QF634 APPLIED QUANTITATIVE RESEARCH METHODS
LECTURE 9

Lecturer: Prof Emeritus Lim Kian Guan

Introduction to Natural Language Processing Applications

What is Natural Language Processing?

- NLP enables computers to process human language and understand meaning and context, along with the associated sentiment and intent behind it, and eventually, use these insights to create something new such as chatbox.

Process toward Getting Understanding out of NLP

First, make words interpretable for computers by converting the raw text first to computable numbers. This process is called vectorization. In Machine Learning, vectorization is a first step, aka feature extraction -- obtaining some distinct features out of the text for the model to train on after converting text to numerical vectors.

- **Word embedding** is a technique used in Natural Language Processing (NLP) to represent words as numeric vectors in an intelligent way, possibly accounting for semantics (not just syntax or grammar).

Vectorization techniques

- **Bag of Words.** (1) Tokenization. All words in all documents (corpus) are listed individually. Document could include a sentence or phrase or clause. (2) Create Vocabulary (listing) for corpus by listing all unique words (and optionally sorting into alphabetical order). Stop words may not be included. (3) A sparse matrix is created for the input corpus. In this sparse matrix, each row is a sentence vector whose length (the columns of the matrix) is equal to the size of the vocabulary, and words that occur in this sentence take the value of its frequency (1 or more) if it occurs in the position in the dictionary. Other elements of row are zeros.

Bag-of-Words

- Bag-of-words model (BoW) is a method to extract features from text for use in language modeling.
- BoW considers each word in a corpus (set of documents) as a feature. A document may be defined as a report, a chapter, a book, a paragraph, or even as small as a sentence.
- BoW is only concerned about whether the known word occurs in the document, and not its locational position with the document.

Take a simple example. Corpus contains 2 documents. Each is a sentence. BoW tokenize the sentence into individual words as tokens.

[1] Tim went fishing on a boat and Tom went too.
[2] Jerry went to play golf today.

Remove the punctuations. We may also remove the stop word “a”. Create a Vocabulary list of all the unique words in the corpus and give each word a value of one in a unique dimension**. If there are N unique words, then there are N dimensions in the Vocabulary list**. In this corpus, the dimension k is associated with the k^{th} unique word. Order of the dimensioning is not important.
[“Tim”, “went”, “fishing”, “on”, “boat”, “and”, “Tom”, “too”, “Jerry”, “to”, “play”, “golf”, “today”] – 13 dimensions.

The two documents (sentences) can be vectorized (represented as numerical vector) as
[1] (1, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0)
[2] (0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1)
Note: “went” occurs twice in [1] and once in [2]. More informative than hot-encoding [1] (1,1,1,1,1,1,1,1,0,0,0,0,0)

Frequency-based embeddings – another representation of a word

In BoW, each word is vectorized in a huge dimension. Another method of representing a word in more context is the word's TF-IDF score.

The TF-IDF score **for a term (word) in a document** is calculated using the following formula:

$$\textbf{TF-IDF}\ (\boldsymbol{t,d,D}) = \boldsymbol{TF}(\mathbf{t,d})\ \boldsymbol{x}\ \boldsymbol{IDF}(\mathbf{t, D})$$

Term frequency is the number of instances of a word/term t in a single document d only, i.e., TF(t,d). For each specific term in the paper, there is an entry with the value being the term frequency.

Frequency of the document is the number of separate documents in which the word/term t appears, i.e., DF(t,D), or the occurrence of word/term in documents. Documents are in the entire corpus D.

If Number of documents in corpus is N, then Inverse Document Frequency, IDF, of the word is the relative appearance of the word/term in the entire corpus D.

$$\mathrm{IDF}(t,D) = N/DF \geq 1$$

IDF is higher if DF is lower relative to N, i.e., term does not appear in many documents of D. Higher IDF indicates rarity. Log_{10} form is sometimes used.

Tf-idf is one of the best metrics to determine how significant a term is to a text in a series or a corpus if it is not a common word across corpus (hence high IDF) and if it occurs heavily in one text d. **It gives a numerical value (embedding) to the significance of the word in the context of the corpus D.** Example: a document representation by words could be (1.5, 0.8, 0.3, 0.6, 1.1, 2.2, 0.1, 0.05, 0, 0, 0, 0, 0). "0" means the word does not occur in that document.

Bigram Models of Vocabulary List

- A related method of extracting features is to create a vocabulary list of grouped words. Grouped words may carry more meaning (semantic content) than BoW. Creating vocabulary of two-word pairs is called a Bigram or 2-gram. Bag of bigrams can be more informative than Bag of (single) words.

[1] Tim went fishing on a boat and Tom went too.

Constructing the bigrams:

“Tim went”, “went fishing”, “fishing on”, “on boat”, “boat and”, “and Tom”, “Tom went”, “went too”

A tri-gram “fishing on boat” would tell more and differentiate from fishing at shore, fishing at pond, etc.

- The dictionary can be expanded to include bigrams and n-grams in general. Some n-grams, e.g., “punishing interest cost”, “escalating rent rates” could be more informative than use of single words in weighting on state of the economy or sentiment of the economy.
- Collocation or commonly co-occurred words also impart better semantic structures. Bigrams, e.g., “customer service”, and Trigrams, e.g., “on the ball”, provide more accurate meanings. The understanding based on a larger dictionary can help understand the semantic structures (meaning) and improve the insights when these are nested in longer sentences.

One important Application: Topics Modelling (Unsupervised)

	Topic 1	Topic 2
Document 1	0.80	0.20
Document 2	0.30	0.70

What this means more exactly is that in document 1, 80% of the number of words belong to Topic 1.

Each unique word in all the documents (corpus) must belong to or be allocated to just one topic. (We avoid more complicated Topic Modeling allowing same word to occur in different topics.)

Topic 1		Topic 2	
Climate	0.18	Defence	0.10
Weather	0.12	Threat	0.08
Heatwave	0.08	Military	0.12
Typhoon	0.09	Evacuate	0.05
Carbon	0.15	Food	0.07
....	P_i	....	Q_i
$\Sigma P_i = 1$		$\Sigma Q_i = 1$	

Here, we have **two documents** and we want to analyze their contents. We suppose the contents fall into **two topics**. **We want to identify the Topics**. As it turns out, the topics that may likely be described as climate risk (Topic 1) and war risk (Topic 2).

Top table's first row shows Document 1 has 80% probability of carrying words related to Topic 1 and 20% probability of carrying words related to Topic 2. Or Document 1 has a distribution of Topic 1 with probability 0.8 and Topic 2 with probability 0.2.

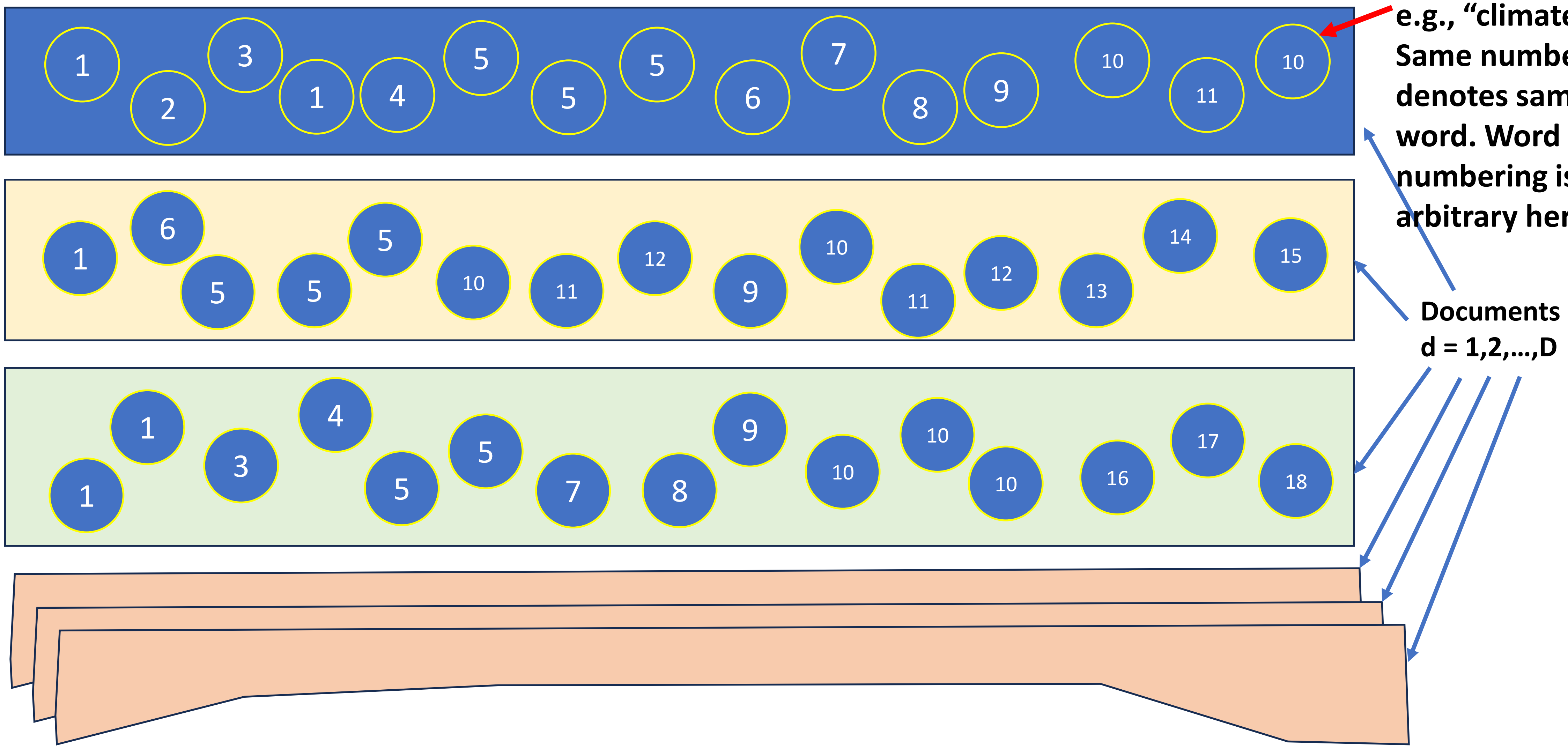
Document 2 has a distribution of Topic 1 with probability 0.3 and Topic 2 with probability 0.7.

Note: It can sometimes be confusing to say "Topics belong to Document", "Document belongs to Topics". Easier to think of a pre-determined 2 topics, and each document has a probability distribution of these 2 topics, T_1 , T_2. We do not know what are these topics or their constituents (affiliated words) but we denote they are different, $T_1 \neq T_2$.

Bottom table shows each topic has a probability distribution of unique words w_k where we can think of w_k as identified by a vector (0, 0, 0, ... ,1, 0,....,0) with 1 occurring at the kth dimension. The total dimension of the word vector is the size of the dictionary or number of unique words in all the documents forming the corpus.

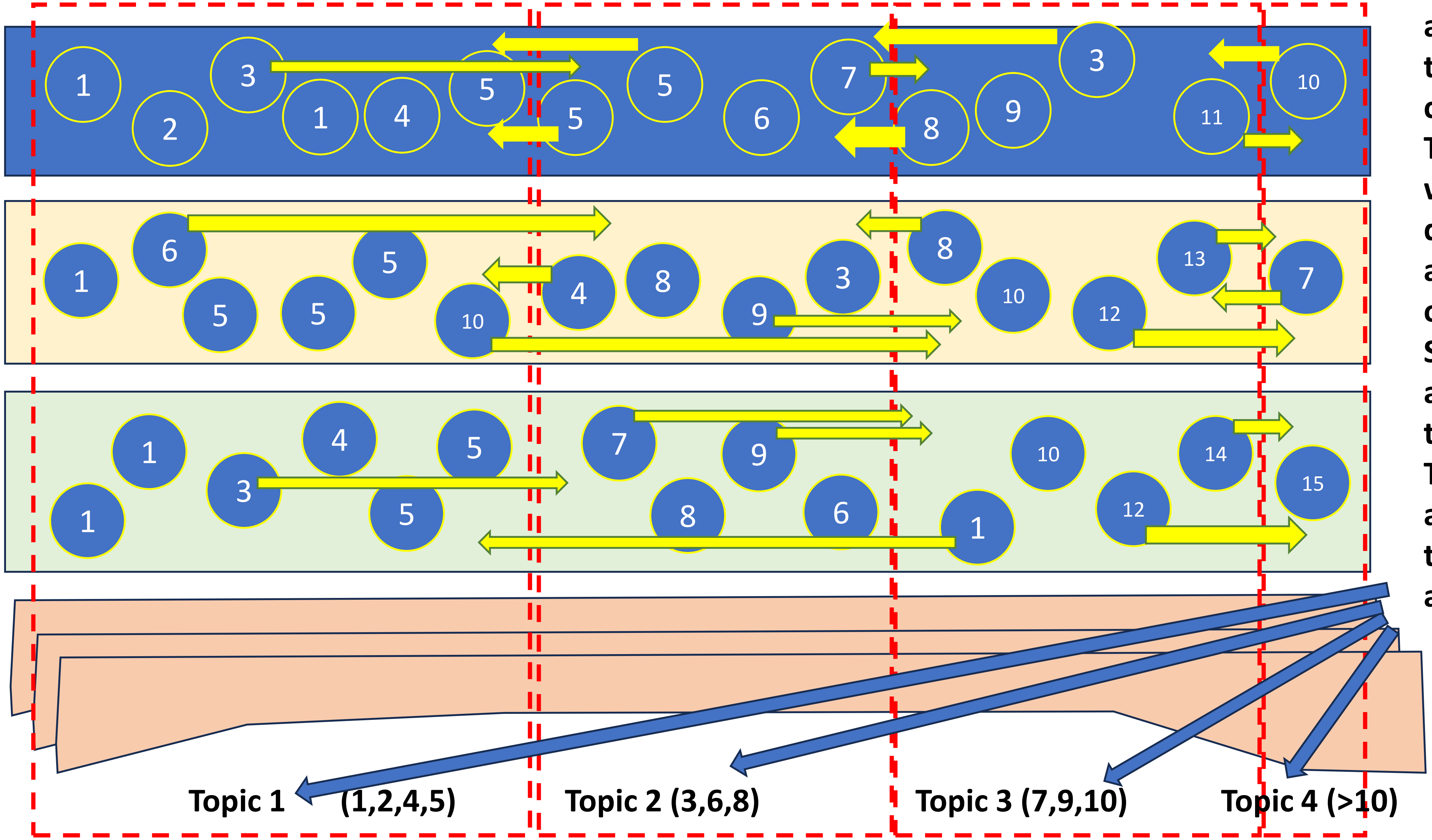
Documents are probability distributions (mixtures) of topics and Topics are probability distributions (mixtures) of words.

Topics Modelling using Latent Dirichlet Allocation



Each is a word, e.g., “climate”. Same number denotes same word. Word numbering is arbitrary here.

Topics Modelling using Latent Dirichlet Allocation

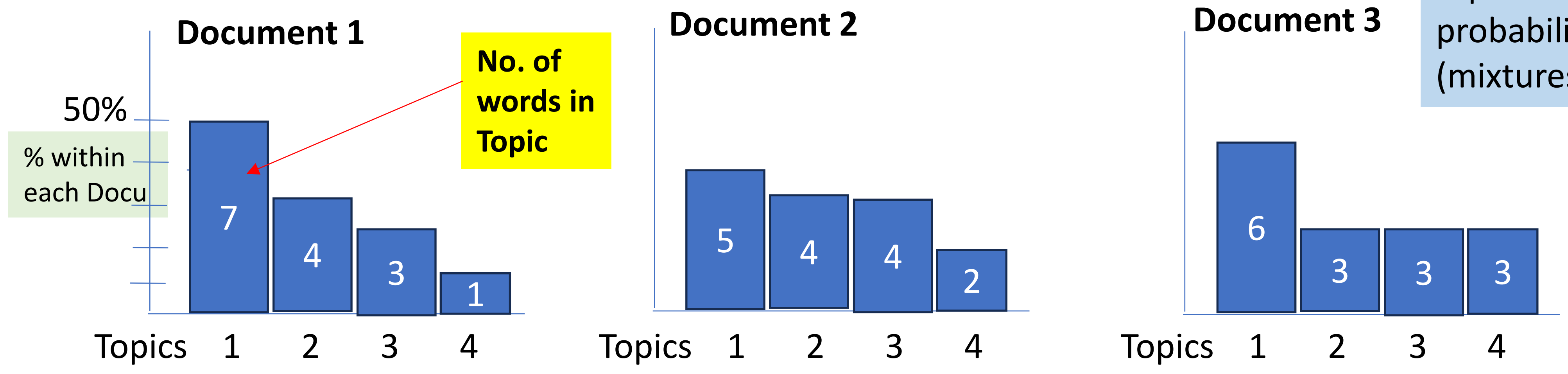


Suppose we apriori believe the Documents contain 4 Topics. Each word in every document is allocated one of the 4 topics. Same words are allocated the same topic. The yellow arrows show the possible allocations.

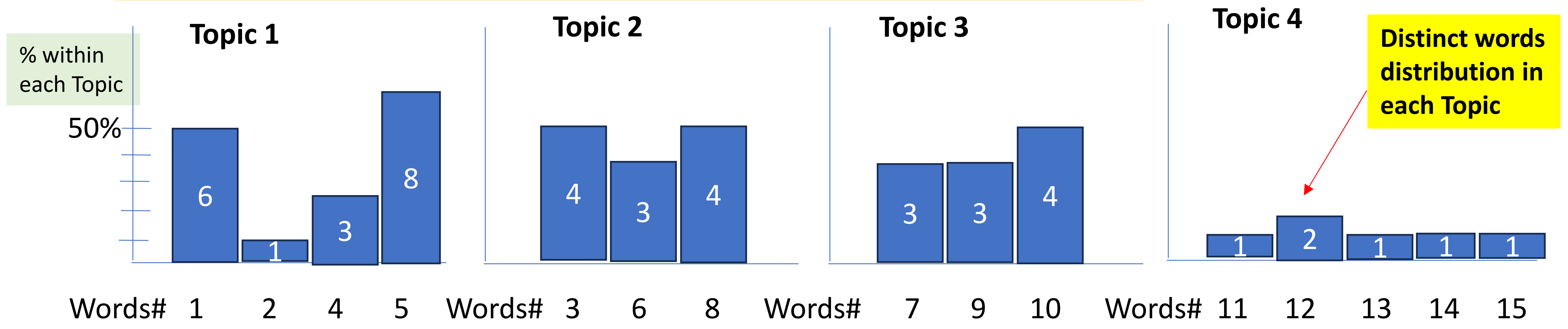
Topics Modelling using Latent Dirichlet Allocation

Documents are probability distributions (mixtures) of topics and Topics are probability distributions (mixtures) of words.

We obtain the Document-Topic Distributions

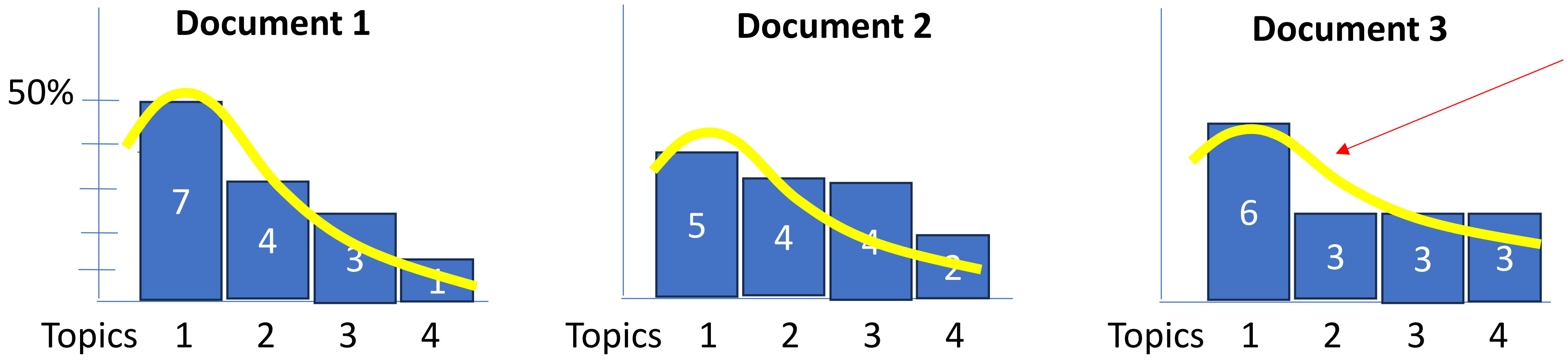


And the Topic-Word Distributions assuming we use only the first 3 documents.



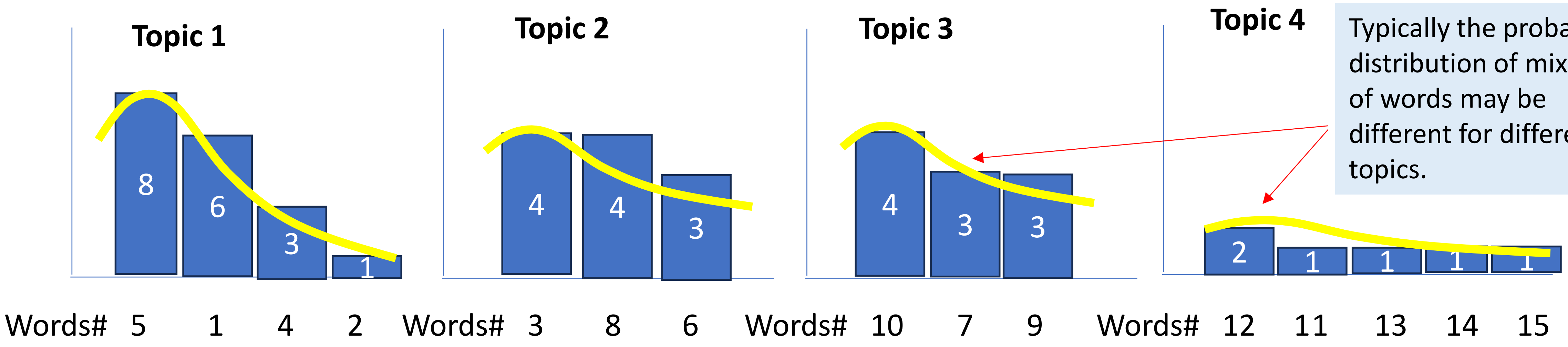
Topics Modelling using Latent Dirichlet Allocation

Idea is to **optimally allocate** words to each topic so that distribution of topics to document and the distributions of words to topics follow a suitable distributions



Typically the probability distribution of mixtures of topics is the same for every document (selected from the same type of publication)

and the Topic-Word distribution for each topic also follows suitable probability distribution



Typically the probability distribution of mixtures of words may be different for different topics.

Topics Modelling using Latent Dirichlet Allocation

- Topic modeling is a technique in NLP **to discover latent (hidden) themes/topics** within a collection (corpus) of textual documents. It is a type of **unsupervised textual analysis**. The textual documents could be newspapers (various publishers). Or it could be number of analysts' reports on stock recommendations. Or, it could be the number of transcribed audio and video recordings of interviews. These are all within a stated period of analysis.
- Assume a given **corpus has D number of documents** (each document could be a sentence or a paragraph or a report/essay or a book or a transcribed interview) denoted as document d_1, d_2, d_3, ….. , d_D .
- Assume **each document contains K number of unique topics** denoted as T_1, T_2, T_3, …… , T_K . For a more tractable solution, we assume each document within the given corpus has the same set of K topics (instead of possibly different sets of topics). **Apriori, we do not know what are the topics – they are hidden or latent**. We only know $T_i \neq T_j$ for $i \neq j$.
- An example is the following. After a corpus of daily news print over a stated time period is "solved" to estimate the "topics", we may be able to interpret the estimated topics as politics, sports, entertainments, economics, personalities, etc.
- **Each topic is characterized by the topic's probability distribution over N unique words**. The N unique words are drawn from all the documents in the corpus. The (unique) words are the basic units of analyses. The words may be denoted by w_1, w_2, w_3, ……, w_N. Each has a unique id such as a vector with unit in a particular dimension and zeros elsewhere (sparse) in a Bag-of-Words representation.
- In any particular document or particular topic, some words may have zero probability of occurrence.

Topics Modelling using Latent Dirichlet Allocation

We now discuss what are suitable Document-Topic and Topic-Word probability distributions.

Dirichlet distribution

$$f(x_1,\ldots,x_K;\alpha_1,\ldots,\alpha_K) = \frac{1}{\mathrm{B}(\boldsymbol{\alpha})}\prod_{i=1}^{K} x_i^{\alpha_i-1} \qquad \sum_{i=1}^{K} x_i = 1 \text{ and } x_i \in [0,1] \qquad \alpha_1, \ldots, \alpha_K > 0$$

$$\mathrm{B}(z_1,z_2) = \frac{\Gamma(z_1)\Gamma(z_2)}{\Gamma(z_1+z_2)}$$ is Beta function, and $\Gamma(z) \equiv \int_0^\infty t^{z-1} e^{-t}\, dt$ Is Gamma function

The Dirichlet distribution is often used as a prior distribution for the multinomial parameter vector (probabilities) $p = (p_1, p_2, \ldots, p_K)$ in Bayesian inference. Thus Dirch is a probability distribution of probabilities.

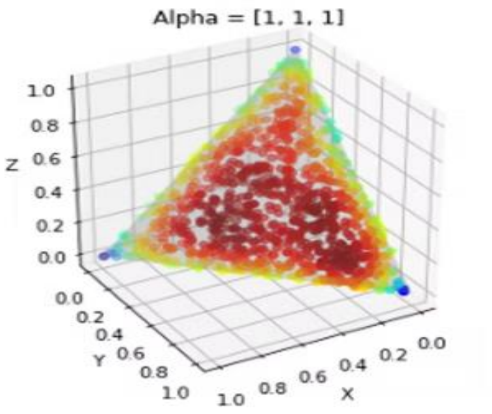
Suppose $X = (x_1, x_2, \ldots, x_K \mid p) \sim \text{Multin}(N; p_1, p_2, \ldots, p_K)$ where x_i is number of occurrences of words in Topic T_i in a document, and $p \sim \text{Dirch}(\alpha)$, then the posterior distribution of p is

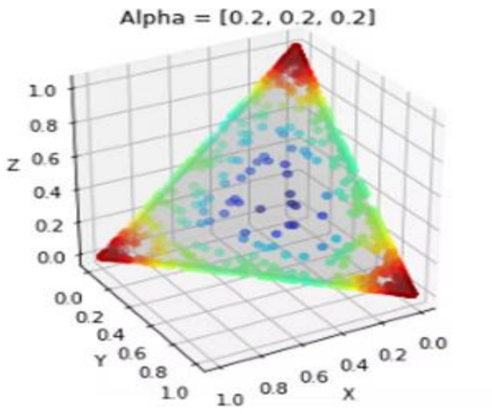
$$\begin{aligned}\pi(p|X) &\propto \frac{n!}{x_1!x_2!\cdots x_k!} p_1^{x_1}\cdots p_k^{x_k} \times \frac{1}{B(\alpha)} p_1^{\alpha_1-1}\cdots p_k^{\alpha_k-1} \\ &\propto p_1^{x_1+\alpha_1-1}\cdots p_k^{x_k+\alpha_k-1} \\ &\sim \text{Dirch}(x_1+\alpha_1,\cdots,x_k+\alpha_k).\end{aligned}$$

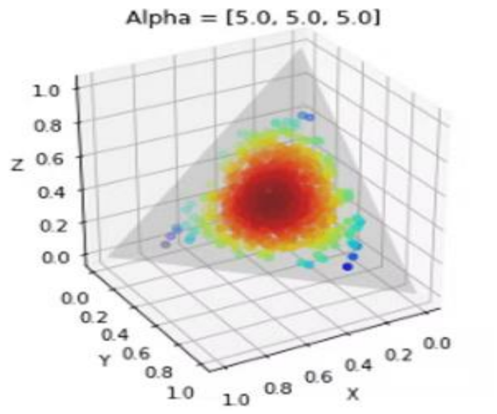
Dirch and Multin are conjugate distributions whereby posterior of p given Multin is also a Dirch.

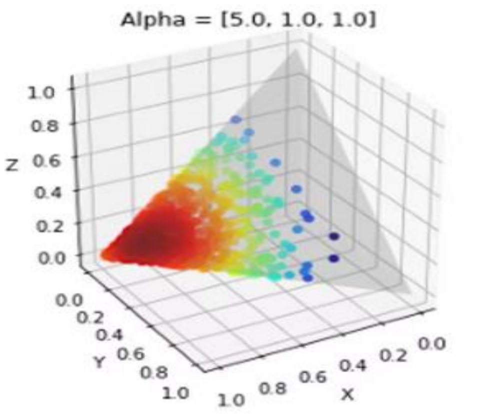
Topics Modelling using Latent Dirichlet Allocation

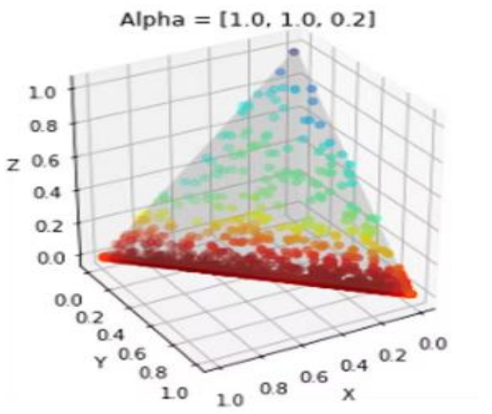
Following shows 3-dimensional Dirichlet distribution (x,y,z) with parameter α = (α_1 α_2 α_3). Smaller alphas mean the densities are concentrated around either x, y, or z (topics T_1, T_2, T_3). Larger alphas mean more spread out densities in x, y, z. Higher alpha in one dimension means document largely carries that particular topic. This can be extended to K dimension.

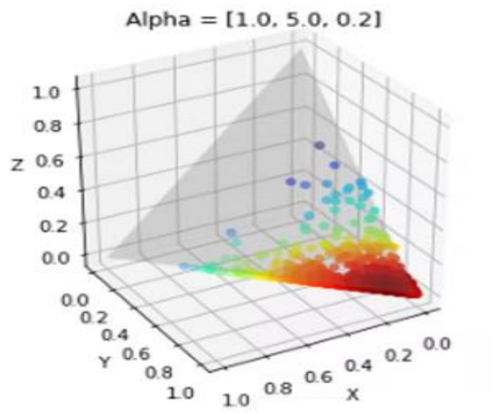








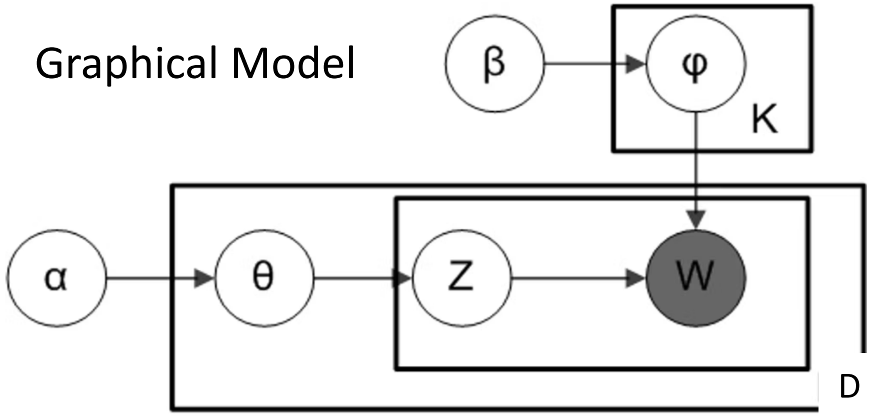




Topics Modelling using Latent Dirichlet Allocation

The way to understand LDA is to think of the **generative process for the units of words**. Then later we work backwards to try to **infer the latent topics from the distribution of words**. We can view this generation in a graphical probability model.

Graphical Model



For $k = 1...K$:

(a) $\phi^{(k)} \sim Dirichlet(\beta)$

For each document $d \in \mathbf{D}$:

(a) $\theta_d \sim Dirichlet(\alpha)$

(b) For each word $w_i \in d$:

i. $z_i \sim Discrete(\theta_d)$

ii. $w_i \sim Disctete(\phi^{(z_i)})$

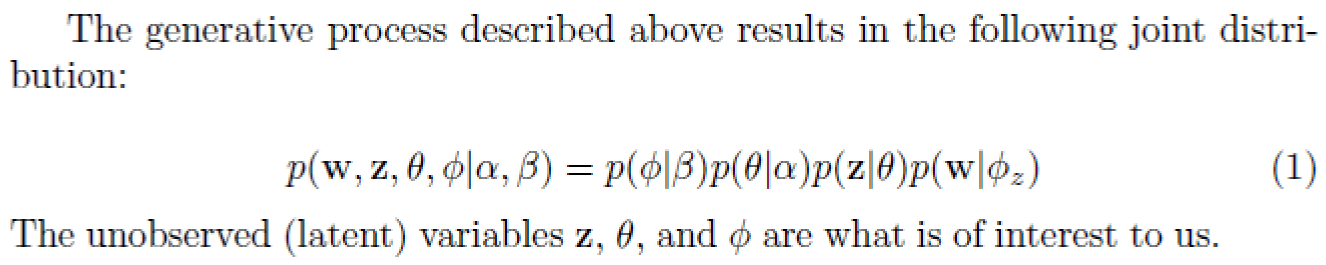
There are D documents, each of which provides a Dirch distribution (with hyperparameter α) of probabilities θ_d of topics in d.

Another Dirch distribution (with hyperparameter β) provides a Dirch distribution of probabilities $\phi^{(k)}$ over words in Topic k. There are K number of such distributions, for k = 1,2,…,K.

The circle colored in grey, *w*, is the observed word, while the other circles represent different latent variables. We let z_i be a random draw from the topic set T. z_i is associated with w_i. z_i is Multin given θ_d . w_i is Multin given $\phi^{(Zi)}$.

Topics Modelling using Latent Dirichlet Allocation

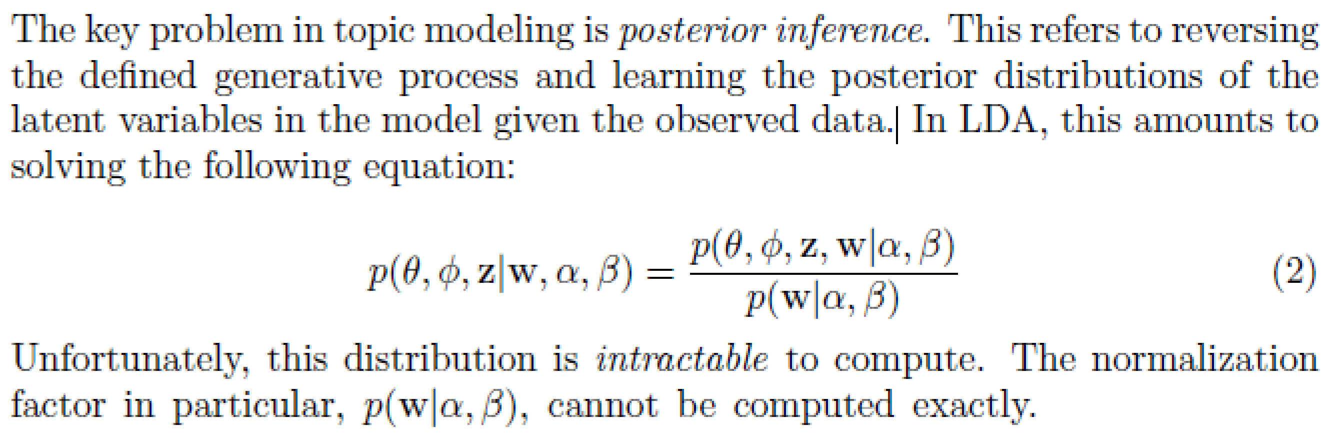
Ref: “A Theoretical and Practical Implementation Tutorial on Topic Modeling and Gibbs Sampling” by W.M. Darling, Guelph, 2011.



The generative process described above results in the following joint distribution:

$$p(\mathbf{w}, \mathbf{z}, \theta, \phi | \alpha, \beta) = p(\phi|\beta) p(\theta|\alpha) p(\mathbf{z}|\theta) p(\mathbf{w}|\phi_z) \tag{1}$$

The unobserved (latent) variables $\mathbf{z}$, θ, and ϕ are what is of interest to us.



The key problem in topic modeling is *posterior inference*. This refers to reversing the defined generative process and learning the posterior distributions of the latent variables in the model given the observed data. In LDA, this amounts to solving the following equation:

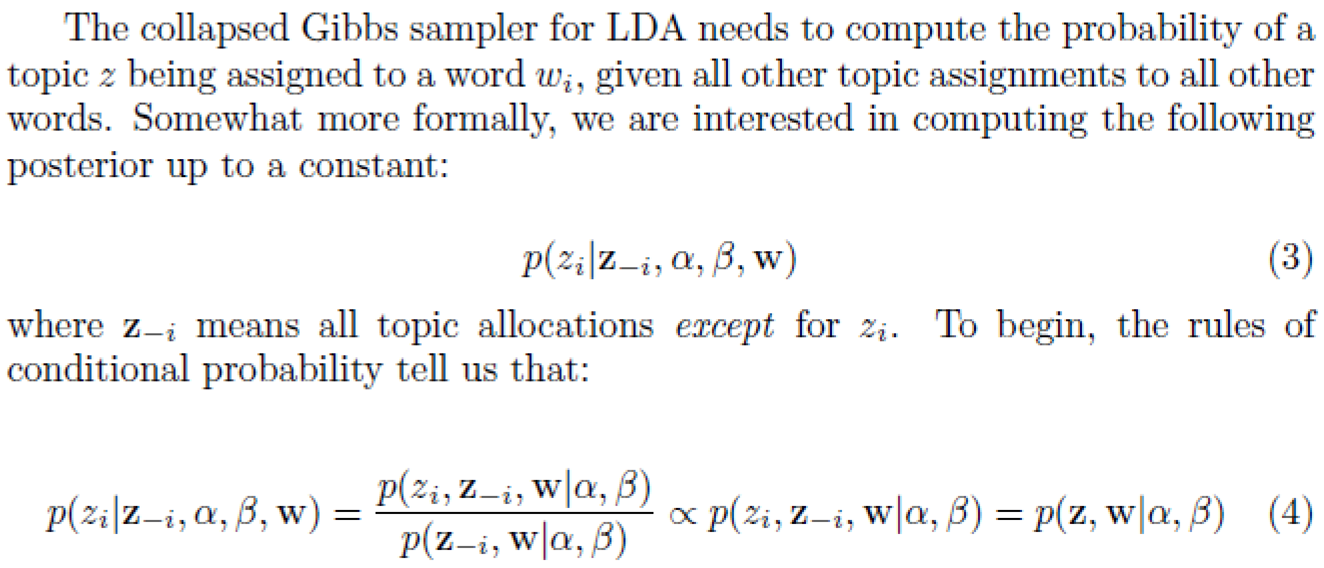
$$p(\theta, \phi, \mathbf{z} | \mathbf{w}, \alpha, \beta) = \frac{p(\theta, \phi, \mathbf{z}, \mathbf{w} | \alpha, \beta)}{p(\mathbf{w}|\alpha, \beta)} \tag{2}$$

Unfortunately, this distribution is *intractable* to compute. The normalization factor in particular, $p(\mathbf{w}|\alpha, \beta)$, cannot be computed exactly.

Topics Modelling using Latent Dirichlet Allocation

Ref: “A Theoretical and Practical Implementation Tutorial on Topic Modeling and Gibbs Sampling” by W.M. Darling, Guelph, 2011.

But Gibbs sampling method can be applied.



The collapsed Gibbs sampler for LDA needs to compute the probability of a topic z being assigned to a word w_i, given all other topic assignments to all other words. Somewhat more formally, we are interested in computing the following posterior up to a constant:

$$p(z_i|\mathbf{z}_{-i}, \alpha, \beta, \mathbf{w}) \tag{3}$$

where $\mathbf{z}_{-i}$ means all topic allocations *except* for z_i. To begin, the rules of conditional probability tell us that:

$$p(z_i|\mathbf{z}_{-i}, \alpha, \beta, \mathbf{w}) = \frac{p(z_i, \mathbf{z}_{-i}, \mathbf{w}|\alpha, \beta)}{p(\mathbf{z}_{-i}, \mathbf{w}|\alpha, \beta)} \propto p(z_i, \mathbf{z}_{-i}, \mathbf{w}|\alpha, \beta) = p(\mathbf{z}, \mathbf{w}|\alpha, \beta) \tag{4}$$

This is shifting one word w_i that belongs to topic z_j to another topic z_i, and checking the new probability of $p(z_i|z_{-I},\alpha,\beta,w)$ that should be improved until it becomes stable for passing through every document and every word in the document.

Applying LDA to a set of Documents to get the Latent Topics

L5-LDA.ipynb

Source: By Susan Li, Towards Data Science, May 31, 2018

- Data set "'abcnews-date-text.csv'" contains 1,244,184 news headlines published over a period of 18 years (2003 Feb -2021 Dec), from Kaggle. **Each headline is a document**.

	publish_date	headline_text
0	20030219	aba decides against community broadcasting lic...
1	20030219	act fire witnesses must be aware of defamation
2	20030219	a g calls for infrastructure protection summit
3	20030219	air nz staff in aust strike for pay rise
4	20030219	air nz strike to affect australian travellers

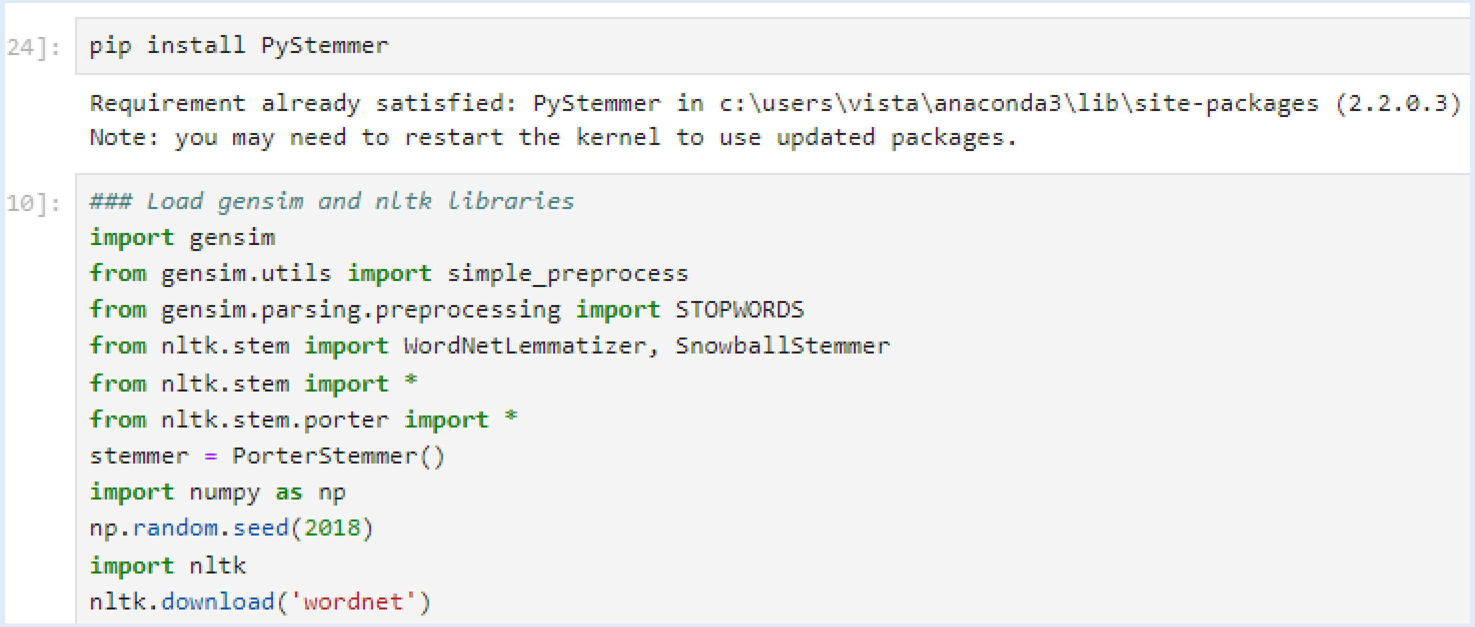
……………………………………………………………………………………

- Pre-process the raw text data:

Tokenization: Split the text into sentences and the sentences into words.
Lowercase the words and remove punctuation. Words that have fewer than 3 characters are removed.
All stop words are removed, e.g., "a", "an", "the", "is", "are", "was", "were", "am", "been", etc. (They typically do not lend much context or semantic meanings.) Includes prepositions.
Words are lemmatized — words in third person are changed to first person and verbs in past and future tenses are changed into present.
Words are stemmed — words are reduced to their root form, e.g., "to want", "wanting", "wants", "wanted" -> "want"

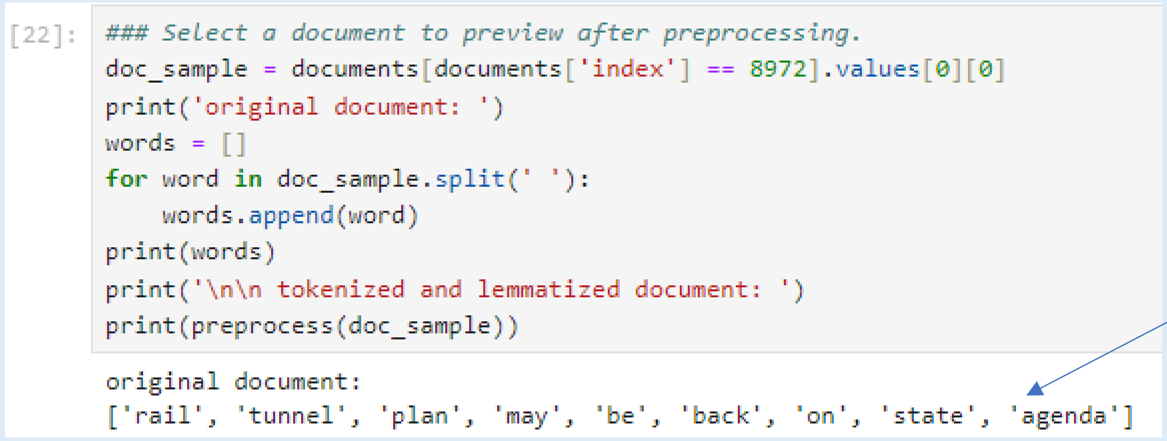
Applying LDA to a set of Documents to get the Latent Topics

Import a whole lot of text analysis apps



```
24]: pip install PyStemmer

     Requirement already satisfied: PyStemmer in c:\users\vista\anaconda3\lib\site-packages (2.2.0.3)
     Note: you may need to restart the kernel to use updated packages.

10]: ### Load gensim and nltk libraries
     import gensim
     from gensim.utils import simple_preprocess
     from gensim.parsing.preprocessing import STOPWORDS
     from nltk.stem import WordNetLemmatizer, SnowballStemmer
     from nltk.stem import *
     from nltk.stem.porter import *
     stemmer = PorterStemmer()
     import numpy as np
     np.random.seed(2018)
     import nltk
     nltk.download('wordnet')
```

WordNet is a large **lexical database corpus** in NLTK (natural language toolkit). The **Natural Language Toolkit**, or more commonly **NLTK**, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. It supports classification, tokenization, stemming, tagging, parsing, and semantic reasoning functionalities.

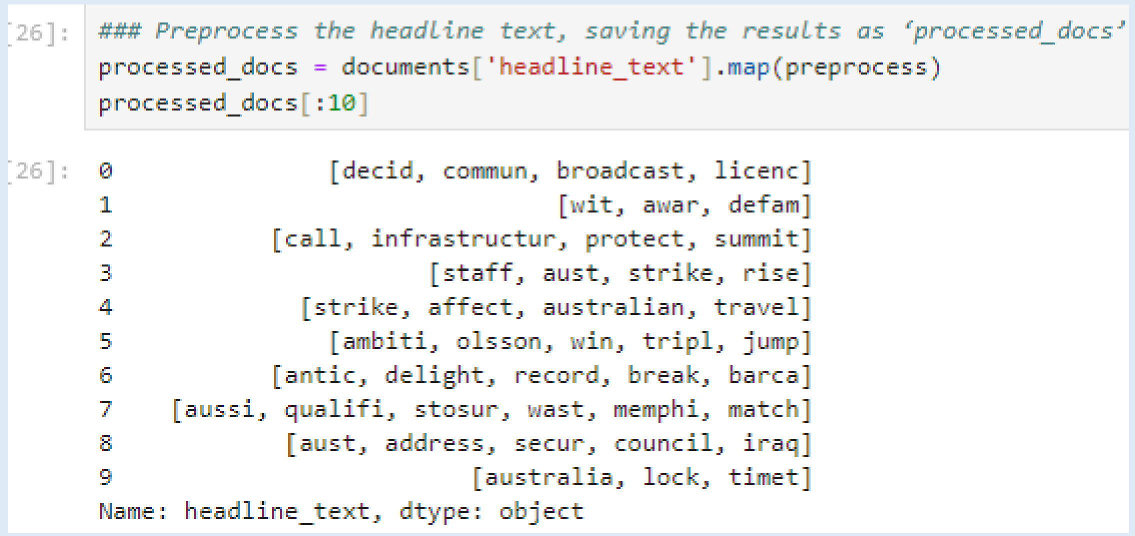


```
[22]: ### Select a document to preview after preprocessing.
      doc_sample = documents[documents['index'] == 8972].values[0][0]
      print('original document: ')
      words = []
      for word in doc_sample.split(' '):
          words.append(word)
      print(words)
      print('\n\n tokenized and lemmatized document: ')
      print(preprocess(doc_sample))

      original document:
      ['rail', 'tunnel', 'plan', 'may', 'be', 'back', 'on', 'state', 'agenda']
```

An example of headline, “rail tunnel plan may be back on state agenda”, after pre-processing

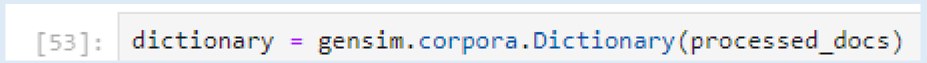
Applying LDA to a set of Documents to get the Latent Topics

Preprocess the entire corpus onto ‘processed_docs’

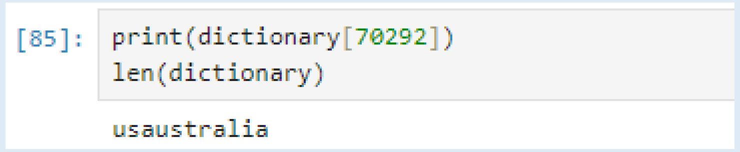


```
[26]:  ### Preprocess the headline text, saving the results as ‘processed_docs’
       processed_docs = documents['headline_text'].map(preprocess)
       processed_docs[:10]

[26]:  0                [decid, commun, broadcast, licenc]
       1                                [wit, awar, defam]
       2            [call, infrastructur, protect, summit]
       3                       [staff, aust, strike, rise]
       4             [strike, affect, australian, travel]
       5               [ambiti, olsson, win, tripl, jump]
       6            [antic, delight, record, break, barca]
       7    [aussi, qualifi, stosur, wast, memphi, match]
       8            [aust, address, secur, council, iraq]
       9                         [australia, lock, timet]
       Name: headline_text, dtype: object
```

Create a dictionary from ‘processed_docs’ containing the number of times a word appears in the training set.



```
[53]:  dictionary = gensim.corpora.Dictionary(processed_docs)
```



```
[85]:  print(dictionary[70292])
       len(dictionary)

       usaustralia
```

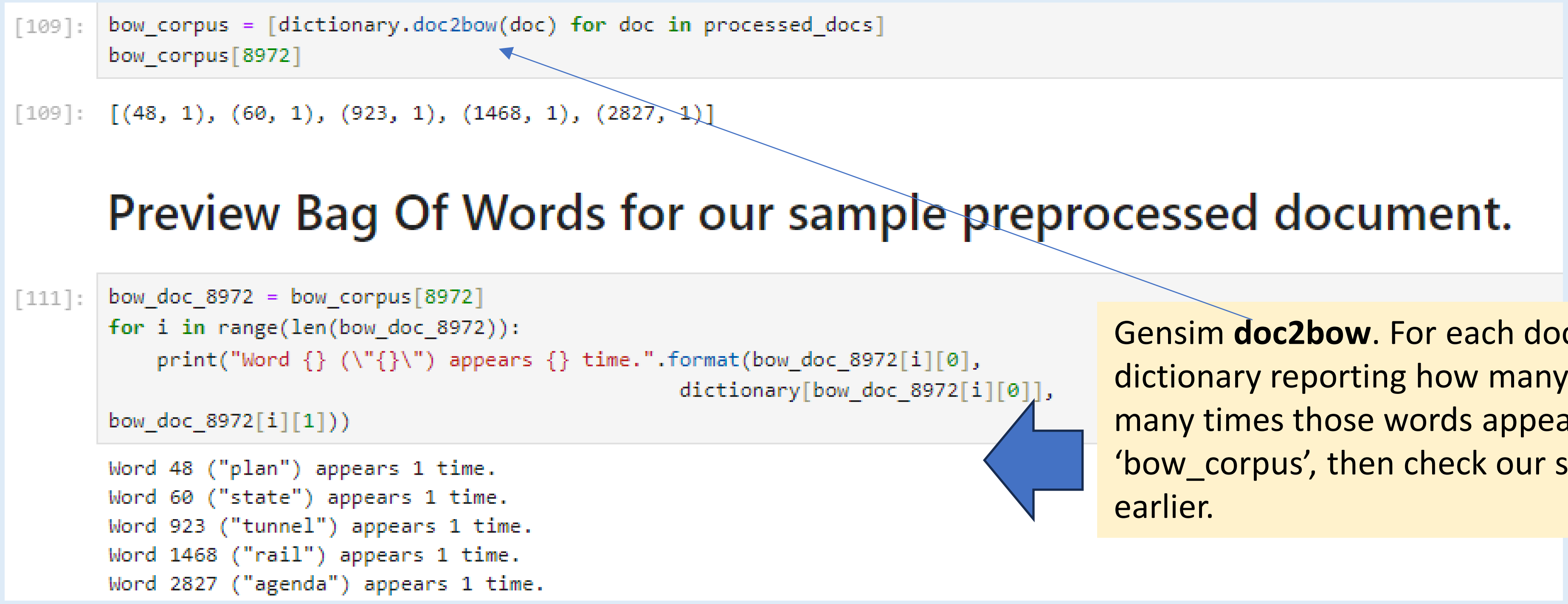
In order to work on text documents, Gensim (a natural language processing package) requires the words (aka tokens) be converted to unique ids. In order to achieve that, Gensim lets you create a Dictionary object that maps each word to a unique id.

Dictionary has 70,292 words

Applying LDA to a set of Documents to get the Latent Topics

By now, we have: ‘processed_docs’ is a sequence of 1,244,184 headlines-sentence-list of words, e.g., [decid, commun, broadcast, licenc], and ‘dictionary’ has 70,293 unique words with id given by Gensim. This id is the dimension number in Bag-of-Words of 70,293 dimensions. Note the last word in this dictionary with 1 in the 70,293 dim and zero elsewhere is ‘usaustralia’.

In the 8972th document: the tokenized and lemmatized document is ['rail', 'tunnel', 'plan', 'state', 'agenda’].



```
[109]: bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
       bow_corpus[8972]

[109]: [(48, 1), (60, 1), (923, 1), (1468, 1), (2827, 1)]
```

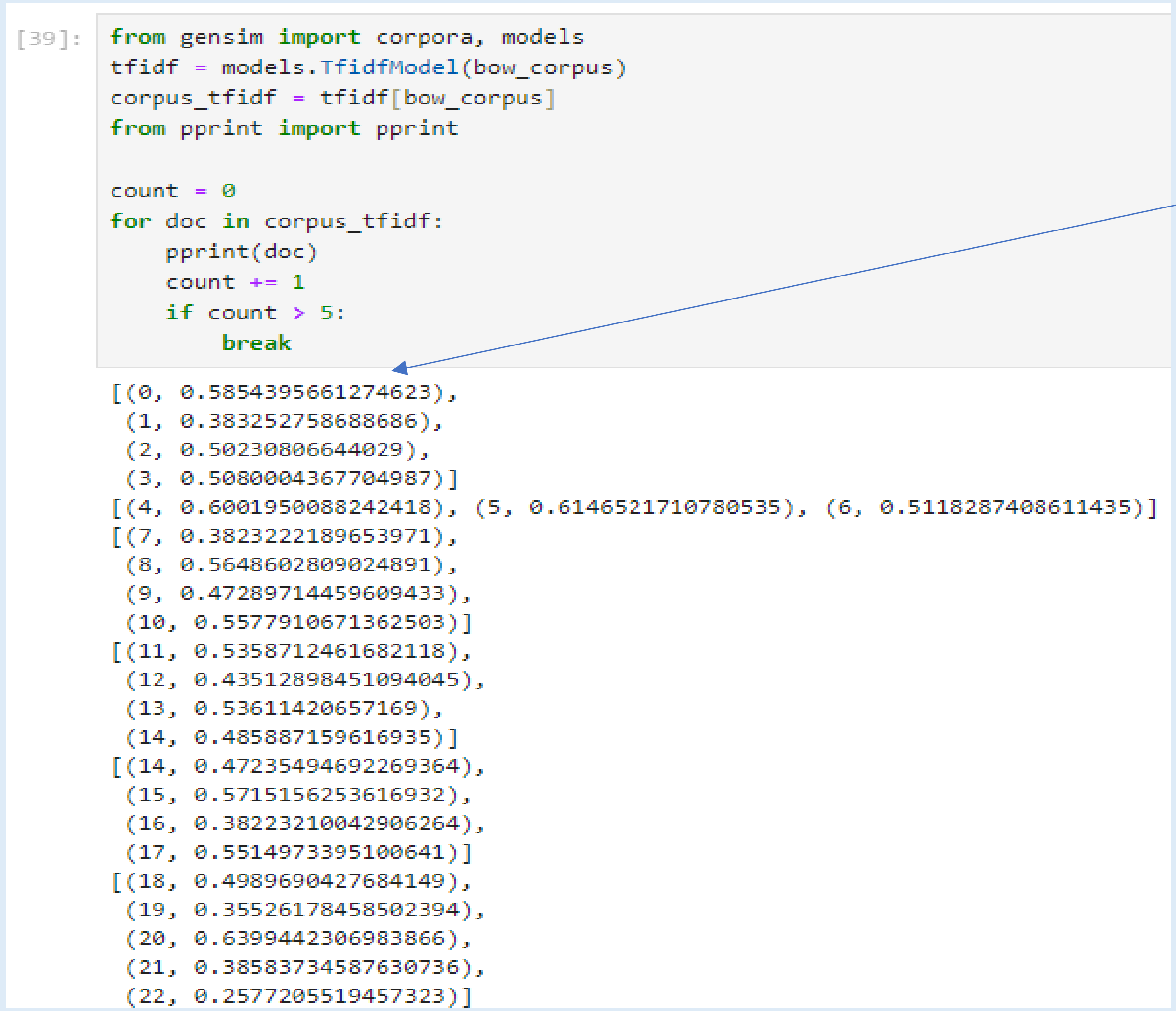
Preview Bag Of Words for our sample preprocessed document.

```
[111]: bow_doc_8972 = bow_corpus[8972]
       for i in range(len(bow_doc_8972)):
           print("Word {} (\"{}\") appears {} time.".format(bow_doc_8972[i][0],
                                                            dictionary[bow_doc_8972[i][0]],
       bow_doc_8972[i][1]))

       Word 48 ("plan") appears 1 time.
       Word 60 ("state") appears 1 time.
       Word 923 ("tunnel") appears 1 time.
       Word 1468 ("rail") appears 1 time.
       Word 2827 ("agenda") appears 1 time.
```

Gensim **doc2bow**. For each document we create a dictionary reporting how many words and how many times those words appear. Save this to ‘bow_corpus’, then check our selected document earlier.

Applying LDA to a set of Documents to get the Latent Topics

From bow_corpus, we can create TF-IDF on each word.



```
[39]: from gensim import corpora, models
      tfidf = models.TfidfModel(bow_corpus)
      corpus_tfidf = tfidf[bow_corpus]
      from pprint import pprint

      count = 0
      for doc in corpus_tfidf:
          pprint(doc)
          count += 1
          if count > 5:
              break
```

```
[(0, 0.5854395661274623),
 (1, 0.383252758688686),
 (2, 0.50230806644029),
 (3, 0.5080004367704987)]
[(4, 0.6001950088242418), (5, 0.6146521710780535), (6, 0.5118287408611435)]
[(7, 0.3823222189653971),
 (8, 0.5648602809024891),
 (9, 0.47289714459609433),
 (10, 0.5577910671362503)]
[(11, 0.5358712461682118),
 (12, 0.43512898451094045),
 (13, 0.53611420657169),
 (14, 0.485887159616935)]
[(14, 0.47235494692269364),
 (15, 0.5715156253616932),
 (16, 0.38223210042906264),
 (17, 0.5514973395100641)]
[(18, 0.4989690427684149),
 (19, 0.35526178458502394),
 (20, 0.6399442306983866),
 (21, 0.38583734587630736),
 (22, 0.2577205519457323)]
```

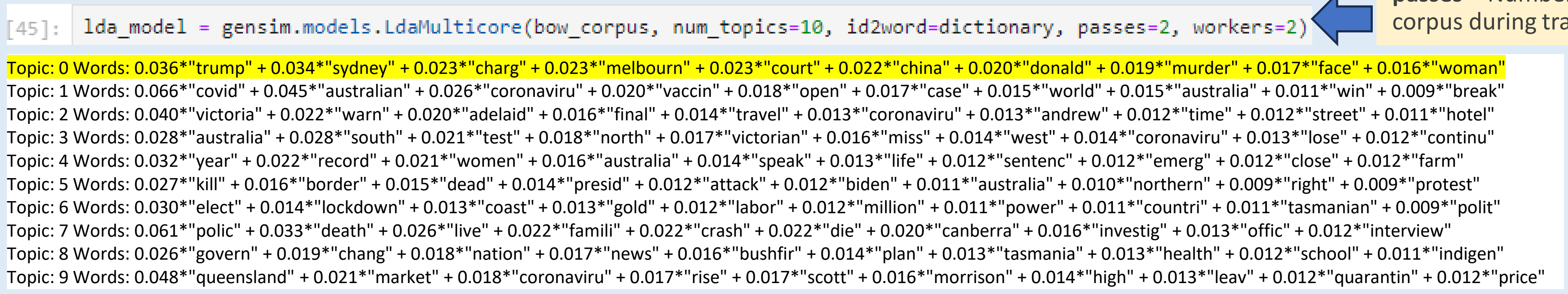
Create tf-idf model object using models.TfidfModel on ‘bow_corpus’ and save it to ‘tfidf’, then apply transformation to the entire corpus and call it ‘corpus_tfidf’. Preview TF-IDF scores for our first 6 documents.

Applying LDA to a set of Documents to get the Latent Topics

Running LDA using Bag of Words

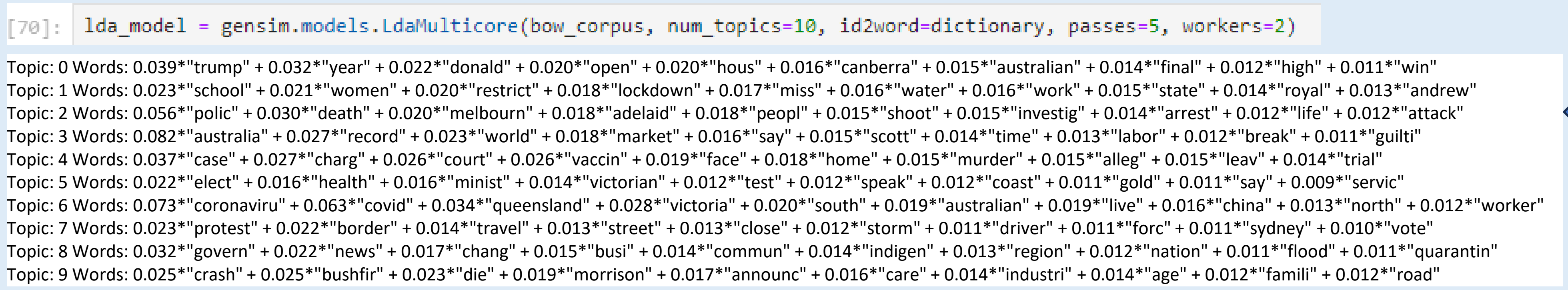
Training LDA model using gensim.models.LdaMulticore and save it to ‘lda_model’

workers - Number of workers processes to be used for parallelization.
passes – Number of passes through the corpus during training.



```
[45]:  lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=10, id2word=dictionary, passes=2, workers=2)
```

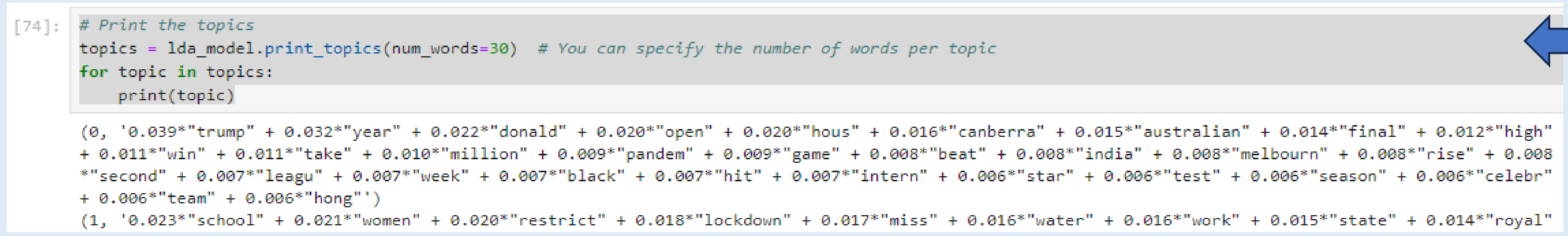
```
Topic: 0 Words: 0.036*"trump" + 0.034*"sydney" + 0.023*"charg" + 0.023*"melbourn" + 0.023*"court" + 0.022*"china" + 0.020*"donald" + 0.019*"murder" + 0.017*"face" + 0.016*"woman"
Topic: 1 Words: 0.066*"covid" + 0.045*"australian" + 0.026*"coronaviru" + 0.020*"vaccin" + 0.018*"open" + 0.017*"case" + 0.015*"world" + 0.015*"australia" + 0.011*"win" + 0.009*"break"
Topic: 2 Words: 0.040*"victoria" + 0.022*"warn" + 0.020*"adelaid" + 0.016*"final" + 0.014*"travel" + 0.013*"coronaviru" + 0.013*"andrew" + 0.012*"time" + 0.012*"street" + 0.011*"hotel"
Topic: 3 Words: 0.028*"australia" + 0.028*"south" + 0.021*"test" + 0.018*"north" + 0.017*"victorian" + 0.016*"miss" + 0.014*"west" + 0.014*"coronaviru" + 0.013*"lose" + 0.012*"continu"
Topic: 4 Words: 0.032*"year" + 0.022*"record" + 0.021*"women" + 0.016*"australia" + 0.014*"speak" + 0.013*"life" + 0.012*"sentenc" + 0.012*"emerg" + 0.012*"close" + 0.012*"farm"
Topic: 5 Words: 0.027*"kill" + 0.016*"border" + 0.015*"dead" + 0.014*"presid" + 0.012*"attack" + 0.012*"biden" + 0.011*"australia" + 0.010*"northern" + 0.009*"right" + 0.009*"protest"
Topic: 6 Words: 0.030*"elect" + 0.014*"lockdown" + 0.013*"coast" + 0.013*"gold" + 0.012*"labor" + 0.012*"million" + 0.011*"power" + 0.011*"countri" + 0.011*"tasmanian" + 0.009*"polit"
Topic: 7 Words: 0.061*"polic" + 0.033*"death" + 0.026*"live" + 0.022*"famili" + 0.022*"crash" + 0.022*"die" + 0.020*"canberra" + 0.016*"investig" + 0.013*"offic" + 0.012*"interview"
Topic: 8 Words: 0.026*"govern" + 0.019*"chang" + 0.018*"nation" + 0.017*"news" + 0.016*"bushfir" + 0.014*"plan" + 0.013*"tasmania" + 0.013*"health" + 0.012*"school" + 0.011*"indigen"
Topic: 9 Words: 0.048*"queensland" + 0.021*"market" + 0.018*"coronaviru" + 0.017*"rise" + 0.017*"scott" + 0.016*"morrison" + 0.014*"high" + 0.013*"leav" + 0.012*"quarantin" + 0.012*"price"
```



```
[70]:  lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=10, id2word=dictionary, passes=5, workers=2)
```

```
Topic: 0 Words: 0.039*"trump" + 0.032*"year" + 0.022*"donald" + 0.020*"open" + 0.020*"hous" + 0.016*"canberra" + 0.015*"australian" + 0.014*"final" + 0.012*"high" + 0.011*"win"
Topic: 1 Words: 0.023*"school" + 0.021*"women" + 0.020*"restrict" + 0.018*"lockdown" + 0.017*"miss" + 0.016*"water" + 0.016*"work" + 0.015*"state" + 0.014*"royal" + 0.013*"andrew"
Topic: 2 Words: 0.056*"polic" + 0.030*"death" + 0.020*"melbourn" + 0.018*"adelaid" + 0.018*"peopl" + 0.015*"shoot" + 0.015*"investig" + 0.014*"arrest" + 0.012*"life" + 0.012*"attack"
Topic: 3 Words: 0.082*"australia" + 0.027*"record" + 0.023*"world" + 0.018*"market" + 0.016*"say" + 0.015*"scott" + 0.014*"time" + 0.013*"labor" + 0.012*"break" + 0.011*"guilti"
Topic: 4 Words: 0.037*"case" + 0.027*"charg" + 0.026*"court" + 0.026*"vaccin" + 0.019*"face" + 0.018*"home" + 0.015*"murder" + 0.015*"alleg" + 0.015*"leav" + 0.014*"trial"
Topic: 5 Words: 0.022*"elect" + 0.016*"health" + 0.016*"minist" + 0.014*"victorian" + 0.012*"test" + 0.012*"speak" + 0.012*"coast" + 0.011*"gold" + 0.011*"say" + 0.009*"servic"
Topic: 6 Words: 0.073*"coronaviru" + 0.063*"covid" + 0.034*"queensland" + 0.028*"victoria" + 0.020*"south" + 0.019*"australian" + 0.019*"live" + 0.016*"china" + 0.013*"north" + 0.012*"worker"
Topic: 7 Words: 0.023*"protest" + 0.022*"border" + 0.014*"travel" + 0.013*"street" + 0.013*"close" + 0.012*"storm" + 0.011*"driver" + 0.011*"forc" + 0.011*"sydney" + 0.010*"vote"
Topic: 8 Words: 0.032*"govern" + 0.022*"news" + 0.017*"chang" + 0.015*"busi" + 0.014*"commun" + 0.014*"indigen" + 0.013*"region" + 0.012*"nation" + 0.011*"flood" + 0.011*"quarantin"
Topic: 9 Words: 0.025*"crash" + 0.025*"bushfir" + 0.023*"die" + 0.019*"morrison" + 0.017*"announc" + 0.016*"care" + 0.014*"industri" + 0.014*"age" + 0.012*"famili" + 0.012*"road"
```

Shows only top ten words in each topic



```
[74]:  # Print the topics
       topics = lda_model.print_topics(num_words=30)  # You can specify the number of words per topic
       for topic in topics:
           print(topic)
```

```
(0, '0.039*"trump" + 0.032*"year" + 0.022*"donald" + 0.020*"open" + 0.020*"hous" + 0.016*"canberra" + 0.015*"australian" + 0.014*"final" + 0.012*"high"
+ 0.011*"win" + 0.011*"take" + 0.010*"million" + 0.009*"pandem" + 0.009*"game" + 0.008*"beat" + 0.008*"india" + 0.008*"melbourn" + 0.008*"rise" + 0.008
*"second" + 0.007*"leagu" + 0.007*"week" + 0.007*"black" + 0.007*"hit" + 0.007*"intern" + 0.006*"star" + 0.006*"test" + 0.006*"season" + 0.006*"celebr"
+ 0.006*"team" + 0.006*"hong"')
(1, '0.023*"school" + 0.021*"women" + 0.020*"restrict" + 0.018*"lockdown" + 0.017*"miss" + 0.016*"water" + 0.016*"work" + 0.015*"state" + 0.014*"royal"
```

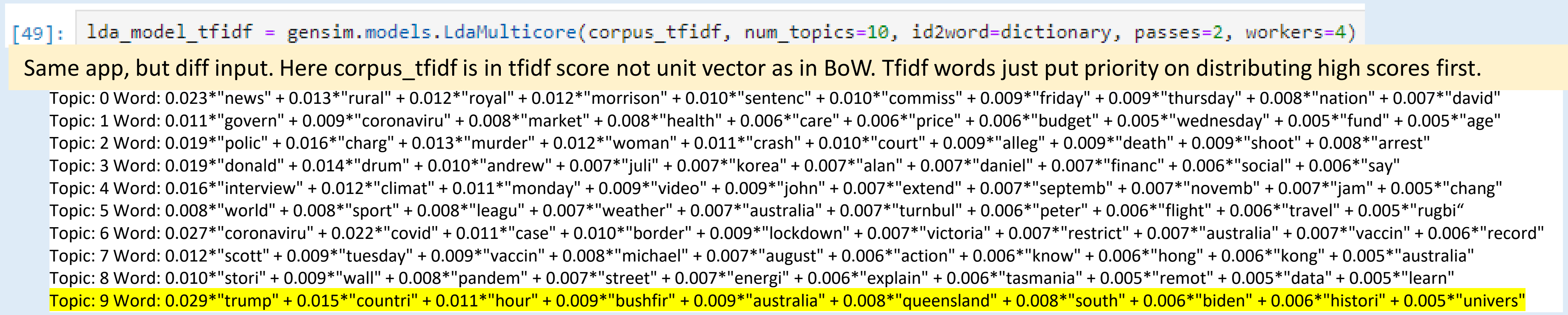
Shows top 30 words in each topic

………………………………………………………………

Applying LDA to a set of Documents to get the Latent Topics

Running LDA using TF-IDF

Training LDA model using gensim.models.LdaMulticore and save it to ‘lda_model_tfidf’



```
[49]:  lda_model_tfidf = gensim.models.LdaMulticore(corpus_tfidf, num_topics=10, id2word=dictionary, passes=2, workers=4)
```

Same app, but diff input. Here corpus_tfidf is in tfidf score not unit vector as in BoW. Tfidf words just put priority on distributing high scores first.

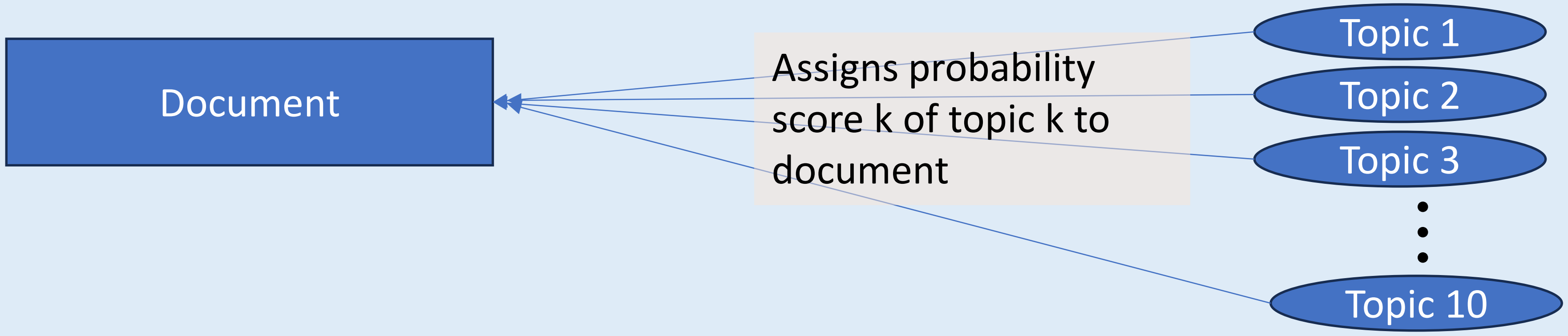
```
Topic: 0 Word: 0.023*"news" + 0.013*"rural" + 0.012*"royal" + 0.012*"morrison" + 0.010*"sentenc" + 0.010*"commiss" + 0.009*"friday" + 0.009*"thursday" + 0.008*"nation" + 0.007*"david"
Topic: 1 Word: 0.011*"govern" + 0.009*"coronaviru" + 0.008*"market" + 0.008*"health" + 0.006*"care" + 0.006*"price" + 0.006*"budget" + 0.005*"wednesday" + 0.005*"fund" + 0.005*"age"
Topic: 2 Word: 0.019*"polic" + 0.016*"charg" + 0.013*"murder" + 0.012*"woman" + 0.011*"crash" + 0.010*"court" + 0.009*"alleg" + 0.009*"death" + 0.009*"shoot" + 0.008*"arrest"
Topic: 3 Word: 0.019*"donald" + 0.014*"drum" + 0.010*"andrew" + 0.007*"juli" + 0.007*"korea" + 0.007*"alan" + 0.007*"daniel" + 0.007*"financ" + 0.006*"social" + 0.006*"say"
Topic: 4 Word: 0.016*"interview" + 0.012*"climat" + 0.011*"monday" + 0.009*"video" + 0.009*"john" + 0.007*"extend" + 0.007*"septemb" + 0.007*"novemb" + 0.007*"jam" + 0.005*"chang"
Topic: 5 Word: 0.008*"world" + 0.008*"sport" + 0.008*"leagu" + 0.007*"weather" + 0.007*"australia" + 0.007*"turnbul" + 0.006*"peter" + 0.006*"flight" + 0.006*"travel" + 0.005*"rugbi“
Topic: 6 Word: 0.027*"coronaviru" + 0.022*"covid" + 0.011*"case" + 0.010*"border" + 0.009*"lockdown" + 0.007*"victoria" + 0.007*"restrict" + 0.007*"australia" + 0.007*"vaccin" + 0.006*"record"
Topic: 7 Word: 0.012*"scott" + 0.009*"tuesday" + 0.009*"vaccin" + 0.008*"michael" + 0.007*"august" + 0.006*"action" + 0.006*"know" + 0.006*"hong" + 0.006*"kong" + 0.005*"australia"
Topic: 8 Word: 0.010*"stori" + 0.009*"wall" + 0.008*"pandem" + 0.007*"street" + 0.007*"energi" + 0.006*"explain" + 0.006*"tasmania" + 0.005*"remot" + 0.005*"data" + 0.005*"learn"
Topic: 9 Word: 0.029*"trump" + 0.015*"countri" + 0.011*"hour" + 0.009*"bushfir" + 0.009*"australia" + 0.008*"queensland" + 0.008*"south" + 0.006*"biden" + 0.006*"histori" + 0.005*"univers"
```

In Gensim LdaMulticore, *alpha* and *eta* are hyperparameters that affect sparsity of the document-topic (theta) and topic-word distributions. Both default to a symmetric 1.0.

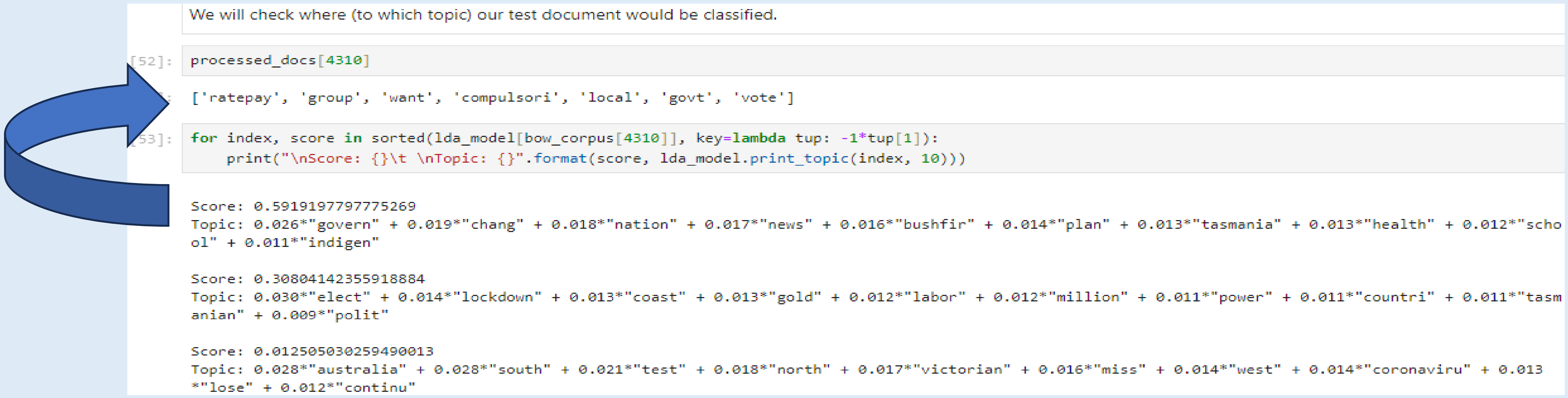
Topic 9 in TF-IDF word representation looks similar to Topic 0 in Bag of Words word representation. Note the word probs in each topic here smaller but more uniform because more spread of words across all topics.

Can you distinguish different topics using the words in each topic and their corresponding weights in that topic?

Performance evaluation of LDA by classifying sample document using Bag of Words representation of words



For a given document, check if the assigned topic with the highest probability score is indeed where the document should largely belong to – as the document can have one or more topics.

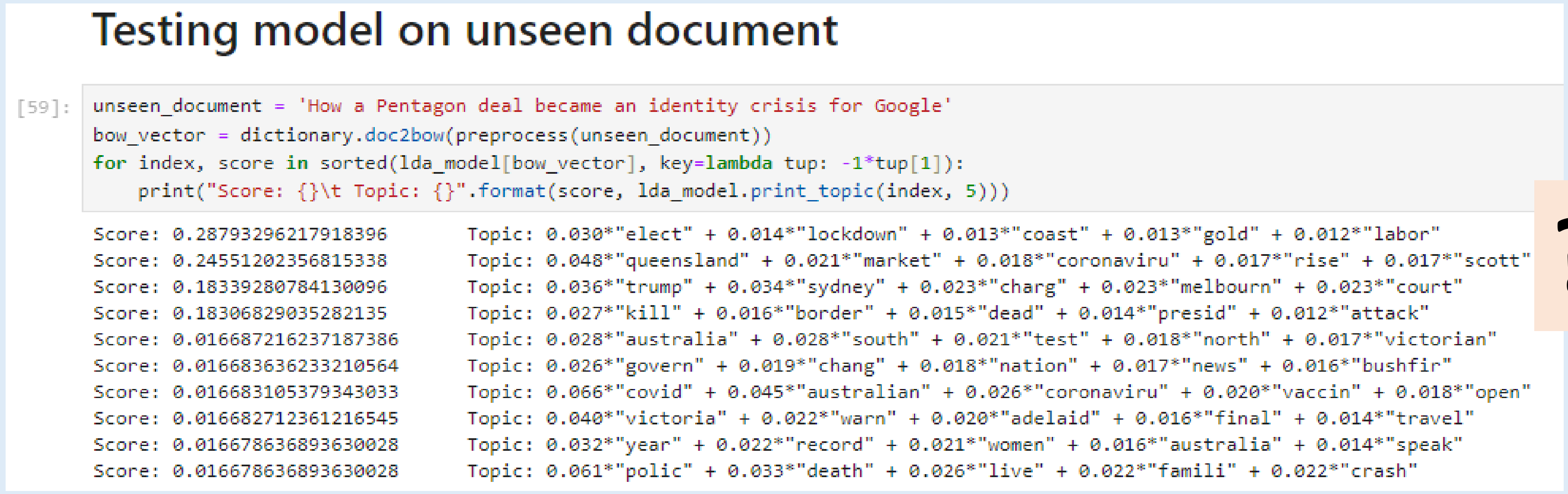


We will check where (to which topic) our test document would be classified.

```
[52]: processed_docs[4310]

      ['ratepay', 'group', 'want', 'compulsori', 'local', 'govt', 'vote']

[53]: for index, score in sorted(lda_model[bow_corpus[4310]], key=lambda tup: -1*tup[1]):
          print("\nScore: {}\t \nTopic: {}".format(score, lda_model.print_topic(index, 10)))
```

```
Score: 0.5919197797775269
Topic: 0.026*"govern" + 0.019*"chang" + 0.018*"nation" + 0.017*"news" + 0.016*"bushfir" + 0.014*"plan" + 0.013*"tasmania" + 0.013*"health" + 0.012*"scho
ol" + 0.011*"indigen"

Score: 0.30804142355918884
Topic: 0.030*"elect" + 0.014*"lockdown" + 0.013*"coast" + 0.013*"gold" + 0.012*"labor" + 0.012*"million" + 0.011*"power" + 0.011*"countri" + 0.011*"tasm
anian" + 0.009*"polit"

Score: 0.012505030259490013
Topic: 0.028*"australia" + 0.028*"south" + 0.021*"test" + 0.018*"north" + 0.017*"victorian" + 0.016*"miss" + 0.014*"west" + 0.014*"coronaviru" + 0.013
*"lose" + 0.012*"continu"
```

Performance evaluation of LDA by classifying an unseen document using Bag of Words representation of words

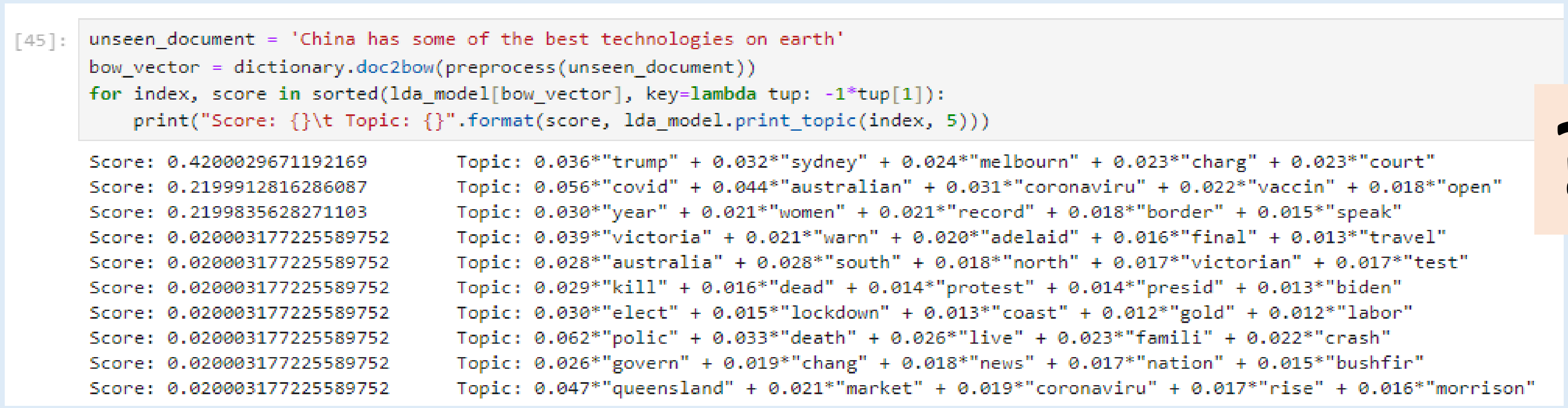


Testing model on unseen document

```
[59]: unseen_document = 'How a Pentagon deal became an identity crisis for Google'
      bow_vector = dictionary.doc2bow(preprocess(unseen_document))
      for index, score in sorted(lda_model[bow_vector], key=lambda tup: -1*tup[1]):
          print("Score: {}\t Topic: {}".format(score, lda_model.print_topic(index, 5)))
```

```
Score: 0.28793296217918396	 Topic: 0.030*"elect" + 0.014*"lockdown" + 0.013*"coast" + 0.013*"gold" + 0.012*"labor"
Score: 0.24551202356815338	 Topic: 0.048*"queensland" + 0.021*"market" + 0.018*"coronaviru" + 0.017*"rise" + 0.017*"scott"
Score: 0.18339280784130096	 Topic: 0.036*"trump" + 0.034*"sydney" + 0.023*"charg" + 0.023*"melbourn" + 0.023*"court"
Score: 0.18306829035282135	 Topic: 0.027*"kill" + 0.016*"border" + 0.015*"dead" + 0.014*"presid" + 0.012*"attack"
Score: 0.016687216237187386	 Topic: 0.028*"australia" + 0.028*"south" + 0.021*"test" + 0.018*"north" + 0.017*"victorian"
Score: 0.016683636233210564	 Topic: 0.026*"govern" + 0.019*"chang" + 0.018*"nation" + 0.017*"news" + 0.016*"bushfir"
Score: 0.016683105379343033	 Topic: 0.066*"covid" + 0.045*"australian" + 0.026*"coronaviru" + 0.020*"vaccin" + 0.018*"open"
Score: 0.016682712361216545	 Topic: 0.040*"victoria" + 0.022*"warn" + 0.020*"adelaid" + 0.016*"final" + 0.014*"travel"
Score: 0.016678636893630028	 Topic: 0.032*"year" + 0.022*"record" + 0.021*"women" + 0.016*"australia" + 0.014*"speak"
Score: 0.016678636893630028	 Topic: 0.061*"polic" + 0.033*"death" + 0.026*"live" + 0.022*"famili" + 0.022*"crash"
```

?



```
[45]: unseen_document = 'China has some of the best technologies on earth'
      bow_vector = dictionary.doc2bow(preprocess(unseen_document))
      for index, score in sorted(lda_model[bow_vector], key=lambda tup: -1*tup[1]):
          print("Score: {}\t Topic: {}".format(score, lda_model.print_topic(index, 5)))
```

```
Score: 0.4200029671192169	 Topic: 0.036*"trump" + 0.032*"sydney" + 0.024*"melbourn" + 0.023*"charg" + 0.023*"court"
Score: 0.2199912816286087	 Topic: 0.056*"covid" + 0.044*"australian" + 0.031*"coronaviru" + 0.022*"vaccin" + 0.018*"open"
Score: 0.2199835628271103	 Topic: 0.030*"year" + 0.021*"women" + 0.021*"record" + 0.018*"border" + 0.015*"speak"
Score: 0.020003177225589752	 Topic: 0.039*"victoria" + 0.021*"warn" + 0.020*"adelaid" + 0.016*"final" + 0.013*"travel"
Score: 0.020003177225589752	 Topic: 0.028*"australia" + 0.028*"south" + 0.018*"north" + 0.017*"victorian" + 0.017*"test"
Score: 0.020003177225589752	 Topic: 0.029*"kill" + 0.016*"dead" + 0.014*"protest" + 0.014*"presid" + 0.013*"biden"
Score: 0.020003177225589752	 Topic: 0.030*"elect" + 0.015*"lockdown" + 0.013*"coast" + 0.012*"gold" + 0.012*"labor"
Score: 0.020003177225589752	 Topic: 0.062*"polic" + 0.033*"death" + 0.026*"live" + 0.023*"famili" + 0.022*"crash"
Score: 0.020003177225589752	 Topic: 0.026*"govern" + 0.019*"chang" + 0.018*"news" + 0.017*"nation" + 0.015*"bushfir"
Score: 0.020003177225589752	 Topic: 0.047*"queensland" + 0.021*"market" + 0.019*"coronaviru" + 0.017*"rise" + 0.016*"morrison"
```

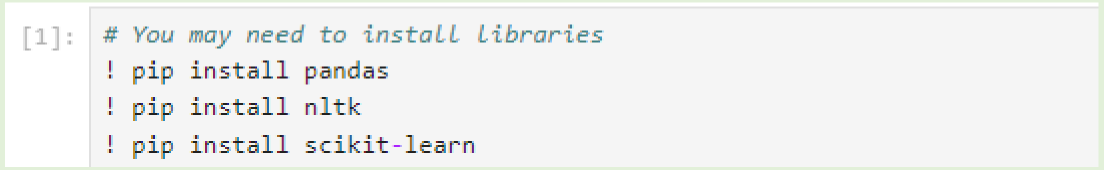
??

Topic Modeling vs. Topic Classification

- Latent Dirichlet Allocation is an important decomposition technique for topic modeling in NLP that can automatically (without supervision) discover hidden topics from documents.
- However, curse of dimensionality makes it difficult to train models when the number of features (words*) is huge and flat (most words occur once or twice) and it reduces the efficiency of the model to output correct topic allocations to a document. *Using Bag-of-Words, dimension of corpus tokens (word id) is too big.
- Other text recognition architectures such as word2vec’s skip-gram or BERT’s pre-trained contextual representations can more efficiently represent words and word connections to effectuate a faster and more accurate LDA process than just using Bag of Words or TF-IDF word representations.
- Topic modeling is preferred when there is not much time to analyze texts and there is no need for detailed analysis – just a few topics of what texts are talking about is enough.
- On the other hand, when time is permissible and more accuracy is required in topic identification on texts, text classification into topics is preferred. The classification, unlike topic modeling, is based on exogenously determined labels of topics, and texts can be trained to predict or identify an appropriate label or topic.
- One example of such text or topic classification is the identification of SPAM. Another example is identifying sentiments.

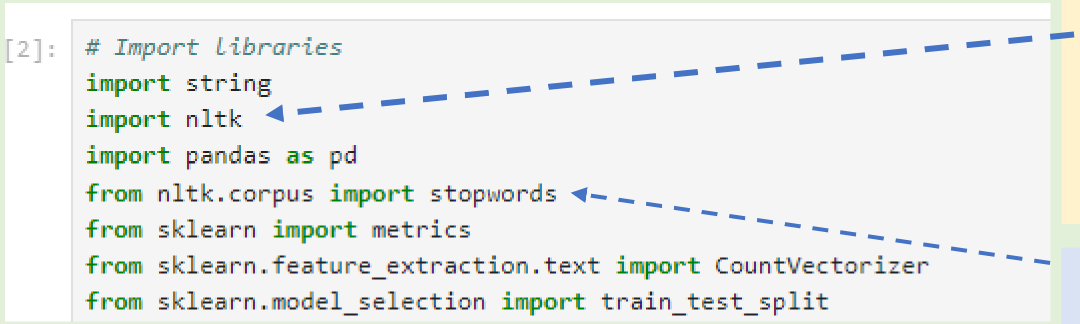
SPAM Detector

L5-spam.ipynb



```
[1]: # You may need to install libraries
     ! pip install pandas
     ! pip install nltk
     ! pip install scikit-learn
```

"spam.csv" (492 kB) is available at Kaggle, viz. https://www.kaggle.com/datasets/tmehul/spamcsv
It contains 5572 lines of email messages in column 2 (or v2). 4825 of these are labelled as "ham" (non-spam) while 747 are labelled as "spam". The labels are shown in column 1 (or v1).
Spam refers to any kind of unsolicited and unwanted digital communication – here in the form of emails.

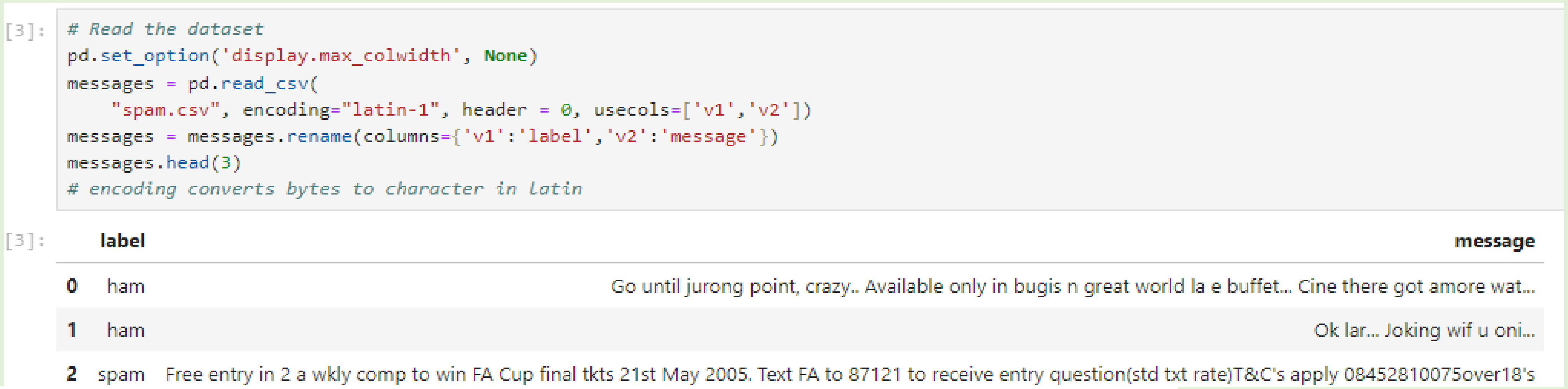


```
[2]: # Import libraries
     import string
     import nltk
     import pandas as pd
     from nltk.corpus import stopwords
     from sklearn import metrics
     from sklearn.feature_extraction.text import CountVectorizer
     from sklearn.model_selection import train_test_split
```

The **Natural Language Toolkit**, or more commonly **NLTK**, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. It supports classification, tokenization, stemming, tagging, parsing, and semantic reasoning functionalities.

A stop wordis a commonly used **word** (such as **"the",** "a", "an", or "in") that a search engine has been programmed to ignore.

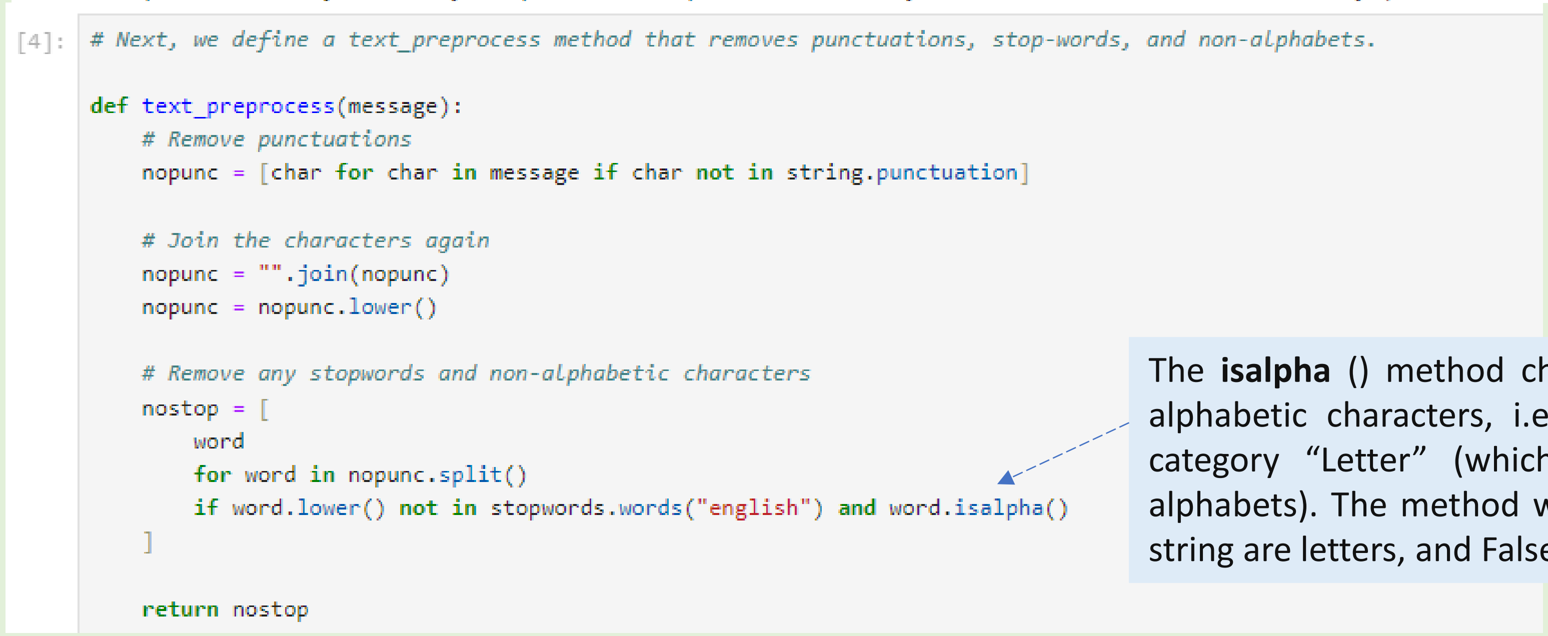
SPAM Detector



```
[3]: # Read the dataset
     pd.set_option('display.max_colwidth', None)
     messages = pd.read_csv(
         "spam.csv", encoding="latin-1", header = 0, usecols=['v1','v2'])
     messages = messages.rename(columns={'v1':'label','v2':'message'})
     messages.head(3)
     # encoding converts bytes to character in latin
```

[3]:

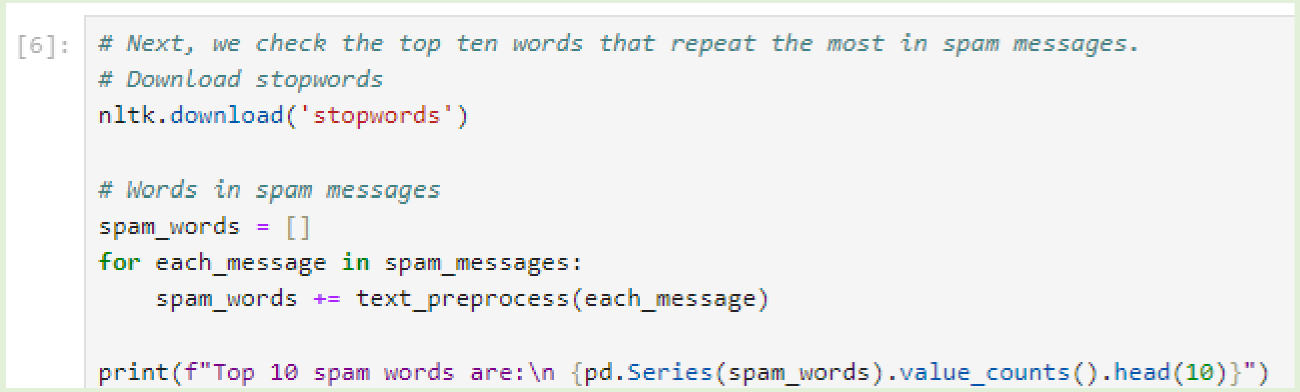
	label	message
0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's



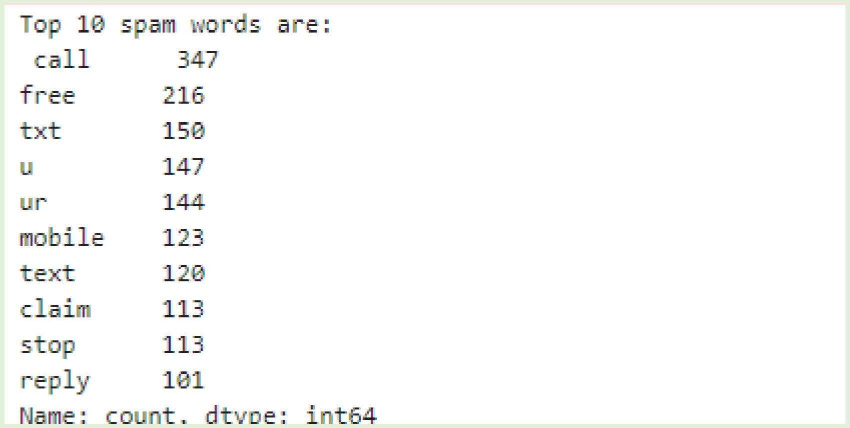
```
[4]: # Next, we define a text_preprocess method that removes punctuations, stop-words, and non-alphabets.

     def text_preprocess(message):
         # Remove punctuations
         nopunc = [char for char in message if char not in string.punctuation]

         # Join the characters again
         nopunc = "".join(nopunc)
         nopunc = nopunc.lower()

         # Remove any stopwords and non-alphabetic characters
         nostop = [
             word
             for word in nopunc.split()
             if word.lower() not in stopwords.words("english") and word.isalpha()
         ]

         return nostop
```

The **isalpha** () method checks if all the characters in a string are alphabetic characters, i.e., whether they belong to the Unicode category “Letter” (which includes letters from all scripts and alphabets). The method will return True if all the characters in the string are letters, and False otherwise.

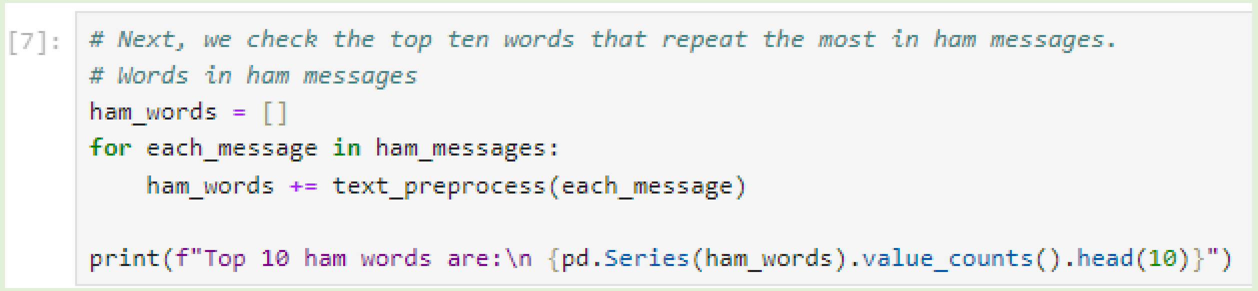
SPAM Detector



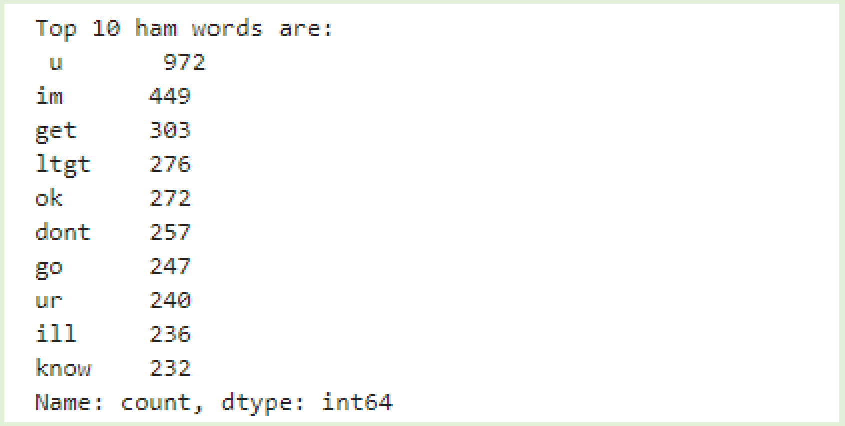
```
[6]: # Next, we check the top ten words that repeat the most in spam messages.
     # Download stopwords
     nltk.download('stopwords')

     # Words in spam messages
     spam_words = []
     for each_message in spam_messages:
         spam_words += text_preprocess(each_message)

     print(f"Top 10 spam words are:\n {pd.Series(spam_words).value_counts().head(10)}")
```



```
Top 10 spam words are:
 call      347
free      216
txt       150
u         147
ur        144
mobile    123
text      120
claim     113
stop      113
reply     101
Name: count, dtype: int64
```

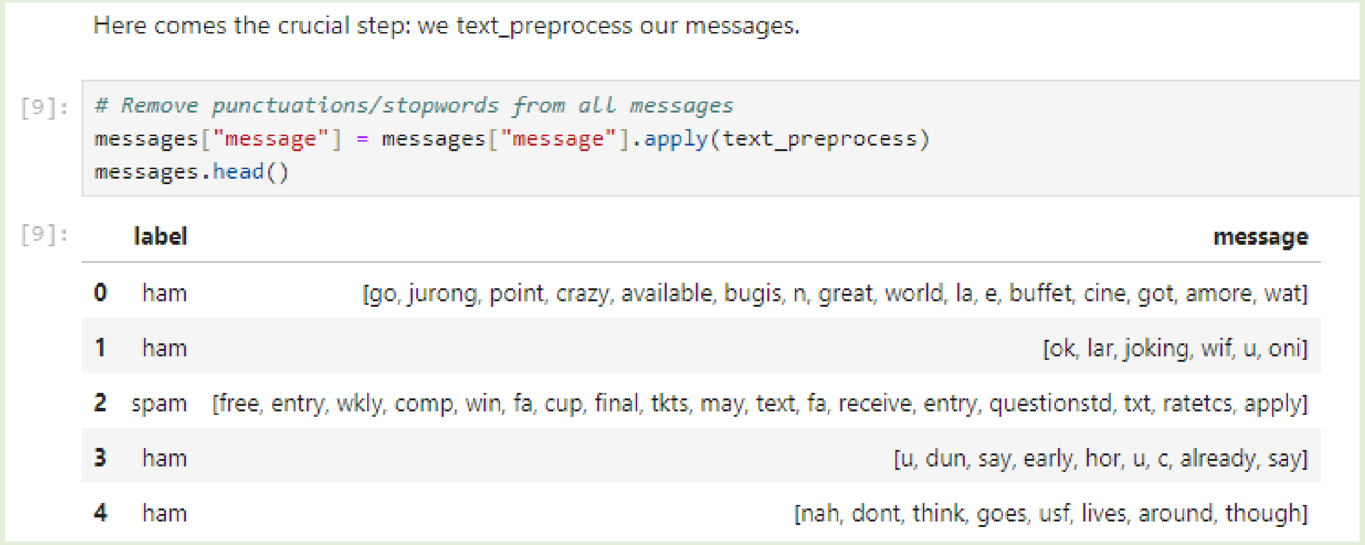


```
[7]: # Next, we check the top ten words that repeat the most in ham messages.
     # Words in ham messages
     ham_words = []
     for each_message in ham_messages:
         ham_words += text_preprocess(each_message)

     print(f"Top 10 ham words are:\n {pd.Series(ham_words).value_counts().head(10)}")
```



```
Top 10 ham words are:
 u      972
im     449
get    303
ltgt   276
ok     272
dont   257
go     247
ur     240
ill    236
know   232
Name: count, dtype: int64
```

SPAM Detector



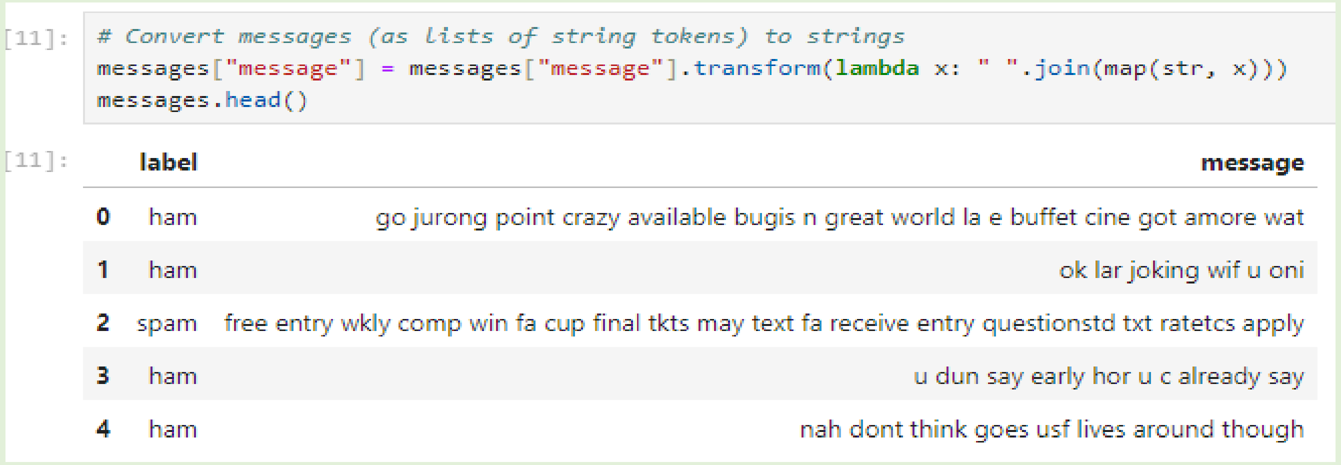
Here comes the crucial step: we text_preprocess our messages.

```
[9]: # Remove punctuations/stopwords from all messages
     messages["message"] = messages["message"].apply(text_preprocess)
     messages.head()
```

[9]:

	label	message
0	ham	[go, jurong, point, crazy, available, bugis, n, great, world, la, e, buffet, cine, got, amore, wat]
1	ham	[ok, lar, joking, wif, u, oni]
2	spam	[free, entry, wkly, comp, win, fa, cup, final, tkts, may, text, fa, receive, entry, questionstd, txt, ratetcs, apply]
3	ham	[u, dun, say, early, hor, u, c, already, say]
4	ham	[nah, dont, think, goes, usf, lives, around, though]

The output produced will be a list of tokens. A string can be understood by a model, not a list of tokens. Hence, we convert the list of tokens to a string.



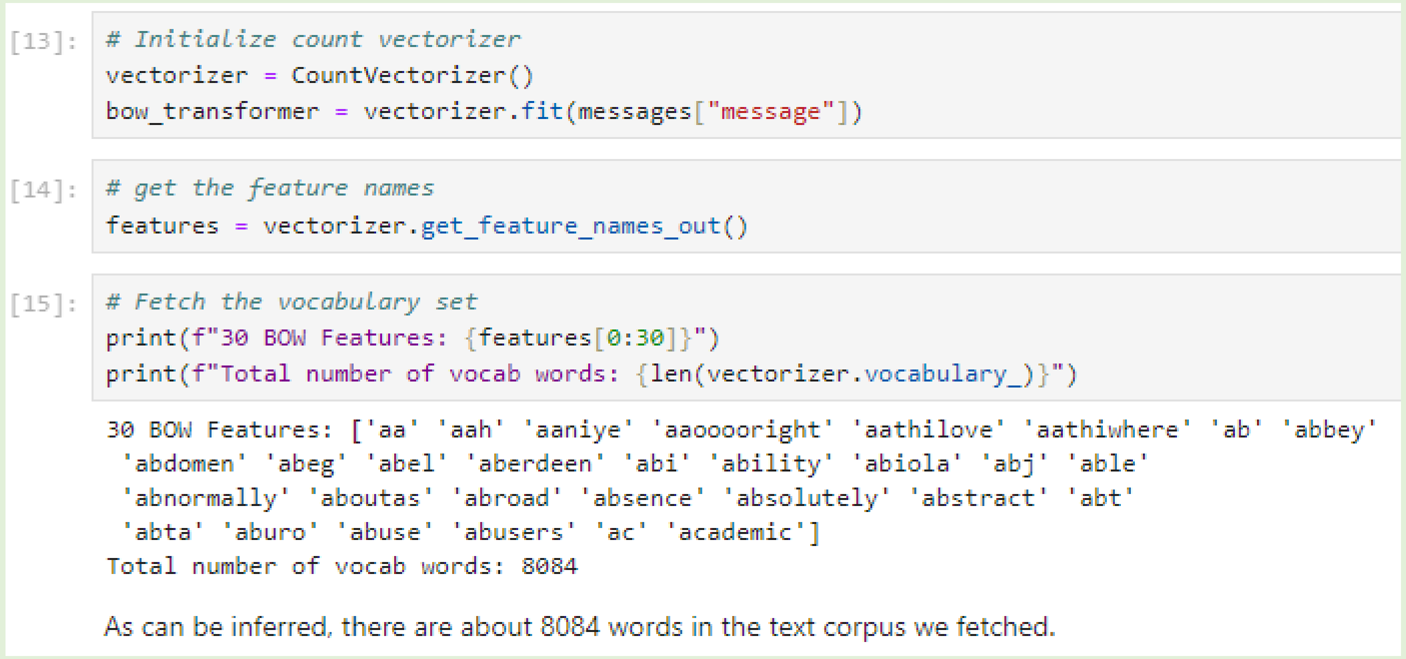
```
[11]: # Convert messages (as lists of string tokens) to strings
      messages["message"] = messages["message"].transform(lambda x: " ".join(map(str, x)))
      messages.head()
```

[11]:

	label	message
0	ham	go jurong point crazy available bugis n great world la e buffet cine got amore wat
1	ham	ok lar joking wif u oni
2	spam	free entry wkly comp win fa cup final tkts may text fa receive entry questionstd txt ratetcs apply
3	ham	u dun say early hor u c already say
4	ham	nah dont think goes usf lives around though

SPAM Detector

The CountVectorizer() class in the scikit-learn library is useful in defining the Bag-of-Words (BoW) approach. We first fit the vectorizer to the messages to fetch the whole vocabulary.



```
[13]: # Initialize count vectorizer
      vectorizer = CountVectorizer()
      bow_transformer = vectorizer.fit(messages["message"])
```

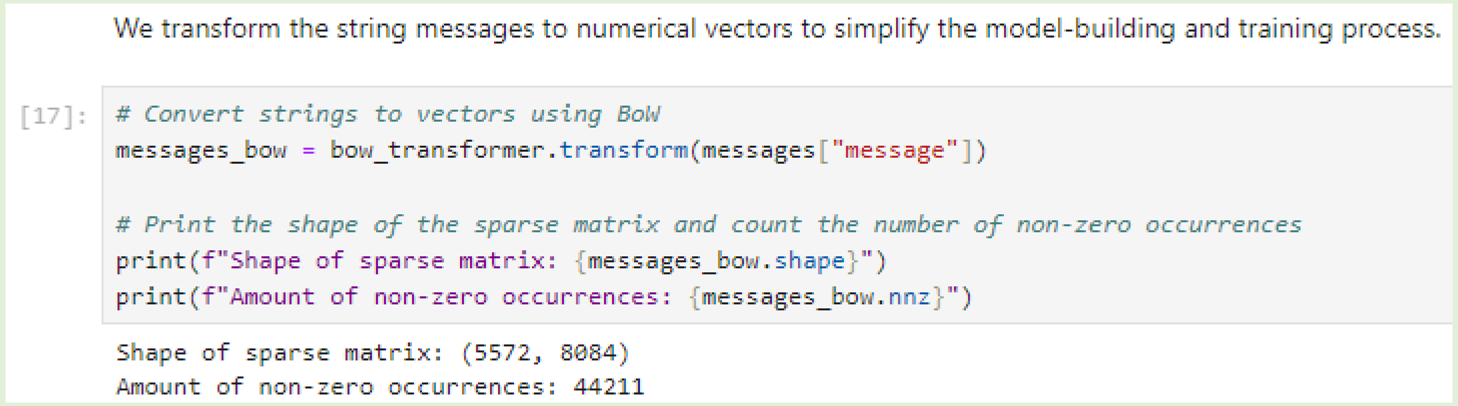
```
[14]: # get the feature names
      features = vectorizer.get_feature_names_out()
```

```
[15]: # Fetch the vocabulary set
      print(f"30 BOW Features: {features[0:30]}")
      print(f"Total number of vocab words: {len(vectorizer.vocabulary_)}")
```

```
30 BOW Features: ['aa' 'aah' 'aaniye' 'aaooooright' 'aathilove' 'aathiwhere' 'ab' 'abbey'
 'abdomen' 'abeg' 'abel' 'aberdeen' 'abi' 'ability' 'abiola' 'abj' 'able'
 'abnormally' 'aboutas' 'abroad' 'absence' 'absolutely' 'abstract' 'abt'
 'abta' 'aburo' 'abuse' 'abusers' 'ac' 'academic']
Total number of vocab words: 8084
```

As can be inferred, there are about 8084 words in the text corpus we fetched.

SPAM Detector



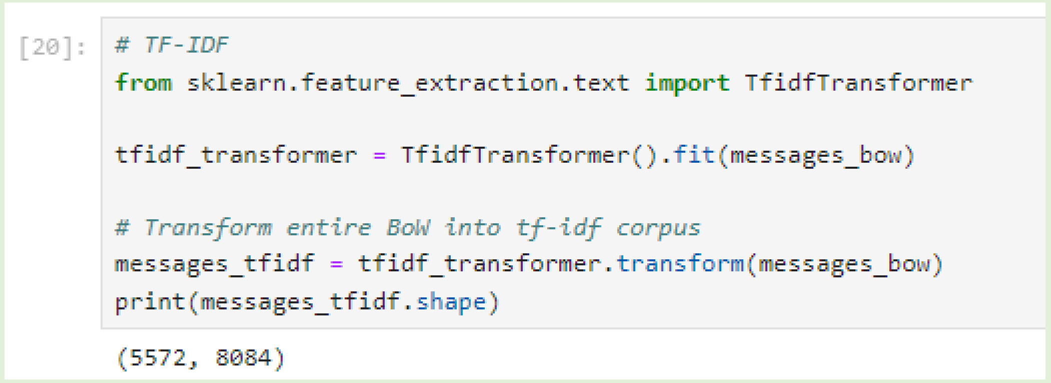
We transform the string messages to numerical vectors to simplify the model-building and training process.

```
[17]: # Convert strings to vectors using BoW
      messages_bow = bow_transformer.transform(messages["message"])

      # Print the shape of the sparse matrix and count the number of non-zero occurrences
      print(f"Shape of sparse matrix: {messages_bow.shape}")
      print(f"Amount of non-zero occurrences: {messages_bow.nnz}")
```

```
Shape of sparse matrix: (5572, 8084)
Amount of non-zero occurrences: 44211
```

BoW builds a sparse matrix mapping the occurrence of every word to the corpus vocabulary. Thus, this approach leads to building a sparse matrix, or a matrix that is mostly comprised of zeros. This format allows for the conversion of the text into an interpretable encoding of linguistic information that a model can make use of.

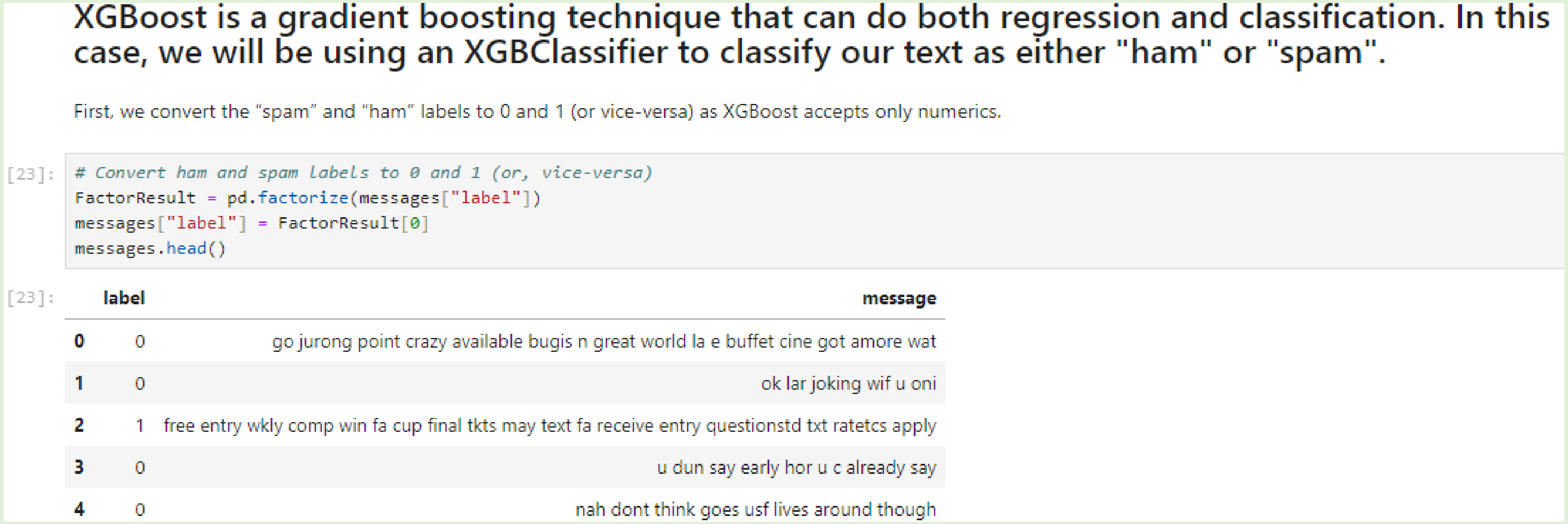
BoW’s technique could be enhanced when combined with TF-IDF. Here, we run our BoW vectors through TF-IDF.



```
[20]: # TF-IDF
      from sklearn.feature_extraction.text import TfidfTransformer

      tfidf_transformer = TfidfTransformer().fit(messages_bow)

      # Transform entire BoW into tf-idf corpus
      messages_tfidf = tfidf_transformer.transform(messages_bow)
      print(messages_tfidf.shape)
```

```
(5572, 8084)
```

SPAM Detector



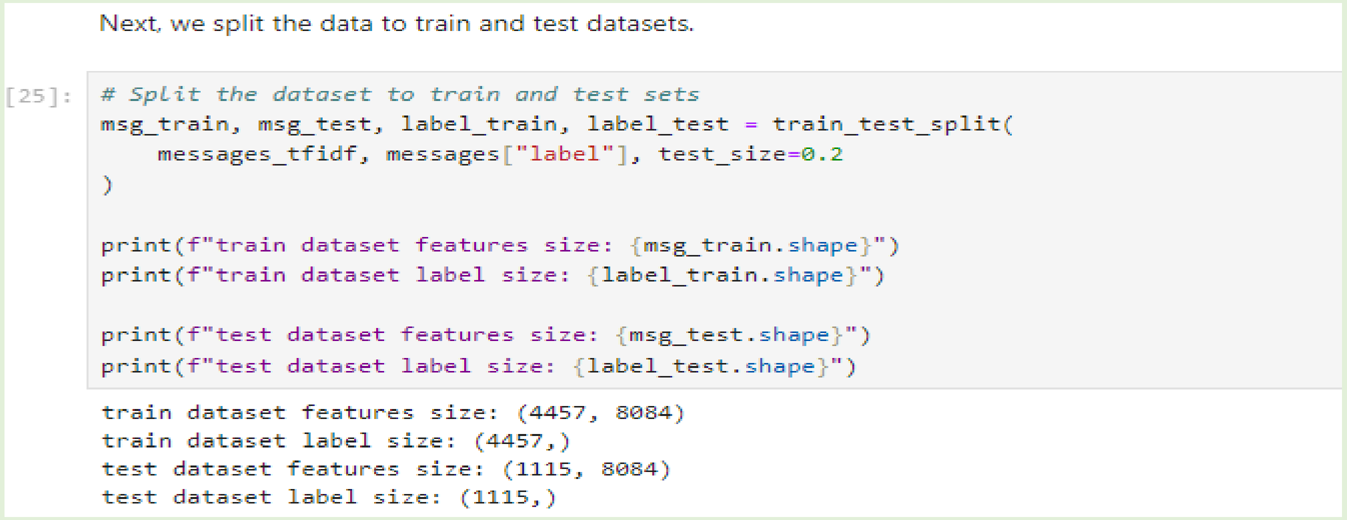
XGBoost is a gradient boosting technique that can do both regression and classification. In this case, we will be using an XGBClassifier to classify our text as either "ham" or "spam".

First, we convert the “spam” and “ham” labels to 0 and 1 (or vice-versa) as XGBoost accepts only numerics.

```
[23]: # Convert ham and spam labels to 0 and 1 (or, vice-versa)
      FactorResult = pd.factorize(messages["label"])
      messages["label"] = FactorResult[0]
      messages.head()
```

[23]:

	label	message
0	0	go jurong point crazy available bugis n great world la e buffet cine got amore wat
1	0	ok lar joking wif u oni
2	1	free entry wkly comp win fa cup final tkts may text fa receive entry questionstd txt ratetcs apply
3	0	u dun say early hor u c already say
4	0	nah dont think goes usf lives around though

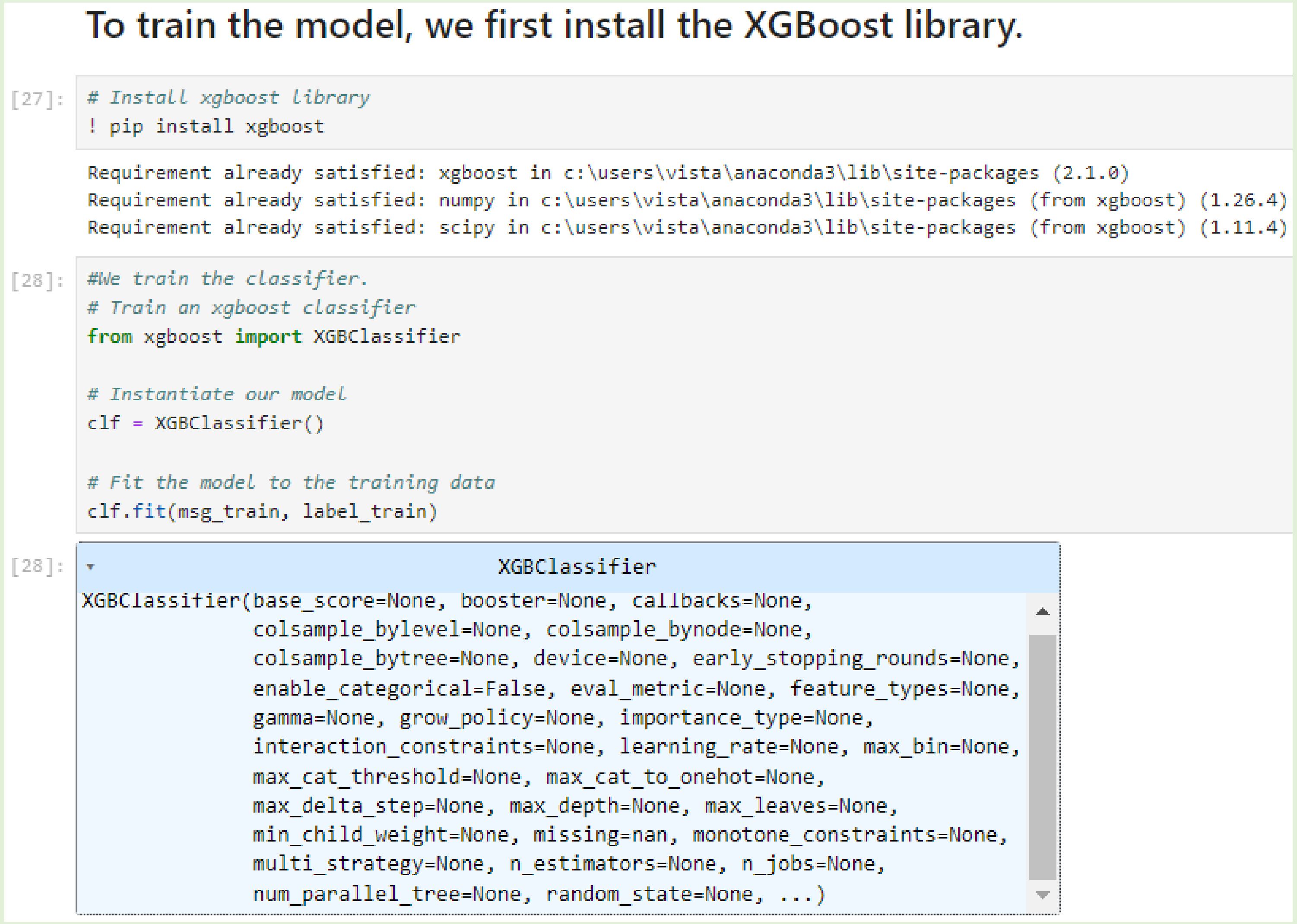


Next, we split the data to train and test datasets.

```
[25]: # Split the dataset to train and test sets
      msg_train, msg_test, label_train, label_test = train_test_split(
          messages_tfidf, messages["label"], test_size=0.2
      )

      print(f"train dataset features size: {msg_train.shape}")
      print(f"train dataset label size: {label_train.shape}")

      print(f"test dataset features size: {msg_test.shape}")
      print(f"test dataset label size: {label_test.shape}")
```

```
train dataset features size: (4457, 8084)
train dataset label size: (4457,)
test dataset features size: (1115, 8084)
test dataset label size: (1115,)
```

SPAM Detector



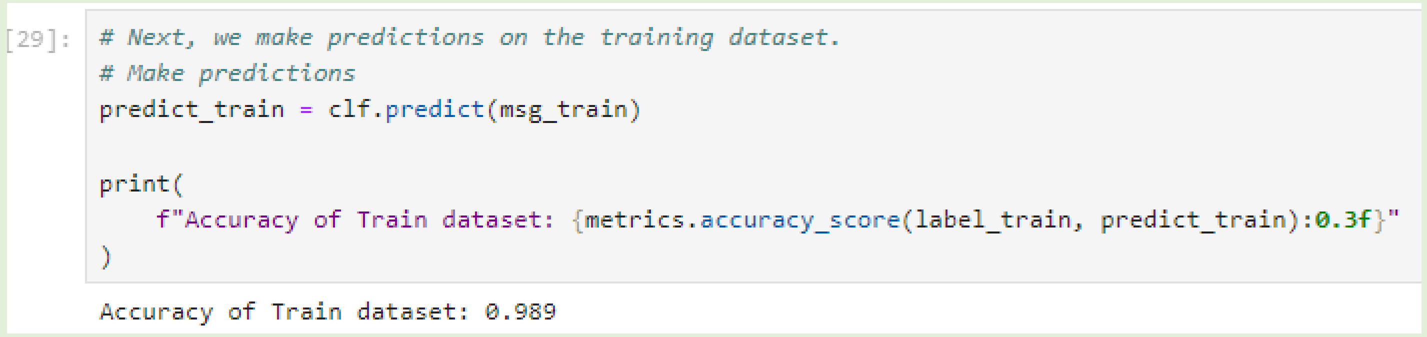
To train the model, we first install the XGBoost library.

```
[27]: # Install xgboost library
      ! pip install xgboost

Requirement already satisfied: xgboost in c:\users\vista\anaconda3\lib\site-packages (2.1.0)
Requirement already satisfied: numpy in c:\users\vista\anaconda3\lib\site-packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\users\vista\anaconda3\lib\site-packages (from xgboost) (1.11.4)
```

```
[28]: #We train the classifier.
      # Train an xgboost classifier
      from xgboost import XGBClassifier

      # Instantiate our model
      clf = XGBClassifier()

      # Fit the model to the training data
      clf.fit(msg_train, label_train)
```

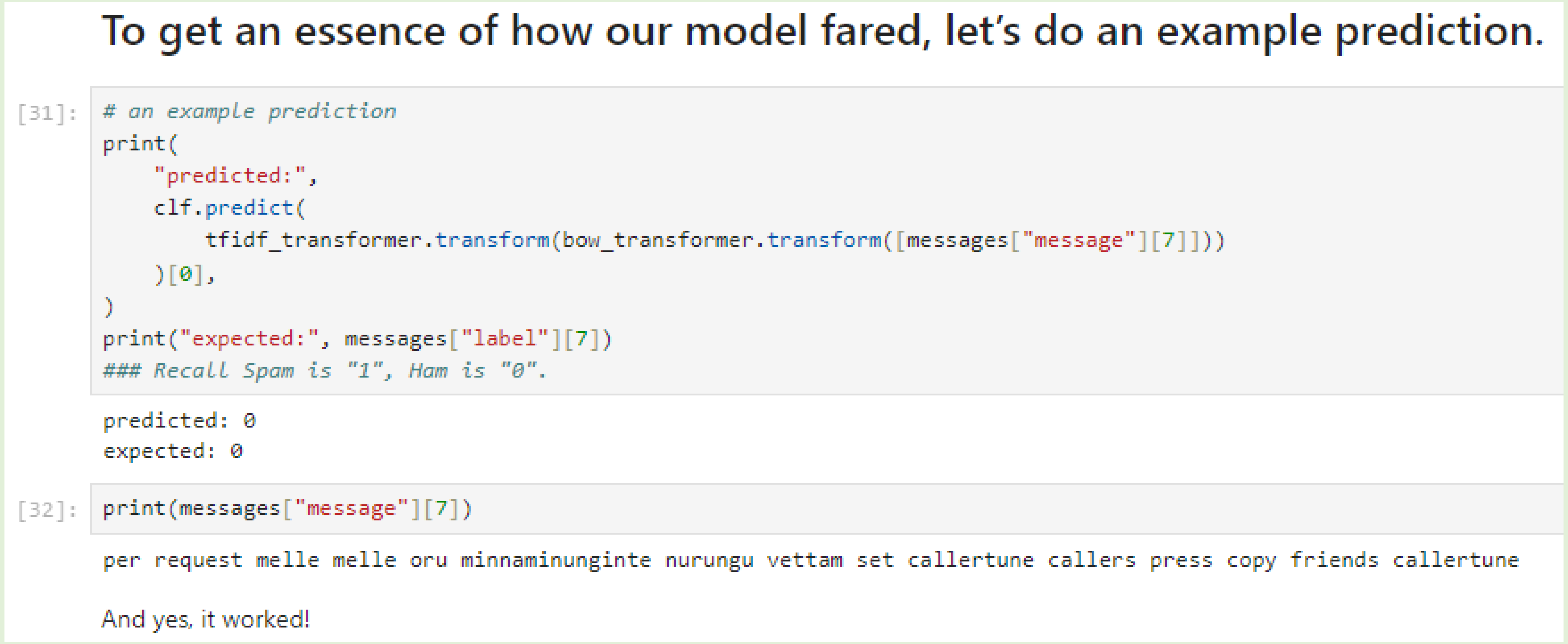
```
[28]:                          XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

SPAM Detector



```
[29]: # Next, we make predictions on the training dataset.
      # Make predictions
      predict_train = clf.predict(msg_train)

      print(
          f"Accuracy of Train dataset: {metrics.accuracy_score(label_train, predict_train):0.3f}"
      )
```

```
Accuracy of Train dataset: 0.989
```



To get an essence of how our model fared, let’s do an example prediction.

```
[31]: # an example prediction
      print(
          "predicted:",
          clf.predict(
              tfidf_transformer.transform(bow_transformer.transform([messages["message"][7]]))
          )[0],
      )
      print("expected:", messages["label"][7])
      ### Recall Spam is "1", Ham is "0".
```

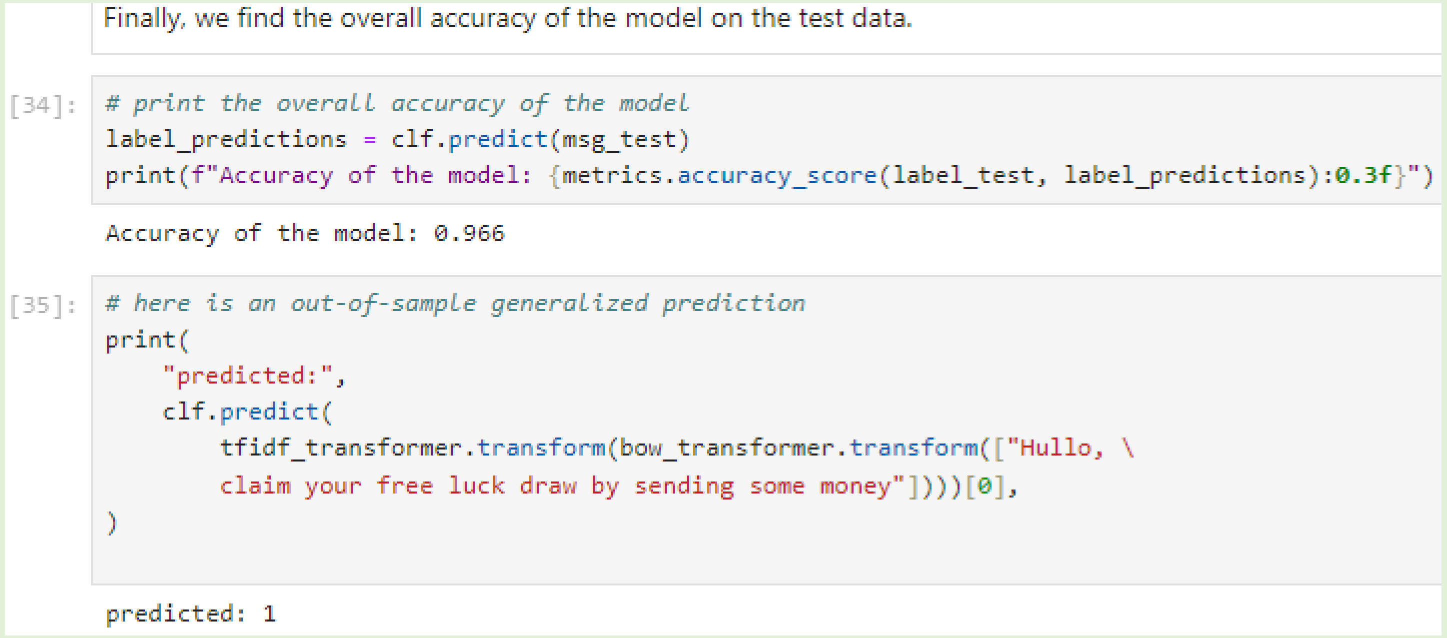
```
predicted: 0
expected: 0
```

```
[32]: print(messages["message"][7])
```

```
per request melle melle oru minnaminunginte nurungu vettam set callertune callers press copy friends callertune
```

And yes, it worked!

SPAM Detector



Finally, we find the overall accuracy of the model on the test data.

```
[34]: # print the overall accuracy of the model
      label_predictions = clf.predict(msg_test)
      print(f"Accuracy of the model: {metrics.accuracy_score(label_test, label_predictions):0.3f}")
```

```
Accuracy of the model: 0.966
```

```
[35]: # here is an out-of-sample generalized prediction
      print(
          "predicted:",
          clf.predict(
              tfidf_transformer.transform(bow_transformer.transform(["Hullo, \
              claim your free luck draw by sending some money"])))[0],
      )
```

```
predicted: 1
```

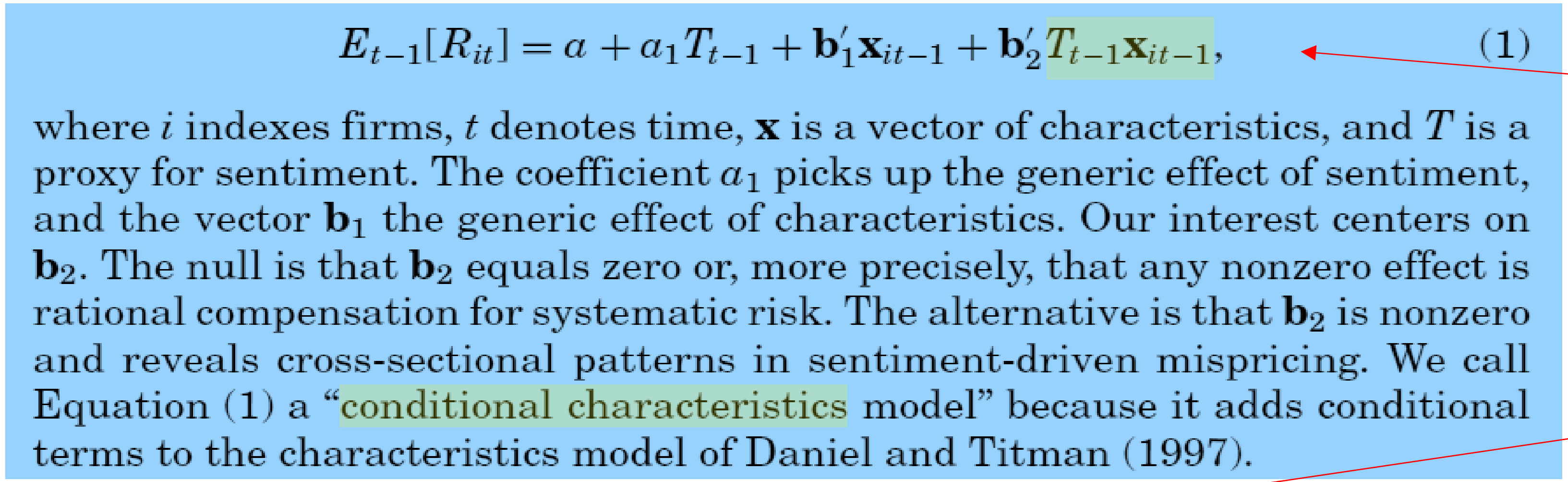
Sentiment Analysis

Ref: “Investor Sentiment and the Cross-Section of Stock Returns” by M. Baker and J. wurgler, Journal of Finance, 2006, No.4

- Studied how investor sentiment can affect the cross-section of stock returns. Essentially, sentiment is shown as another (interactive) factor affecting stock returns, although more significantly on the less transparent stocks.
- Showed investor sentiment has larger effects on securities whose valuations are less transparent (more difficult to value), viz., small stocks, young stocks, high volatility stocks, unprofitable stocks, non-dividend-paying stocks, extreme growth stocks, and distressed stocks.
- When sentiments are low (no good feelings about these stocks), prices of these stocks are low. In the next period when sentiments are better, prices rise – hence the subsequent returns are relatively high. When sentiments are high (good feelings about these stocks), prices of these stocks are high. In the next period when sentiments are lower, prices fall – hence the subsequent returns are relatively low.
- **Forms sentiment index measure T each month** using a composite of proxies such as closed-end fund discount** (average net asset values NAV of closed-end fund shares less their market prices), NYSE share turnover* (share trading volume or liquidity), number of IPOS*, average first-day returns on IPOs*, equity share in new issues* (proportion of equity to long-term debt in firm combined seasoned financing issuances), and dividend premium** (relative demand for stocks with higher dividend payouts). [* means higher sentiment if value is higher, ** means lower sentiment if value if higher]
- An open-end fund issues new shares if buyers want them and takes shares out of circulation if holders sell them. Trading is not in exchange. Fund buys and sells based on the funds’ NAV calculated the day before. If a large number of shares are redeemed, the fund may sell some of its investments to pay the selling investors. Closed-end fund, unlike open-end fund, does not issue any new shares. Shares are traded in the market (exchange) and the traded price can fall below NAV if investor feels the fund’s asset pool is going to perform badly in the future.

Sentiment Analysis

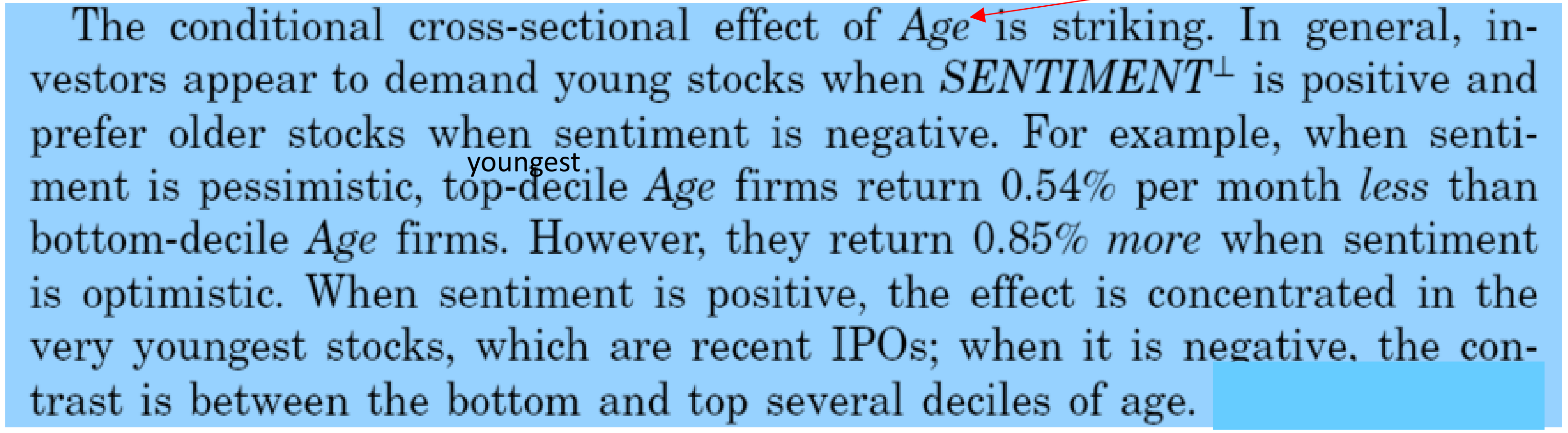
Ref: “Investor Sentiment and the Cross-Section of Stock Returns” by M. Baker and J. wurgler, Journal of Finance, 2006, No.4



$$E_{t-1}[R_{it}] = a + a_1 T_{t-1} + \mathbf{b}_1'\mathbf{x}_{it-1} + \mathbf{b}_2' T_{t-1}\mathbf{x}_{it-1}, \tag{1}$$

where i indexes firms, t denotes time, $\mathbf{x}$ is a vector of characteristics, and T is a proxy for sentiment. The coefficient a_1 picks up the generic effect of sentiment, and the vector $\mathbf{b}_1$ the generic effect of characteristics. Our interest centers on $\mathbf{b}_2$. The null is that $\mathbf{b}_2$ equals zero or, more precisely, that any nonzero effect is rational compensation for systematic risk. The alternative is that $\mathbf{b}_2$ is nonzero and reveals cross-sectional patterns in sentiment-driven mispricing. We call Equation (1) a “conditional characteristics model” because it adds conditional terms to the characteristics model of Daniel and Titman (1997).

Cross-sectional regression of R_{it} on lagged firm characteristics /features. Age is also a feature in x_{it-1}. $b_2 \neq 0$



The conditional cross-sectional effect of *Age* is striking. In general, investors appear to demand young stocks when *SENTIMENT*$^{\perp}$ is positive and prefer older stocks when sentiment is negative. For example, when sentiment is pessimistic, youngest top-decile *Age* firms return 0.54% per month *less* than bottom-decile *Age* firms. However, they return 0.85% *more* when sentiment is optimistic. When sentiment is positive, the effect is concentrated in the very youngest stocks, which are recent IPOs; when it is negative, the contrast is between the bottom and top several deciles of age.

Further, predicted $E_{t-1}[R_{it}]$ of R_{it} in (1) are sorted into bins by each decile of x_{it-1} (for every feature) and each binary category of positive and negative sentiment measures. Sorting is regardless of t. In each bin, the future returns are averaged.

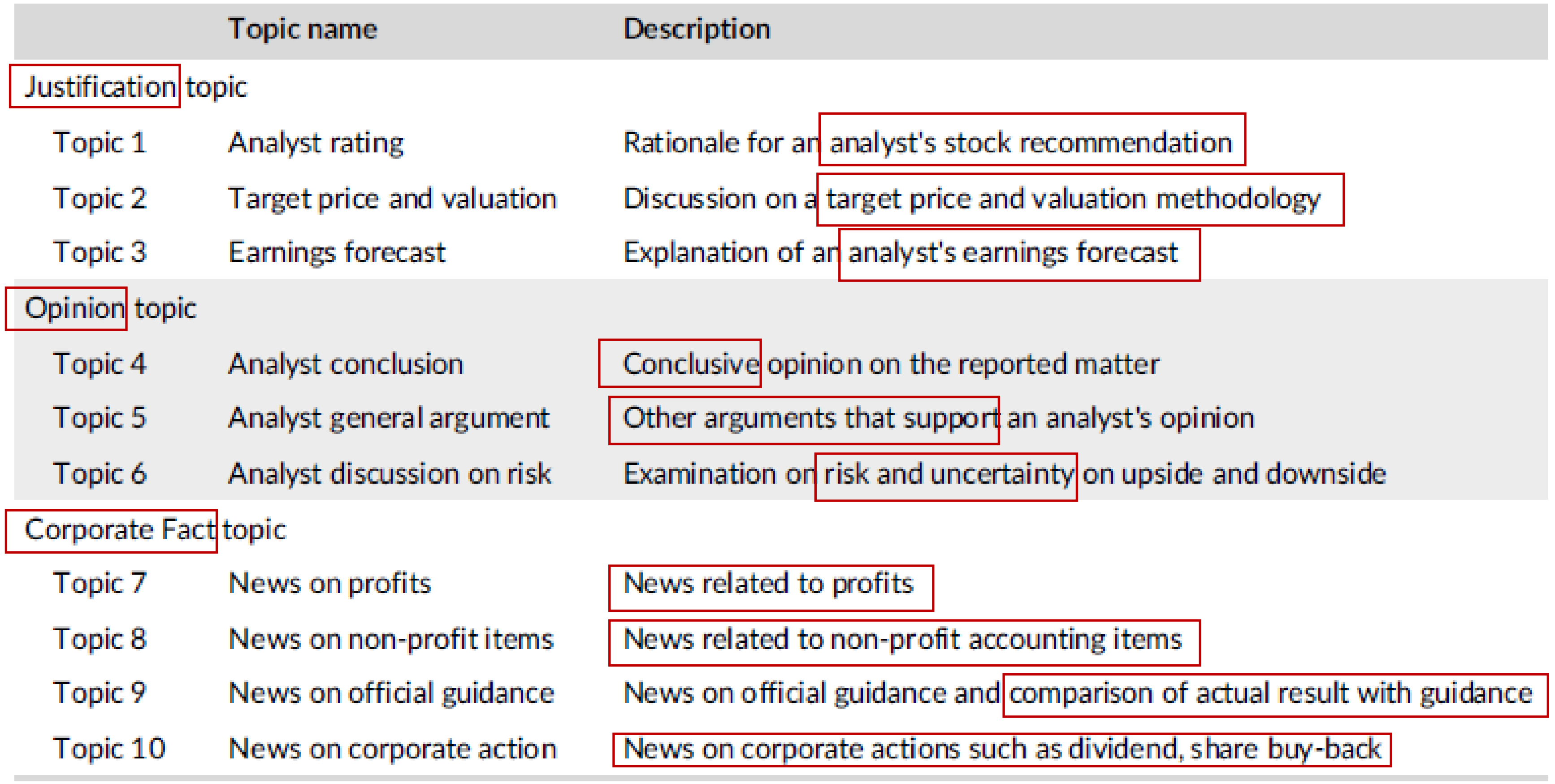
Sentiment Analysis

Ref: “Topic tones of analyst reports and stock returns: A deep learning approach”, by H. Iwasaki, Ying Chen, and Jun Tu, International Review of Finance, June 2023.

- Finance literature showed existence of sentiment effect in stock pricing. Sentiments are expressed in texts or transcribed videos and audios, such as news, social media platforms (Facebook, Youtube, Whatsapp, Instagram, WeChat, TikTok, Douyin, Telegram, Snapchat, etc.) internet and micro-blogging that sits on other popular social media platforms such as Twitter, Reddit, Threads. The process and technology to convert these text sentiments into nominal or else ratio scale number representations for input into ML algorithm to compute is called Sentiment Analysis in ML. These can then be used to predict stock trend.
- This study analyzes the topic tone of a unique dataset consisting of 113,043 sell-side Japanese analyst reports (in Japanese) covering 820 unique firms in Japanese stock markets from Jan 2016 to June 2018. Topic tone $\equiv$ Sentiment.
- An analyst report usually has three main sections: (1) Justification: supportive justifications on the numerical forecasts in the report (2) Opinion: analyst's views on certain issues or events in the firm (3) Corporate Fact: important news and information within the firm.
- Sell-side analysts’ firms include broker-dealers, investment banks, and market makers that give investment services such as stock buy, hold, or sell recommendations to their clients. “sell-side” activities also involve helping in selling new securities, M&A and fund raising/IPO advices. Buy-side analysts’ firms include asset managers, hedge funds, and other companies that trade securities for their clients. They evaluate an investment's potential and whether it aligns with a fund's investment strategy

Sentiment Analysis

- Paper expands 3 sections into 10 detailed topic classifications.

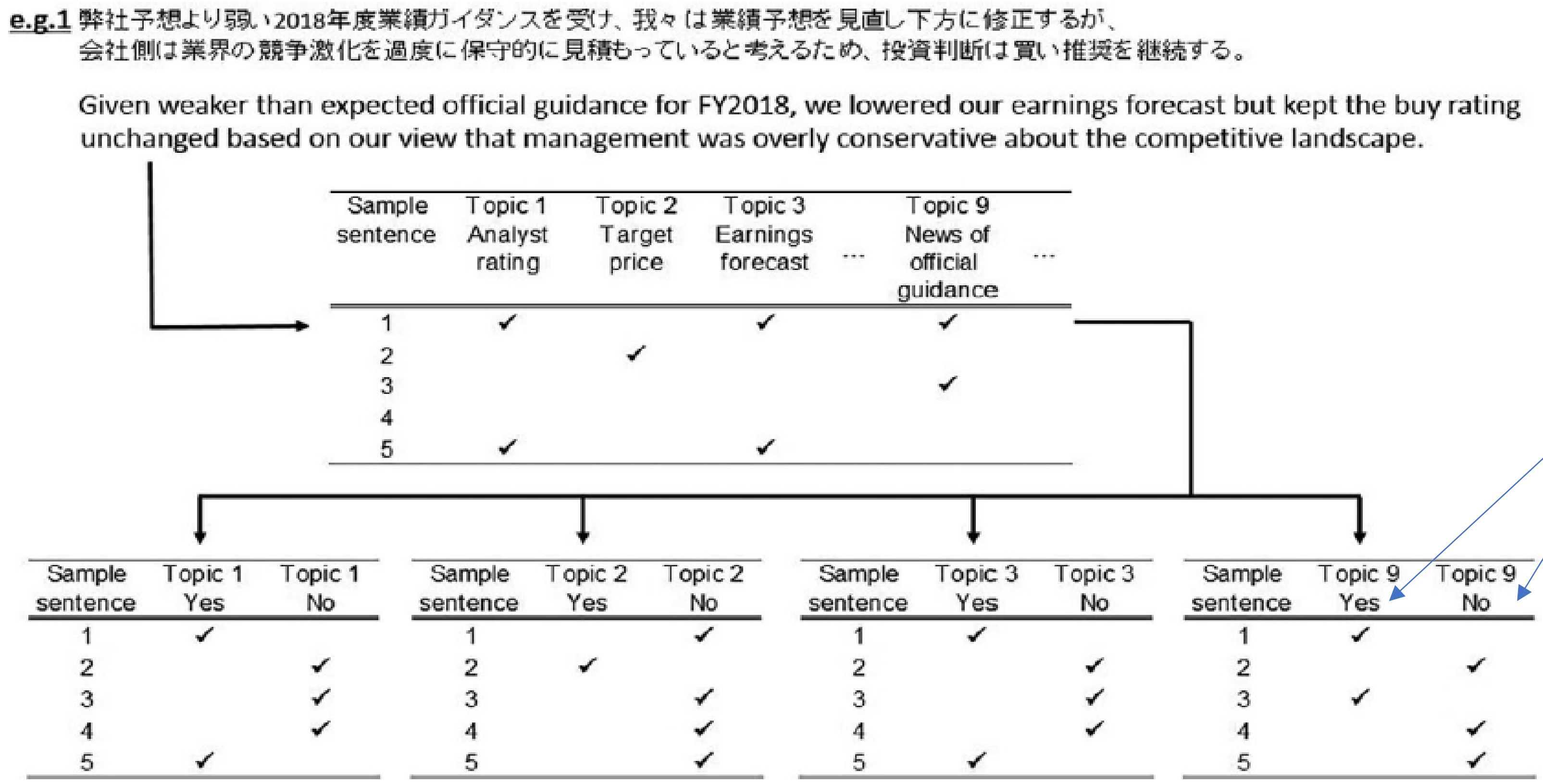


	Topic name	Description
Justification topic		
Topic 1	Analyst rating	Rationale for an analyst's stock recommendation
Topic 2	Target price and valuation	Discussion on a target price and valuation methodology
Topic 3	Earnings forecast	Explanation of an analyst's earnings forecast
Opinion topic		
Topic 4	Analyst conclusion	Conclusive opinion on the reported matter
Topic 5	Analyst general argument	Other arguments that support an analyst's opinion
Topic 6	Analyst discussion on risk	Examination on risk and uncertainty on upside and downside
Corporate Fact topic		
Topic 7	News on profits	News related to profits
Topic 8	News on non-profit items	News related to non-profit accounting items
Topic 9	News on official guidance	News on official guidance and comparison of actual result with guidance
Topic 10	News on corporate action	News on corporate actions such as dividend, share buy-back

Sentiment Analysis

Creating Topic-Tones

- In each sentence of any one report, one or more of the topics could be identified as being associated with that sentence. As an example, see below of a sentence [1] translated from the original Japanese sentence. It is associated with topics 1,3, and 9. **To classify tone, we assign one label out of three: positive, neutral, or negative.** Using the annotated corpus, we train 10 binary classifiers as topic models and one trinary classifier as a tone model **for every sentence in every report**.



e.g.1 弊社予想より弱い2018年度業績ガイダンスを受け、我々は業績予想を見直し下方に修正するが、
会社側は業界の競争激化を過度に保守的に見積もっていると考えるため、投資判断は買い推奨を継続する。

Given weaker than expected official guidance for FY2018, we lowered our earnings forecast but kept the buy rating unchanged based on our view that management was overly conservative about the competitive landscape.

Sample sentence	Topic 1 Analyst rating	Topic 2 Target price	Topic 3 Earnings forecast	···	Topic 9 News of official guidance	···
1	✓		✓		✓	
2		✓				
3					✓	
4						
5	✓		✓			

Sample sentence	Topic 1 Yes	Topic 1 No
1	✓	
2		✓
3		✓
4		✓
5	✓	

Sample sentence	Topic 2 Yes	Topic 2 No
1		✓
2	✓	
3		✓
4		✓
5		✓

Sample sentence	Topic 3 Yes	Topic 3 No
1	✓	
2		✓
3		✓
4		✓
5	✓	

Sample sentence	Topic 9 Yes	Topic 9 No
1	✓	
2		✓
3	✓	
4		✓
5		✓

Each sentence is processed by 10 different binary topic classifiers individually, as illustrated in Figure.

Sentiment Analysis

Creating Topic-Tones

- Paper constructs topic-tones using DNN (deep neural network) supervised classifications. The process commences with creating a training corpus that is used to train the classifiers. Initially, a random selection of 3000 sentences is chosen from the training data set. Then, each sentence is manually assigned **topic class(es)** and **a tone label**, by a Japanese native author who has more than 10 years of industry experience, with double-checking performed by his research analyst.
- In the annotation process, multi-label topic assignment in which a sentence can have more than one topic is allowed.
- Because of the subtle (nuanced) shades in meanings of the Japanese language, use of unsupervised LDA is likely less accurate in the topic modeling than by native human industry experts.
- There are 11 labeled training corpus each comprising 3000 sentences. One training corpus with the manually annotated tone labels together with the sentences is used by a DNN to train the NN to classify tone (positive, neutral, or negative) given a new sentence. It can do so for example using bag-of-words representation for each word in the sentence as a feature in the prediction.
- The other 10 training corpus each has a manually annotated topic k label (binary "Yes" or "No" for the topic k) together with the sentences that is used by the DNN to train the NN to classify topic ("Yes" or "No") given a new sentence. It can do so for example using bag-of-words representation for each word in the sentence as a feature in the prediction.
- The prediction is then extended to all the training set data reports and their report level sentences apart from the 3000 in the training corpus. In this way, all reports and their sentences become labelled with both topics and tone.

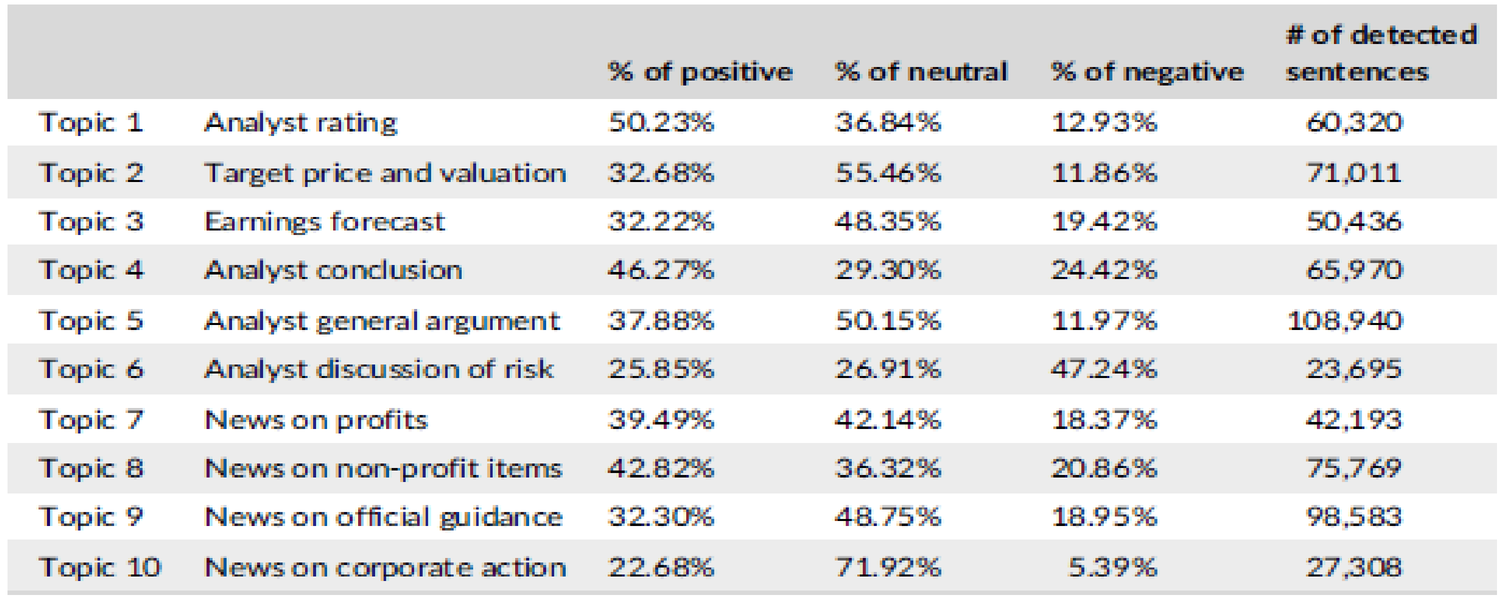
Sentiment Analysis

- After the sentence-level topics and tones are evaluated, we aggregate them to form a report-level variable. For topic i =1,2,…,10, we calculate the topic tone at report level:

$$TT_i = Tone_i^{pos} - Tone_i^{neg}$$

where $Tone_i^{pos}$ ($Tone_i^{neg}$) is computed as the percentage of positive (negative) sentences within a specific topic in a report. Each report is associated with one stock, so each report level variable can be used as feature to predict/explain the stock's future returns.

TABLE 4 Distribution of topic tones.



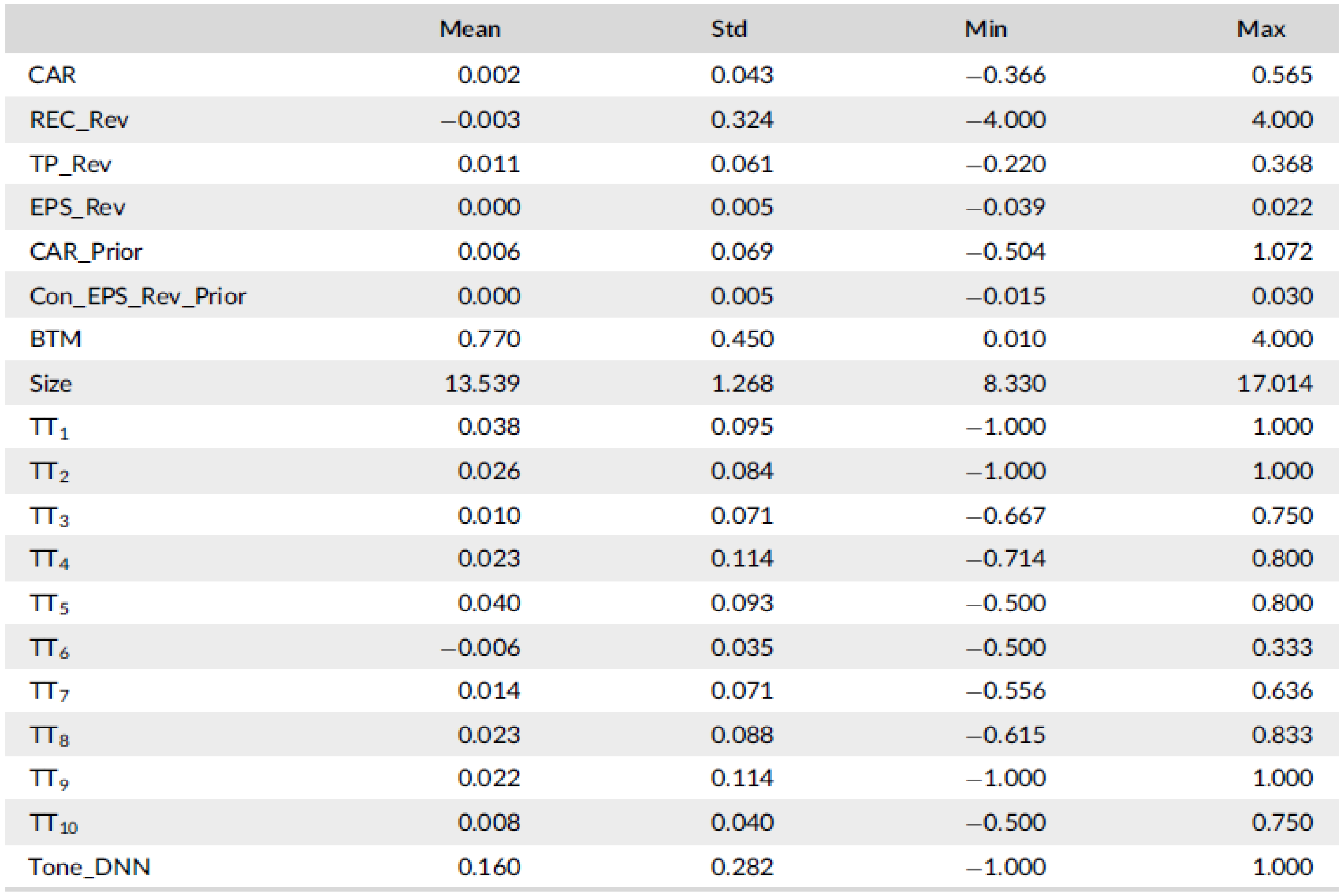
		% of positive	% of neutral	% of negative	# of detected sentences
Topic 1	Analyst rating	50.23%	36.84%	12.93%	60,320
Topic 2	Target price and valuation	32.68%	55.46%	11.86%	71,011
Topic 3	Earnings forecast	32.22%	48.35%	19.42%	50,436
Topic 4	Analyst conclusion	46.27%	29.30%	24.42%	65,970
Topic 5	Analyst general argument	37.88%	50.15%	11.97%	108,940
Topic 6	Analyst discussion of risk	25.85%	26.91%	47.24%	23,695
Topic 7	News on profits	39.49%	42.14%	18.37%	42,193
Topic 8	News on non-profit items	42.82%	36.32%	20.86%	75,769
Topic 9	News on official guidance	32.30%	48.75%	18.95%	98,583
Topic 10	News on corporate action	22.68%	71.92%	5.39%	27,308

Note: This table displays a distribution of topic tones. Depending on topics, the tone distributions can be skewed (e.g., positive skew in Topic 1, negative skew in Topic 6).

This is at aggregate level across all reports in the corpus.

We provide the summary statistics of the target CAR, control, and topic tone variables TT_i in Table 5. The mean, std, etc. are taken across all reports in the corpus. Each report produces one TT_1, TT_2, etc.

TABLE 5 Summary statistics of variables.



	Mean	Std	Min	Max
CAR	0.002	0.043	−0.366	0.565
REC_Rev	−0.003	0.324	−4.000	4.000
TP_Rev	0.011	0.061	−0.220	0.368
EPS_Rev	0.000	0.005	−0.039	0.022
CAR_Prior	0.006	0.069	−0.504	1.072
Con_EPS_Rev_Prior	0.000	0.005	−0.015	0.030
BTM	0.770	0.450	0.010	4.000
Size	13.539	1.268	8.330	17.014
TT_1	0.038	0.095	−1.000	1.000
TT_2	0.026	0.084	−1.000	1.000
TT_3	0.010	0.071	−0.667	0.750
TT_4	0.023	0.114	−0.714	0.800
TT_5	0.040	0.093	−0.500	0.800
TT_6	−0.006	0.035	−0.500	0.333
TT_7	0.014	0.071	−0.556	0.636
TT_8	0.023	0.088	−0.615	0.833
TT_9	0.022	0.114	−1.000	1.000
TT_{10}	0.008	0.040	−0.500	0.750
Tone_DNN	0.160	0.282	−1.000	1.000

Note: This table presents summary statistics for the variables in the training data set. CAR refers to the cumulative two-day abnormal returns starting from the analyst report issue date, where the abnormal returns are calculated as the difference between the raw stock return and the benchmark index return (i.e., TOPIX return).

REC_Rev is the analyst recommendation revision. TP_Rev represents the target price revision, and EPS_Rev denotes the earnings forecast revision. CAR_Prior is the cumulative 20-day abnormal return before the analyst report publication date. Con_EPS_Rev_Prior measures IBES consensus EPS forecast (mean) revisions that are made within 20 days windows prior to the analyst report publication date. BTM stands for book-to-market. Size is the logarithm of the firm's market capitalization.

Tone_DNN is the overall tone, whereas TTi is the topic tone for a specific topic i. All the topic tones are standardized to have mean zero and unit variance.

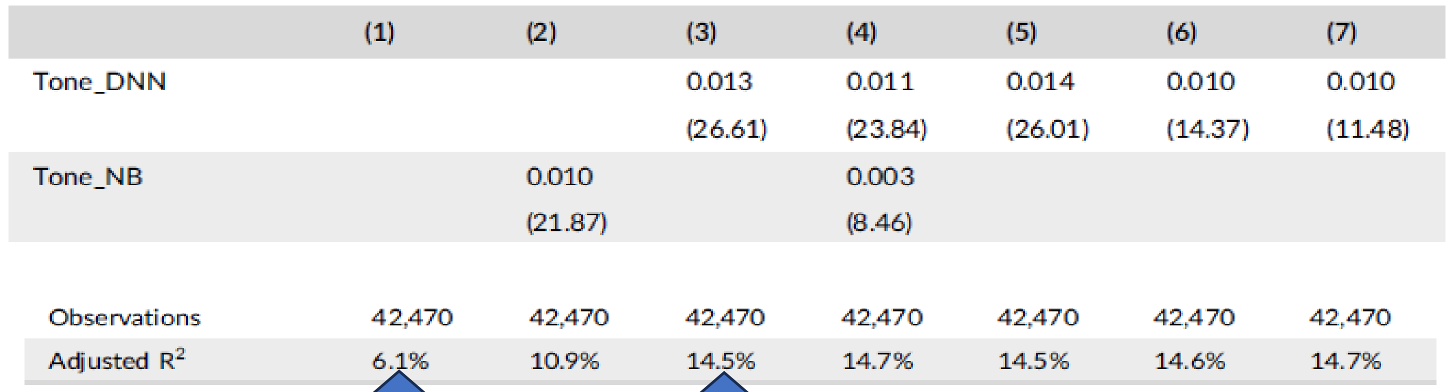
Informativeness of overall DNN tone

We may treat this as cross-sectional across firm-reports although there are also reports across 2 years

$$CAR = \alpha_0 + \beta_1 Tone\_DNN + \beta_2 Tone\_NB + controls + \epsilon$$

CAR refers to the cumulative two-day abnormal returns starting from the analyst report issue date and the abnormal returns are calculated as the difference between the raw individual stock return and the benchmark index return (i.e., TOPIX return). Tone_DNN is the percentage of positive sentences minus the percentage of negative sentences in a report without considering topics, and it serves as the overall topic measure of an analyst report. Tone_NB is constructed in the same labeling procedure but based on Naïve Bayes prediction. Note that this overall tone measure, Tone_DNN, is closely related to, but not identical to the sum of topic tones, TT1,…,TT10, due to the multi-label topic assignment, in which one sentence can have more than one topic.

TABLE 6 Informativeness of the overall DNN tone.



	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Tone_DNN			0.013	0.011	0.014	0.010	0.010
			(26.61)	(23.84)	(26.01)	(14.37)	(11.48)
Tone_NB		0.010		0.003			
		(21.87)		(8.46)			
Observations	42,470	42,470	42,470	42,470	42,470	42,470	42,470
Adjusted R^2	6.1%	10.9%	14.5%	14.7%	14.5%	14.6%	14.7%

Only control variables used

Tone_DNN + control variables used

Informativeness of topic tones

$$CAR = \alpha_0 + \sum_i \beta_i DetailedTone_i + controls + \epsilon$$

DetailedTone stands for TT or other categories of tone, such as Tone_Justif, Tone_Quali, Tone_Opn or Tone_Fact. Tone_Justif (Tone_Quali) is the aggregate topic tone from Topic 1 to Topic 3 (Topic 4 to Topic 10), while Tone_Opn (Tone_Fact) is aggregate topic tone from Topic 4 to Topic 6 (Topic 7 to Topic 10). All the topic tones are standardized to have mean zero and unit variance for the convenience of comparing the magnitude of the estimated loading.

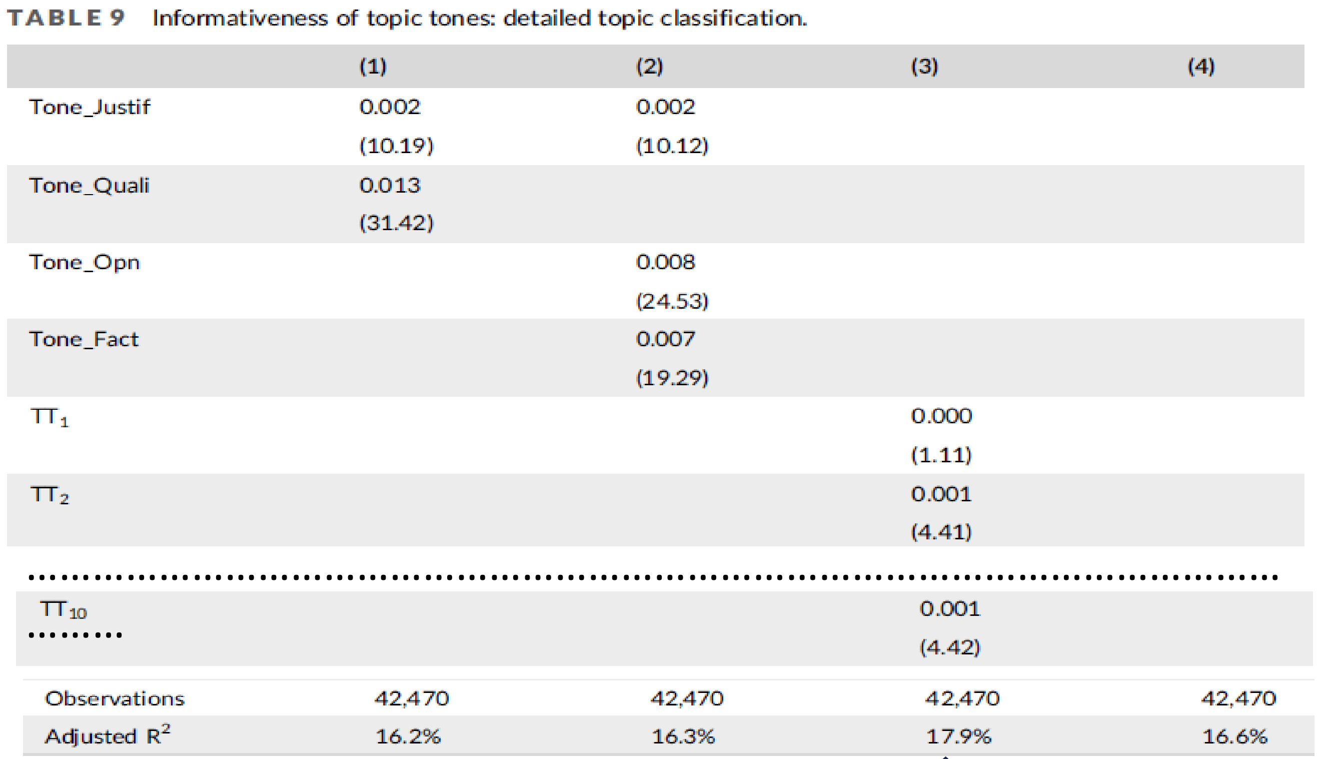


TABLE 9 Informativeness of topic tones: detailed topic classification.

	(1)	(2)	(3)	(4)
Tone_Justif	0.002	0.002		
	(10.19)	(10.12)		
Tone_Quali	0.013			
	(31.42)			
Tone_Opn		0.008		
		(24.53)		
Tone_Fact		0.007		
		(19.29)		
TT_1			0.000	
			(1.11)	
TT_2			0.001	
			(4.41)	
....				
TT_{10}			0.001	
....			(4.42)	
Observations	42,470	42,470	42,470	42,470
Adjusted R^2	16.2%	16.3%	17.9%	16.6%

Detailed Tone_Topics + control variables used

Sentiment Analysis

- The empirical study demonstrates the benefits of utilizing deep neural network (DNN) models to extract topic-tones for predicting cumulative two-day abnormal returns surrounding analyst report issue dates.
- Specifically, in the training dataset, this proposed approach yields a significant improvement in R^2 from 6.1% (baseline model without using any information in the analyst report text) to 14.5% (including an overall aggregate DNN tone) and further to 17.9% (including detailed topic tones).
- They find that the tone of Justification topics is less informative, while Opinion and Corporate Fact topics carry more relevant information. The latter have higher regression impact on CAR.
- The research adds to the literature on the information role of analysts by introducing supervised topic-tone analysis. This approach overcomes LDA's shortcoming (Huang et al., 2018; Lowry et al., 2020) that requires researchers to rely on frequent words for topic interpretation in the absence of explicit labels.
- While previous research has mostly focused on the overall textual tone (e.g., Huang et al., 2014; Li, 2010; Tetlock et al., 2008), the paper's approach of finer topic classifications provides a deeper understanding of textual information and can enable investors to identify important aspects of an analyst report more quickly.

Prediction-based instead of just frequency-based Word Embeddings to Improve Textual Analyses

- We have seen Frequency-based word embeddings such as BoW, TF-IDF, etc.
- Prediction-based Word embeddings are newer ways of representing words as dense vectors (most dimensions carry non-zero values) in a multi-dimensional space, where closer distance and direction between vectors reflect relationship among the corresponding words and **semantic meaning**.
- Word embedding vectors are trained typically by neural networks using a large amount of training text data and adjusting the vector representations depending on the **context** in which words appear.
- One popular method for training word embeddings is Word2Vec, which uses a neural network to predict the surrounding words of a target word in a given **context**. This and other semantic advances form methods and dictionaries for deeper AI in LLM.

End of Class