PROJECT REPORT

ON

“USER CHARACTERIZATION”

Submitted to

PROF. SUDARSHAN IYENGAR

BY

HARSHA VARDHAN	2015EEB1053
ANKAN BAL	2016CSB1031
SIDDHARTH BANRA	2016CSB1061

UNDER THE GUIDANCE OF
AMIT KUMAR VERMA

DEPARTMENT OF COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROPAR

2018-2019

AFFILIATED TO
INDIAN INSTITUTE OF TECHNOLOGY
ROPAR

INTRODUCTION

Today a lot of open source knowledge based website sources are there where users can contribute and learn from them. This is called crowd-sourcing. Crowd-sourcing can give entities access to a wider set of expertise at a faster pace and at a lower cost than traditional routes. On the other hand, those entities can't guarantee that the crowd they reach has the expertise, experience or resources to deliver what they need nor whether they're actually reaching the best sources to deliver the best possible outcome.

One of the most common example is Wikipedia. People can freely contribute to different pages and then get information from it. Now since there are a lot of edits and changes to the articles, its sometimes difficult to know whether the source is trustworthy or not. Here is a hypothesis- In a Wikipedia article, the users contributing to the same article will have mutually exclusive interests and disinterests.

We can think about it as this way, if the information contributed by the first user was all the way correct, then the second user wouldnt have bothered contributing to the same topics again. It is possible that maybe the second users contribution is also not 100% trustworthy and that is why we made the above assumption.

OBJECTIVE

Our objective in this project is to check whether our hypothesis to check whether some users contributing to a croud-sourced article have mutually exclusive interests by programming it in python using dummy Wikipedia data.

THEORY

Topic modelling is a type of statistical modelling for discovering the abstract ”topics” that occur in a collection of documents. **Latent Dirichlet Allocation(LDA)** is an example of topic model and is used to classify text in a document to a particular topic. It builds a topic per document model and words per topic model, modelled as Dirichlet distributions.

In natural language processing, **latent Dirichlet allocation (LDA)** is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics. LDA is an example of a topic model.

In LDA, each document may be viewed as amixtureof various topics where each document is considered to have a set of topics that are assigned to it via LDA. This is identical toprobabilistic latent semantic analysis(pLSA), except that in LDA the topic distribution is assumed to have a sparse Dirichletprior. The sparse Dirichlet priors encode the intuition that documents cover only a small set of topics and that topics use only a small set of words frequently. In practice, this results in a better disambiguation of words and a more precise assignment of documents to topics. LDA is a generalization of thepLSAmodel, which is equivalent to LDA under a uniform Dirichlet prior distribution.

For example, an LDA model might have topics that can be classified as **CAT_related** and **DOG_related**. A topic has probabilities of generating various words, such as milk, meow, and kitten, which can be classified and interpreted by the viewer as "**CAT_related**". Naturally, the word cat itself will have high probability given this topic. The **DOG_related** topic likewise has probabilities of generating each word: puppy, bark, and bone might have high probability. Words without special relevance, such as "the" , will have roughly even probability between classes (or can be placed into a separate category). A topic is neither semantically nor epistemologically strongly defined. It is identified on the basis of automatic detection of the likelihood of term co-occurrence. A lexical word may occur in several topics with a different probability, however, with a different typical set of neighboring words in each topic.

Each document is assumed to be characterized by a particular set of topics. This is similar to the standard bag of words model assumption, and makes the individual words exchangeable.

Word Embedding is a language modeling technique used for mapping words to vectors of real numbers. It represents words or phrases in vector space with several dimensions. Word embeddings can be generated using various methods like neural networks, co-occurrence matrix, probabilistic models, etc.

Word embedding is the collective name for a set of language modelling and feature learning techniques in natural language processing(NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. Conceptually it involves a mathematical embedding from a space with one dimension per word to a continuous vector space with a much lower dimension.

Methods to generate this mapping include neural networks, dimensionality reduction on the word co-occurrence matrix, probabilistic models, explainable knowledge base method, and explicit representation in terms of the context in which words appear.

Word and phrase embeddings, when used as the underlying input representation, have been shown to boost the performance in NLP tasks such as syntactic parsing and sentiment analysis.

Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

Word2Vec consists of models for generating word embedding. These models are shallow two layer neural networks having one input layer, one hidden layer and one output layer. Word2Vec utilizes two architectures :

1. **CBOW (Continuous Bag of Words)** : CBOW model predicts the current word given context words within specific window. The input layer contains the context words and the output layer contains the current word. The hidden layer contains the number of dimensions in which we want to represent current word present at the output layer.

2. **Skip Gram** : Skip gram predicts the surrounding context words within specific window given current word. The input layer contains the current word and the output layer contains the context words. The hidden layer contains the number of dimensions in which we want to represent current word present at the input layer.

The basic idea of word embedding is words that occur in similar context tend to be closer to each other in vector space. For generating word vectors in Python, modules needed are nltk and gensim.

ALGORITHM USED

1. Extract all edits in a Wikipedia article.
   We have used wikipedia dump data which is in xml format. We have used xml.etree.ElementTree modeule to paese the xml. This modules helps us to find all the edits done by a particular user.

2. Cluster all edits of a single user in the article in one text file
   After extracting all the edits from the xml file we have clustered the data. We have stored all the edits done by a single user in a single text file. Therefore the number of text files is equal to the number of the users and each text file contains edits done by that particular user.

3. Extract topics corresponding to each text using a topic modeling model.
   In this segment we are extracting topics from the text files which we have

stored for each user. We have trained a LDA model to extract topics from the text file. After extracting the topics from the text files we store them in a list. Each user has its own list of topics.

4. Compare the list of topics. The list of topics are similar if the sum of dot products of corresponding word2vec vectors of topic lists are greater than a threshold.
   In this step we have found the similarity between the list of topics of different users. The list of topics are similar if the sum of dot products of corresponding word2vec vectors of topic lists are greater than a threshold.

5. We obtain similarity quotient as

$$\frac{\sum_{i=1,j=2}^{n-1,n}(similar\_topics/unique\_topics)}{number\_of\_users} \tag{1}$$

STEPS

The following algorithm has been used for this project:

- An xml dump data from Wikipedia is given to the main program(new_file.py) as command line arguments. The xml file is parsed and username/ip address(if anonymous) and their contributions are extracted. The names are stored in a list and the contributions for all the different users are stored as key-value pairs. Using the dictionary the edits are traversed and then stored in separate files with the same username/ip .txt naming convention stored inside the folder with the xml file name.

- The models required to model the topics are then trained using a separate python program and then the trained models are saved into a file so that time can be saved for running the main code later in the future.

- After this the topics are modelled using the model just trained by reading the edits of contributors and feeding them to model just trained. The modelled topics are then stored in a list.

- The modelled topics are then converted to vectors and then their similarity is calculated using dot product. Similar words are set for a threshold of more then 70. The similarity/uniqueness is used to calculate the User Similarity Coherence which gives the rating how much the page contents have similarity among users.

LIBRARIES USED

The following are some concepts that are used in this project.

1. XML file parsed using xml.etree.ElementTree module
2. Training the models for topic modelling is done using gensim and pandas modules
3. The conversion of words into vectors is done using numpy module
4. The nltk(natural language toolkit) and spacy modules are used to classify or tokenize language specific words.
5. The pickle module is used to save the trained module dictionary into .pkl file.

EXPLANATIONS OF MAJOR CODE SEGMENTS

There are 4 major python files required to run the whole project.

- The train_lda.py file trains the model needed to model the topics.

  ```
  lda\_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                              id2word=id2word,
                                              num\_topics=10,
                                              random\_state=100,
                                              update\_every=1,
                                              chunksize=100,
                                              passes=10,
                                              alpha='auto',
                                              per\_word\_topics=True)

  lda\_model.save('lda.model')
  ```

  This part of the code is the main part as this generates the trained model dictionary from the different attributes and then saves them in a lad.model file. This is the file that takes longest time to run. After once, it is needed to be run again as the trained models are saved.

- The testing.py file contains the function to use the trained models to generate topics.

```
def getTopicForQuery (question,lda,dic):
    temp = question.lower()
    ques_vec = []
    dictionary1, corpus, important_words = pre_processing(
                                                 temp,dic)
    ques_vec = dic.doc2bow(important_words)
    topic_vec = []
    topic_vec = lda[ques_vec]
    final = []
    topic_vec[0].sort(key =lambda x:x[1])
    final = lda.print_topic(topic_vec[0][len(
                                 topic_vec[0])-1][0])
    lda.print_topics(20)
    lda.show_topics(20)
    a=re.findall(r"[a-zA-Z']+",final)
    return a
```

This function generates the topics from the dictionary of trained models and returns a list of topics for every user from the material they contributed.

- The driver.py file contains the function to use the topics to find similarity among them.

```
def filter(list_of_topics):
    for user_topics in list_of_topics:
        for topic in user_topics:
            if topic.lower() not in words or not
                                 topic.isalpha():
                user_topics.remove(topic)
    return list_of_topics

def remove_stopwords(list_of_topics):
    for user_topics in list_of_topics:
        for topic in user_topics:
            if topic in stop_words:
                user_topics.remove(topic)
    return list_of_topics

def drive(mat):
     mat = filter(mat)
     mat = remove_stopwords(mat)
     n=len(mat)
     tot_user_similarity=0.0
```

```
for i in range(n-1):
    for j in range(i+1,n):
        tot=0
        a=len(mat[i])
        b=len(mat[j])
        mm=min(a,b)
        for k in range(mm):
            similarity=np.dot(model[mat[i][k]]
                                        ,model[mat[j][k]])
            if similarity>=70:
                tot+=1
            unique=a+b-tot
        st="similarity between topics of user "+str(i)
                                                    +" and user
        "+str(j)
        print(st)
        tmp=tot/unique
        print(tmp)
    tot_user_similarity+=tmp
print("Total similarity between all users contributing
                                                to same page
or The User Similarity Coherence is ")
print(tot_user_similarity/n)
```

The function remove_stopwords() clears the list containing the list of topics of all the stopwords as they are not counted towards word similarity counting. The function filter() removes any strange occurances of any topics such as not in alphabetical form or not in lower alphabetics. The function drive() takes the list of topics as input and then among all the topics of all users, it calculates the similarity. Upon similarity more than 70%, then it counts it for the similarity coherence.

- The new_file.py is the main file that uses the functions get the similarities.

```
def users\_list(threshlold,root):
    a = collections.defaultdict(list)
    t=int(threshlold)
    for c in root:
        for n in c.iter():
            if n.tag=='title':
                title=n.text
            if n.tag=='username':
                u=n
            if n.tag==’ip’:
                u=n
            if n.tag==’text’:
                b=int(n.get(’bytes’))
                if b>t:
                    if title not in a[u.text]:
                        a[u.text].append(n.text)
    return a

for i in range(len(username)):
    count += 1
    a = open(os.path.join(sys.argv[1],
    str(username[i]).replace(’:’,’-’).replace(’?’,’-’).
                                             replace(’|’,’-’)
    .replace(’/’,’-’).replace("\\","-")+’.txt’)
                             ,’w’,encoding = ’utf-8’)
    for j in range(0,len(text[i])):
        new_str = re.sub(’[ˆa-zA-Z0-9\n\.]’, ’ ’, text[i][j])
        text[i][j] = new_str
        lda_model=gensim.models.LdaModel.load(’lda.model’)
        id2word={}
        f = open("file.pkl","rb")
        id2word=pickle.load(f)
        pp=getTopicForQuery(new_str,lda_model,id2word)
        a.write(new_str)
        a.write(’\n-------------------|||||||||||||||||||
        |||||||||||||||||||||||||||||||||||||||||||||----
        -----------------------\n’)
    mat[i]=pp
    a.close()
```

The function users_list() takes the xml filename as the input as parses it to get the text of the username tag of all the users in the file and creates a folder where contributions of all users are stored in separate text files. The next code segment is the part where the functions are called and the list of all topics is converted into a 2D matrix which is used later for drive() function as the list of topics(the topics themselves are a list, hence the 2D matrix)

OUTPUT

```
$ python3 new_file.py Yugoslav_torpedo_boat_T5

[nltk_data] Downloading package stopwords to
[nltk_data]     /home/ankanb49/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/ankanb49/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package words to /home/ankanb49/nltk_data...
[nltk_data]   Package words is already up-to-date!
No. of contributors for this page:-12
Topics generated for all these users are:
[’use’, ’problem’, ’system’, ’also’, ’work’, ’bit’, ’time’, ’sale’,
’may’, ’make’]
[’use’, ’problem’, ’system’, ’also’, ’work’, ’bit’, ’time’, ’sale’,
’may’, ’make’]
[’use’, ’problem’, ’system’, ’also’, ’work’, ’bit’, ’time’, ’sale’,
’may’, ’make’]
[’use’, ’problem’, ’system’, ’also’, ’work’, ’bit’, ’time’, ’sale’,
’may’, ’make’]
[’use’, ’problem’, ’system’, ’also’, ’work’, ’bit’, ’time’, ’sale’,
’may’, ’make’]
[’use’, ’problem’, ’system’, ’also’, ’work’, ’bit’, ’time’, ’sale’,
’may’, ’make’]
[’use’, ’problem’, ’system’, ’also’, ’work’, ’bit’, ’time’, ’sale’,
’may’, ’make’]
[’use’, ’problem’, ’system’, ’also’, ’work’, ’bit’, ’time’, ’sale’,
’may’, ’make’]
[’use’, ’problem’, ’system’, ’also’, ’work’, ’bit’, ’time’, ’sale’,
’may’, ’make’]
[’use’, ’problem’, ’system’, ’also’, ’work’, ’bit’, ’time’, ’sale’,
’may’, ’make’]
[’use’, ’problem’, ’system’, ’also’, ’work’, ’bit’, ’time’, ’sale’,
’may’, ’make’]
[’use’, ’problem’, ’system’, ’also’, ’work’, ’bit’, ’time’, ’sale’,
’may’, ’make’]
Total similarity between all users contributing to same page or The User
Similarity Coherence is
5.5
```

INFERENCE

From the program created, it is quite evident that for small number of users, their list of topics for their contributions are mostly coming similar. So, the hypothesis cannot be directly calculated from the results coming from the program.

CONCLUSION

Hence we conclude from the results of our program that our hypothesis is wrong. The interests/topics for users come too similar for them to prove our hypothesis right.

References

[1] https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

[2] https://en.wikipedia.org/wiki/Word2vec

[3] https://en.wikipedia.org/wiki/Word_embedding

[4] https://radimrehurek.com/gensim/tutorial.html

[5] https://spacy.io/usage/examples

[6] https://docs.python.org/2/library/xml.etree.elementtree.html