HEDONOMETER

Submitted by: Chinnu Pittapally
A20351852
cpittapa@hawk.iit.edu

Team:
1) Sachin Krishna Murthy (A20354077)
2) Chinnu Pittapally (A20351852)
3) Raghavan Kothandaraman (A20365507)

Abstract

Happiness is what most people want, whether in a family or an organization. But how do we know if people are happy or not? Hedonometer is a device to measure the happiness. So from where do we get the measure of happiness? In an organization, the views are expressed mostly in the form of emails. Hence, we are trying to extract keywords from emails that determine happiness of the employees of the organization.

Hedonometer allows us to tune the relative importance of the most emotionally charged words. It removes neutral words from consideration when determining the happiness. Using Hedonometer approach, sentimental analysis on email dataset will be performed by extracting the bodies from each email, computing the sentiment score and obtaining the sentiment words that can be used to predict the feelings of the people in an organization. Also, it is possible to obtain the total sentiment score with respect to the timeline with the help of date field in the data set.

I Data

The Enron email dataset was collected and prepared by the CALO Project. It contains data from about 150 users; mostly senior management of Enron. The corpus contains a total of about 0.5M email messages.

Enron Corporation (former New York Stock Exchange ticker symbol ENE) was an American energy, commodities, and Services Company based in Houston, Texas. Before its bankruptcy on December 2, 2001, Enron employed approximately 20,000 staff and was one of the world's major electricity, natural gas, communications and pulp and paper companies, with claimed revenues of nearly $111 billion during 2000.

The Email Dataset has a set of 20 text files. Also, it seems like some of text file has unique topic or area of interest in the chain of email in the document. Although we do notice not all the email in text file might not be related. All of the email has docID, segmentNumber, date and Body. Each email can be uniquely identified by filename, docID and segmentNumber. Another important field in the email data set is the date field. For our experiments, the date field is very important to track the timeline and sentiment score during the period.

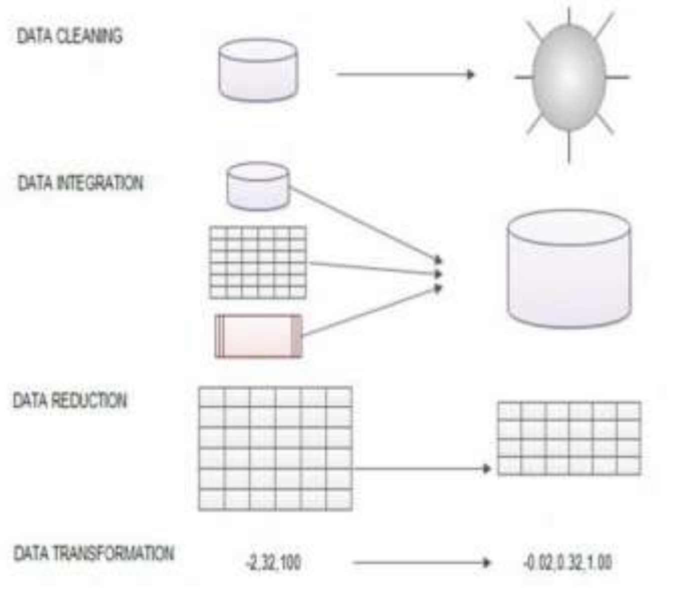
II Experiments

Lexical Analysis approach of sentiment analysis is used for our experiments below.

II.A Data Pre-processing

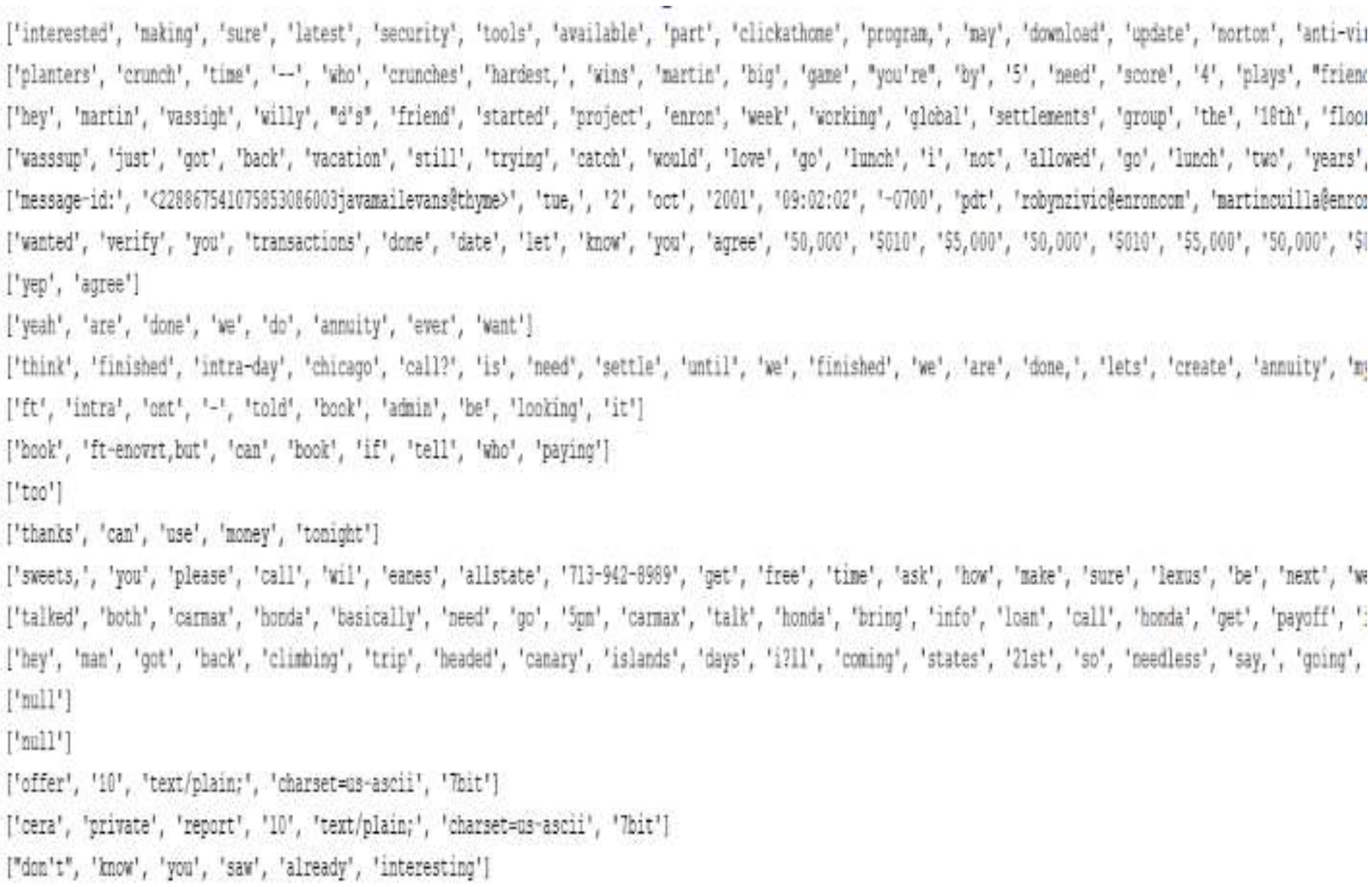
Data pre-processing is important technique that involves transforming raw data into a unique format. There is too much noise, incomplete and inconsistencies in the corpus of email data set.

Different forms of data pre-processing are represented in the figure below:



After loading all the email dataset into a corpus, we start with our data pre-processing steps. Since only the body and date are the important fields we need, we take the body of the dataset and tokenize it. Then we start with data cleaning by removing noise. For our email dataset, we remove little words like “the”, “and”, or “if”, which are known as stop words. Next comes data transformation, we do normalizing for the email data set. Shifting everything to lower case, stemming words, this is an attempt to find the root of the word.

Below are the words from the file after tokenizing. From the below list of words we can see, there is some noise and null values. We feed this list of words to get the pre-processed more clean data to be used for our further analysis of sentiment score calculation.

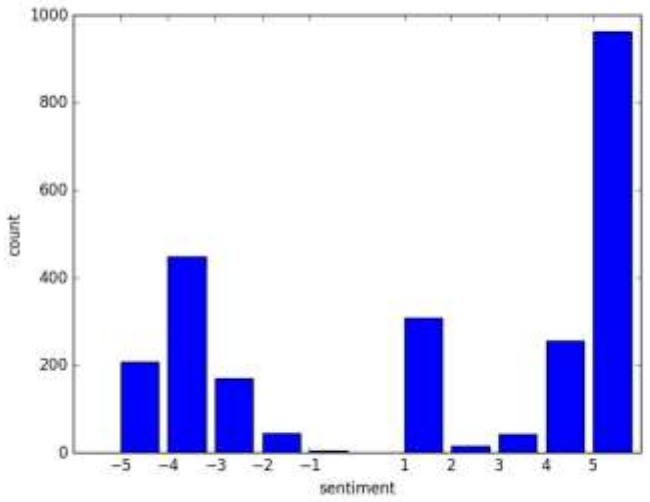


```
['interested', 'making', 'sure', 'latest', 'security', 'tools', 'available', 'part', 'clickathome', 'program,', 'may', 'download', 'update', 'norton', 'anti-vi
['planters', 'crunch', 'time', '--', 'who', 'crunches', 'hardest,', 'wins', 'martin', 'big', 'game', "you're", 'by', '5', 'need', 'score', '4', 'plays', "frien
['hey', 'martin', 'vassigh', 'willy', "d's", 'friend', 'started', 'project', 'enron', 'week', 'working', 'global', 'settlements', 'group', 'the', '18th', 'floo
['wassup', 'just', 'got', 'back', 'vacation', 'still', 'trying', 'catch', 'would', 'love', 'go', 'lunch', 'i', 'not', 'allowed', 'go', 'lunch', 'two', 'years',
['message-id:', '<22886754107585308600 3javamailevans@thyme>', 'tue,', '2', 'oct', '2001', '09:02:02', '-0700', 'pdt', 'robynzivic@enroncom', 'martincuilla@enro
['wanted', 'verify', 'you', 'transactions', 'done', 'date', 'let', 'know', 'you', 'agree', '50,000', '$010', '$5,000', '50,000', '$010', '$5,000', '50,000', '$
['yep', 'agree']
['yeah', 'are', 'done', 'we', 'do', 'annuity', 'ever', 'want']
['think', 'finished', 'intra-day', 'chicago', 'call?', 'is', 'need', 'settle', 'until', 'we', 'finished', 'we', 'are', 'done,', 'lets', 'create', 'annuity', 'm
['ft', 'intra', 'ont', '-', 'told', 'book', 'admin', 'be', 'looking', 'it']
['book', 'ft-enovrt,but', 'can', 'book', 'if', 'tell', 'who', 'paying']
['too']
['thanks', 'can', 'use', 'money', 'tonight']
['sweets,', 'you', 'please', 'call', 'wil', 'eanes', 'allstate', '713-942-8989', 'get', 'free', 'time', 'ask', 'how', 'make', 'sure', 'lexus', 'be', 'next', 'w
['talked', 'both', 'carmax', 'honda', 'basically', 'need', 'go', '5pm', 'carmax', 'talk', 'honda', 'bring', 'info', 'loan', 'call', 'honda', 'get', 'payoff', '
['hey', 'man', 'got', 'back', 'climbing', 'trip', 'headed', 'canary', 'islands', 'days', 'i?ll', 'coming', 'states', '21st', 'so', 'needless', 'say,', 'going',
['null']
['null']
['offer', '10', 'text/plain;', 'charset=us-ascii', '7bit']
['cera', 'private', 'report', '10', 'text/plain;', 'charset=us-ascii', '7bit']
["don't", 'know', 'you', 'saw', 'already', 'interesting']
```

II.B Sentiment Analysis Experiments

For our first set of experiments, we used AFINN dictionary, AFINN is a list of English words rated for valence with an integer between minus five (negative) and plus five (positive). The words have been manually labelled by Finn Årup Nielsen in 2009-2011.

Below is the distribution of positive and negative words in the AFINN dictionary.



We conducted many experiments with the AFINN dictionary, but the results were not satisfactory, since AFINN dictionary had only 2477 words, many sentences did not contribute to any sentiment score due to absence of words in sentiment dictionary. One additional issue was adding the booster sentiment score, for e.g.: ‘very good’ should have a higher score than ‘good’, also in the cases of negation.

New Approach -VADER

We started using a different approach on the sentiment score analysis called VADER. VADER Sentiment Analysis. VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments. The VADER sentiment lexicon is sensitive both the polarity and the intensity of sentiments expressed in emails.

It has 7517 words, which includes emoticons, sentiment related acronyms, slang words. Vader Sentiment Lexicon consists of features with validated valence scores that indicates both sentiment polarity that is positive or negative and the sentiment intensity on a scale from -4 which is extremely negative to +4 which is extremely positive. B_INCR and B_DECR empirically derived mean sentiment intensity rating increase for booster words. It handles the booster words, from the list of booster words, if a word has a booster word “fabulously” , it will add B_INCR = 0.293 value to the word. In the same way if a negative booster word like "scarcely": B_DECR= -0.293 will be added to the word. Basically it checks if the preceding words increase, decrease, or negate/nullify the valence.

The function “negated” is to handle the negative words appearing in the sentences. This function includes the parameter “nwords” which is the set of words in the dictionary and the parameter “NEGATE” which is the set of words defined above. This function adds on to the negative sentimental score of the email content by detecting the occurrence of the negative word in the sentence.

The function def sentiment (text): is the main function which returns the sentiment score of an email text which is passed to the function. This function returns s value which is a list of 4 values,

neg: for negative words scores,
neu: for neutral word score,
pos: for positive word score and
compound: for the normalized score considering the neg, pos and neu scores .

The compound value is normalized between -1 and 1. So for an email text provided to the sentiment function, a value between -1 and 1 is returned as the compound score.

To understand, the calculation of the neg, neu, pos and compound score, we can see the examples below:

Thank you Doug!I relayed the update verbally to Jennifer Medcalf, here is the written version. Expect a copy on a note I'm going to send EBS.
Thank : 1.5
    For the date:dec2000
    {'neg': 0.0, 'neu': 0.892, 'pos': 0.108, 'compound': 0.4199}

For the email text above, we can see that the words thank although has a positive value, it has certain valence in neutral and positive score. And the compound score of the email is only .4199.

Another example,

```
This is an excellent update.  Thanks for putting this together.Dale/Patrick - lets regroup on how we
want to move this onward.  Seems like SJ's suggestion of our spending more time with Bill Dwyer is a
good one.
excellent : 2.7
Thanks : 1.9
want : 0.3
like : 1.5
good : 1.9
        For the date:dec2000
        {'neg': 0.0, 'neu': 0.693, 'pos': 0.307, 'compound': 0.9062}

Thanks.  You're the best!
Thanks : 1.9
best : 3.2
        For the date:dec2000
        {'neg': 0.0, 'neu': 0.213, 'pos': 0.787, 'compound': 0.8122}
```

From reading the above emails, we can visually see it has high positive valence. The VADER algorithm was able to compute high compound score for the emails above.

```
The PRELIMINARY DPR indicates a Maturity Gap Risk violation for Gas Trading as follows: 12/26
Maturity/Gap Risk:  202 Bcf 12/27 Maturity/Gap Risk:  205 Bcf Maturity Gap Risk Limit:    200
BcfPlease provide an explanation for the memos.  If you have any questions, please call me at 5-
4541.
Risk : -1.1
violation : -2.2
Risk : -1.1
Risk : -1.1
Risk : -1.1
please : 1.3
        For the date:dec2000
        {'neg': 0.219, 'neu': 0.737, 'pos': 0.043, 'compound': -0.8074}
I cant' seem to make my gambling problem go away.bills +3 250denver -7 250jack +3 1/2 250
problem : -1.7
        For the date:dec2000
        {'neg': 0.153, 'neu': 0.847, 'pos': 0.0, 'compound': -0.4019}
wish you guys were going a week earlier.  i'm going next weekend.  too much vegas is bad for the
soul....and the wallet.
wish : 1.7
bad : -2.5
        For the date:dec2000
        {'neg': 0.139, 'neu': 0.754, 'pos': 0.107, 'compound': -0.2023}
```

So is the case with negative emails as seen in the above examples.

Some of the emails with both positive and negative score can also be examined.

```
sorry didnt respond to your message.   don't know how to do that instant messenger thing anymore.
volume very, very light.  most of stated volume in spreads and TAS.  No one seems to want to be in
the office this week.   Everyone wants to get this year over with.  Keep pumping in the fundamental
info.   very good stuff and i'm not getting it anywhere else.
sorry : -0.3
No : -1.2
want : 0.3
good : 2.193
            For the date:dec2000
           {'neg': 0.051, 'neu': 0.882, 'pos': 0.066, 'compound': 0.2484}

the market had to get to a price whereby these guys shut down.  there is just not enough gas to allow
everybody who wants to to burn it.  the elasticity of demand is in the industrial sector.  million dollar
question is have we gotten to a high enough price whereby we end the year with gas in the ground
and deliverability to meet a late cold snap.  shutdown of processing, distillate and resid switching,
loss of industrial load...maybe we have.good holidays,
allow : 0.9
demand : -0.5
loss : -1.3
holidays : 1.6
            For the date:dec2000
           {'neg': 0.046, 'neu': 0.899, 'pos': 0.055, 'compound': 0.1779}
```

II.C Results Summary

A dictionary object was created to store the corresponding date (monthyear) along with the score value. After the sentiment calculation, we needed to aggregate the values in a particular month year together. For this, we did summation of the sentiment scores for each month-year combination.

Note: The results explained below are from the sample subset that was executed. Sample subset is used for better understanding of the results. The final results graph is explained in detail later.

Aggregated results of month-year from sample subset

```
total score for date dec2000:
81.6185
total score for date oct2000:
9.1377
total score for date nov2000:
41.7987
total score for date jan2001:
7.378
total score for date sep2000:
3.6767
total score for date feb2001:
0.0
total score for date jul2000:
0.8957
```

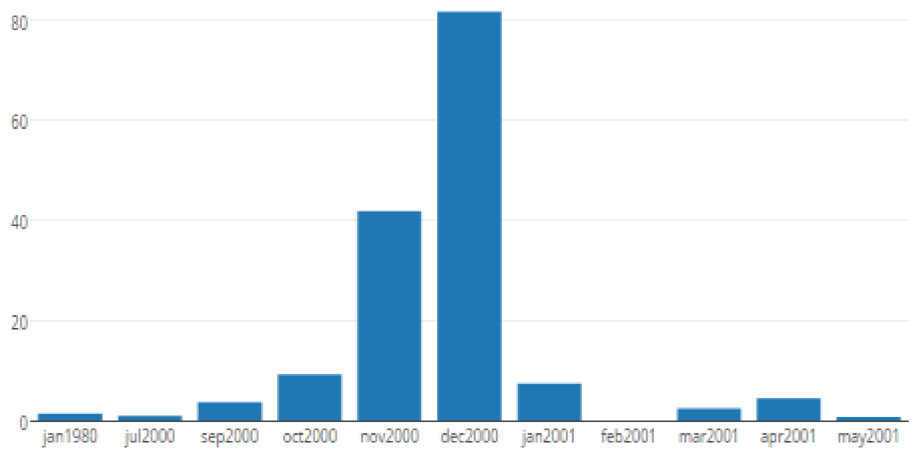
```
total score for date jan1980:
1.3324
total score for date apr2001:
4.3769
total score for date mar2001:
2.3996
total score for date may2001:
0.6129
```

We ordered the results we got in the order of dates.

Ordered results of the date with score value

```
[{'date': 'jan1980', 'score': 1.3324}, {'date': 'jul2000', 'score': 0.8957000000000002}, {'date': 'sep2000', 'score': 3.6767}, {'date': 'oct2000', 'score': 9.1377}, {'date': 'nov2000', 'score': 41.79869999999999}, {'date': 'dec2000', 'score': 81.6185}, {'date': 'jan2001', 'score': 7.378}, {'date': 'feb2001', 'score': 0.0}, {'date': 'mar2001', 'score': 2.3996}, {'date': 'apr2001', 'score': 4.3769}, {'date': 'may2001', 'score': 0.6129000000000001}]
```

<u>Visual Representation using bar graphs.</u>



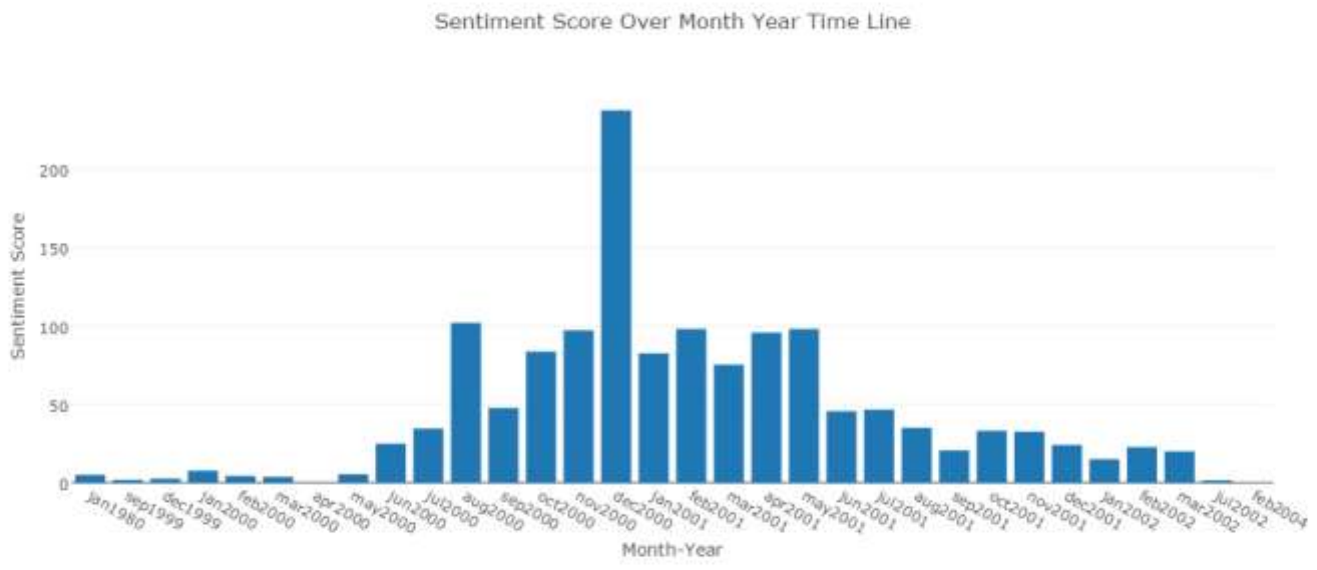
From the above bar graph which is run on a smaller sample subset, we can see a variation of dates from jan 1980 to may 2001. Although there is a huge gap between jan 1980 and jul 2000. Probably a few set of emails from jan 1980 were in the sample subset dataset we ran, hence the observation.

The sample email set observed had maximum number of emails dated from oct 2000 to jan 2001. This could be one of the reasons why these bar values are higher than most of the other value.

We do observe a very high value for dec 2000, one of the probable reasons for high value is that maximum number of emails in the subset is from dec 2000, also most emails had a positive compound score. The reason for the positive compound score could be due to the holiday season in December too.

III Analysis

The results explained below are from 2 files that contained emails ranging from January 1980 to February 2004 that was executed.



From the above graph, we can get a lot of information on the change of sentiments over time. We can clearly see a high value for months from aug2000 to may2001. Also, we do observe much lower levels of sentiment for dec2001 and jan2002.

To further analyze the reason behind sentiment variation in the months, we tried to extract the words that contributed to the score.

In the above graph, we can clearly see dec2000 having a very high score, so we got the most frequent words with high value of sentiment which would have contributed to overall high sentiment score for the month.

Dec2000 Top 5 words with high positive score

- good
- significant
- ensure
- best
- improve

Jan2001 Top 5 words with high negative score

- loss
- critical
- low
- risk
- cut

Enron company findings and results observed in graph

As part of analyzing the results, we read about the company and found most of the results are situational. August 2000 the company shares started hitting a high value. We also did notice December 2000, the company shares hit another big high. The company started facing some issues during the month of oct2001 which is also observed in the graph results. Following months, there was a lot of change in the organization. In December 2001, the company went into bankruptcy; many employees were laid off during this period. By January 2002, a criminal investigation had started.

Although we observe a lot of trends in the graph results, we do know that it is not the absolute. Many factors influence email chains in a company. Some of the situations like financial crises etc., a company is going through might be shared only by the top company management and hence the sentiments might not get reflected in rest of the organization.

Conclusion

To conclude, from our experiments, we learned VADER performed well and gave much better results than AFINN dictionary. Additionally, VADER was able to successfully handle the short coming faced when using AFINN dictionary. We were successfully able to detect the change of sentiments in the organization over the timeline with their email dataset. Many interesting observations can also be concluded from the graph results as mentioned in the analysis section above. It is interesting to note, the month of the December for every year has different measure of sentiment in the organization, highlighting the different phases the company was going through. We were also able to extract the top words that reflected the sentiment scores.

APPENDIX

Python Code

```
import os, math, re, sys, fnmatch, string
import plotly.plotly as py
import plotly.graph_objs as go
from datetime import datetime
py.sign_in('cpittapa', 'y3vi46b2iv')
reload(sys)
def make_lex_dict(f):
    return dict(map(lambda (w, m): (w, float(m)), [wmsr.strip().split('\t')[0:2] for wmsr in
open(f) ]))

f = 'vader_sentiment_lexicon.txt'
try:
    WORD_VALENCE_DICT = make_lex_dict(f)
except:
    f = os.path.join(os.path.dirname(__file__),'vader_sentiment_lexicon.txt')
    WORD_VALENCE_DICT = make_lex_dict(f)
B_INCR = 0.293
B_DECR = -0.293
c_INCR = 0.733

REGEX_REMOVE_PUNCTUATION = re.compile('[%s]' % re.escape(string.punctuation))

PUNC_LIST = [".", "!", "?", ",", ";", ":", "-", "'", "\"",
                "!!", "!!!", "??", "???", "?!?", "!?!", "?!?!", "!?!?"]
NEGATE = ["aint", "arent", "cannot", "cant", "couldnt", "darent", "didnt", "doesnt",
              "ain't", "aren't", "can't", "couldn't", "daren't", "didn't", "doesn't",
              "dont", "hadnt", "hasnt", "havent", "isnt", "mightnt", "mustnt", "neither",
              "don't", "hadn't", "hasn't", "haven't", "isn't", "mightn't", "mustn't",
              "neednt", "needn't", "never", "none", "nope", "nor", "not", "nothing",
"nowhere",
              "oughtnt", "shant", "shouldnt", "uhuh", "wasnt", "werent",
              "oughtn't", "shan't", "shouldn't", "uh-uh", "wasn't", "weren't",
              "without", "wont", "wouldnt", "won't", "wouldn't", "rarely", "seldom",
"despite"]
BOOSTER_DICT = {"absolutely": B_INCR, "amazingly": B_INCR, "awfully": B_INCR, "completely":
B_INCR, "considerably": B_INCR,
                "decidedly": B_INCR, "deeply": B_INCR, "effing": B_INCR, "enormously": B_INCR,
                "entirely": B_INCR, "especially": B_INCR, "exceptionally": B_INCR,
"extremely": B_INCR,
                "fabulously": B_INCR, "flipping": B_INCR, "flippin": B_INCR,
                "fricking": B_INCR, "frickin": B_INCR, "frigging": B_INCR, "friggin": B_INCR,
"fully": B_INCR, "fucking": B_INCR,
                "greatly": B_INCR, "hella": B_INCR, "highly": B_INCR, "hugely": B_INCR,
"incredibly": B_INCR,
                "intensely": B_INCR, "majorly": B_INCR, "more": B_INCR, "most": B_INCR,
"particularly": B_INCR,
                "purely": B_INCR, "quite": B_INCR, "really": B_INCR, "remarkably": B_INCR,
                "so": B_INCR,  "substantially": B_INCR,
                "thoroughly": B_INCR, "totally": B_INCR, "tremendously": B_INCR,
                "uber": B_INCR, "unbelievably": B_INCR, "unusually": B_INCR, "utterly":
B_INCR,
                "very": B_INCR,
                "almost": B_DECR, "barely": B_DECR, "hardly": B_DECR, "just enough": B_DECR,
                "kind of": B_DECR, "kinda": B_DECR, "kindof": B_DECR, "kind-of": B_DECR,
                "less": B_DECR, "little": B_DECR, "marginally": B_DECR, "occasionally":
B_DECR, "partly": B_DECR,
                "scarcely": B_DECR, "slightly": B_DECR, "somewhat": B_DECR,
                "sort of": B_DECR, "sorta": B_DECR, "sortof": B_DECR, "sort-of": B_DECR}

SPECIAL_CASE_IDIOMS = {"the shit": 3, "the bomb": 3, "bad ass": 1.5, "yeah right": -2,
                       "cut the mustard": 2, "kiss of death": -1.5, "hand to mouth": -2}

def negated(list, nWords=[], includeNT=True):
    nWords.extend(NEGATE)
    for word in nWords:
        if word in list:
            return True
    if includeNT:
```

```python
        for word in list:
            if "n't" in word:
                return True
    if "least" in list:
        i = list.index("least")
        if i > 0 and list[i-1] != "at":
            return True
    return False

def normalize(score, alpha=15):
    normScore = score/math.sqrt( ((score*score) + alpha) )
    return normScore

def wildCardMatch(patternWithWildcard, listOfStringsToMatchAgainst):
    listOfMatches = fnmatch.filter(listOfStringsToMatchAgainst, patternWithWildcard)
    return listOfMatches


def isALLCAP_differential(wordList):
    countALLCAPS= 0
    for w in wordList:
        if w.isupper():
            countALLCAPS += 1
    cap_differential = len(wordList) - countALLCAPS
    if cap_differential > 0 and cap_differential < len(wordList):
        isDiff = True
    else: isDiff = False
    return isDiff

#check if the preceding words increase, decrease, or negate/nullify the valence
def scalar_inc_dec(word, valence, isCap_diff):
    scalar = 0.0
    word_lower = word.lower()
    if word_lower in BOOSTER_DICT:
        scalar = BOOSTER_DICT[word_lower]
        if valence < 0: scalar *= -1
        if word.isupper() and isCap_diff:
            if valence > 0: scalar += c_INCR
            else:  scalar -= c_INCR
    return scalar

def sentiment(text):

    if not isinstance(text, unicode) and not isinstance(text, str):
        text = str(text)

    wordsAndEmoticons = text.split() #doesn't separate words from adjacent punctuation (keeps
emoticons & contractions)
    text_mod = REGEX_REMOVE_PUNCTUATION.sub('', text) # removes punctuation
    wordsOnly = text_mod.split()
    # get rid of empty items or single letter "words" like 'a' and 'I' from wordsOnly
    for word in wordsOnly:
        if len(word) <= 1:
            wordsOnly.remove(word)
    # now remove adjacent & redundant punctuation from [wordsAndEmoticons] while keeping
emoticons and contractions

    for word in wordsOnly:
        for p in PUNC_LIST:
            pword = p + word
            x1 = wordsAndEmoticons.count(pword)
            while x1 > 0:
                i = wordsAndEmoticons.index(pword)
                wordsAndEmoticons.remove(pword)
                wordsAndEmoticons.insert(i, word)
                x1 = wordsAndEmoticons.count(pword)

            wordp = word + p
            x2 = wordsAndEmoticons.count(wordp)
            while x2 > 0:
                i = wordsAndEmoticons.index(wordp)
                wordsAndEmoticons.remove(wordp)
                wordsAndEmoticons.insert(i, word)
                x2 = wordsAndEmoticons.count(wordp)

    # get rid of residual empty items or single letter "words" like 'a' and 'I' from
wordsAndEmoticons
```

```
    for word in wordsAndEmoticons:
        if len(word) <= 1:
            wordsAndEmoticons.remove(word)


    isCap_diff = isALLCAP_differential(wordsAndEmoticons)

    sentiments = []
    for item in wordsAndEmoticons:
        v = 0
        i = wordsAndEmoticons.index(item)
        if (i < len(wordsAndEmoticons)-1 and item.lower() == "kind" and \
           wordsAndEmoticons[i+1].lower() == "of") or item.lower() in BOOSTER_DICT:
            sentiments.append(v)
            continue
        item_lowercase = item.lower()
        if item_lowercase in WORD_VALENCE_DICT:
            #get the sentiment valence
            v = float(WORD_VALENCE_DICT[item_lowercase])


            if item.isupper() and isCap_diff:
                if v > 0: v += c_INCR
                else: v -= c_INCR


            n_scalar = -0.74
            if i > 0 and wordsAndEmoticons[i-1].lower() not in WORD_VALENCE_DICT:
                s1 = scalar_inc_dec(wordsAndEmoticons[i-1], v,isCap_diff)
                v = v+s1
                if negated([wordsAndEmoticons[i-1]]): v = v*n_scalar
            if i > 1 and wordsAndEmoticons[i-2].lower() not in WORD_VALENCE_DICT:
                s2 = scalar_inc_dec(wordsAndEmoticons[i-2], v,isCap_diff)
                if s2 != 0: s2 = s2*0.95
                v = v+s2
                if wordsAndEmoticons[i-2] == "never" and (wordsAndEmoticons[i-1] == "so" or
wordsAndEmoticons[i-1] == "this"):
                    v = v*1.5
                elif negated([wordsAndEmoticons[i-2]]): v = v*n_scalar
            if i > 2 and wordsAndEmoticons[i-3].lower() not in WORD_VALENCE_DICT:
                s3 = scalar_inc_dec(wordsAndEmoticons[i-3], v,isCap_diff)
                if s3 != 0: s3 = s3*0.9
                v = v+s3
                if wordsAndEmoticons[i-3] == "never" and \
                   (wordsAndEmoticons[i-2] == "so" or wordsAndEmoticons[i-2] == "this") or \
                   (wordsAndEmoticons[i-1] == "so" or wordsAndEmoticons[i-1] == "this"):
                    v = v*1.25
                elif negated([wordsAndEmoticons[i-3]]): v = v*n_scalar


                onezero = u"{} {}".format(wordsAndEmoticons[i-1], wordsAndEmoticons[i])
                twoonezero = u"{} {} {}".format(wordsAndEmoticons[i-2], wordsAndEmoticons[i-
1], wordsAndEmoticons[i])
                twoone = u"{} {}".format(wordsAndEmoticons[i-2], wordsAndEmoticons[i-1])
                threetwoone = u"{} {} {}".format(wordsAndEmoticons[i-3], wordsAndEmoticons[i-
2], wordsAndEmoticons[i-1])
                threetwo = u"{} {}".format(wordsAndEmoticons[i-3], wordsAndEmoticons[i-2])
                if onezero in SPECIAL_CASE_IDIOMS:
                    v = SPECIAL_CASE_IDIOMS[onezero]
                elif twoonezero in SPECIAL_CASE_IDIOMS:
                    v = SPECIAL_CASE_IDIOMS[twoonezero]
                elif twoone in SPECIAL_CASE_IDIOMS:
                    v = SPECIAL_CASE_IDIOMS[twoone]
                elif threetwoone in SPECIAL_CASE_IDIOMS:
                    v = SPECIAL_CASE_IDIOMS[threetwoone]
                elif threetwo in SPECIAL_CASE_IDIOMS:
                    v = SPECIAL_CASE_IDIOMS[threetwo]
                if len(wordsAndEmoticons)-1 > i:
                    zeroone = u"{} {}".format(wordsAndEmoticons[i], wordsAndEmoticons[i+1])
                    if zeroone in SPECIAL_CASE_IDIOMS:
                        v = SPECIAL_CASE_IDIOMS[zeroone]
                if len(wordsAndEmoticons)-1 > i+1:
                    zeroonetwo = u"{} {}".format(wordsAndEmoticons[i], wordsAndEmoticons[i+1],
wordsAndEmoticons[i+2])
                    if zeroonetwo in SPECIAL_CASE_IDIOMS:
                        v = SPECIAL_CASE_IDIOMS[zeroonetwo]
                if threetwo in BOOSTER_DICT or twoone in BOOSTER_DICT:
                    v = v+B_DECR

            # check for negation case using "least"
            if i > 1 and wordsAndEmoticons[i-1].lower() not in WORD_VALENCE_DICT \
                and wordsAndEmoticons[i-1].lower() == "least":
                if (wordsAndEmoticons[i-2].lower() != "at" and wordsAndEmoticons[i-2].lower()
!= "very"):
                    v = v*n_scalar
            elif i > 0 and wordsAndEmoticons[i-1].lower() not in WORD_VALENCE_DICT \
                and wordsAndEmoticons[i-1].lower() == "least":
                v = v*n_scalar
        sentiments.append(v)

    # check for modification in sentiment due to contrastive conjunction 'but'
    if 'but' in wordsAndEmoticons or 'BUT' in wordsAndEmoticons:
        try: bi = wordsAndEmoticons.index('but')
        except: bi = wordsAndEmoticons.index('BUT')
        for s in sentiments:
            si = sentiments.index(s)
            if si < bi:
                sentiments.pop(si)
                sentiments.insert(si, s*0.5)
            elif si > bi:
                sentiments.pop(si)
                sentiments.insert(si, s*1.5)
    for i,sentiment_score in enumerate(sentiments):
        if sentiment_score:
            print wordsAndEmoticons[i] + " : " + str(sentiment_score)
    if sentiments:
        sum_s = float(sum(sentiments))

        ep_count = text.count("!")
        if ep_count > 4: ep_count = 4
        ep_amplifier = ep_count*0.292
        if sum_s > 0:  sum_s += ep_amplifier
        elif  sum_s < 0: sum_s -= ep_amplifier

        qm_count = text.count("?")
        qm_amplifier = 0
        if qm_count > 1:
            if qm_count <= 3: qm_amplifier = qm_count*0.18
            else: qm_amplifier = 0.96
            if sum_s > 0:  sum_s += qm_amplifier
            elif  sum_s < 0: sum_s -= qm_amplifier

        compound = normalize(sum_s)


        pos_sum = 0.0
        neg_sum = 0.0
        neu_count = 0
        for sentiment_score in sentiments:
            if sentiment_score > 0:
                pos_sum += (float(sentiment_score) +1) # compensates for neutral words that
are counted as 1
            if sentiment_score < 0:
                neg_sum += (float(sentiment_score) -1) # when used with math.fabs(),
compensates for neutrals
            if sentiment_score == 0:
                neu_count += 1

        if pos_sum > math.fabs(neg_sum): pos_sum += (ep_amplifier+qm_amplifier)
        elif pos_sum < math.fabs(neg_sum): neg_sum -= (ep_amplifier+qm_amplifier)

        total = pos_sum + math.fabs(neg_sum) + neu_count
        pos = math.fabs(pos_sum / total)
        neg = math.fabs(neg_sum / total)
        neu = math.fabs(neu_count / total)

    else:
        compound = 0.0; pos = 0.0; neg = 0.0; neu = 0.0

    s = {"neg" : round(neg, 3),
         "neu" : round(neu, 3),
         "pos" : round(pos, 3),
         "compound" : round(compound, 4)}
    return s
if __name__ == '__main__':
    time1=datetime.now()
    reachedbody = 0
    para = []
    final=[]
    email=[]
    cnt = 0
    i=1
    for i in range(1,3):
        file_name = "sample.txt"
        with open(file_name,"r") as datafromfile:
            for line in datafromfile:
                line_processed = re.sub('[(){}!.]','',line)
                line_processed = line_processed.lower().split()
                line = line.replace('\n', '')
                if len(line_processed):
                    if line_processed[0] == 'docid:':
                        if reachedbody == 1:
                            reachedbody = 0
                            para_str = ''.join(para)
                            print para_str
                            ss = sentiment(para_str)
                            if(len(date)>9):
                                d=date[5]+date[9]
                                print "\t For the date:" +d
                            print "\t" + str(ss)
                            email.append({'score': ss['compound'], 'date': d, 'flag':
'False'})
                        para = []
                        date=""
                        cnt=cnt+1
                if len(line_processed):
                    if line_processed[0] == 'segment':
                        date=line_processed[1:]

                if len(line_processed):
                    if line_processed[0] == 'content:':
                        reachedbody = 1
                        para.append(line[9:])
                        continue
                if reachedbody == 1:
                    para.append(line)
date=[]
for i in range(len(email)):
    print(email[i])

for i in range(len(email)):
    total=email[i]['score']
    f=0
    if ((email[i]['flag'] == 'False') & (len(email[i]['date'])>0) ):
        for j in range(len(email)):
            if(email[i]['date']==email[j]['date']):
                total=total+email[j]['score']
                email[j]['flag'] = 'True'
                f=1
    if(f==1):
        print 'total score for date '+email[i]['date']+':'
        print total
        final.append({'score': total,'date':email[i]['date']})
        date.append(email[i]['date'])
date=[]
score=[]
import datetime
final.sort(key=lambda final: datetime.datetime.strptime(final['date'], '%b%Y'))
print final
for i in range(len(final)):
    date.append(final[i]['date'])
    score.append(final[i]['score'])
data = [
    go.Bar(
        x=date,
        y=score
    )
]
plot_url = py.plot(data, filename='Sentiment score over month peroid')
```