



# Topic Analysis

15-05-2020

---

Mallika Bhagora - 17BCP007

Drishti Sabhaya - 17BCP019

Harshil Gandhi - 17BCP022

## Introduction

Topic analysis is a Natural Language Processing (NLP) technique that allows us to automatically extract meaning from texts by identifying recurrent themes or topics. In other words, topic analysis (often referred to as *topic detection* or *topic modeling*) is a machine learning technique that organizes and understands large collections of text data, by assigning tags or categories according to each individual text's topic or theme.

As an example, take a look at the following text below:

*"The user interface is quite straightforward and easy to use."*

A classifier can take this text as an input, analyze its content, and then automatically assign relevant tags, such as *UI* and *Easy To Use* that represent this text.


The two most common approaches for topic analysis with machine learning are **topic modeling** and **topic classification**.

**Text/Topic classification** (a.k.a. *text categorization* or *text tagging*) is the task of assigning a set of predefined categories to free-text. Text classifiers can be used to organize, structure, and categorize pretty much anything. For example, new articles can be organized by topics, support tickets can be organized by urgency, chat conversations can be organized by language, brand mentions can be organized by sentiment, and so on.

In machine learning classification, examples of text and the expected categories are used to train a classification model. This model learns from the training data to recognize patterns in order to make a classification into the categories you define. The various classification models which can be used are: **Naive Bayes**, **Support Vector Machines**, **Multinomial Logistic Regression** and also Deep learning algorithms such as **Convolutional Neural Network** and **Recurrent Neural Network**.

**Topic modeling** is an unsupervised machine learning technique. This means it can infer patterns and cluster similar expressions without needing to define topic tags or train data beforehand. This type of algorithm can be applied quickly and easily, but there's a downside – they are rather *inaccurate*.

The two of the most popular algorithms of topic modelling are : **Latent Semantic Analysis** (LSA) and **Latent Dirichlet Allocation** (LDA).



Latent Semantic Analysis is the 'traditional' method for topic modeling. It is based on a principle called the distributional hypothesis: words and expressions that occur in similar pieces of text will have similar meanings. The general idea is that for every word in each document, you can count the frequency of that word and group together the documents that have high frequencies of the same words.

Latent Dirichlet Allocation involves some advanced mathematical probability topics. Imagine a fixed set of topics. We define each topic as represented by an (unknown) set of words. These are the topics that our documents cover, but we don't know what they are yet. LDA tries to map all the (known) documents to the (unknown) topics in a way such that the words in each document are mostly captured by those topics.

So these are the two approaches for topic analysis in which here we will be using both of them in order to analyse a topic with better accuracy.

## Literature Survey

### Text classification

As it first emerged in the late 1990s as "text data mining" or just "text mining" and has a long history. Many techniques have been studied to improve the performance. The commonly used text representation is bag-of-words. Not words but phrases, word sequences or N-grams are sometimes.

Most of them focused on words or N-grams extracted from the whole document with feature selection or feature weighting scheme. Some of the previous work aimed at the integration of document contents and citation structure. Basic lexical analysis might count frequencies of words and terms in order to carry out elementary functions such as attempting to classify documents by topic.

Literature overviews of text classification usually reveal its crucial elements, techniques, and solutions, proposing the further development of this field.

Aas and Eikvil (1999) have previously enumerated and described the text classification steps, namely, pre-processing, vector space model creation, dimensionality reduction (feature selection and projection), training of a classification function, and performance measurement.

Aggarwal and Zhai (2012) described text processing similarly to Aas and Eikvil (1999), but they provided more examples. Additionally, the authors presented a more in depth discussion on each element of the text classification process.

The last two works of Guzella and Caminhas (2009) and Adeva et al. (2014) present a classification process focusing on solutions for domain-specific problems.

All the works discussed above contribute significantly to text classification.

The text classification includes six primary elements: (1) data acquisition, (2) data analysis and labelling, (3) feature construction and weighting, (4) feature selection and/or projection, (5) model training, and (6) solution evaluation.

## Topic Modelling

A **topic model** is a kind of a probabilistic generative model that has been used widely in the field of computer science with a specific focus on text mining and information retrieval in recent years.

The origin of a topic model is latent semantic indexing (LSI) (Deerwester et al. 1990); it has served as the basis for the development of a topic model. Nevertheless, LSI is not a probabilistic model; therefore, it is not an authentic topic model. Based on LSI, probabilistic latent semantic analysis (PLSA) (Hofmann 2001) was proposed by Hofmann and is a genuine topic model. Published after PLSA, latent Dirichlet allocation (LDA) proposed by Blei et al. (2003) is an even more complete probabilistic generative model and is the extension of PLSA.

Since the emergence of topic models, researchers have introduced this approach into the fields of biological and medical document mining. Because of its superiority in analysis of large-scale document collections, better results have been obtained in such fields as biological/biomedical text mining.

A popular IBM service called Watson Discovery is a powerful tool for Natural Language Understanding of the document. This service can ingest, enrich, normalize and search the unstructured data (JSON, PDF, HTML, TEXT) with speed and accuracy. This service uses smart document understanding to differentiate each part of the document and trains itself.

## Proposed System

Topic classification was implemented using 2 algorithms:

1. Multinomial Naive Bayes
2. Random Forest Classifier



Topic Modelling was implemented using 2 algorithms:

1. Latent Semantic Analysis (LSA)
2. Latent Dirichlet Analysis (LDA)

## Topic Classification

The dataset used here is the stack-overflow data. This dataset has nearly 40000 instances and 2 columns.

The first column corresponds to the posts that contain text data from users of the website, it contains various types of queries. Corresponding to each post there is a tag cell that contains the tag of the query. Posts are categorised into 20 tags or topics.

Implementation involved data cleaning, splitting data into train-test sets, creating a pipeline of CountVectorizer (Bag of words), Tf-idf Vectorizer and classification model - MNB & RFC.

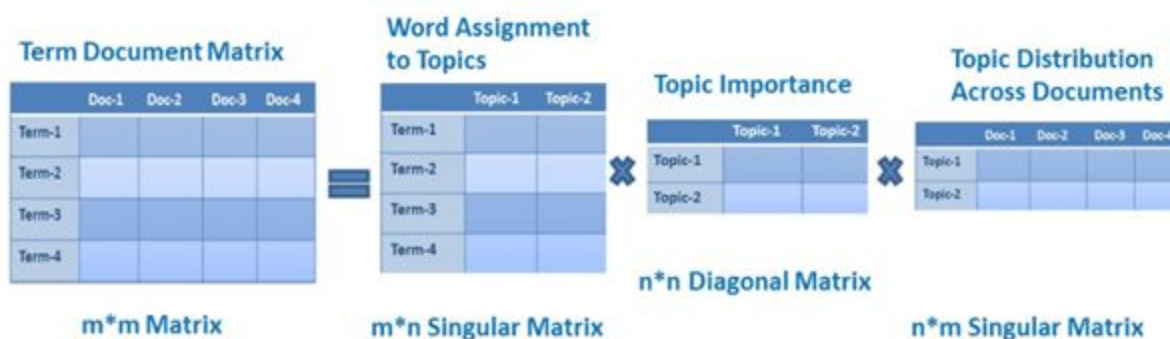
## Topic Modelling

The 20newsgroup dataset is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups.

Importing this dataset was easy using sklearn.dataset package. Although documents were partitioned over 20 different newsgroups, our topic modelling algorithms took only the documents and extracted out most frequent words for each topic over the set of documents. These extracted words for each topic can help us find the correct newsgroup based on how relevant are those words to the particular newsgroup.

### LSA

This model follows a traditional approach. This particular model was implemented using a technique called truncated SVD or truncated Singular-value-decomposition. It is a linear algebra algorithm for factoring a matrix into a product of three matrices. We will try to get the intuition for it from the figure below.



## LDA

LDA is a probabilistic model that assumes each topic is a mixture of an underlying set of words, and each document is a mixture of over a set of topic probabilities.

The generative process of LDA can be described as, given  $M$  number of documents,  $N$  number of words and prior  $K$  number of topics, the model trains to output:

$\Psi_i$ , the distribution of words for each topic  $K$ .

$\Phi_i$ , the distribution of topics for each document  $i$ .

Implementation of both algorithms include data cleaning, making a bag of words model and then applying respective models. Followed by the output topics for each of the models.

## Implementation and Results

For below implementations, to check whether the preprocessing happened correctly, we will make a word cloud using wordcloud package. This technique is called exploratory analysis.

### Topic Classification

> Importing libraries and dataset.





> Applying Multinomial Naive Bayes Algorithm to the preprocessed data.

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer

clf = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('clf', MultinomialNB()),
                 ])
clf.fit(X_train, y_train)
```

> Predicting for test set and calculating accuracy along with classification report.



```

from sklearn.metrics import classification_report
y_pred = clf.predict(X_test)

print('accuracy %s' % accuracy_score(y_pred, y_test))
print(classification_report(y_test, y_pred))

```

```

(base) drishti@drishti:~/Desktop/projects$ python nb.py
accuracy 0.7506

```

	precision	recall	f1-score	support
.net	0.62	0.65	0.64	492
android	0.93	0.85	0.89	542
angularjs	0.93	0.93	0.93	513
asp.net	0.71	0.75	0.73	492
c	0.75	0.85	0.80	513
c#	0.70	0.53	0.60	485
c++	0.79	0.75	0.77	504
css	0.70	0.90	0.79	492
html	0.57	0.67	0.62	454
ios	0.63	0.65	0.64	507
iphone	0.67	0.59	0.63	498
java	0.75	0.78	0.77	488
javascript	0.80	0.62	0.69	494
jquery	0.75	0.80	0.77	495
mysql	0.66	0.81	0.72	503
objective-c	0.72	0.65	0.68	504
php	0.83	0.76	0.79	509
python	0.87	0.87	0.87	520
ruby-on-rails	0.91	0.93	0.92	488
sql	0.77	0.67	0.72	507
accuracy			0.75	10000
macro avg	0.75	0.75	0.75	10000
weighted avg	0.75	0.75	0.75	10000

## > Applying Random Forest Algorithm

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer

clf = Pipeline([('vect', CountVectorizer()),
                ('tfidf', TfidfTransformer()),
                ('clf', RandomForestClassifier(n_estimators=100, criterion="entropy", random_state=0)),
                ])
clf.fit(X_train, y_train)

```

> Predicting Test set results and calculating the accuracy. Code is the same as above.

## Topic Modelling

> Importing libraries and dataset.

```
import numpy as np
import pandas as pd
#importing 20_newsgroup data
from sklearn.datasets import fetch_20newsgroups
dataset=fetch_20newsgroups(shuffle=True, random_state=1, remove=('headers', 'footers', 'quotes'))
print(dataset.target_names)
documents=dataset.data
news_20=pd.DataFrame({'Document':documents})
```

> Data cleaning

```
#data preprocessing
import nltk
import re
from nltk.corpus import stopwords
stopwords=stopwords.words('english')

#removing unwanted chars and words with len less than 3
news_20['clean_text']=news_20['clean_text'].str.replace('[^a-zA-Z#]', ' ')
news_20['clean_text']=news_20['clean_text'].apply(lambda x:x.lower())
news_20['clean_text']=news_20['clean_text'].apply(lambda x: " ".join([word for word in x.split() if len(word)>3]))
```

> Performing lemmatization

```
#tokenize-->removing stop words --> lemmatization -->detokenize
#initializing tagger component which is the POS tagger tht gives POS tag to each token
#it is needed for lemmatization
import spacy
nlp=spacy.load('en', disable=['parser', 'ner'])#keeping only tagger component
def lemmatization(text, tags=['NOUN', 'ADJ']):
    output=[]
    for sentence in text:
        doc=nlp(" ".join(sentence))
        output.append([token.lemma_ for token in doc if token.pos_ in tags])
    return output

tokenize_doc=news_20['clean_text'].apply(lambda x:x.split())
tokenize_doc=tokenize_doc.apply(lambda x: [word for word in x if word not in stopwords])
tokenize_doc=lemmatization(tokenize_doc)
detokenize_doc=[]
for i in range(len(news_20)):
    t=" ".join(tokenize_doc[i])
    detokenize_doc.append(t)

news_20['clean_text']=detokenize_doc
```

### > Removing non-important words

```
#removing non-meaningful words -- using words corpus from nltk
nltk.download('words')
words=set(nltk.corpus.words.words())
news_20['clean_text']=news_20['clean_text'].apply(lambda x: " ".join([word for word in x.split() if word in words])
```

### > Performing exploratory analysis

```
#exploratory analysis -- to check if preprocessing happened correctly.
from wordcloud import WordCloud
long_string=", ".join(list(news_20['clean_text'].values))
wordcloud=WordCloud(background_color="white", max_words=500, contour_color="steelblue")
wordcloud.generate(long_string)
wordcloud.to_image()
```

### > Creating bag of words model

```
#creating tf-idf vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer=TfidfVectorizer(max_features=1000, smooth_idf=True, max_df=0.5)
X=vectorizer.fit_transform(news_20['clean_text'])
```

### > Topic modelling - applying truncated SVM

```
#topic modelling -- using Truncated SVD Concept
from sklearn.decomposition import TruncatedSVD
svd_model=TruncatedSVD(n_components=20, algorithm='randomized', n_iter=100, random_state=122)
svd_model.fit(X)
len(svd_model.components_) #components are the topics of our model - term topic matrix i.e. 20x1000
```

### > Printing out most important words in each of 20 topics

```
#printing out most important words in each of the 20 topics
terms=vectorizer.get_feature_names()
for i, comp in enumerate(svd_model.components_):
    terms_comp=zip(terms, comp) #1000x1000 matrix
    sorted_terms=sorted(terms_comp, key=lambda x:x[1], reverse=True)[:7]
    print('Topic ' + str(i) + ":")
    for t in sorted_terms:
        print(t[0])
```

## &gt; Output

```

drishti@driht: ~/Desktop/projects
(base) drishti@driht:~/Desktop/projects$ python LDA.py
[alt.athens, 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']
[nltk_data] Downloading package words to /home/drishti/nltk_data...
[nltk_data] Package words is already up-to-date!
Topic 0:
['people', 'other', 'good', 'time', 'year', 'thing', 'problem']
Topic 1:
['window', 'file', 'thank', 'card', 'driver', 'program', 'system']
Topic 2:
['game', 'team', 'year', 'player', 'season', 'good', 'last']
Topic 3:
['thank', 'card', 'mail', 'drive', 'address', 'price', 'offer']
Topic 4:
['thank', 'file', 'mail', 'program', 'address', 'people', 'information']
Topic 5:
['card', 'people', 'driver', 'window', 'thank', 'game', 'video']
Topic 6:
['good', 'window', 'year', 'problem', 'bike', 'time', 'thing']
Topic 7:
['drive', 'people', 'file', 'problem', 'disk', 'game', 'hard']
Topic 8:
['file', 'good', 'card', 'driver', 'people', 'year', 'price']
Topic 9:
['year', 'last', 'problem', 'file', 'thank', 'time', 'driver']
Topic 10:
['window', 'people', 'drive', 'year', 'chip', 'government', 'disk']
Topic 11:
['problem', 'team', 'system', 'player', 'good', 'chip', 'thank']
Topic 12:
['team', 'drive', 'time', 'player', 'driver', 'question', 'number']
Topic 13:
['bike', 'chip', 'time', 'thank', 'thing', 'game', 'driver']
Topic 14:
['system', 'time', 'card', 'thank', 'year', 'thing', 'window']
Topic 15:
['time', 'people', 'bike', 'team', 'mail', 'list', 'address']
Topic 16:
['bike', 'system', 'other', 'list', 'team', 'right', 'driver']
Topic 17:
['driver', 'time', 'system', 'mail', 'address', 'good', 'list']
Topic 18:
['program', 'thing', 'driver', 'version', 'system', 'people', 'available']
Topic 19:
['thing', 'address', 'mail', 'chip', 'window', 'name', 'same']
(base) drishti@driht:~/Desktop/projects$

```

> Same preprocessing applies for the LDA modelling. All steps upto bag of words model remains the same. Applying Latent Dirichlet Analysis model.

```

#topic modelling -- using LDA - Latent Dirichlet Analysis
from sklearn.decomposition import LatentDirichletAllocation as LDA
lda_model=LDA(n_components=20, random_state=122)
lda_model.fit(X)
len(lda_model.components_)    #components are the topics of our model - term topic matrix i.e. 20x1000

```

## &gt; Output

```

drishti@driht: ~/Desktop/projects
(base) drishti@driht:~/Desktop/projects$ python LDA.py
[alt.athens, 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']
[nltk_data] Downloading package words to /home/drishti/nltk_data...
[nltk_data] Package words is already up-to-date!
Topic 0:
['drive', 'disk', 'hard', 'system', 'boot', 'controller', 'morality']
Topic 1:
['chip', 'government', 'clipper', 'encryption', 'key', 'phone', 'system']
Topic 2:
['driver', 'card', 'color', 'image', 'node', 'graphic', 'video']
Topic 3:
['road', 'part', 'traffic', 'detector', 'radar', 'finger', 'wheel']
Topic 4:
['number', 'phone', 'mouse', 'month', 'true', 'magazine', 'company']
Topic 5:
['source', 'code', 'available', 'library', 'vote', 'luck', 'public']
Topic 6:
['people', 'child', 'woman', 'country', 'attack', 'government', 'fire']
Topic 7:
['doctor', 'patient', 'science', 'medical', 'disease', 'bank', 'pain']
Topic 8:
['game', 'team', 'year', 'player', 'season', 'last', 'good']
Topic 9:
['article', 'ticket', 'water', 'dollar', 'option', 'thing', 'reader']
Topic 10:
['printer', 'print', 'voltage', 'laser', 'marriage', 'period', 'circuit']
Topic 11:
['people', 'other', 'thing', 'life', 'religion', 'time', 'such']
Topic 12:
['bike', 'engine', 'mile', 'motorcycle', 'ride', 'front', 'insurance']
Topic 13:
['thank', 'mail', 'address', 'list', 'name', 'group', 'information']
Topic 14:
['space', 'food', 'launch', 'helmet', 'shuttle', 'energy', 'orbit']
Topic 15:
['car', 'auto', 'dealer', 'battery', 'stuff', 'test', 'face']
Topic 16:
['long', 'time', 'student', 'hole', 'keyboard', 'seat', 'summer']
Topic 17:
['card', 'price', 'sale', 'monitor', 'offer', 'port', 'good']
Topic 18:
['version', 'word', 'language', 'love', 'service', 'figure', 'clock']
Topic 19:
['window', 'file', 'program', 'problem', 'server', 'application', 'thank']
(base) drishti@driht:~/Desktop/projects$

```



## Conclusion

### Topic Classification

Multinomial naive bayes algorithm gives the accuracy of 75%.

Random Forest Classifier with 100 estimators gives the accuracy of 77%.

Further the accuracy can be informed by changing the working parameters.

That can be a future scope of the project.

### Topic Modelling

As can be seen from the output, the both the model provides a list of most frequent words for each of the 20 topics.

#### LSA

For eg.

Topic 1: ['window', 'file', 'thank', 'card', 'driver', 'program', 'system'] can be given the tag - comp.os.ms-windows.misc.

Topic 2: ['game', 'team', 'player', 'year', 'season', 'good', 'last'] can be given the tag - rec.sport.baseball or rec.sport.hockey. Either of the tags tells us that the topic talks about sport.

#### LDA


For eg.

Topic 7: ['doctor', 'patient', 'science', 'medical', 'disease', 'bank', 'pain'] can be given the tag - sci.med. This topic talks about medicine or biology science.

Further analysing each topic, we can provide appropriate tags for each of them.

LDA gives a better output than LSA, as the topics in LDA more firmly determine the tags we have.

Topic analysis makes it possible to detect different topics within a large set of text data in a fast and simple way. By using this incredibly powerful tool, you can automate many processes in different areas of your business, from customer service to social media monitoring. From sales and marketing to customer support and product teams, topic



analysis offers endless possibilities across different industries and areas within a company. The various applications of topic analysis have been used in Social Media Monitoring, Customer Service, Product Analytics, Business Intelligence etc.

## References

<https://monkeylearn.com/text-classification/>

<http://lsa.colorado.edu/papers/dp1.LSAintro.pdf>

<http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf>

[https://github.com/gokriznastic/20-newsgroups\\_text-classification/blob/master/Multinomial%20Naive%20Bayes-%20BOW%20with%20TF.ipynb](https://github.com/gokriznastic/20-newsgroups_text-classification/blob/master/Multinomial%20Naive%20Bayes-%20BOW%20with%20TF.ipynb)