Aspect Based Sentiment Analysis using Bidirectional Residual Gated Units on Electronic Reviews

(IN ENGLISH AND HINDI)

A Report on the Final Year Project for the partial fulfillment of the Degree of
Bachelor of Technology
in
Computer Science & Engineering.

By:

Syed Muntazar Abbas
059, CSE 2K16

Avanish Gupta
014 , CSE 2K16

Gaurav Yadav
023, CSE 2K16

To:
Dept. of Computer Science & Engineering,
University Institute of Engineering and Technology,
CHHATRAPATI SHAHUJI MAHARAJ UNIVERSITY, KANPUR.

MAY 2020

BONAFIDE CERTIFICATE

This is to certify that this Project Report on: “Aspect Based Sentiment Analysis using Bidirectional Residual Gated Units” is the bonafide work of Syed Muntazar Abbas (059, CSE 2K16), Avanish Gupta (014, CSE 2K16) and Gaurav Yadav (023, CSE 2K16) carried out at home due to the Covid-19 precautionary lockdown imposed since March 2020 , for the better part of this semester.

(SIGNATURE)

Dr. Alok Kumar
Assistant Professor
Department of CSE
UIET,CSJM University
Kanpur

Abstract

In this work I present a deep learning approach to Aspect based Sentiment Analysis using Recurrent Neural Networks containing a specialized 'Residual Gated Unit' as it's recurrent unit as compared to a vanilla RNN that simple contains a straightforward previous hiddenstate connection. The approach divides the problem in two parts: a. Aspect Labeling and b. Detecting Sentiment Polarity of detected aspect labels. Referring to the literature on ABSA , this kind of approach is called the pipeline method. The proposed model hence is composed of two sub-models: one for aspect labeling and the other for aspect polarity detection. I also employ a Sentiment Lexicon in order to increase the performance of the model for polarity detection. Global word Embeddings have been used as input ,coupled with domain specific embeddings to enhance understanding. The entire model was implemented using Pytorch Library for Python and was experimented on both Hindi and English Reviews.

We also experiment with certain machine learning techniques by using dataset specific heuristics of gathering all most occurring aspect terms and a matrix driven approach to identifying those aspects within sentences.

Acknowledgment

I would like to thank Dr. Alok Kumar for introducing me to the domain of Natural Language Processing , and encouraging me to learn more about the subject in order to understand and grasp the problem statement with more command. His continuous guidance motivated us to keep striving and looking for new ideas that can be applied in order to present a novel and possibly better solution to the challenge he acquainted us with this project. I am thankful for this project as it gave me an opportunity to explore new topics and research papers in Deep Learning as well as NLP and thereby made me more knowledgeable in the process of doing so.

I also would like to express gratitude to all other teaching staff of the Computer Science & Engineering Department for their continual support.

Contents

Introduction

Background

Ever since product companies were formed, they always relied on consumer feedback to evaluate their product and to tweak and tune it to more suit the public to which they cater. Before online shopping and blogs about products came about, these companies relied on feedback forms to receive and analyze criticism of their products through manual hardbound forms distributed at retail stores or malls. Both the task of gathering this large amount of information and analyzing it are difficult and require a lot of man power. The internet gave rise to users of different products to express their sentiments on some online platform about the product and discuss it with other users. With increasing volume of these reviews of products, it became even more difficult to analyze them. As artificial intelligence and natural language processing developed further , it came to realize that with effective modeling of language and machine learning models, this process of analyzing reviews could be automated and could profit the product companies heavily by saving them the hours and man power required for manual analysis. Sentiment Analysis started as the simple problem of analyzing a specific document and the overall sentiment expressed within it written for a specific product or service. But with the advent of online shopping, the reviews expressed were more fine grained and often ambiguous. The reviews themselves didn't just talk about the product in it's entirety but specific aspects of the product and how they wished if it were a little different instead of the way they were. Simply redesigning an entire product based on one 'overall' negative review is infeasible. If only there was a system to collect all reviews of one specific product and tabulate all of it's different aspects along with whether the reviewers loved or hated them. This is the very goal of Aspect Based Sentiment Analysis.

Problem

Aspect Based Sentiment Analysis aims to summarize a bunch of reviews of a product by highlighting the aspect about which the review is written and it's associated sentiment. In general practice there are datasets in one particular domain that contain reviews over a variety of products, and what an ABSA model is supposed to do is to extract aspect words from it and label them with positive, negative or neutral sentiment labels. Over the years there have been a multitude of models both hardcoded and AI (learning) based that have aimed to tackle this problem , but hardcoded logic always seems overlook many forms of expression that is used in natural language, hence deep learning which is used to learn a hierarchy of concepts can be utilized to understand language and it's many forms of expressions better than machine learning and hand coded methods. In the last couple of years , most of the methods proposed in this problem sphere involved deep learning as some part of their methodologies and have surpassed the former methods in performance immensely.

<u>Problem Statement:</u> Given a set of sentences: S={s_1,s_2,s_3...s_n}, for each sentence s_i:
Part A:label each individual token or word with 'B' <Begin> , 'I' <Inside> or 'O' <Outside> , where 'B' indicates started token of an aspect, 'I' indicated inside token of an aspect, 'O' indicated all other tokens which are not aspect words.

Part B: Given an aspect labeled sentence ('B's 'I's and 'O's) , identify the sentiment polarity expressed towards these labeled words (aspects) within the sentence. That is , label each token again with 'P' <Positive> , 'N' <Negative> , 'E' <Neutral> or 'O' <Other/Outside> . Where 'P' indicates positive polarity, 'N' indicates negative polarity, 'E' indicates negative polarity and 'O' indicates not an aspect or other word.

Literature Review

Most of the reviewed work on ABSA deals with either aspect extraction or aspect sentiment classification but not both. Since I have mainly employed deep learning for my work, I hereby present a short summary of most of the models proposed in the recent years that are related to the one that I have used in one form or the other:

Liu et al. Utilized BiReGU recurrent networks to label both aspects and sentiments along with a cross shared unit to infer information from both tasks to aid each other's performance in labeling. BiReGU: Bidirectional Residual Gated Units.

Xu et al. Utilized the BERT language model which is pretrained on a large english corpus like wikipedia to predict sentences. They used transfer learning to post train this model in order to extract aspect words from product reviews leveraging it's pretrained knowledge of english.

Wang et al. utilized LSTM for Twitter sentiment classification by simulating the interactions of words during the compositional process. Multiplicative operations between word embeddings through gate structures are used to provide more flexibility and to produce better compositional results compared to the additive ones in simple recurrent neural network. Similar to bidirectional RNN, the unidirectional LSTM can be extended to a bidirectional LSTM by allowing bidirectional connections in the hidden layer.

Wang et al. proposed a regional CNN-LSTM model, which consists of two parts: regional CNN and LSTM, to predict the valence arousal ratings of text.

Wang et al. Described a joint CNN and RNN architecture for sentiment classification of short texts, which takes advantage of the coarse-grained local features generated by CNN and long-distance dependencies learned via RNN.

Guggilla et al. presented a LSTM-and CNN-based deep neural network model, which utilizes word2vec and linguistic embeddings for claim classification (classifying sentences to be factual or feeling).

Huang et al. proposed to encode the syntactic knowledge (e.g., part-of-speech tags) in a tree-structured LSTM to enhance phrase and sentence representation.

Akhtar et al. proposed several multi-layer perceptron based ensemble models for fine-gained sentiment classification of financial microblogs and news.

Guan et al.[62]employed a weakly-supervised CNN for sentence (and also aspect) level sentiment classification. It contains a two-step learning process: it first learns a sentence representation weakly supervised by overall review ratings and then uses the sentence (and aspect) level labels for fine-tuning.

Teng et al. proposed a context-sensitive lexicon-based method for sentiment classification based on a simple weighted-sum model, using bidirectional LSTM to learn the sentiment strength, intensification and negation of lexicon sentiments in composing the sentiment value of a sentence.

Yu and Jiang studied the problem of learning generalized sentence embeddings for cross-domain sentence sentiment classification and designed a neural network model containing two separated CNNs that jointly learn two hidden feature representations from both the labeled and unlabeled data.

Zhao et al. introduced a recurrent random walk network learning approach for sentiment classification of opinionated tweets by exploiting the deep semantic representation of both user posted tweets and their social relationships.

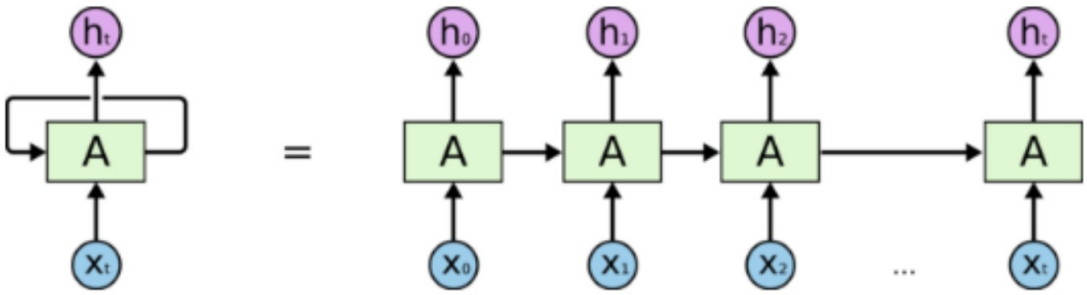
Mishra et al. utilized CNN to automatically extract cognitive features from the eye-movement (or gaze) data of human readers reading the text and used them as enriched features along with textual features for sentiment classification.

Qian et al. presented a linguistically regularized LSTM for the task. The proposed model incorporates linguistic resources such as sentiment lexicon, negation words and intensity words into the LSTM so as to capture the sentiment effect in sentences more accurately.

Methodology

BiReGU:

In this work I propose to use a special form of Recurrent Neural Network known as 'Residual Gated Unit'. It employs residual connections between each successive timestep of the sequence. A Recurrent Neural Network is basically a Deep Neural Network that accepts a sequence of data of variable length. The basic task of an RNN is sequence labeling, either many to one or many to many. The RNN uses self – connections, i.e, after taking one input , it can go on to take further inputs in the sequence along with the it's previous output in order to learn the sequence entirely and not just the individual inputs. Such a network is useful in sentence classification because it can understand the overall meaning of a sentence which is a permutation of words with a specific meaning.



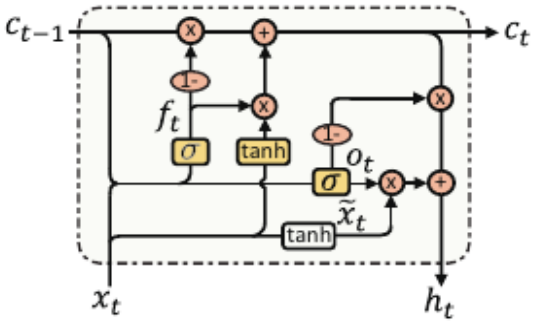
An unrolled recurrent neural network.

Here, x0..xn are elements of a sequence (or sentence), and the RNN creates hidden representations at each step of the sequence until the very end.

Parameter sharing: the unit within an RNN is the box labeled with 'A' in the figure, and the same unit is used at each step, just with a different input. Hence the same neural network can learn about all forms of input in the sequence.
So the RNN is able to classify a sequence as a whole and not just individual elements. This enables it to understand the context of each sentence input to it when it is used in language modeling.

Normal RNN vs a Residual Gated Unit: A normal (or vanilla) RNN uses basic connections from the previous timestep in the sequence from the very first to the last element , this connection is an n dimentional vector input which

summarizes the past inputs to the RNN. A normal RNN suffers with long sequence forgetting problem , when it encounters rather long sequences it cannot remember an input from 10 timesteps before which might affect the current output. Plus, a long sequence often causes vanishing gradients, that is during the training process the backpropagation through time from the last to first timestep has to calculate gradients one by one, if the gradient is less than one for a unit, it might diminish as the algorithm backpropagates further into the past. The Residual Gated unit aims to alleviate these problems by using forget gates and skip connections. Forget gates are used to learn whether to remember or discard information from some timestep and skip connections ensure that information from one timestep is effectively communicated to the future ones.



A Residual Gated Unit

The Bidirectional Residual Gated Unit, proposed by **Liu et al.** Used two ReGU RNNs , one trained from left to right on a given sequence, the other trained from right to left and the resulting hidden representations are concatenated. That is , eg. if the hiddensize is specified as 300, the final hidden representation will have a dimension of 600. For each timestep a hidden representation of this size is created and fed to a **Softmax Classifier.** Which is trained along with the RNN to predict the correct class label for the word /token at the particular timestep.

Unit Equations:

Given input x(t) at time t and the previous memory cell c(t−1), the new memory cell c(t) is calculated via the following equation:

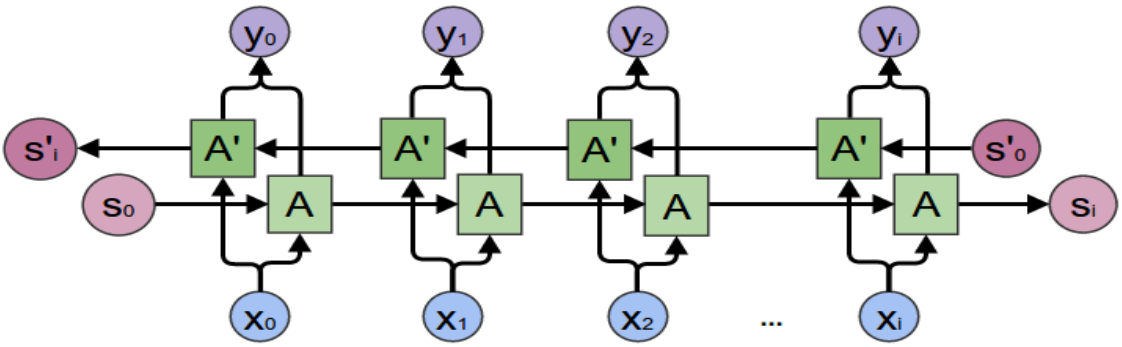
$$c_t = (1 - f_t) \odot c_{t-1} + f_t \odot \tanh(W_i x_t)$$

$$f_t = \sigma\left(W_f x_t + U_f c_{t-1}\right)$$

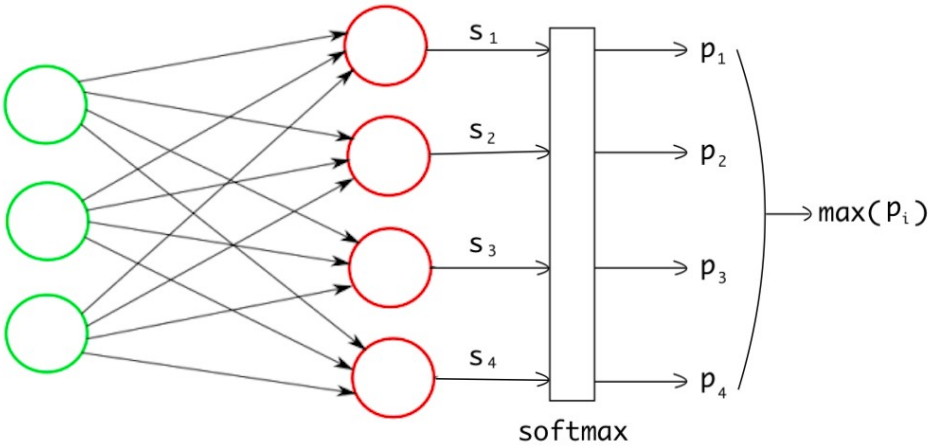
$$h_t = (1 - o_t) \odot c_t + o_t \odot \tilde{x}_t,$$

$$\sigma\left(W_o x_t + U_o c_{t-1}\right)$$

FIGURES:



A Bidirectional RNN



A Softmax Classifier

Softmax Layer

A **Softmax** layer is applied at each timestep of the RNN to predict the label for that position in the sequence.

The softmax layer applies the softmax function :

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

This gives a score between 0 and 1 for each class (for multiclass classification) which we can use as a distribution predicted by the model for a given input.

Word Embeddings:

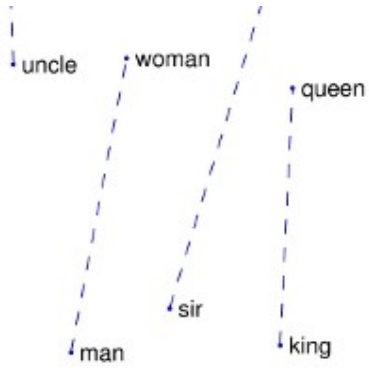
In order to encode natural language words in a numeric form to use as input in the model, we need an efficient way to represent them as vectors. Word embeddings are a way to encode semantic information of words in fixed dimensional vectors. There is one way to represent each word as a vector , the one-hot encoding, where the size of the vector is the size of the vocabulary and the each ith dimension represents each word in the vocabulary, for each word, one of the values in the vector is one while all the others are zero. Neither does this method encode any semantic knowledge, but it also very high in dimensionality. Hence word embeddings were introduced to encode word meaning in a smaller dimensional vector. The methodology is as follows:
Distributional Semantics states that any two words that are used in the same context will have the same meaning. Hence the Glove word embeddings use unsupervised classification from a corpus to determine a list of words
that would most likely occur along a given input word using a **word co – occurrence matrix** . A neural network is used to perform the training. The input is the one hot encoded word vector. The product of the input and weight matrix of the network for each word gives the word embedding value of the that word. This is derived after the network has been fully trained. The weight matrix is called the **Embedding layer.**

The Advantage of using Embeddings is that since it captures semantic information, hence words that are similar in meaning or are being used within the same context will have similar embedding values. Hence if the embedding is calculated in a 3d space (3d embeddings) , the points representing two similar words will be closer.

Such a procedure can be used to derive word embeddings of any natural language out there. Hence **fasttext** has trained embedding matrices for over 157 different languages using corpus derived from wikipedia entries of that language.

Double Embeddings

As proposed by Liu et al., using glove embeddings that are trained on wikipedia or general english language text, may not be sufficient for Aspect based Sentiment Analysis, since usually we need a model for one domain (eg.laptops, restaurants etc). Hence we can use the general embeddings concatenated with domain specific embeddings. That is , word vectors that are trained on corpus that is only based on the domain on which we are working on. In this model I have used domain specific embeddings that are trained on amazon reviews of electronics so that we can get better representations of specific technical terms and gadget names that may not occur in general english text . The size of domain embeddings is 100, hence the effective word vector size becomes 400.



<<words used in similar context tend to be represented near each other using word embeddings.

The **gensim** library in python was used to train domain specific word embeddings on amazon electronic reviews corpus. By using additional domain specific embeddings , we eliminate the problem of technical words being out of the general embeddings word vocabulary.

Sentiment Lexicon

A Sentiment Lexicon is a collection of words in a natural language annotated with their corresponding sentiment polarity scores based on how they are used in real life conversations and other text. I have employed the use of Sentiment lexicons both in Hindi and English as an additional feature for each input of the second layer which is used for polarity detection. This feature essentially indicates the sentiment value of the input word as follows: 0 for neutral word, 1 for positive word and -1 for negative word.

The reason to use this addition was to aid the network in understanding which words affect the polarity of labeled aspect words in the sentence. Instead of spending training time in learning which words represent negative or positive emotions, this lexicon makes the job easier.

The following Lexicons were used:

1. Opinion Lexicon (a list of 6800 positive and negative words) by **Hu and Liu , KDD-2004**

2.Hindi SentiWordnet by **A. Das and S. Bandyopadhyay.(SentiWordNet for Indian Languages), 2010**

The Final Model

Leveraging the tools described above and used widely in Natural language processing, the final model proposed here comprises of dual stacked BiReGU Layers. The first layer is a bidirectional ReGU used to identify aspect words within the given sentence by predicting a label for each word out of 3 possible classes (B, I and O), the hidden representation vector of each timestep is concatenated with the sentiment polarity score of the word at that timestep using the respective Sentiment Lexicons mentioned above, and the new h+1 dimensional vectors (h is the hiddensize) are fed to the second BiReGU layer which predicts the polarity label of each aspect labeled word (or timestep in the sequence) out of 4 possible classes (P,N,E or O)

Implementation

The final model was implemented using both built in deep learning libraries as well as manual computations. Since the well established deep learning frameworks in python are based on C as a backend, they are much faster than the native python implementation. Hence for the final working model, I have used Pytorch to implement it, The Glove embeddings were downloaded from Stanford NLP website, since training those embeddings would be infeasible on my system, and would require a lot of memory. The domain specific embeddings were trained using gensim library which contain Continuous Bag of Words model for embedding training. The corpus used was the Amazon Electronics Review dataset in json. The entire model is implemented in two different .py files , one for aspect and the other for polarity detection.

Shortcomings of the available Hindi Dataset:

We obtained an annotated list of hindi review sentences written in various domains from IIT Patna's AI-NLP-ML group website which has been described below. The main flaw in this dataset is the presence of english words transliterated to hindi. For example : यूजर्स ('Users') and सेशंस ('sessions') . Since the dataset was created by machine translation from english reviews to hindi , there were many technical terms simply written in english eg. 'IEEE 802.11' etc. Hence using the hindi word embeddings , a lot of OUT OF VOCABULARY words were encountered for which randomly generated vectors were defined.

Datasets

English:

Semeval 2014 Laptop Review Dataset
excerpt:
No. of Sentences containing Aspect words: 1488

```
<text>
I charge it at night and skip taking the cord with me because of the good
battery life.
</text>
<aspectTerms>
<aspectTerm term="cord" polarity="neutral" from="41" to="45"/>
<aspectTerm term="battery life" polarity="positive" from="74" to="86"/>
</aspectTerms>
```

Hindi:

Hindi Electronics Review Dataset with annotated aspects and polarity labels by **Md Shad Akhtar, Asif Ekbal, Pushpak Bhattacharyya;** ***Aspect Based Sentiment Analysis in Hindi: Resource Creation and Evaluation, 2016***
excerpt:
No. of Sentences containing Aspect words: 3318

```
<text>
फेसबुक का सिक्योरिटी चैकअप फीचर पॉपअप की तरह यूजर्स को दिखाइ देगा।
</text>
<aspectTerms>
<aspectTerm from="10" polarity="neu" term="सिक्योरिटी चैकअप फीचर" to="31"/>
</aspectTerms>
```

Experiments and Evaluation

I trained the model with a learning rate of 0.1 and an L2 regularization parameter of 0.001 to prevent overfitting. There is an inherent **Class Imbalance** in the dataset, because since each sentence may contain some aspects or none at all, the model mostly encounters the 'O' label. Hence it is difficult for the model to learn to detect aspects if it gets sentences that have no aspect terms. The dataset was filtered in order to only contain sentences that have an aspect term. This diminished the size of the dataset substantially but helped in increasing performance greatly. In order to further address the class imbalance , the loss of class 'O' was given a lower weight compared to 'B' and 'I' , since in a sentence only a handful of words are aspect terms. The **Cross Entropy Loss** was used and **Stochastic Gradient Descent** was used to optimize the parameters.

The training algorithm used was **BackPropagation through Time,** that is backpropagation used in a normal multilayer perceptron , instead here the gradients for each parameter are aggregated over all timesteps of the sequence.

The model is trained for 100 epochs for both aspect and polarity detection.

Dataset Breakup:

English: Training- 1000 Sentences
Test- 488 Sentences
Hindi: Training- 1000 Sentences
Test- 500 Sentences

Evaluation Results

The F1 Score was used as the evaluation metric.

English	Hindi
Aspect Term Labeling:- F1 Score: 79.35%	Aspect TermLabeling:- F1 Score: 67.36%
Polarity Detection:- F1 Score: 67.34%	Polarity Detection:- F1 Score: 53.47%

Other Experimented Approaches

Aspect Term Extraction

Machine Learning Classifiers:

1. Read data from <xml> file.
2. Put data under two label i. e. text data and aspect information
3. Get a tagged text list after POS tagging over text data.
4. Apply filter over the tagged text list (Noun, Adjective ,Verb, Adverb)
5. Take most frequent aspect terms using frequency distribution.
6. Now generate a matrix of text * most common aspects and put value as NaN , Positive and Negative as per given in the dataset.

Text	cord	battery life	service center	"sales" team	tech guy	quality	GUI	applications	use	...	Price	Value	WiFi	update programs	MS applications	Internet tabs	noises	bottom of the computer	t
charge night skip taking cord good battery life	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
bought hp pavilion dv4-1222nr laptop many problems computer	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
tech guy said service center 1-to-1 exchange direct concern sales team retail shop bought netbook	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
investigated netbooks saw toshiba nb305-n410bl	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

7. Now put the value in form of 0 and 1 where 0 is for NaN and 1 is for positive or negative.

Text	cord	battery life	service center	"sales" team	tech guy	quality	GUI	applications	use	...	Price	Value	WiFi	update programs	MS applications	Internet tabs	noises	bottom of the computer
charge night skip taking cord good battery life	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
bought hp pavilion dv4-1222nr laptop many problems computer	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
tech guy said service center 1-to-1 exchange direct concern sales team retail shop bought netbook	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
investigated netbooks saw toshiba nb305-n410bl	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

8. Now take X as text data and Y as aspect information.

9. Now by using count-vectorizer transform X and Y.

Apply the classification model for aspect term extraction .Here we have used OneVsRest Classification technique (it is used for multiclass classifier where one class is true and rest other is false for any model) and used Multinomial Naïve bayes, Support Vector Classifier, Linear Support Vector Classifier and SGD Classifier.

We have trained our model using the above model and the result of each classifier is as follows.

Model used	Accuracy	Recall	F1 Score
Multinomial Naive Bayes	58%	34	41
SVC	84%	78	80.3
Linear_SVC	87%	83	84.7
SGD Classifier	93.2%	89	91.1

10. After aspect term extraction we will get the output as :-
Ex:- The battery life of the product is good.

The	O
battery	B
life	I
Of	O
The	O
Product	O
Is	O
Good	O

Conclusion & Future Work

We can definitely by witnessing the performance of these methods on ABSA that we have come a long way in automating the process of natural language understanding to a great extent. Deep learning frameworks of today really help in speeding up the process of realizing the model and inferring from it eventually. We still need some universal good quality datasets for this problem domain. But since deep learning alleviates the shortcomings of less data , by leaning complex functions easily , Recurrent Neural Networks of any form are the current state of the art in language modeling and other downstream tasks like ABSA given the scarcity of good annotated sentiment datasets and the difficulty of finding them. The model presented here is definitely not par any of the current best methods, but it still is better than most machine learning and hand coded methods used throughout the literature in solving this problem.
A comparison with other deep learning methods were not presented because of different datasets they were used with and the different tools used to build them, hence comparing evaluation scores cited in a paper written years ago with old tools and botched dataset with the current work would be unfair, and due to constraints of time it was infeasible to reproduce each and every such method , given this field of work was such a novelty to me, and I had no experience until I worked on this project itself.

The problem of Sentiment analysis is an interesting one indeed, and one wonders why such an easy problem for us is so difficult for a computer to do. Hence I would like to explore the possibility of using attention vectors and sentiment lexicons in order to boost polarity detection. Better representation of technical terms is needed in order to increase aspect detection performance. Better domain specific word embeddings are needed for the domain in which we intend to work. I used only general hindi text embeddings for hindi version of this model, domain specific embeddings for hindi would greatly increase it's peformance for both the subtasks. Hence in the future I would like to make these additions to the current presented model.

References

Deep Learning for Sentiment Analysis: A Survey Lei Zhang, LinkedIn Corporation, lzhang32@gmail.comShuai Wang, University of Illinois at Chicago, shuaiwanghk@gmail.comBing Liu, University of Illinois at Chicago, liub@uic.edu

Jeffrey Pennington, Richard Socher, and Christo-pher D. Manning. 2014. **Glove: Global vectors for word representation**. In EMNLP, pages 1532–1543

Alex Graves and Jürgen Schmidhuber. 2005. **Frame-wise phoneme classification with bidirectional LSTM and other neural network architectures**. Neu-ral Networks, 18(5-6):602–610.

Duyu Tang, Bing Qin, Xiaocheng Feng, and Ting Liu.2016a. **Effective lstms for target-dependent senti-ment classification**. In COLING, pages 3298–3307.

DOER: Dual Cross-Shared RNN for Aspect Term-Polarity Co-Extraction. Huaishao Luo, Tianrui Li, Bing Liu, Junbo Zhang. ACL, 2019

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. **Deep residual learning for image recog-nition**. In CVPR, pages 770–778.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jef-frey Dean. 2013. **Efficient estimation of word representations in vector space**. arXiv preprintarXiv:1301.3781.

Himabindu Lakkaraju, Richard Socher, and Chris Man-ning. 2014. **Aspect specific sentiment analysis using hierarchical deep learning.** In NIPS Workshop on Deep Learning and Representation Learning.

Appendix

Implemented Python Code:

BiReGU Layer:

```
import torch as t
dev='cpu'

class BiReGU(t.nn.Module):
    def __init__(self,Din,hiddensize,classify=0,out=0,type='A'):
        super().__init__()
        #self.H1A=t.tensor([])
        #self.H1P=t.tensor([])
        #self.ax2=t.tensor([])
        self.type=type
        self.sig=t.nn.Sigmoid()
        self.tanh=t.nn.Tanh()
        self.soft=t.nn.Softmax(dim=0)
        self.classify=classify
        self.hiddensize=hiddensize
        self.out=out

        self.onea=(t.nn.Linear(Din,4*hiddensize))
        self.oneb=t.nn.Linear(hiddensize,2*hiddensize)
    #    self.onep=t.nn.Linear(hiddensize,2*hiddensize)
        self.onea2=(t.nn.Linear(Din,4*hiddensize))
        self.oneb2=t.nn.Linear(hiddensize,2*hiddensize)
        #self.onep2=t.nn.Linear(hiddensize,2*hiddensize)

    #    self.sixa=t.nn.Linear(hiddensize,hiddensize)
    #    self.sixb=t.nn.Linear(hiddensize,hiddensize)
    #    self.seven=t.nn.Linear(2*hiddensize,3)
    #    self.eight=t.nn.Linear(2*hiddensize,5)
        if classify==1:

            self.five=t.nn.Linear(2*hiddensize,out)

    def forward(self,x,n):
        o=t.tensor([],requires_grad=True)
        H=t.tensor([],requires_grad=True)
        second=t.tensor([],requires_grad=True)
        ct=t.zeros(self.hiddensize,device=dev,dtype=t.float32,requires_grad=True
)
        ct2=t.zeros(self.hiddensize,device=dev,dtype=t.float32,requires_grad=Tru
e)
        #aht=t.zeros(2*self.hiddensize,device=dev,dtype=t.float32,requires_grad=
True)
        #ctp=t.zeros(self.hiddensize,device=dev,dtype=t.float32,requires_grad=Tr
ue)
        #ctp2=t.zeros(self.hiddensize,device=dev,dtype=t.float32,requires_grad=T
rue)
        #print(x.shape)
        for i,xt in enumerate(x):
            xt2=x[-1-i]
            #print(i)
            #ft=self.one(xt,ct)
            a=self.onea(xt)
            b=self.oneb(ct)
        #    hta=self.sixa(a[4*self.hiddensize:5*self.hiddensize])
```

```
        #    d=self.onep(ctp)
            ft=self.sig(a[:self.hiddensize]+b[:self.hiddensize])
            ct=(1-ft)*ct+ft*self.tanh(a[self.hiddensize:2*self.hiddensize])
                                ot=self.sig(a[2*self.hiddensize:3*self.hiddensize]
+b[self.hiddensize:])
            ht=(1-ot)*ct+ot*self.tanh(a[3*self.hiddensize:4*self.hiddensize])


        #    print(ht.shape)
            a=self.onea2(xt2)
            b=self.oneb2(ct2)
            #hta2=self.sixb(a[4*self.hiddensize:5*self.hiddensize])
        #    d=self.onep2(ctp2)
            ft2=self.sig(a[:self.hiddensize]+b[:self.hiddensize])
            ct2=(1-ft2)*ct2+ft2*self.tanh(a[self.hiddensize:2*self.hiddensize])
                               ot2=self.sig(a[2*self.hiddensize:3*self.hiddensize]
+b[self.hiddensize:])
                                                                          ht2=(1-
ot2)*ct2+ot2*self.tanh(a[3*self.hiddensize:4*self.hiddensize])

            ht=t.cat((ht,ht2))
        #    htp=t.cat((htp,htp2))
        #    hta=t.cat((hta,hta2))

        #    print(ht.shape)
            if self.classify==1:
                y_=self.soft(self.five(ht)).unsqueeze(0)
                o=t.cat((o,y_))

            H=t.cat((H,ht.unsqueeze(0)))
            #sec=self.soft(self.eight(htp))

            #aux2_y_=self.soft(self.seven(hta))
            #ax2=t.cat((ax2,aux2_y_.unsqueeze(0)))
            #second=t.cat((second,sec.unsqueeze(0)))
        #print(o.shape,H.shape)
        return o,H

def train(models,train,lr,regu,epochs=5):
    loss=t.nn.CrossEntropyLoss(weight=t.tensor([3,3,0.3]))
   # loss2=t.nn.CrossEntropyLoss()
    #loss3=t.nn.CrossEntropyLoss()
    opt=t.optim.SGD(model.parameters(),lr=lr,weight_decay=regu)

    for e in range(epochs):
        at=0
        #bt=0
        #ct=0
        for i,x in enumerate(train):
            opt.zero_grad()
           # print(len(x[0]))
            #print(x[0].shape)
            #print('this is sentence:',i)
            output=model(x[0],len(x[0]))[0]
            #print(output[0].shape,output[1].shape,output[2].shape)
           # print([int(t.argmax(x)) for x in output])
           # print(x[1])
           # print()
            a=loss(output,x[1])
            #at+=a
            #b=loss(output2,x[2])
            #c=loss(output[2],x[2])
            a.backward()
                                                    #Error=loss(output[0],x[1])
+loss2(output[1],x[2])#+loss3(output[2],x[2])
```

```
            opt.step()
            at+=a
            #bt+=b
            #ct+=c
        print('loss for epoch,',e,':',at)
    #for e in range(epochs):
    print('Done:')


    return model

ds=t.load('dataset_hindi_onlyaspect')
print(len(ds))
train_=ds[:100]
print(train_[0][1])
#part1:

model=BiReGU(300,150,1,3,'A')
#model2=BiReGU(150,150,1,5,'P')
#model2=BiReGU(401,150,1,5,'P')
model=t.load('trainedmyrnn_hindi')
model=train(model,train_,0.3,0.0001,120)
t.save(model,'trainedmyrnn_hindi')
```

Machine Learning Approach 1 Python Code:

B/I/O formatting:

```
import pandas as pd


# convert into bio format
class BIO:

             def   convert_into_bio(self,   text,   predicted_aspect_terms_list,
common_words_list):
        text_list = text.split()
        # print(text_list)
        df = pd.DataFrame({'text': text_list, 'BIO': 'O'})

                bio_dict  =  self.bio_mapper(common_words_list=common_words_list,
prediction=predicted_aspect_terms_list)
        bio_filtered_dict = self.filter_predicted_words(bio_dict)

        for item in bio_filtered_dict:
            final_list = self.matcher(text, item)
            if final_list[1] - final_list[0] == 0:
                df.at[final_list[0], 'BIO'] = 'B'
            else:
                df.at[final_list[0], 'BIO'] = 'B'
                for index in range(final_list[0] + 1, final_list[1] + 1):
                        df.at[index, 'BIO'] = 'I'

        return df

    def bio_mapper(self, common_words_list, prediction):
        bio_dict = dict()

        for i in range(len(common_words_list)):
            bio_dict[common_words_list[i]] = prediction[i]

        return bio_dict
```

```
    def matcher(self, text, word):
        split_word = word.split()
        split_text = text.split(word)

        if len(split_text) > 1:
            start = len(split_text[0].split())
            end = start + len(split_word) - 1
        return start, end

    def filter_predicted_words(self, _dict):

        final_list = []

        for key, value in _dict.items():
            if value == 1:
                final_list.append(key)

        return final_list;


if __name__ == '__main__':
    text = "The battery life is really good and its size is reasonable"
    x = BIO()
    prd = [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

    common_words = ['OS',
                    'Vista',
                    'Windows',
                    'Windows 7',
                    'applications',
                    'battery',
                    'battery life',
                    'carry',
                    'charge',
                    'cost',
                    'design',
                    'display',
                    'extended warranty',
                    'features',
                    'games',
                    'gaming',
                    'graphics',
                    'hard drive',
                    'keyboard',
                    'keys',
                    'look',
                    'memory',
                    'motherboard',
                    'mouse',
                    'operating system',
                    'performance',
                    'power',
                    'power supply',
                    'price',
                    'processor',
                    'program',
                    'programs',
                    'quality',
                    'runs',
                    'screen',
                    'service',
                    'shipping',
                    'size',
                    'software',
```

```
                    'speakers',
                    'speed',
                    'system',
                    'use',
                    'value',
                    'warranty',
                    'warrenty',
                    'weight',
                    'windows',
                    'work',
                    'works']

    p=x.convert_into_bio(text, prd, common_words_list=common_words)
```

POS Tagger:

```
import nltk
from nltk.corpus import stopwords
from string import punctuation


class POSTagger:

    @staticmethod
    def pos_tagger(text):
        text = nltk.word_tokenize(text)
        stopwords_en = stopwords.words('english')
        stopwords_en_withpunct = set(stopwords_en).union(set(punctuation))
        text = [word for word in text if word not in stopwords_en_withpunct]
        tagged_pos_list = nltk.pos_tag(text)
        return tagged_pos_list

    def filter_pos_tag(self, tagged_text):
        final_text_list = []
                                                                matching_tag       =
['NN','NNS','NNP','NNPS','RB','RBR','RBS','JJ','JJR','JJS','VB','VBD','VBG','VBN
','VBP','VBZ']

        for word, tag in tagged_text:
            final_text = []
            if tag in matching_tag:
                final_text.append(word)
                final_text_list.append((' '.join(final_text)))
        return self.combine(final_text_list)

    @staticmethod
    def combine(filtered_tags):
        filtered_string = ' '.join(filtered_tags)
        return filtered_string


if __name__ == '__main__':
    x = POSTagger()
       k  =  x.pos_tagger("The  battery  life  is  really  good  and  its  size  is
reasonable")
    y = x.filter_pos_tag(k)
```

Model Code:

```
import pandas as pd
from sklearn.externals import joblib
```

```
from sklearn.feature_extraction import DictVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn import svm
from scipy.sparse import hstack
import warnings

from preprocessing.xml_2_dataframe import Xml2DataFrame
from preprocessing.pos_tagger import POSTagger

# path of training dataset

path_train = r'Laptops_Train_v2.xml'
path_test = r'C:Laptops_Test_Gold.xml'
new_test_path = r'test.xml'

# xml parser
def get_xml_data(path):
    xml2df = Xml2DataFrame()
    xml_dataframe = xml2df.process_data(path)
    return xml_dataframe

# Making list to train
train_dataframe = get_xml_data(path_train)
# print(train_dataframe.head())
train_text_list = train_dataframe['text']
train_aspects_list = list(train_dataframe['aspect_info'])
print(train_text_list.head())

# getting pos_tag for each review

def pos_tag(review):
    tagged_text_list = []
    pos_tagger = POSTagger()
    for text in review:
        tagged_text_list.append(pos_tagger.pos_tagger(text=text))
    return tagged_text_list


# pos_tag for  filtering of noun,adjective,verb,adverb

def filter_tag(tagged_reviews):
    filtered_list = []
    pos_tagger = POSTagger()
    for tagged_review in tagged_reviews:
        filtered_list.append(pos_tagger.filter_pos_tag(tagged_review))
    return filtered_list

# pos_tagging

tagged_text_list_train = pos_tag(train_text_list)

# getting the final train list after filtering

final_train_text_list = filter_tag(tagged_text_list_train)
print(final_train_text_list[:5])

#Selecting only 20 most common aspect.

def get_most_common_aspect(aspect_list):
    import nltk
    aspect_terms = []
    aspect_list = list(aspect_list.aspect_info)
    for inner_list in aspect_list:
        if inner_list is not None:
            for _dict in inner_list:
```

```
                # for key in _dict:
                aspect_terms.append(_dict.get('term'))

    most_common_aspect = [k for k, v in
nltk.FreqDist(aspect_terms).most_common(1000)]
    return most_common_aspect

# generating the data frame

def get_data_frame(text_list, train_aspects_list, most_common_aspect):
    data = {'Text': text_list}
    df = pd.DataFrame(data)
    for inner_list in train_aspects_list:
        if inner_list is not None:
            for _dict in inner_list:

                if _dict.get('term') in most_common_aspect:
                    df.loc[train_aspects_list.index(inner_list),
_dict.get('term')] = _dict.get('polarity')
    return df

# generate data frame for aspect extraction
def get_aspect_data_frame(df, most_common_aspect):
    for common_aspect in most_common_aspect:
        df[common_aspect] =
df[common_aspect].replace(['positive','negative','neutral','conflict'],
[1,1,1,1])
    df = df.fillna(0)
    return df

# getting and printing the most common aspect.

most_common_aspect = get_most_common_aspect(train_dataframe)
print(most_common_aspect)

#get data frame
df_train = get_data_frame(final_train_text_list,train_aspects_list,
most_common_aspect)
df_train.head()

# get aspect term dataframe   with most common aspect.
df_train_aspect = get_aspect_data_frame(df_train, most_common_aspect)
df_train_aspect.head()

df_train_aspect = df_train_aspect.reindex(sorted(df_train_aspect.columns),
axis=1)
df_train_aspect.head()

# Similar for test list
test_dataframe = get_xml_data(path_test)     # getting the test data from
test.xml file
test_text_list = train_dataframe['text']     # putting test data in data frame.
test_aspects_list = list(train_dataframe['aspect_info'])      # putting it into
a list
tagged_text_list_test = pos_tag(test_text_list)           # pos_tagging of the
test dataset
final_test_text_list = filter_tag(tagged_text_list_test)    # final test dataset
df_test = get_data_frame(final_test_text_list,test_aspects_list,
most_common_aspect)   # get aspect term dataframe for test data
df_test_aspect = get_aspect_data_frame(df_test, most_common_aspect)
df_test_aspect = df_test_aspect.reindex(sorted(df_test_aspect.columns), axis=1)

# Sort the data frame according to aspect's name and separate data(X) and
target(y)

# for training dataset
X_train= df_train_aspect.Text
```

```
y_train = df_train_aspect.drop('Text', 1)
print(y_train[:5])


# for testing dataset.
X_test = df_test_aspect.Text
y_test = df_test_aspect.drop('Text', 1)
final_most_common_aspect = list(y_train)
#list(y_train)

# Change y_train to numpy array
import numpy as np
y_train = np.asarray(y_train, dtype=np.int64)
y_test = np.asarray(y_test, dtype=np.int64)
print(y_train[:5])

# Generate word vecotors using CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer(max_df=1.0, stop_words='english')
X_train_dtm = vect.fit_transform(X_train)
X_test_dtm = vect.transform(X_test)

# defining the training model for aspcet term extraction
# multinomial naive bayes class classifier.
# SVM
# SGD classifier
nb_classif = OneVsRestClassifier(MultinomialNB()).fit(X_train_dtm, y_train)
C = 1.0
# SVregularization parameter
svc = OneVsRestClassifier(svm.SVC(kernel='linear', C=C)).fit(X_train_dtm,
y_train)    # fiting the training data into the model
lin_svc = OneVsRestClassifier(svm.LinearSVC(C=C)).fit(X_train_dtm, y_train)
sgd = OneVsRestClassifier(SGDClassifier(max_iter=1000)).fit(X_train_dtm,y_train)

# Predict the test data using classifiers
y_pred_class = nb_classif.predict(X_test_dtm)
y_pred_class_svc = svc.predict(X_test_dtm)
y_pred_class_lin_svc = lin_svc.predict(X_test_dtm)
y_pred_class_sgd = sgd.predict(X_test_dtm)

# Following code to test metrics of all aspect extraction classifiers

# Results

from sklearn import metrics
# printing accuracy of the model
print("Accuracy of getting the aspect term using different model:-  ")
print("Using multinomial naive bayes classifier:-
",metrics.accuracy_score(y_test,y_pred_class))
print("Using Support vector classifier:-
",metrics.accuracy_score(y_test,y_pred_class_svc))
print("Using Linear Support Vector Classifier :-
",metrics.accuracy_score(y_test,y_pred_class_lin_svc))
print("Using SGD classifier :-
",metrics.accuracy_score(y_test,y_pred_class_sgd))


# printing the  recall of the model
print("Recall of getting the aspect term using different model :- ")
print("Using multinomial naive bayes classifier:-
",metrics.recall_score(y_test,y_pred_class,average='micro'))
print("Using Support vector classifier:-
",metrics.recall_score(y_test,y_pred_class_svc,average='micro'))
print("Using Linear Support Vector Classifier :-
",metrics.recall_score(y_test,y_pred_class_lin_svc,average='micro'))
print("Using SGD classifier :-
",metrics.recall_score(y_test,y_pred_class_sgd,average='micro'))
```

```
# printing the f1 score of the model
print(" F1 score of getting the aspect term using different model:-  ")
print(metrics.f1_score(y_test,y_pred_class,average='micro'))
print("Using Support vector classifier:-
",metrics.f1_score(y_test,y_pred_class_svc,average='micro'))
print("Using Linear Support Vector Classifier :-
",metrics.f1_score(y_test,y_pred_class_lin_svc,average='micro'))
print("Using SGD classifier :-
",metrics.f1_score(y_test,y_pred_class_sgd,average='micro'))

# printing the classification  report  of different model

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    print(metrics.classification_report(y_test, y_pred_class))
    print(metrics.classification_report(y_test, y_pred_class_svc))
    print(metrics.classification_report(y_test, y_pred_class_lin_svc))
    print(metrics.classification_report(y_test, y_pred_class_sgd))

def get_dict_aspect(y,most_common_aspect):
    position=[]
    for innerlist in y:
        position.append([i for i, j in enumerate(innerlist) if j == 1])
    sorted_common=sorted(most_common_aspect)
    dict_aspect=[]
    for innerlist in position:
        inner_dict={}
        for word in sorted_common:
            if sorted_common.index(word) in innerlist:
                inner_dict[word]= 5
            else:
                inner_dict[word]=0
        dict_aspect.append(inner_dict)
    return dict_aspect

# Generating extra feature that indicates which aspect category is present in
the review
train_dict_aspect=get_dict_aspect(y_train, most_common_aspect)
d_train=DictVectorizer()
X_train_aspect_dtm = d_train.fit_transform(train_dict_aspect)

# y_test is used to generated extra feature in order to test the performance of
2nd classifer.
#Use y_pred_class_svc(Highest performer for aspect classification) as input for
extra feature to test the overall performace.
test_dict_aspect=get_dict_aspect(y_test, most_common_aspect)
d_test=DictVectorizer()

from BIO_format import BIO

def BIO_format(text, predicted_output, common_words):
    bio_obj = BIO()
    df = bio_obj.convert_into_bio(text, predicted_output,
common_words_list=common_words)
    return df

# Aspect term extractor of user's input.

#user_input = "it is of high quality, has a killer GUI, is extremely stable, is
highly expandable, is bundled with lots of very good applications, is easy to
use, and is absolutely gorgeous."

user_input=input("Enter the comment:- ")
# Preprocessing and vectorizing
```

```
tagged_user_input = pos_tag([user_input])
print(tagged_user_input)
filter_tagged_user_input = filter_tag(tagged_user_input)
print(filter_tagged_user_input)

user_input_series = pd.Series(filter_tagged_user_input)
print(user_input_series)
user_input_series_dtm = vect.transform(user_input_series)
print(user_input_series_dtm)
# print(user_input_series[:5])

predict_aspect= sgd.predict(user_input_series_dtm)
print(predict_aspect)
# predict_aspect_data = predict_aspect[0]
extra_feature=get_dict_aspect(predict_aspect, most_common_aspect)
extra_feature_dtm=DictVectorizer().fit_transform(extra_feature)
predict_aspect

df = BIO_format(user_input, predict_aspect[0], final_most_common_aspect)
df
```

Machine Learning Approach 2 Python Code: Gaurav Yadav

```
import pandas as pd
from sklearn.externals import joblib
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction import DictVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn import svm
import xml.etree.ElementTree as ET
from lxml import etree
from scipy.sparse import hstack
import numpy as np
import warnings

path_train          =          r'C:\Users\gaurav\Desktop\sentiment          code\
ABSA16_Laptops_Train_English_SB2.xml'
path_test =r'C:\Users\gaurav\Desktop\sentiment code\EN_LAPT_SB2_TEST.xml'

#For stanford POS Tagger
home = r'C:\Users\gaurav\anaconda3\stanford-tagger-4.0.0'
from nltk.tag.stanford import StanfordPOSTagger as POS_Tag
from nltk import word_tokenize
_path_to_model = home + '/models/english-bidirectional-distsim.tagger'
_path_to_jar = home + '/stanford-postagger.jar'
stanford_tag = POS_Tag(model_filename=_path_to_model, path_to_jar=_path_to_jar)
import os
java_path = "C:\Program Files\Java\jdk-11.0.2\bin\java.exe"
os.environ['JAVAHOME'] = java_path

#xml parser
def get_list(path):
    tree=ET.parse(path)
    root = tree.getroot()
    text_list = []
    opinion_list = []
    for review in root.findall('Review'):
        text_string=""
        opinion_inner_list=[]
        for sent in review.findall('./sentences/sentence'):
            text_string= text_string+ " "+ sent.find('text').text
        text_list.append(text_string)
        for opinion in review.findall('./Opinions/Opinion'):
```

```
            opinion_dict = {
                                        opinion.get('category').replace('#','_'):
opinion.get('polarity')
            }
            opinion_inner_list.append(opinion_dict)
        opinion_list.append(opinion_inner_list)
    return text_list,opinion_list

#generate data frame
def get_data_frame(text_list,opinion_list,most_common_aspect):
    data={'Review':text_list}
    df = pd.DataFrame(data)
    if opinion_list:
        for inner_list in opinion_list:
            for _dict in inner_list:
                for key in _dict:
                    if key in most_common_aspect:
                        df.loc[opinion_list.index(inner_list),key]=_dict[key]
    return df

#generate data frame for aspect extraction task
def get_aspect_data_frame(df,most_common_aspect):
    for common_aspect in most_common_aspect:
        df[common_aspect]=df[common_aspect].replace(['positive','negative','neut
ral','conflict'],[1,1,1,1])
    df = df.fillna(0)
    return df

def get_dict_aspect(y,most_common_aspect):
    position=[]
    for innerlist in y:
        position.append([i for i, j in enumerate(innerlist) if j == 1])
    sorted_common=sorted(most_common_aspect)
    dict_aspect=[]
    for innerlist in position:
        inner_dict={}
        for word in sorted_common:
            if sorted_common.index(word) in innerlist:
                inner_dict[word]= 5
            else:
                inner_dict[word]=0
        dict_aspect.append(inner_dict)
    return dict_aspect

#Selecting only 20 most common aspect.
def get_most_common_aspect(opinion_list):
    import nltk
    opinion= []
    for inner_list in opinion_list:
        for _dict in inner_list:
            for key in _dict:
                opinion.append(key)
    most_common_aspect = [k for k,v in nltk.FreqDist(opinion).most_common(20)]
    return most_common_aspect

#Filter the word with tag- noun,adjective,verb,adverb
def filterTag(tagged_review):
    final_text_list=[]
    for text_list in tagged_review:
        final_text=[]
        for word,tag in text_list:
                                                        if     tag     in
['NN','NNS','NNP','NNPS','RB','RBR','RBS','JJ','JJR','JJS','VB','VBD','VBG','VBN
','VBP','VBZ']:
                final_text.append(word)
        final_text_list.append(' '.join(final_text))
    return final_text_list
```

```
#Stage 1:
#Making list to train
train_text_list,train_opinion_list = get_list(path_train)

most_common_aspect = get_most_common_aspect(train_opinion_list)
print(most_common_aspect)

#This takes time to tag. Already tagged and saved. So, loading file ...
#tagged_text_list_train=pos_tag(train_text_list)

print(train_text_list)

print(train_opinion_list)

This takes time to tag. Already tagged and saved. So, loading file ...
tagged_text_list_train=posTag(train_text_list)
joblib.dump(tagged_text_list_train, 'tagged_text_list_train.pkl')
tagged_text_list_train=joblib.load('tagged_text_list_train.pkl')

#train list after filter
final_train_text_list=filterTag(tagged_text_list_train)
```