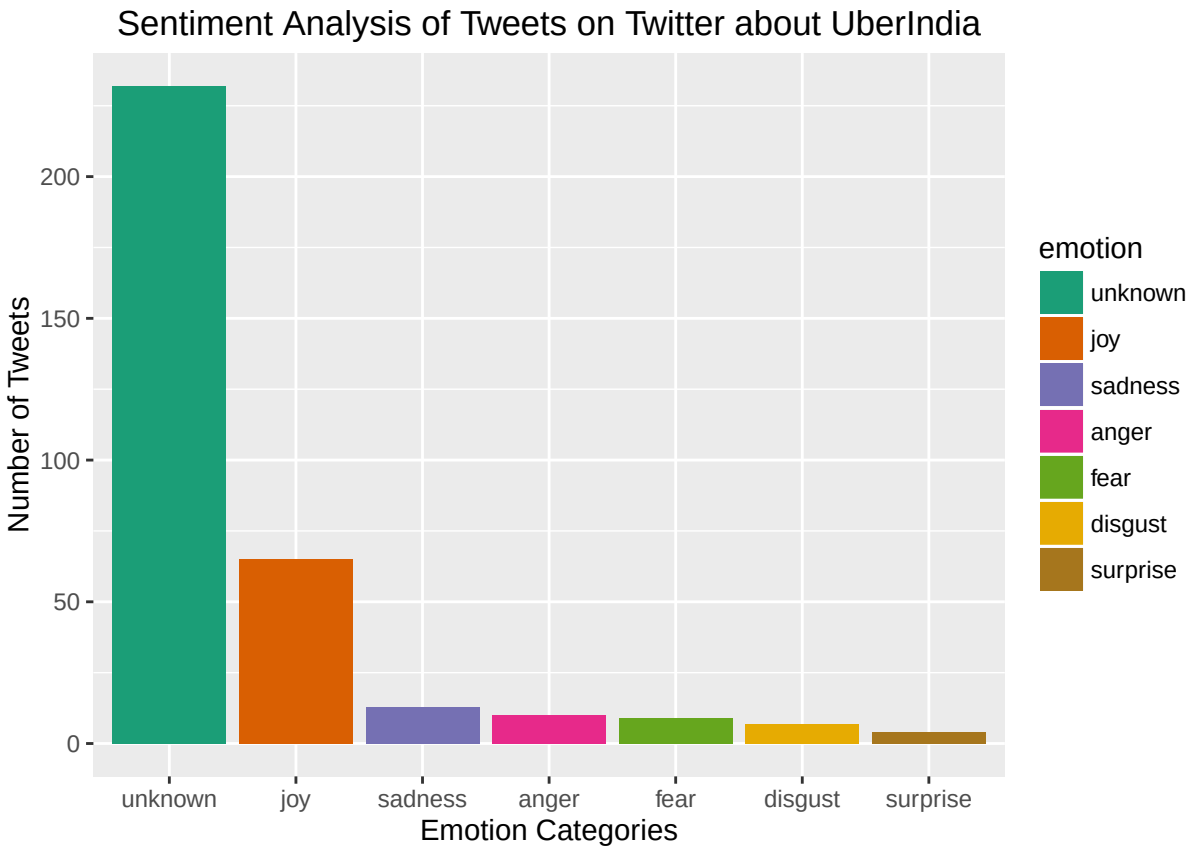

```
plotSentiments1(OlaSentimentDataFrame
                , 'Sentiment Analysis of Tweets on Twitter about OlaCabs')
```




```
plotSentiments1(TaxiForSureSentimentDataFrame
                , 'Sentiment Analysis of Tweets on Twitter about TaxiForSure')
```

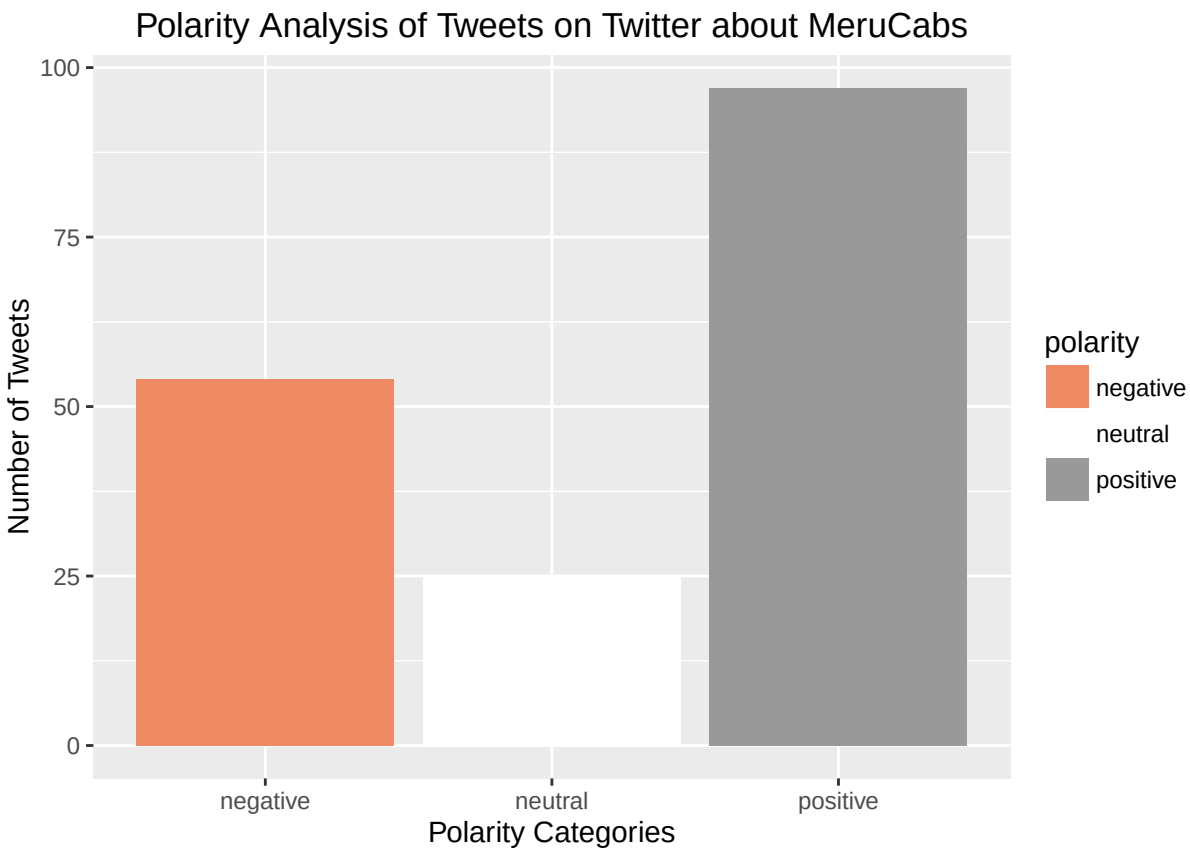



```
plotSentiments1(UberSentimentDataFrame
                , 'Sentiment Analysis of Tweets on Twitter about UberIndia')
```

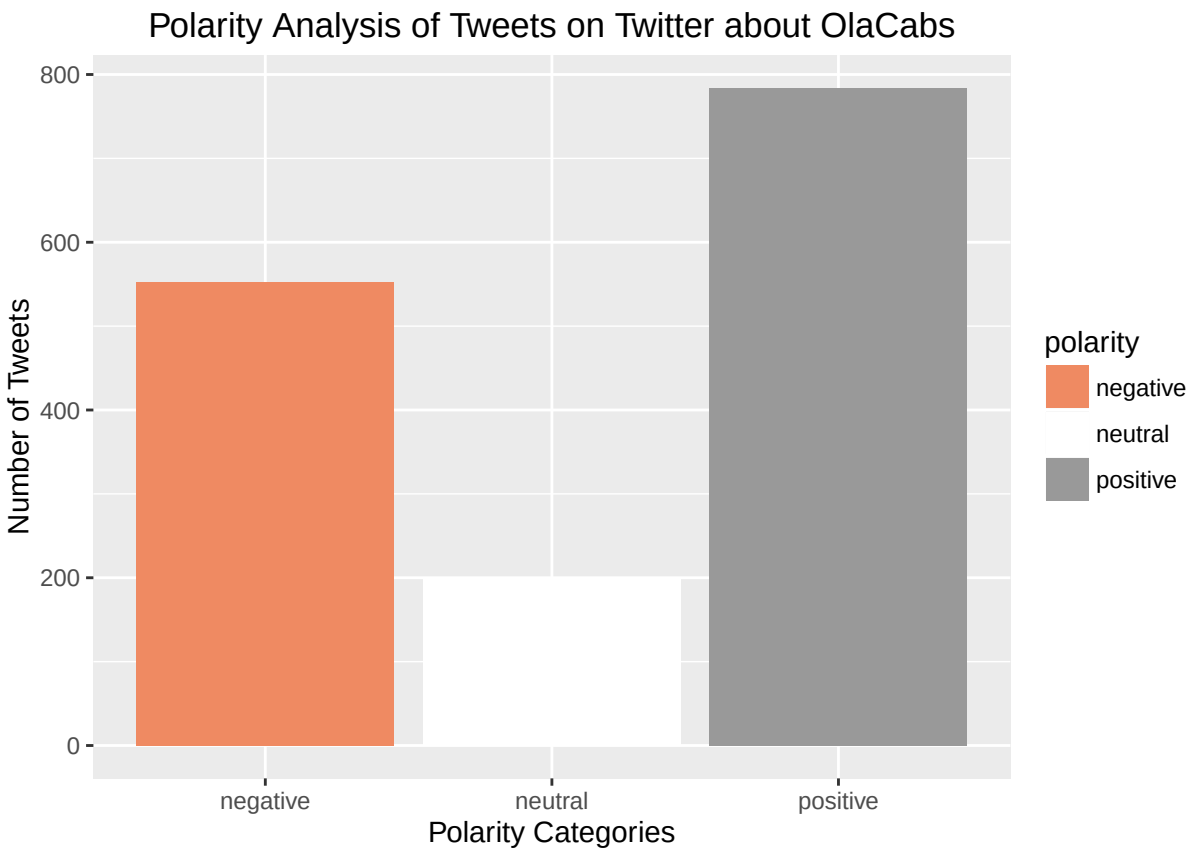


These sentiments basically reflect observations broadly similar to those we made with the basic word-matching algorithm. The number of tweets with joy constitute the largest part of tweets for all these organizations, indicating that these organizations are trying their best to provide good business in the country. The sadness tweets are less numerous than the joy tweets. However, if compared with each other, they indicate the overall market share versus the level of customer satisfaction with each service provider. Similarly, these graphs can be used to assess the level of dissatisfaction in terms of anger and disgust in the tweets. Let's now consider only the positive and negative sentiments present in the tweets:

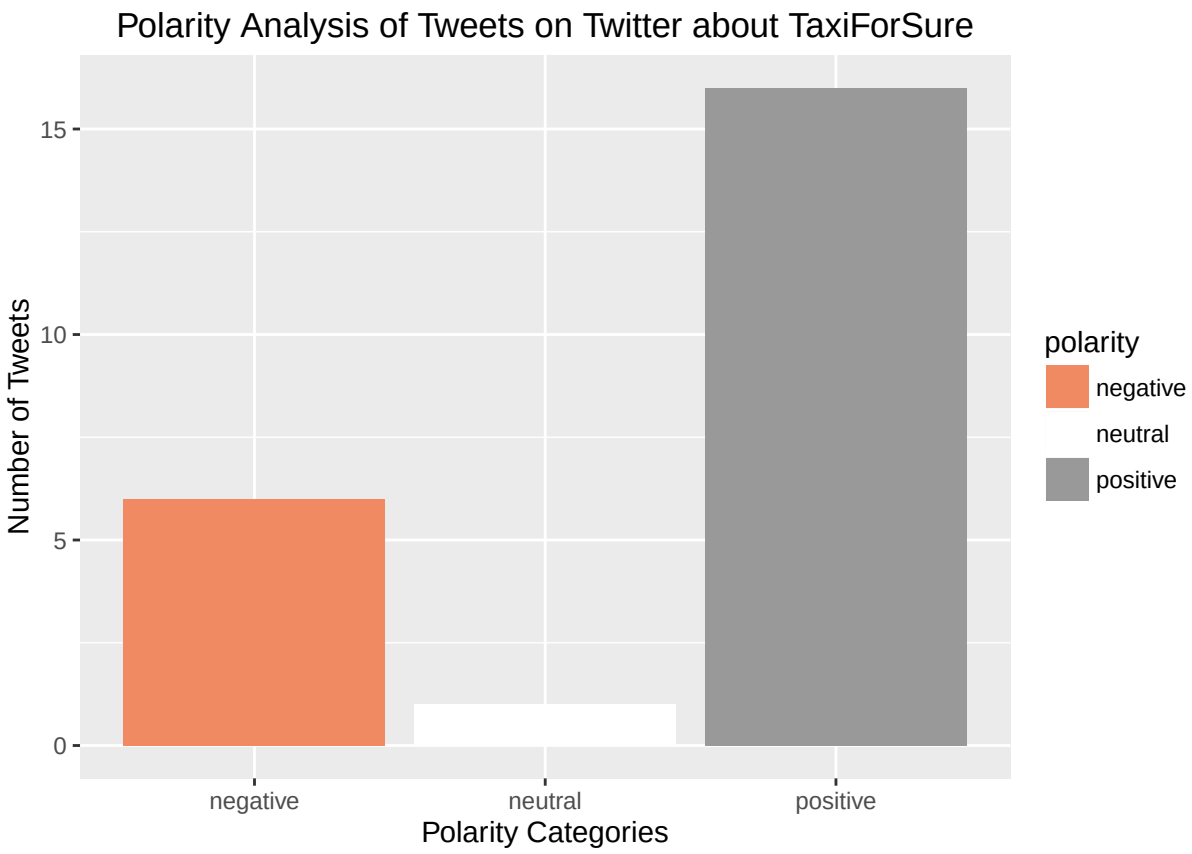
```
# Similarly we will plot distribution of polarity in the tweets
plotSentiments2 <- function (sentiment_dataframe,title) {
  library(ggplot2)
  ggplot(sentiment_dataframe, aes(x=polarity)) +
  geom_bar(aes(y=..count.., fill=polarity)) +
  scale_fill_brewer(palette="RdGy") +
  ggtitle(title) +
  theme(legend.position='right') + ylab('Number of Tweets') + xlab('Polarity Categories')
}

plotSentiments2(MeruSentimentDataFrame
                , 'Polarity Analysis of Tweets on Twitter about MeruCabs')
```



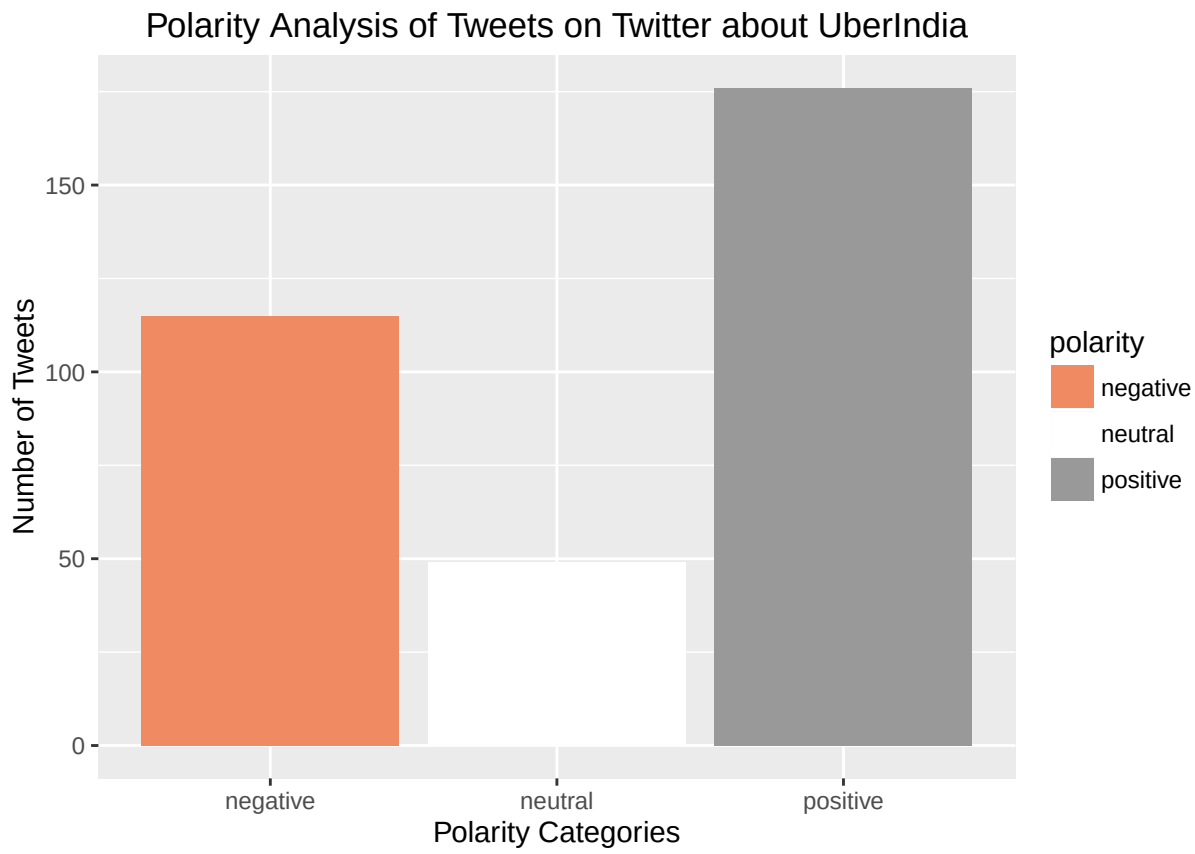

```
plotSentiments2(OlaSentimentDataFrame
                , 'Polarity Analysis of Tweets on Twitter about OlaCabs')
```




```
plotSentiments2(TaxiForSureSentimentDataFrame
                , 'Polarity Analysis of Tweets on Twitter about TaxiForSure')
```



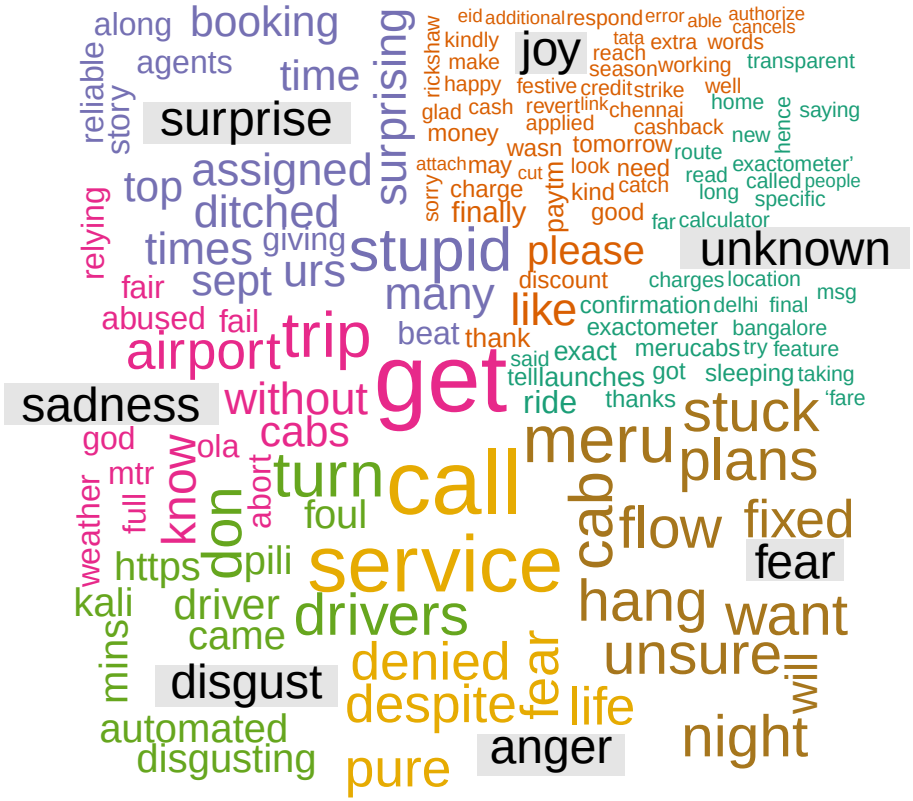

```
plotSentiments2(UberSentimentDataFrame
                , 'Polarity Analysis of Tweets on Twitter about UberIndia')
```



Next, we try to get a sense of the overall content of the tweets by using the word clouds.

```
removeCustomeWords <- function (TweetsCleaned) {
  for(i in 1:length(TweetsCleaned)){
    TweetsCleaned[i] <- tryCatch({
      TweetsCleaned[i] = removeWords(TweetsCleaned[i]
                                     ,c(stopwords("english"), "care"
                                        , "guys", "can", "dis", "didn"
                                        ,"guy" ,"booked", "plz"))
    TweetsCleaned[i]
    }, error=function(cond) {
        TweetsCleaned[i]
        }, warning=function(cond) {
              TweetsCleaned[i]
            })
  }
  return(TweetsCleaned)
}

getWordCloud <- function(sentiment_dataframe, TweetsCleaned, Emotion) {
  emos = levels(factor(sentiment_dataframe$emotion))
  n_emos = length(emos)
  emo.docs = rep("", n_emos)
  TweetsCleaned = removeCustomeWords(TweetsCleaned)
  for (i in 1:n_emos){
    emo.docs[i] = paste(TweetsCleaned[Emotion == emos[i]], collapse=" ")
  }
```

```
  corpus = Corpus(VectorSource(emo.docs))
  tdm = TermDocumentMatrix(corpus)
  tdm = as.matrix(tdm)
  colnames(tdm) = emos
  require(wordcloud)
  suppressWarnings(comparison.cloud(tdm, colors = brewer.pal(n_emos, "Dark2"), scale = c(3,.5)
                                    , random.order = FALSE, title.size = 1.5))
}

getWordCloud(MeruSentimentDataFrame, MeruTweetsCleaned,MeruEmotion)
```




```
getWordCloud(OlaSentimentDataFrame, OlaTweetsCleaned, OlaEmotion)
```


```
getWordCloud(TaxiForSureSentimentDataFrame, TaxiForSureTweetsCleaned, TaxiForSureEmotion)
```


```
getWordCloud(UberSentimentDataFrame, UberTweetsCleaned, UberEmotion)
```


joy
sadness
unknown
anger
surprise
fear
disgust
get
non
horror
horrific
tweeted
please
drivers
high
court
uber
cab
ride
cooperative
behaviour
expectations
explosion
influenzaas
interested