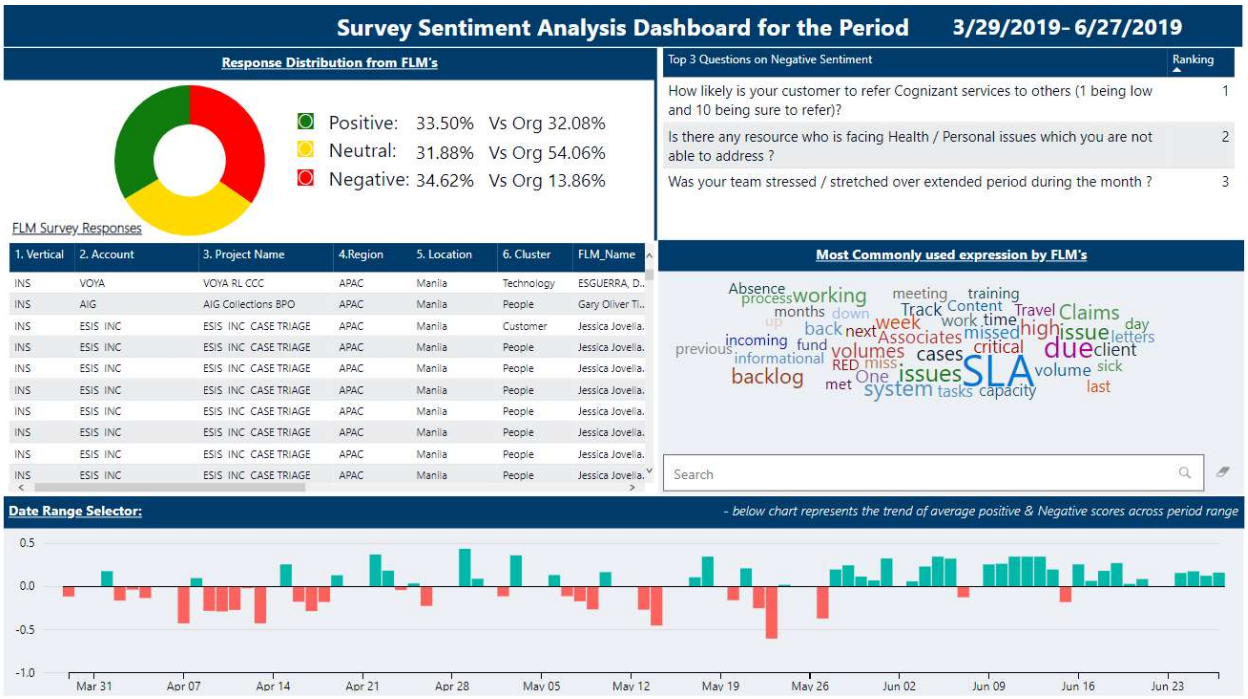
Delivery Health Dashboard – Sentiment Analysis



Context: Floor Managers (FLM) in Business Process Outsourcing (BPO) centers typically update a daily situation report. This report contains important information about the health of the processes managed by the respective FLM. The quantitative data is already being analyzed, but BPO leadership can benefit from the qualitative inputs/ comments the FLM are providing as well.

Brainstorming with BPO business stakeholders lead to the conclusion that the sentiment of the FLMs' comments provided valuable insights into process health. In addition, this sentiment could serve as a trigger for leadership attention/ interventions.

Solution approach: Options like transfer learning from movie and product review comments were considered. But it didn't work very well because the domain had very specific vocabulary that could not be analyzed by a model trained on review comments. For example, the word phrase "no developmental feedback" is considered good in operations. Further, many of the delivery centers are located in regions where English is not the native language. Hence, the FLM's language skills were limited.

After some reading and experimentation, the R package 'sentimentr', described here, was found to produce reliable output. It came with a fairly large lexicon of words and their emotion scores. We were able modify the out of the box (OOB) lexicon and add domain specific phrases to it. sentimentr also had the interesting ability the perform "valence shifting", which is described by the package author as follows:

> **So what are these valence shifters?**
>
> A negator flips the sign of a polarized word (e.g., "I do not like it."). See lexicon::hash_valence_shifters[y==1] for examples. An amplifier (intensifier) increases the impact of a polarized word (e.g., "I really like it."). See lexicon::hash_valence_shifters[y==2] for examples. A de-amplifier (downtoner) reduces the impact of a polarized word (e.g., "I hardly like it."). See lexicon::hash_valence_shifters[y==3] for examples. An adversative conjunction overrules the previous clause containing a polarized word (e.g., "I like it but it's not worth it."). See lexicon::hash_valence_shifters[y==4] for examples.

Here is how it works for a BPO dataset: (sentiment_by() is the OOB functionality & customsentimentby() is what we added to catch domain specific phrases)

sentimentr's powerful "valence shifter" functionality that can score a polarized word and negate, de-amplify or amplify it. Observe 'ave_sentiment' column.

```
> sentiment_by("happy")
   element_id word_count sd ave_sentiment
1:          1          1 NA          0.75
> sentiment_by("not happy")
   element_id word_count sd ave_sentiment
1:          1          2 NA    -0.5303301
> sentiment_by("not very happy")
   element_id word_count sd ave_sentiment
1:          1          3 NA   -0.08660254
> sentiment_by("very happy")
   element_id word_count sd ave_sentiment
1:          1          2 NA     0.9545942
```

Adding domain specific +ve & -ve phrases to the OOB emotion lexicon

words	polarity	source
no change	1	custom
not applicable	1	custom
nice work	1	custom
no developmental feedback	1	custom
no negative feedback	1	custom
no indication	1	custom

words	polarity	source
reverse transition	-1	custom
move process	-1	custom
rebadging	-1	custom
in house	-1	custom
never visit	-1	custom
no face to face	-1	custom

Scoring sentiment - OOB v/s domain specific modifications - negative statement

```
> sentiment_by("reverse transition")
   element_id word_count sd ave_sentiment
1:          1          2 NA             0
> customSentimentBy("reverse transition")
   element_id word_count sd ave_sentiment
1:          1          2 NA    -0.7071068
```

Scoring sentiment - OOB v/s domain specific modifications - positive statement

```
> sentiment_by("no developmental feedback")
   element_id word_count sd ave_sentiment
1:          1          3 NA             0
> customSentimentBy("no developmental feedback")
   element_id word_count sd ave_sentiment
1:          1          3 NA     0.5773503
```

The workflow:

Simulated Power BI report, including sentiment scores, drill down by metadata and text analysis

R code:

```
library(sentimentr)#sentiment scoring

library(tm)#preprocessing to deal with the values listed in the variable stopwords

library(stringr)#knock out excess spaces

library(dplyr)#data wrangling



setwd('C:\\Users\\654829\\Documents\\BPSDHD')

data1a <- readxl::read_excel('QuestionTrendDump_07092019-1 v1.xlsx')

data1b <- readxl::read_excel('QuestionTrendDump_07092019-2.xlsx')

data1b <- data1b %>% select(-...18)#removing a redundant column that is causing rbind trouble

data1 <- readxl::read_excel('UC  New data set for clusters v2.xlsx', sheet='Sheet1')


colnames(data1a)

colnames(data1b)

#data1 = data1a

data1 = rbind(data1a, data1b)


data1 <- data1 %>% arrange(Survey_Closed_Date)


data1$USER_COMMENT <- as.character(data1$USER_COMMENT)

data1$USER_COMMENT <- str_squish(data1$USER_COMMENT)#knock out excess spaces

stopwords = c('NULL', 'NA', 'n/a', 'na',  'N/A', 'Na', 'N.A.', 'N/a', '(', ')')    #words to remove

data1$USER_COMMENT2 <- removeWords(data1$USER_COMMENT, stopwords)     #Add the list

y <- lapply(data1$USER_COMMENT2, function(x) gsub("^$|^ $", NA, x)) #marking rows with blanks or just spaces as NA for removal later

data2 <- data1

data2$USER_COMMENT2 <- unlist(y)

sum(is.na(data2$USER_COMMENT2))#check for invalid records
```

```
data3 <- data2[!is.na(data2$USER_COMMENT2),]

data3 <- data3[sapply(strsplit(data3$USER_COMMENT," "),length)>2,]#knocking out phrases that are too short

#data3$USER_COMMENT2 <- gsub("[[:punct:]]", "", data3$USER_COMMENT2)#removes punctions

#data3$USER_COMMENT2 <- str_replace_all(data3$USER_COMMENT2, "[^[:alnum:]]", " ")#retains only alphanumeric characters

#data3$USER_COMMENT2 <- str_replace_all(data3$USER_COMMENT2, "\\s+"," ")#removes additional whitespaces inserted above

data3$USER_COMMENT2 <- tolower(data3$USER_COMMENT2)

data3x = data3

data3 <- data3[5001:5500,]


dataCustomDict <- readxl::read_excel('Word classification positive negative20190423.xlsx', sheet='Word directory')


posWords <- na.omit(dataCustomDict$Positive)#drop NA introduced by differing lengths of postive and negative word columns

#posWords <- gsub("[[:punct:]]", "", posWords)#removes punctions

posWords <- tolower(posWords)#sentimentr OOB dictionary update requires lower case

posWords <- posWords[!duplicated(posWords)]#removes duplicates within list

sum(duplicated(posWords))


negWords <- na.omit(dataCustomDict$Negative)#drop NA introduced by differing lengths of postive and negative word columns

#negWords <- gsub("[[:punct:]]", "", negWords)#removes punctions

negWords <- tolower(negWords)#sentimentr OOB dictionary update requires lower case

negWords <- negWords[!duplicated(negWords)]#removes duplicates within list

sum(duplicated(negWords))


#check for duplicate words across +ve and -ve lists

allCustomWordsList = c(posWords, negWords)
```

```
anyDuplicated(allCustomWordsList)#check for duplicate dictionary entries

allCustomWordsList[duplicated(allCustomWordsList)]#check for duplicate dictionary entries


#build dictionary with scores

posPolarity <- rep.int(1, length(posWords))

posPolarityDict <- data.frame(posWords, posPolarity, stringsAsFactors = FALSE)

colnames(posPolarityDict) <- c('words', 'polarity') #col names have to match for rbind later

negPolarity <- rep.int(-1, length(negWords))

negPolarityDict <- data.frame(negWords, negPolarity, stringsAsFactors = FALSE)

colnames(negPolarityDict) <- c('words', 'polarity') #col names have to match for rbind later

polarityDict <- rbind(posPolarityDict, negPolarityDict)

polarityDict$source <- 'custom'


#loading out of the box dictionary

OOBDict <- as.data.frame(lexicon::hash_sentiment_jockers_rinker)

colnames(OOBDict) <- c('words', 'polarity')

OOBDict$source <- 'OOB'


#creating a new dictionary that includes OOB and custom words

custPolarityDict <- rbind(OOBDict, polarityDict)

duplicates <- custPolarityDict %>% group_by(words) %>% filter(n()>1 & source == 'custom') %>% arrange(words)#finding custom words clashing with OOB words

duplicatesOOB <- custPolarityDict %>% group_by(words) %>% filter(n()>1 & source == 'OOB') %>% arrange(words)

duplicatesCheck <- cbind(duplicates, duplicatesOOB)


custPolarityDictNoDupes <- anti_join(custPolarityDict, duplicates, by=c('words', 'source'))#removing clashing custom words

custPolarityDictNoDupes <- custPolarityDictNoDupes %>% select(c('words', 'polarity'))
```

```
custPolarityDict <- as_key(custPolarityDictNoDupes)
is_key(custPolarityDict)#check for valid dictionary object

##custom function to use custom dictionary
customSentimentBy <- function(user_comment){
  sentiment_by(user_comment, polarity_dt = custPolarityDict)

}

#applying the custom function to user comments
SentimentExtractCustom <-mapply(customSentimentBy, data3$USER_COMMENT2)
sentimentrSentimentScoreCustom <- as.numeric(SentimentExtractCustom[4,])

data4 = data3
data4$sentimentrSentimentScoreCustom <- sentimentrSentimentScoreCustom

data4 %>% filter(str_detect(USER_COMMENT2, "target")) %>% select(USER_COMMENT)
#text <- data4 %>% filter(str_detect(USER_COMMENT, "issue")) %>% arrange(sentimentrSentimentScoreCustom) %>%
#  select(USER_COMMENT, sentimentrSentimentScoreCustom)

write.csv(data4, 'BPSDHDsentimentScore20190917.csv')
```