COSC 2671 Social Media and Network Analytics

Lab 4

Sentiment Analysis

Learning outcomes:

- Learn how to implement sentiment analysis for real social media data (Twitter)
- Gain understanding of limitations of sentiment analysis
- Continue to familiarise with Jupyter Notebooks and/or other IDEs

Requirements:

- A PC with Internet connection and Python installed (preference is version 3.X, but 2.7 is also okay, but unfortunately we don't have resources to provide support if there are version incompatibility issues)

Resources:

- This lab worksheet (available on Canvas)
- Associated code in lab04Code.zip (available on Canvas)

Python Packages Required:

- nltk
- matplotlib
- colorama
- pandas

Introduction

In this lab, we will perform unsupervised sentiment analysis on a set of Tweets. We will also visualise the outputs of sentiment analysis using time series and a basic text colouring package, colorama, to highlight the identified positive and negative sentiment words in the tweets. We will also observe some limitations of sentiment analysis, which isn't a bad thing, but just an indication that the problem is hard, subjective and the "no free lunch" theorem in machine learning – there is no algorithm that is good for all problems and datasets.

Data & Pre-processing

Download the attached code for this week (lab04Code.zip) and unzip into your working directory. We will reuse the RMIT Twitter dataset we used in the week 3 lab. This is available in the lab3Code.zip.

There are a number of files within this week's lab. They are:

- TwitterProcessing.py
- wordCountingSA.ipynb
- negativeWords

- positiveWords
- rmitCsTwitterTimeline.json (from last week)

TwitterProcessing.py is a python script (we will shortly explain why we have this structure), like python notebooks also contains python code etc, but it defines code to clean up Twitter data. wordCountingSA.ipynb is the Jupyter notebook that has most of the code to perform some sentiment analysis on tweets and which you will modify and add in the exercises for this lab. negativeWords and positiveWords are sets of positive and negative words, that we will use for sentiment analysis.

First returning to TwitterProcessing.py. As preprocessing for Twitter can be generic and same across multiple Twitter analysing notebooks, we have defined a class and a some functions within that python script, which allows us to load it into our notebooks as needed and don't have to constantly cut and paste the same code between notebooks.

Open the file. You don't have to understand the class structure, but if you have a look at the code, you'll notice many of the same elements are there as what we had last week for data pre-processing. There are a few additions, to clean terms that aren't useful for sentiment analysis, e.g., http links. Consider the following code extract from the file, within the *process*() function:

```
# regex pattern for http
regexHttp = re.compile("^http")

# regexHttp.match() will return None if the pattern is not in the
# string tok, i.e, tok is not a http url
return [tok for tok in tokensStripped if … and regexHttp.match(tok)
== None]
```

There are also some additional regular expressions to remove terms such as single quotes (‘).

```
stopwordList = stopwords.words('english') + punct + ['rt', 'via',
'...', '…', '"', "'"]
```

Unsupervised Sentiment Analysis

Typically for sentiment analysis, we don't have sentiment labels – this is the case for the RMIT Twitter data and likely for your assignment 1. Hence, this week we implement and explore unsupervised sentiment analysis approaches.

We will study two different approaches, to provide a contrast between them and to illustrate some important properties, such as the appropriateness of the lexicon corpus and addition of other lexical features.

Opinion Word Counting Sentiment Analysis

The simplest approach is to count the number of positive words vs negative words when analysing the sentiment of a document, e.g., tweet. This is the approach we take for this first sentiment analysis.

A set of positive and negative words, extracted from movie reviews, are available within the code zip file. These words are focused on determining the sentiment of movie reviews. Within **wordCountingSA.py**, we have provided the skeleton to perform this type of sentiment analysis to label each tweet as positive, negative or neutral (have equal number of positive and negative words).

Exercise 1:

Task 1: Essentially what we want to do is for each tweet, label it according to the following approach (this is pseudo code, not actual python code):

```
numberPosWords = countPositiveWords(tweet)
numberNegWords = countNegativeWords(tweet)
sentimentValue = numberPosWords - numberNegWords
```

We can classify a tweet as positive if sentimentValue > 0, negative if sentimentValue < 0, and neutral if sentimentValue == 0.

Within **wordCountingSA.ipynb**, in the 4th cell (the one starting with def countWordSentimentAnalysis()) in lines 30-46 (between the TODO comment blocks), implement the above pseudo code. This is within the for loop that goes through each a tweet, and what we are doing is for each tweet, we compute the sentiment value/score and store it within the variable '*sentiment'*. We use the set of positive and negative words to help with this.

Note that you will have to implement the counting method (e.g., countPositiveWords) or code to do so, e.g., as a loop or using list comprehension.
To help, the following python code would count the number of words in list tweetWords that appears in the set (of words) setExampleWords.

```
wordCount = 0
for word in tweetWords:
      if word in setExampleWords:
            wordcount += 1
```

Extension material: For those who like list comprehension, the equivalent can be done as:

```
wordCount = len([word for word in tweetWords if word in
setExampleWords])
```

After you've implemented above using one of the methods, or your own approach, run the notebook from 1st cell to the 2nd last cell (leave the last timeseries cell until later).

The 6th cell in the notebook:

```
# input file of set of positive words
```

```
posWordFile = 'positive-words.txt'
# input file of set of negative words
negWordFile = 'negative-words.txt'
# input file of set of tweets (json format)
tweetsFile = 'rmitCsTwitterTimeline.json'
# flag to determine whether to print out tweets and their sentiment
flagPrint = True
# specify the approach to take, one of [count, vader]
# change this to use a different sentiment approach
approach = 'count'
```

specifies the parameters for this notebook. Note it is considered good practice to define the parameter type of variables within one location/cell and modify them as needed.

Ensure you specify *approach* = *count* in the 6th cell, line 14 in bold, which uses this counting approach to analyse the sentiment. The other arguments are hopefully self-explanatory but ask the demonstrators if unsure. The other argument that might be worth discussing is flagPrint, which is a flag to specify to print out the tokens of each tweet and the overall sentiment. In addition, we have used colorama, a package that can change the colour of the text. In this case, positive words are in red, negative words are in blue, and neutral is in the original colour of the terminal. This can be helpful to quickly identify why a tweet has certain sentiment value and help explain things.

Vader Sentiment Analysis

In this section, we will examine a different lexicon set, one that was built from social media, hence it is expected to be more relevant to analysing the sentiment of tweets. In addition, this 2nd approach uses other lexical features, including:

- punctuation (e.g., the presence '!' increases the sentiment score of preceding sentence)
- capitalisation (e.g., "SUPER mum" vs "super mum", it increases/decrease sentiment score a constant amount)
- degree modifier (e.g., "super cool" vs "kind of cool", the first one increases the intensity of cool, while second one decrease) – Vader has a dictionary of these modifiers
- presence of but, shifts polarity (e.g., "I like apples but not during summer", this will reduce the positive degree of "like")

You will explore and determine if vader is more relevant for processing of tweets or not. In addition, this lexicon set has a sentiment score, from -4 to 4 instead of just negative or positive, which has its advantages, as it allows us to take about degrees of positive or negative sentiment.

Return to **countingWordsSA.ipynb**. We first tokenize it using the same way as before (we will explore if we don't do this). Consider the function **vaderSentimentAnalysis()**, 5th cell. It is almost identical to countWordSentimentAnalysis(), apart from the way it computes the sentiment:

```
dSentimentScores = sentAnalyser.polarity_scores(" ".join(lTokens))
```

sentAnalyser is an object of SentimentIntensityAnalyzer, which is a class implemented in nltk for sentiment analysis using the vader lexicon. We call sentAnalyser.polarity_scores(…), which analyses the sentiment/polarity of the input string. The sentiment/polarity is based on summing the positive sentiment words versus the negative sentiment words (recall that vader has degree of positiveness and negativeness, rather than binary values) modified by the lexical features described above. polarity_scores() returns a dictionary of:

dSentimentScores = {
"neg":
"neu":
"pos":
"compound:":
}

where dSentimentScores['neg'] is the negative sentiment value, dSentimentScores['pos'] is the positive sentiment value and dSentimentScores['neu'] is the neural sentiment value (roughly think of it as the amount of neutral words). All of these are normalised to the range of 0 to 1, with 1 been strongest.
dSentimentScores['compound'] is the overall sentiment, between -1 (negative) to 1 (positive).

To run this, it is the almost the same as when we run the count, but we just need to do two things. The first is to change the 6th cell to **approach = vader**, and run rerun this cell. Then rerun the 2nd last cell, which has the following code (unchanged):

```
lSentiment = []
if approach == 'count':
    lSentiment = countWordSentimentAnalysis(setPosWords, setNegWords,
    tweetsFile, flagPrint, tweetProcessor)
elif approach == 'vader':
    lSentiment = vaderSentimentAnalysis(tweetsFile, flagPrint,
    tweetProcessor)
```

This will print out each tweet, and the vader sentiment scores (neg, pos, neu) its average sentiment score (compound, ranging from -1 to 1).

Exercise 2:
Task 1: Read a few of the tweets and the sentiment scores given by vader and see if you agree or disagree. How is it different from our previous approach of using the lexicon from movie reviews (if unsure, rerun the first approach, using parameter of approach=count)? Which is do you think is doing a better job?

Next, vader has lexical features that our tokeniser may have removed. Remove our processing step in vaderSentimentAnalysis():

```
lTokens = tweetProcessor.process(tweetText)
```

Instead, pass the tweetText directly to polarity_scores(), i.e. change:

```
dSentimentScores = sentAnalyser.polarity_scores(tweetText)
```

Also within the "if bprint" score, uncomment the following print statement:

```
# print(tweetText)
```

And comment out (recall use # at start of line to comment out):

```
print(*lTokens, sep=', ')
```

Rerun the script from the 5th cell (the one defining vaderSentimentAnalysis() **)**. Do you think it makes a difference?

In addition, if you disagree with the identified sentiment of some of tweets, what does it suggest about sentiment analysis in general (hint: think of the difficulty and whether we can use more features)?

(Desirable, potentially useful for your assignments)

Task 2: One desirable way to summarise the sentiment is to draw a time series of the overall count every day. We use another important package in python, **pandas**, excellent at doing data processing, particularly time series. Along with numpy, scipy (scientic python), sklearn, they form a nice set of python packages to do many data science tasks.

To use pandas for visualising the time series of tweets, we need to first put the data into what is called a panda 'data frame' (those familiar with R will know the data frame concept well). Consider the following code:

```
import pandas as pd
import matplotlib.pyplot as plt
…

lData = [[2018-03-18 08:00:01, 10], [2018-03-18 08:10:01, 5]]
series = pd.DataFrame(lData, columns=['date', 'value'])
series.set_index('date', inplace=True)
series[['sentiment']] = series[['sentiment']].apply(pd.to_numeric)
```

Given some sample data (lData) with two columns, the first been the date, the 2nd is a numeric value, then pd.DataFrame() creates a pandas data frame, with column titles of 'date' and 'value' (these can be called anything you like, think of it as your labelling of the columns). Then series.set_index() specifies which column is used to index the instances (in our case the date, as this will be used by pandas to determine what should the x-axis be when plotting the time series). Finally, the last line ensures the column 'value' has a numeric type (pandas guesses what the type should be, and sometimes makes mistakes, so this line ensures the 2nd column is numeric).

Once we have the date in the relevant data frame, it is relatively simple to draw it:

```
series.plot()
plt.show()
```

Using this code extract example, add code to draw the time series in the last cell. Note that both countWordSentimentAnalysis() and vaderSentimentAnalysis() return a lSentiment, which is a list of [dates, sentiment]. Hint: Compare lSentiment with lData from above code extract – are they the same type? Does this help with your implementation of a time series?

To run the time series option, run the last cell.

It might be desirable to change the resolution of the time series. pandas has a built in method for this already. Consider the following:

```
newSeries = series.resample('1D').sum()
```

What resample() does is to change the resolution of the time series (in our case, 'series'). The '1D' is the resolution, in this case, one day. Others include '1H' and '1M' (hour and minute). There are other resolutions, check pandas documentation. The ".sum()" part tells pandas what to do when it finds multiple instances of the same day – in our case, we tell it to add all the sentiment values up, so we get the aggregate sentiment values for each day. The output is saved to 'newSeries' and we can plot this instead using 'newSeries.plot()' and 'plt.show()'. Implement this and experiment with different resolutions.