CS 410 Course Project Final Report

12.06.2021

—

Ratan Bajpai

MCS-DS Student

University of Illinois at Urbana-Champaign

Champaign, IL

1. Overview

This document contains the final report for the course project for CS 410. It also contains the software documentation and how to run the software.

2. Team Captain / Member (Team: Sentient)

1. Ratan Bajpai - rbajpai2@illinois.edu

3. Free Topic - Sentiment Analysis

This course project attempted to implement and analyze sentiment analysis for the "Twitter Samples" (from NLTK corpus) dataset using two approaches. The first approach is by using Amazon Comprehend NLP service and the second is the python based Natural Language Toolkit (NLTK). We analyzed the results from both and have presented our findings in this report.

4. Project Details

a. Project Task

The project task is:

i. Implement sentiment analysis on "twitter_samples" dataset using Amazon Comprehend.
ii. Implement the above sentiment analysis task using NLTK.
iii. Analyze and compare the results using both approaches.
iv. List any practical considerations to improve results.

This project task is interesting as it will compare the results of Amazon Comprehend NLP service which is a much newer system for NLP (launched in November 2017) as compared to NLTK (launched in 2001). If available we will try to list the NLP algorithms used by both systems.

b. Planned Approach

As listed in the project task, the planned approach is to implement sentiment analysis using both Amazon Comprehend and NLTK, and compare the results.

Also, we will try to see if there are any parameters in both systems that can be fine tuned to give better results. The two approaches are described in the sections down below. It also details how software is implemented.

c. Tools, Systems or Datasets

The following tools, systems and datasets will be used:

i. Amazon Comprehend, Amazon S3

ii. Boto3

iii. NLTK

iv. Twitter Samples Dataset: https://www.nltk.org/nltk_data/ (need to search for twitter_samples on this page to get to the data)

5. Twitter Samples Dataset

The "Twitter Samples" dataset which is available as a part of NLTK Corpora is collected by using Twitter APIs and filtering positive and negative sentiment tweets by using the following emojis. It contains almost 5000 positive sentiment tweets and 5000 negative sentiment tweets.

a. Positive Sentiment Tweets

Positive sentiment tweets are generated by filtering for the following emojis:

':-)', ':)', ';)', ':o)', ':]', ':3', ':c)', ':>', '=]', '8)', '=)', ':}', ':^)', ':-D', ':D', '8-D', '8D', 'x-D', 'xD', 'X-D', 'XD', '=-D', '=D', '=-3', '=3', ':-))', ":'-)", ":')", ':*', ':^*', '>:P', ':-P', ':P', 'X-P', 'x-p', 'xp', 'XP', ':-p', ':p', '=p', ':-b', ':b', '>:)', '>;)', '>:-)', '<3'

b. Negative Sentiment Tweets

Negative sentiment tweets are generated by filtering for the following emojis:

':L', ':-/', '>:/', ':S', '>:[', ':@', ':-(', ':[', ':-||', '=L', ':<', ':-[', ':-<', '=\\', '=/', '>:(', ':(', '>.<', ":'-(", ":'(", ':\\', ':-c', ':c', ':{', '>:\\', ';('

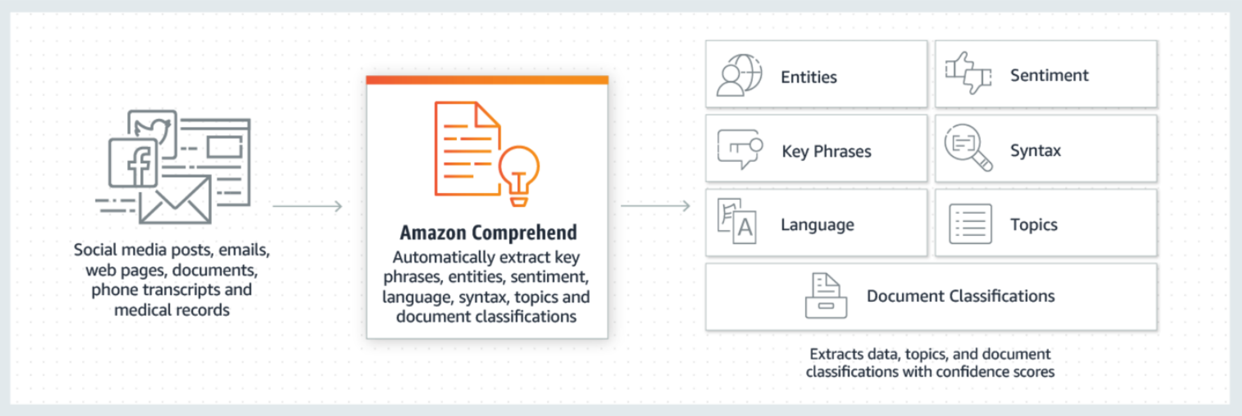
6. Amazon Comprehend

Amazon Comprehend is a high-level NLP service by AWS. It is a relatively newer service and was introduced around November 2017. It uses machine learning to gather insights and relationships from text documents. It provides a rich set of APIs so that the users can accomplish various NLP tasks such as entity / event detection, personal information identification, sentiment analysis, topic modeling etc.

7. How It Works

Amazon Comprehend continuously trains models using a large collection of text data. It then uses this pre-trained model to analyze a set of input documents to gather and generate various insights about them. This general overview is depicted in the figure below (source: Amazon AWS)

In the figure below, Amazon Comprehend ingests large quantities of a variety of text data. It then continuously trains models and automatically extracts key phrases, entities, topics etc. To use this service we need to invoke a specific API for a specific task and give it the input text document or set of documents. The API processes the input data based on pre-trained machine learning models and returns the output which could be dominant language, entities (names, places, items), positive or negative sentiment and topic phrases. Amazon Comprehend uses specific machine learning models for specific tasks. For example, for topic modeling it uses Latent Dirichlet Allocation or LDA.

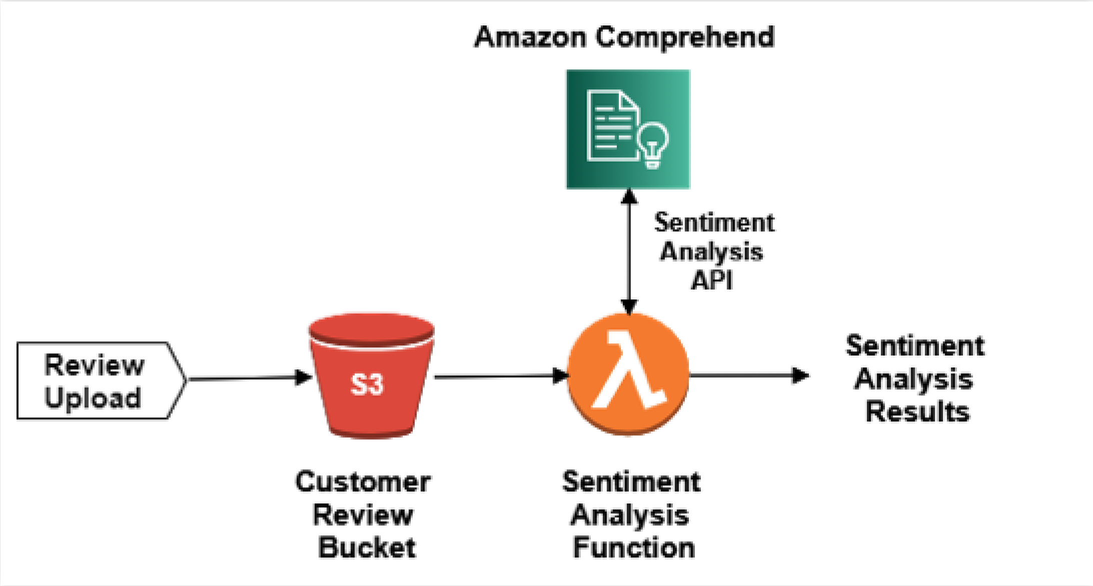

Social media posts, emails, web pages, documents, phone transcripts and medical records
Amazon Comprehend
Automatically extract key phrases, entities, sentiment, language, syntax, topics and document classifications
Entities
Sentiment
Key Phrases
Syntax
Language
Topics
Document Classifications
Extracts data, topics, and document classifications with confidence scores

8. Sentiment Analysis Using Amazon Comprehend

We did sentiment analysis using Amazon Comprehend on the “Twitter Samples” dataset which has a set of tweets that are labeled positive or negative. The dataset was downloaded to flat files using a python script and Amazon Comprehend sentiment analysis APIs were used to sentiment analysis (labeling positive and negative tweets) on part of the “Twitter Samples” dataset.

For each set of 5000 positive and negative tweets, the dataset was split into sets 3750 and 1250 tweets. For the NLTK part which is explained in later sections, each set of approximately 3750 positive and negative sentiment tweets (total approximately 7500) was used for training the model and the remaining set of approximately 1250 sets each (total 2500) was used as the test set. To have an even comparison between Amazon Comprehend and NLTK we used the test set of 2500 tweets to predict sentiment using Amazon Comprehend.

The following picture shows the general architecture of a system which does sentiment analysis using Amazon Comprehend. The difference between this architecture and the one we implemented was instead of using an S3 bucket, the tweets data was stored in our local machine.



The following API of Amazon Comprehend was used to do sentiment analysis for each tweet in the test dataset. Since we know the sentiment of each tweet in the test dataset, accuracy of prediction was calculated. Since we saw with NLTK that the emojis in the tweet can become dominant features in predicting the sentiment of a tweet, we tried two test datasets, one with the emojis and one without the emojis. Results for both are indicated in the later sections for both Amazon Comprehend and NLTK.

a. DetectSentiment API

This API takes the following request / response structure (source: Amazon Comprehend website):

- Request

```
{
   "LanguageCode": "string",
   "Text": "string"
}
```

Here the “LanguageCode” denotes what language the text string is in, i.e. “en”, “de”, “es” etc. A list of these codes along with description is available on Amazon Comprehend website. The “Text” is the actual text string (tweet text) for which the sentiment analysis has to be done.

- Response

```
{
   "Sentiment": "string",
   "SentimentScore": {
      "Mixed": number,
      "Negative": number,
      "Neutral": number,
      "Positive": number
   }
}
```

The response contains the “Sentiment” result, which could be “POSITIVE”, “NEGATIVE”, “NEUTRAL” or “MIXED”. Also it contains a score for each of those sentient values. Below is an example of such a response.

- Example Response

```
{
    "SentimentScore": {
        "Mixed": 0.003354295156896 1143,
        "Positive": 0.9869875907897949,
        "Neutral": 0.008563132025301456,
        "Negative": 0.0010949420975521207
    },
    "Sentiment": "POSITIVE",
 }
}
```

In this example response, the score of the positive sentiment is clearly the highest, hence the text is characterized as positive sentiment. These kinds of sentiment scores are useful as the API user can use their own thresholding mechanisms to decide the sentiment value in case the scores are not clearly in favor of one of the sentiment values.

b. Sentiment Analysis Lambda

This lambda function (sentiment_analysis_lambda.py) runs in AWS and needs the tweet text and language code as input to invoke the DetectSentiment API. The code for this function is available under “src/amazon_comprehend” in the github repository.

c. Sentiment Analysis Client

The sentiment analysis client (sentiment_analysis_amazon_comprehend.py) can be run on a local machine. The code for this client is available under “src/amazon_comprehend”. It uses the boto3 library from Amazon to get the handle of the lambda function running in AWS. It prepares the credentials to be used for accessing the AWS lambda. This client reads the tweet text from the data files stored under “data/amazon_comprehend” one at a time in the main loop and invokes the AWS lambda. It then parses the response from the AWS lambda, compares it with the labeled test data and determines whether AWS Comprehend did the correct classification. After processing all the test data, it calculates the accuracy of the sentiment classification.

d. Results For Data With Emojis

Amazon comprehend gave the following results for the test data with emojis. SInce Amazon Comprehend classifies in four categories (“Positive”, “Negative”, “Neutral”, “Mixed”), only the results that matched the positive and negative labels of the test data were considered accurate when calculating accuracy.

```
Correct positive count:  934
Correct negative count:  938
Neutral count:  740
Mixed count:  162
Other count:  0
Total count:  2774
Accuracy:  0.675
```

e. Results For Data Without Emojis

Following are the results for data emojis.

```
Correct positive count:  741
Correct negative count:  545
Neutral count:  1378
Mixed count:  110
Other count:  0
Total count:  2774
Accuracy:  0.464
```

f. Some Thoughts on Amazon Comprehend Classification

We can clearly see that the accuracy dropped when we removed the emojis from the test dataset. So it seems like the Amazon Comprehend treats them as additional features to help classify. We can see that a significant amount of tweets were classified as "Neutral" or "Mixed". On a closer look, we can see that even though the NLTK Corpora classified these tweets only as "Positive" or "Negative" based on the associated emojis, if we remove the emojis, some of these tweets are indeed "Neutral" or "Mixed". That may be the reason why Amazon Comprehend did not do a very good job on this test data set which contained tweets that were only labeled as "Positive" or "Negative" tweets. Below are a few examples as classified by Amazon Comprehend.

@johnniewalkerph a photographer took a photo of us last night, where can i find it? Or all the photos of the event. (classification NEUTRAL)

@marcherlord1 Leave a note on the fridge. (classification NEUTRAL)

However some of them do feel wrongly classified. Some examples are below:

I will keep fighting for what I want to be. (classification NEUTRAL, but is POSITIVE)

It's always hard leaving my heart. I'mo sulk all day today. (classification NEUTRAL, but is NEGATIVE)

9. Natural Language Toolkit

The second technique that we used to classify the same dataset as described in some of the previous sections is by using the Natural Language Toolkit (NLTK) which is a python library. We wanted to see how Amazon Comprehend compares to some of the low level libraries. Amazon Comprehend does not need any training and is a simple to use high level set of APIs for NLP tasks. However, NLTK is a low level library in which the model first needs to be trained before it can do sentiment classification. Also a lot of work needs to be done before the data can be fed as input for training the model. These steps that were implemented in the program (sentiment_analysis_nltk.py) are described below. The source for this program is available under “src/nltk” in the github repository.

a. Tokenization

The tweet text is first tokenized using an in-built tokenizer that comes with NLTK. This tokenizer is pre-trained so that it can handle various scenarios with names etc. It tokenizes the tweet into individual tokens (or words) as shown in the example below.

Input Text: #FollowFriday @France_Inte @PKuchly57 @Milipol_Paris for being top engaged members in my community this week :)

Output Tokens: ['#FollowFriday', '@France_Inte', '@PKuchly57', '@Milipol_Paris', 'for', 'being', 'top', 'engaged', 'members', 'in', 'my', 'community', 'this', 'week', ':)']

b. Part of Speech Tagging

This step involves attaching a part of speech tag (noun, noun phrase, verb etc.) to each token. The built-in “pos_tag()” function from the NLTK library is used to do POS tagging. After this step is performed, the output tokens would look something like below.

After POS Tagging: [('#FollowFriday', 'JJ'), ('@France_Inte', 'NNP'), ('@PKuchly57', 'NNP'), ('@Milipol_Paris', 'NNP'), ('for', 'IN'), ('being', 'VBG'), ('top', 'JJ'), ('engaged', 'VBN'), ('members', 'NNS'), ('in', 'IN'), ('my', 'PRP$'), ('community', 'NN'), ('this', 'DT'), ('week', 'NN'), (':)', 'NN')]

c. Tweet Text Cleanup

Each tweet may contain Twitter handles, hashtags, URLs, and punctuation marks. This step includes cleanup of all the characters that are not useful or necessary for the classification task. Regular expressions and python's "re" is used to accomplish this step.

d. Stop Words Removal

This step involves removing the common stop words like "is", "the", "and", "for" etc. NLTK has a set of common stop words that can be obtained as a list using an API call. We used the NLTK's built-in set for stop words removal.

e. Normalizing the Tokens

Normalizing means reducing a word or token to its canonical form. Normalization can be achieved via lemmatization or stepping. We used NLTK's "Word Lemmatizer" to do lemmatization. This lemmatizer used the word's context and it's part of speech (POS) tag to normalize the word into its canonical form. That's the reason we did POS for the tokens in the previous steps. The steps "b" through "e" are done in the function "clean_and_lemmatize_tokens()" of the program. After all these steps, the tweet indicated in previous sections looks like the one below.

Original Tweet: #FollowFriday @France_Inte @PKuchly57 @Milipol_Paris for being top engaged members in my community this week :)

Lemmatized Tokens: ['top', 'engage', 'member', 'community', 'week']

f. Creating Training Dataset

For each tweet data the lemmatized tokens become the features of the data vector. For creating each input data, a flag is attached to each feature to indicate if that feature is included in that training data. Also, each training data has to be labeled as a "Positive" or "Negative" tweet. After some processing the lemmatized tokens were transformed to one of the following example input data. Here "True" means that we have included all features in this example.

Input Data for Classifier: ({'top': 'True', 'engage': 'True', 'member': 'True', 'community': 'True', 'week': 'True'}, 'Positive')

g. Classifier

A "NaiveBaseClassifier" model which is one of the models available in NLTK was used for training and classification of the data. Each set of 5000 positive and negative tweets were split into 3750 for training (total 7500 training data items) and 1250 for testing (total 2500 testing data items). Same data was used for testing as in the case of Amazon Comprehend for an even comparison. Again two sets of data, one with emojis and one without emojis was used for training and testing. The results of NLTK sentiment analysis are listed below.

h. Results For Data With Emojis

Testing accuracy is: 0.996
Most Informative Features

```
    :( =           'True'     Negati : Positi =    2214.3 : 1.0
    :) =           'True'     Positi :   Negati =  1073.8 : 1.0
    followed =     'True'     Negati : Positi =    34.3 : 1.0
    follower =     'True'     Positi :   Negati =  26.2 : 1.0
    glad =         'True'     Positi :   Negati =  25.7 : 1.0
    x15 =          'True'     Negati : Positi =    23.7 : 1.0
    arrive =       'True'     Positi :   Negati =  22.2 : 1.0
    sad =          'True'     Negati : Positi =    21.7 : 1.0
    sick =         'True'     Negati : Positi =    19.7 : 1.0
    community = 'True'       Positi :   Negati =  16.3 : 1.0
    ugh =          'True'     Negati : Positi =    13.7 : 1.0
    miss = 'True'     Negati : Positi =    13.3 : 1.0
    aw =           'True'     Negati : Positi =    13.0 : 1.0
    definitely =   'True'     Positi :   Negati =  13.0 : 1.0
    didnt =        'True'     Negati : Positi =    12.3 : 1.0
    follback =     'True'     Positi :   Negati =  12.3 : 1.0
    shame =        'True'     Negati : Positi =    12.3 : 1.0
    bestfriend =   'True'     Positi :   Negati =  11.0 : 1.0
    appreciate =   'True'     Positi :   Negati =  10.6 : 1.0
    ignore =       'True'     Negati : Positi =    10.3 : 1.0
```

i. Results For Data Without Emojis

Testing accuracy is: 0.7368
Most Informative Features

```
    followed =     'True'     Negati : Positi =    34.3 : 1.0
    follower =     'True'     Positi : Negati =    26.2 : 1.0
    glad =         'True'     Positi : Negati =    25.7 : 1.0
    x15 =          'True'     Negati : Positi =    23.7 : 1.0
    arrive =       'True'     Positi : Negati =    22.2 : 1.0
    sad =          'True'     Negati : Positi =    21.7 : 1.0
    p =            'True'     Positi : Negati =    21.4 : 1.0
    sick =         'True'     Negati : Positi =    19.7 : 1.0
```

```
community = 'True'       Positi : Negati =     16.3 : 1.0
justin =           'True'       Negati : Positi =     15.0 : 1.0
ugh =              'True'       Negati : Positi =     13.7 : 1.0
miss = 'True'       Negati : Positi =     13.3 : 1.0
aw =               'True'       Negati : Positi =     13.0 : 1.0
definitely =       'True'       Positi : Negati =     13.0 : 1.0
follback =         'True'       Positi : Negati =     12.3 : 1.0
shame =            'True'       Negati : Positi =     12.3 : 1.0
bestfriend =       'True'       Positi : Negati =     11.0 : 1.0
appreciate =       'True'       Positi : Negati =     10.6 : 1.0
ignore =           'True'       Negati : Positi =     10.3 : 1.0
opportunity = 'True'       Positi : Negati =     10.3 : 1.0
```

j. Some Thoughts on NLTK Classification

We can see that the “NaiveBayesClassifier” performed much better when emojis were included in the tweet text data. This is evident from the fact that the top two features used by the classifier in this case were the emoji “:)” for positive tweet sentiment and emoji “:(“ for negative tweet sentiment. Some of the other positive and negative features are quite obvious. However “followed” was treated as negative and “follower” was treated as positive which is interesting.

10. How To Run The Project

a. NLTK

For running the NLTK part of the project, the following steps can be followed.

i. Install python 3
ii. Install PIP, the python package manager
iii. Install NLTK using PIP ($ pip install nltk)
iv. Download the “Twitter Samples” Corpora from the **python prompt** (>>> nltk.download(‘twitter_samples’)
v. Clone the CourseProject repository (https://github.com/ratanbajpai/CourseProject)
vi. Go to the src/nltk directory
vii. For sentiment analysis with emojis, run the following command: $ python3 sentiment_analysis_nltk.py 1

viii. For sentiment analysis without emojis, run the following command: $ python3 sentiment_analysis_nltk.py 2

ix. You should see similar results as indicated for NLTK in the prior section.

b. Amazon Comprehend

In the case of Amazon Comprehend, since our credentials for executing the AWS lambda cannot be put on github or shared with other users due to Amazon's security policy, the user will have to create their own AWS lambda function. The user can paste the code provided in the file "sentiment_analysis_lambda.py" in the body of the lambda function they create. Also an IAM role with appropriate permissions need to be created and attached to the lambda function. The steps for all this are as follows.

i. On AWS create an IAM role with permissions: ComprehendFullAccess, AWSLambdaExecute.

ii. Create an AWS lambda function for python 3.8 runtime. Select "use an existing role" and select the IAM role created above.

iii. Paste the code from the file "src/amazon_comprehend/sentiment_analysis_lambda.py" in the body of the lambda function.

iv. Deploy the lambda function.

v. Create AWS security credentials (https://docs.aws.amazon.com/powershell/latest/userguide/pstools-appendix-sign-up.html)

vi. In the CourseProject repository go to the src/amazon_comprehend directory

vii. Provide the security credentials in the "sentiment_analysis_amazon_comprehend.py" by editing the line: *lambda_client = boto3.client('lambda', region_name='us-east-1', aws_access_key_id='put your access key id here, aws_secret_access_key='put your secret access key here')*

viii. For sentiment analysis with emojis, run the following command: $ python3 sentiment_analysis_amazon_comprehend.py 1

ix. For sentiment analysis without emojis, run the following command: $ python3 sentiment_analysis_amazon_comprehend.py 2

x. You should see similar results as indicated for Amazon Comprehend in the prior section. It takes a while to execute the "sentiment_analysis_amazon_comprehend.py" script. For each case there will be roughly 2500 calls made to the AWS lambda and Amazon Comprehend API. **This could take several minutes to execute.**

NOTE: Amazon Comprehend is a paid service. If you do not have the free tier, you may get charged for executing the Amazon Comprehend part of this project!!

11. Conclusions and Final Thoughts

By comparing the results from Amazon Comprehend and NLTK, we can see that for both the testing datasets (with and without emojis), NLTK performed much better than Amazon Comprehend. There may be several factors for this result. One could be that NLTK's classifier was trained on a similar dataset which was drawn from the same one as the test dataset. Whereas Amazon Comprehend trains on a wide variety of text data which may not be as specific for this data domain. Also when we analyzed some of the example tweets which were classified as "Neutral" by Amazon Comprehend, those tweets did seem neutral if we remove the positive or negative emoji.

This exercise does not conclude that Amazon Comprehend is inferior for sentiment classification although the accuracy data shown here might indicate that. We have to try datasets which are generated differently as positive and negative sentiments than just going by the emojis. We can try a dataset which is labeled based on the meaning of words and sense of the sentence instead of emojis.

We can see there was a lot more programming effort needed to implement the sentiment analysis task using NLTK. Whereas in the case of Amazon Comprehend, the coding effort was much simpler. No specific knowledge of NLP is needed to use Amazon Comprehend APIs.

For future work we can use more varied datasets to compare both tools. Also classifiers other than the "NaiveBayesClassifier" which are available in NLTK can be tried to compare the results. Amazon Comprehend does not have any model selection or tunable parameters, so not much can be tried there except using the APIs as is.