6103-proposal-milestone3

Strayn

# 0.Prepare for the analysis

Our goal is to build a **text miner**(or semantic analyzer, whatever you would call it), that requires a machine to label the right sentiment to the text we have, basically it requires 3 things to be done:  
1. **Get the data**, which contains text and sentiment.  
2. **Build a miner** to understand the text.  
3. **Build a learner** to find relationship between text and sentiment.

For the 1st step, our kaggle dataset contains everything we need in 4 categories:  
who: **USERS**  
what: **TWEETS**(with text and labeled sentiment)  
when: **TIME**  
where: **LOCATIONS**

But hold on, where does the sentiment come from?  
Well, the sentiment might be given manually or by a smart semantic analyzer which was built by other dataset, but let's assume the labels are right because there are hundres like us studying on this dataset.

The raw data is shown as following structure:

## 'data.frame': 14640 obs. of 15 variables:  
## $ tweet\_id : num 5.7e+17 5.7e+17 5.7e+17 5.7e+17 5.7e+17 ...  
## $ airline\_sentiment : Factor w/ 3 levels "negative","neutral",..: 2 3 2 1 1 1 3 2 3 3 ...  
## $ airline\_sentiment\_confidence: num 1 0.349 0.684 1 1 ...  
## $ negativereason : Factor w/ 10 levels "Bad Flight","Cancelled Flight",..: NA NA NA 1 3 3 NA NA NA NA ...  
## $ negativereason\_confidence : num NA 0 NA 0.703 1 ...  
## $ airline : Factor w/ 6 levels "American","Delta",..: 6 6 6 6 6 6 6 6 6 6 ...  
## $ airline\_sentiment\_gold : Factor w/ 3 levels "negative","neutral",..: NA NA NA NA NA NA NA NA NA NA ...  
## $ name : Factor w/ 7701 levels "0504Traveller",..: 1023 3444 7666 3444 3444 3444 1349 5646 1835 7665 ...  
## $ negativereason\_gold : Factor w/ 13 levels "Bad Flight","Cancelled Flight",..: NA NA NA NA NA NA NA NA NA NA ...  
## $ retweet\_count : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ text : Factor w/ 14427 levels "1/2 @united Just be honest with your customers. The new seats are designed to fit more rows on the plane. More rows = more mone"| \_\_truncated\_\_,..: 14323 14218 14063 14128 13917 14234 14358 14223 14321 14130 ...  
## $ tweet\_coord : Factor w/ 832 levels "[0.0, 0.0]","[10.42269832, -75.54540857]",..: NA NA NA NA NA NA NA NA NA NA ...  
## $ tweet\_created : Factor w/ 14247 levels "2015-02-16 23:36:05 -0800",..: 14212 14170 14169 14168 14166 14165 14164 14160 14158 14106 ...  
## $ tweet\_location : Factor w/ 3081 levels "?????",".",". ",..: NA NA 1360 NA NA NA 2323 1426 2302 1426 ...  
## $ user\_timezone : Factor w/ 85 levels "Abu Dhabi","Adelaide",..: 31 63 28 63 63 63 63 63 63 31 ...

Clearly the information of the who and what is answered by the text, but when and where can also be important in logic. For example, a negative text might be caused by a flight delay, and a delay might be a consequence of a bad weather, they are to be known as the context of text creating, which is also important features for us to dig.

First thing first, let's RELOAD first!

#sourcing tools  
library(twitteR) #read from twitter streaming api  
library(jsonlite) # read from json  
library(readr) # read files  
  
#dataframe tools  
library(dplyr)   
library(plyr)   
library(dplyr) # data manipulation  
library(stringr) # text manipulation  
  
#visualization tools  
library(Amelia) #data explore  
library(ggplot2)   
library(ggthemes)   
library(wordcloud)  
  
#semantic analysis tools  
library(tm) #get corpus  
library(SnowballC) # stemming   
library(NLP) #root   
  
#machine learning tools  
library(caret) #K folds cross validation  
library(MASS) # stepAIC  
library(e1071) #NaiveBayes  
library(leaps) #model selection  
library(bestglm)#model selection   
library(boot) # cv.glm()  
  
#our functions   
source(file('airline\_utilities.R'))

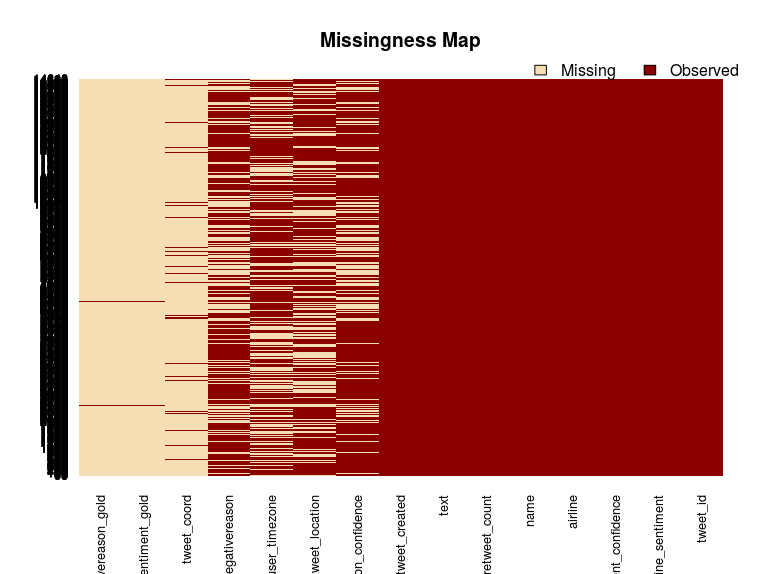
# 1.See our data

To see the data we have to **clean it** first, here are our codes to do so:  
*To prevent distracting readers from the massive codes, our codes are packed up in functions, and if you are intersted to the details, check our* [*github gallery*](https://github.com/gwds/dmg6/blob/master/code_samples/data_clean-Strayn)*.*

#clean the CLASS   
tweets$tweet\_created <- as.POSIXct(tweets$tweet\_created)  
tweets[,sapply(tweets,is.factor)] <- sapply(tweets[,sapply(tweets,is.factor)],as.character)

## Warning: closing unused connection 5 (airline\_utilities.R)

#clean the CONTENT  
  
#clean URLs  
tweets$text <- gsub('((http|ftp|https)://)(([a-zA-Z0-9\\.\_-]+\\.[a-zA-Z]{2,6})|([0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}))(:[0-9]{1,4})\*(/[a-zA-Z0-9\\&%\_\\./-~-]\*)?','URL\_LINK\_TWEETS',tweets$text)  
  
tweets <- clean\_location(tweets) # clean the tweet\_coord column

After cleaning we found our data badly broken with NAs, how bad? Check the missmap here:  These broken points are mainly in the locations, timezone and sentiment labels, sentiment labels are important, but why do we have to fix the locations and timezones?  
Well, that's because they are important to understand the text, because we think the meaning of the text should be determined by 2 points: **a. The text itself.**  
**b. The context where the text was created**

The timezone and location, which are when and where are related to the point b. But right now it's not the priority, we should keep on for other attributes to get a whole view first.  
Let's just take down the problem and fix it latter.

Well, what about the features of text?

# 2.Text Explorating

We said we have to understand the text, that means to break down the feature text from a big one to smaller fragments which are good at answering our question, which is sentiment in this case.

Ok break it down... But how? To understand a text [we have to realize that there are 3 levels out there](http://www.cstr.ed.ac.uk/emasters/course/natural_lang.html):  
**a. Lexical level:** stemming, features extracting. **b. Syntactic level:** grammars and parsing.  
**c. Semantic & Pragmatic level:** logical forms and reasoning, entities identifying.

In this milestone we will focus on the step a, and try to extract important features out of text and make them the predictors to feed our learner(model like logistics regression, naive bayes and svm).

To extract feature meaning maximize the signal, basically we have 2 ways to do so: 2.1. We get rid ot the noise to make the real information significant. For example stopwords are such noise which contains "and, is, a, the" but no information. 2.2. We group the similiar information to make them stronger signal, which is known as stemming. For example words like "complicated, complicate, complicating" contain same information, we have to let our learner know.

To realize the 2 steps we want sentence like: > RT @ScoopDaa: @LALEGGERS Low 70's Forecast For @JetBlue 2016 @LBMarathon = Official Program <https://t.co/BoVcrgnOCo> <https://t.co/qNeLzqlN39>

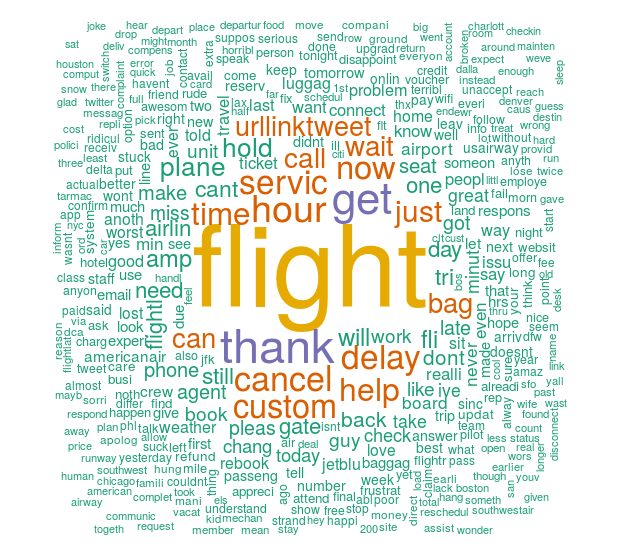
to be processed as this  
> rt scoopdaa teamintrain low s forecast jetblue lbmarathon offici program

Here are our codes to do so:

temp <- gsub('^@\\w+','',tweets$text) #get rid of the entities  
corpus <- VCorpus(VectorSource(temp)) #save text into corpus  
corpus <- tm\_map(corpus, content\_transformer(tolower)) # transfor to lower case  
corpus <- tm\_map(corpus, PlainTextDocument, lazy = T) # creat a plain text document  
corpus <- tm\_map(corpus, removePunctuation) # remove punctuation  
corpus <- tm\_map(corpus, removeWords, stopwords(kind = "english")) # remove stop words  
corpus <- tm\_map(corpus, stripWhitespace) #stripe whitespace from the corpus text  
corpus <- tm\_map(corpus, stemDocument) #get the stem of the words in our sentence

*Noticed that the URLs should have already been removed by the 1st cleaning step, thus it's not shown at above code.*  
*Also noticed that the tolower function might encounter error when dealing with emoji or other non-graphical text, we set a pre cleaning function to deal with it, called tryTolower, again you can check it in our* [*github gallery*](https://github.com/gwds/dmg6/blob/master/code_samples/data_clean-Strayn)*.*

#This is the wordcloud after text cleaning:  
wordcloud(corpus,min.freq = 50,random.order = FALSE,scale=c(7, .6),colors=brewer.pal(6, "Dark2"))

 \*\*\* And after doing so we get a corpus which contains necessary information we have, but a [Document Terms Matrix](https://www.quora.com/What-is-a-term-document-matrix) will be more friendly to analysis:

dtm <- DocumentTermMatrix(corpus) #change corpus to Document Terms Matrix  
dtm <- removeSparseTerms(dtm, 0.97) #remove terms which have tiny probability to show up  
findFreqTerms(dtm,5)

## [1] "airlin" "amp" "back" "bag"   
## [5] "book" "call" "can" "cancel"   
## [9] "cant" "custom" "day" "delay"   
## [13] "dont" "fli" "flight" "flightl"   
## [17] "gate" "get" "got" "help"   
## [21] "hold" "hour" "just" "make"   
## [25] "need" "now" "one" "plane"   
## [29] "pleas" "seat" "servic" "still"   
## [33] "thank" "time" "tri" "urllinktweet"  
## [37] "wait" "will"

In the above codes we extract the important words among texts, and if we group them by the sentiment we have, we will have indicator for certain sentiment! And that's our goal! We've done so in our sample codes, but since the feature engineering requires merging dataset from internet, it's still in the process, so the main idea we want to show in this milestone is to give a whole picture for our thinking process, and if you are interested in analysis detail, please check our [github gallery](https://github.com/gwds/dmg6/tree/master/code_samples/data_clean-Strayn).

# 3. How to make a learner.

Well, a learner needs to be fed by data, and that data has to change along the constants we want to fix by our model. For example if we want to know how much weather influence the sentiments of tweets, at least we have to see different weathers showing up in our dataset.  
So we have to build a learner with features that are both able to be extended by merging other dataset and are logically related to our question, which is the relationship between sentiment and text in this case.

Our dataset is about user, tweets, time and location, tweets text is apparently the best predictor for tweets sentiment, but sometimes the anlysis for them can be hard, so at this milestone we want to know how well the features other than "text itself" can predict the sentiment of text.

For this quick test we choose to ignore the broken part of our data, which are locations, timezones and sentiment reasons, and focus on data we can get from twitter API, which are airline(we can search tweets by topic of airline), retweet\_count, tweet\_created. CLearly this is not good enough, so we also decide to extract some quick information from the text,which are length of the text, the number of @ appearing in the text. Here are codes for that:

library(stringr)  
tweets <- add\_feature(tweets) # count the length of every text column, count the number of @ in it too  
#get rid of broken columns  
broken\_colnum <- c(4,5,7,9,13,14,15,16)  
cfix\_tweets <- tweets[-broken\_colnum]  
cfix\_tweets[,sapply(cfix\_tweets,is.character)] <- as.data.frame(lapply(cfix\_tweets[,sapply(cfix\_tweets,is.character)],as.factor))  
#show the formula we care  
FORMULA <- get\_formula(2,c(4,6,8,9,10,11),cfix\_tweets)  
FORMULA

## airline\_sentiment ~ airline + retweet\_count + tweet\_created +   
## at\_count + at\_countD + text\_length  
## <environment: 0x81eac80>

Above is the model formula we made by our dataset, and now let's try predict with it to see how well it could do:

#get train/test dataset  
train <- get\_tt\_data(cfix\_tweets)[['train']]  
test <- get\_tt\_data(cfix\_tweets)[['test']]  
  
#original model we set in mind  
log\_model1 <- glm(FORMULA, data=train, family=binomial)  
log\_pred1 <- predict(log\_model1, newdata=test, type="response")  
results1 <-table(test$airline\_sentiment, log\_pred1>.5)  
results1

##   
## FALSE TRUE  
## negative 2400 392  
## neutral 443 456  
## positive 348 353

We see from the results that it can predict the negative sentiment very well, 95.42% of the negative sentiment are correctly predicted by our model. But the positive sentiment are poorly predicted, only 24.24% correctly predicted.

Well, we will improve that in the futural milestones, but right now let's continue to model simplification. There might be features that provides similiar information, which made our model somehow fatter than it could be, so we should cut off them, to do so we use stepAIC [to most adequately describe the unknown](http://stats.stackexchange.com/questions/577/is-there-any-reason-to-prefer-the-aic-or-bic-over-the-other), and here are the codes for us to do so:

library(MASS)  
#find the best formula by stepAIC  
model\_full <- glm(FORMULA,train,family = binomial)  
model\_0 <- glm(airline\_sentiment~1,train,family = binomial)  
feature\_selection <- stepAIC(model\_0,scope=FORMULA,direction = 'forward')

## Start: AIC=13471.54  
## airline\_sentiment ~ 1  
##   
## Df Deviance AIC  
## + text\_length 1 12042 12046  
## + airline 5 12677 12689  
## + tweet\_created 1 13428 13432  
## + at\_count 1 13441 13445  
## + at\_countD 2 13443 13449  
## + retweet\_count 1 13464 13468  
## <none> 13470 13472  
##   
## Step: AIC=12045.61  
## airline\_sentiment ~ text\_length  
##   
## Df Deviance AIC  
## + airline 5 11436 11450  
## + at\_count 1 11951 11957  
## + at\_countD 2 11956 11964  
## + tweet\_created 1 12011 12017  
## <none> 12042 12046  
## + retweet\_count 1 12040 12046  
##   
## Step: AIC=11449.72  
## airline\_sentiment ~ text\_length + airline  
##   
## Df Deviance AIC  
## + at\_count 1 11357 11373  
## + at\_countD 2 11360 11378  
## + tweet\_created 1 11431 11447  
## + retweet\_count 1 11433 11449  
## <none> 11436 11450  
##   
## Step: AIC=11372.78  
## airline\_sentiment ~ text\_length + airline + at\_count  
##   
## Df Deviance AIC  
## + tweet\_created 1 11352 11370  
## + retweet\_count 1 11354 11372  
## <none> 11357 11373  
## + at\_countD 2 11357 11377  
##   
## Step: AIC=11370.17  
## airline\_sentiment ~ text\_length + airline + at\_count + tweet\_created  
##   
## Df Deviance AIC  
## + retweet\_count 1 11349 11369  
## <none> 11352 11370  
## + at\_countD 2 11352 11374  
##   
## Step: AIC=11368.99  
## airline\_sentiment ~ text\_length + airline + at\_count + tweet\_created +   
## retweet\_count  
##   
## Df Deviance AIC  
## <none> 11349 11369  
## + at\_countD 2 11349 11373

Therefore we find the formula with the best prediction efficiency to be:

## airline\_sentiment ~ text\_length + airline + at\_count + tweet\_created

We compare the prediction power for the 2 formulas:

#simplified model by AIC  
log\_model2 <- glm(BEST\_FORMULA, data=train, family=binomial)  
log\_pred2 <- predict(log\_model2, newdata=test, type="response")  
results2<-table(test$airline\_sentiment, log\_pred2>.5)  
  
#orginal model prediction power  
results1

##   
## FALSE TRUE  
## negative 2400 392  
## neutral 443 456  
## positive 348 353

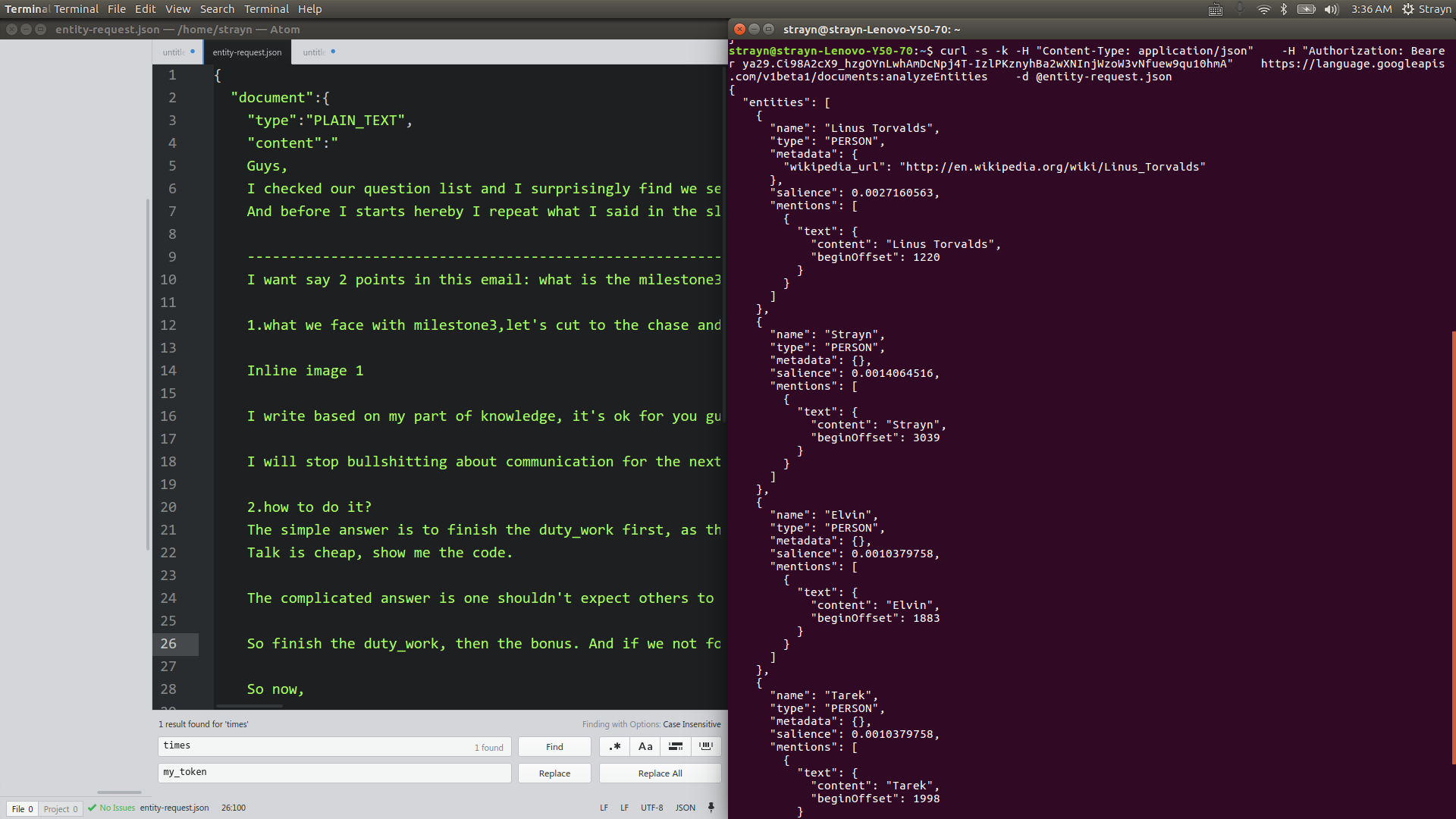
#simplified model prediction power  
results2

##   
## FALSE TRUE  
## negative 2406 386  
## neutral 442 457  
## positive 345 356

From the results, we can tell that the simplified model could do almost the same as our orginal model, so we considering the computing efficiency we should replace the old one with the simplified one.

The same process could be applied by other machine learner, such as decision tree, svm or naive bayes model. Basically this is how far we can achieve without changing model or dataset.

# Finally, how improve?

4 ways for us to make it better:  
1. Fit the prediction results we get from our learner with what we get from semantic API by google, fowllowing is an example of this, more information can be found in our [github gallery](https://github.com/gwds/dmg6/tree/master/results_reports). 

1. We will fix our dataset by online API  
   To recover location information, we can group them by [Geocoding API](https://developers.google.com/maps/documentation/geocoding/intro). To make the tweet time customized by user's timezone, we can use Geocoding API and [Timezon API](https://developers.google.com/maps/documentation/timezone/intro) to do so. To add more feature to know the reason of negative sentiment, we can use location to find weather information by [open weather api](https://openweathermap.org/history), or if we want to know more about the flight information we can use [skyscanner api](http://business.skyscanner.net/portal/en-GB/Documentation/ApiOverview).
2. Explore deeper to twitter streaming api. Our data show text from users to certain topic, well different people have different habit of expressing feelings, so we better customized their sentiment with their h`istorical record on twitter, and [user stream api](https://dev.twitter.com/streaming/userstreams) can help us do that.
3. Find other dataset. Ultimately we are trying to build a text miner, and that is not neccesearily built on twitter data, we can do it by Amazon or Yelp customer review data too. Those data will have better sentiment measurement, such as score features, so that we can improve our miner easily.

Basically this is what we've done in this milestone, EXPECT US!