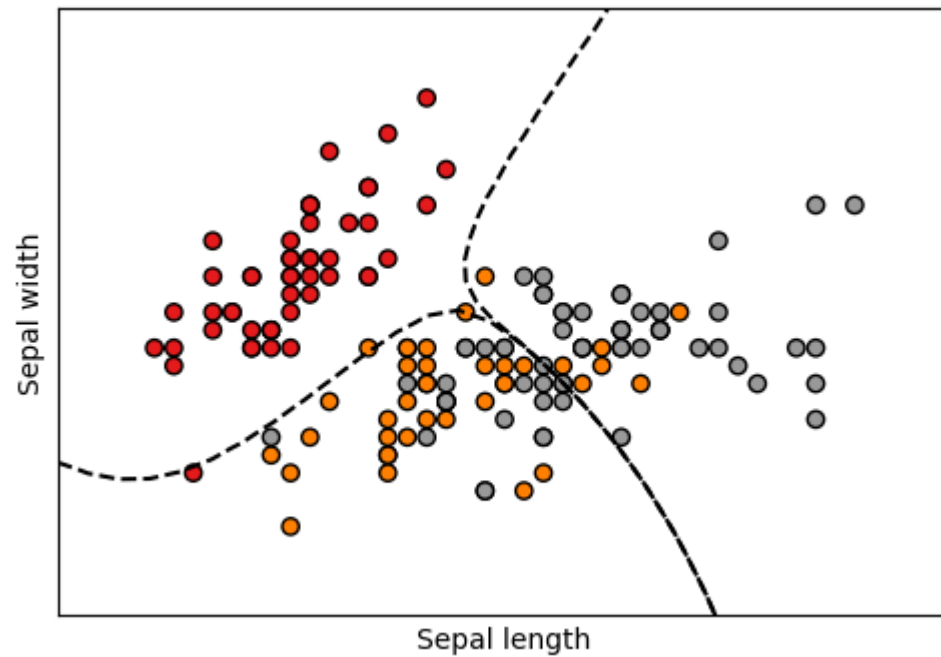


Naive Bayes Classifier



Naïve Bayes

- ▶ What is Naive Bayes?
- ▶ Naive Bayes and Machine Learning
- ▶ Why do we need Naive Bayes?
- ▶ Understanding Naive Bayes Classifier
- ▶ Advantages of Naive Bayes Classifier
- ▶ Demo - Text Classification using Naive Bayes

Introducing Bayes Theorem



Here, the sample space is:

$\{HH, HT, TH, TT\}$

1. $P(\text{Getting two heads}) = 1/4$
2. $P(\text{At least one tail}) = 3/4$
3. $P(\text{Second coin being head given first coin is tail}) = 1/2$
4. $P(\text{Getting two heads given first coin is a head}) = 1/2$

Bayes Rule

This procedure is based on *Bayes Rule*, which says: if you have a hypothesis h and data D which bears on the hypothesis, then:

$$(1) \quad P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

$P(h)$: independent probability of h : *prior probability*

$P(D)$: independent probability of D

$P(D|h)$: conditional probability of D given h : *likelihood*

$P(h|D)$: cond. probability of h given D : *posterior probability*

A diagram illustrating Bayes' Theorem. The equation is presented in a large, bold, dark blue font. Four green labels with leader lines point to specific parts of the equation: 'Posterior Probability' points to $P(A | B)$, 'Likelihood' points to $P(B | A)$, 'Prior Probability' points to $P(A)$, and 'Evidence' points to $P(B)$. The background is a light gray with a faint, stylized image of a magnifying glass.

$$P(A | B) = \frac{P(B | A) P(A)}{P(B)}$$

Posterior Probability

Likelihood

Prior Probability

Evidence

Applying Bayes Theorem



Here, the sample space is:

{HH, HT, TH, TT}

1. $P(\text{Getting two heads}) = 1/4$
2. $P(\text{Atleast one tail}) = 3/4$

3. $P(\text{Second coin being head given first coin is tail}) = 1/2$
4. $P(\text{Getting two heads given first coin is a head}) = 1/2$

THESE TWO USE SIMPLE
PROBABILITIES CALCULATED DIRECTLY
FROM THE SAMPLE SPACE

Applying Bayes Theorem

IN THIS SAMPLE SPACE, LET **A** BE THE
EVENT THAT SECOND COIN IS HEAD
AND **B** BE THE EVENT THAT FIRST COIN
IS TAIL



In the sample space:

{HH, HT, TH, TT}

$P(\text{Second coin being head given first coin is tail})$

$$= P(A|B)$$

$$= [P(B|A) * P(A)] / P(B)$$

$$= [P(\text{First coin being tail given second coin is head}) * P(\text{Second coin being head})] / P(\text{First coin being tail})$$

$$= [(1/2) * (1/2)] / (1/2)$$

$$= 1/2 = 0.5$$

BT calculates the
conditional probability of
the occurrence of an event
based on prior knowledge
of conditions that might be
related to the event

Where is Naïve Bayes Used?

Face
Recognition



Weather
Prediction



Medical
Diagnosis



News
Classification



Naïve Bayes Classifier

Naive Bayes Classifier is based on Bayes' Theorem which gives the conditional probability of an event A given B

Bayes Theorem

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)}$$

where:

$P(A|B)$ = Conditional Probability of A given B

$P(B|A)$ = Conditional Probability of B given A

$P(A)$ = Probability of event A

$P(B)$ = Probability of event B

Shopping Example

To predict whether a person will purchase a product on a specific combination of Day, Discount and Free Delivery using Naive Bayes Classifier



Shopping Dataset

We have a small sample dataset of 30 rows for our demo

	A	B	C	D
1	Day	Discount	Free Delivery	Purchase
2	Weekday	Yes	Yes	Yes
3	Weekday	Yes	Yes	Yes
4	Weekday	No	No	No
5	Holiday	Yes	Yes	Yes
6	Weekend	Yes	Yes	Yes
7	Holiday	No	No	No
8	Weekend	Yes	No	Yes
9	Weekday	Yes	Yes	Yes
10	Weekend	Yes	Yes	Yes
11	Holiday	Yes	Yes	Yes
12	Holiday	No	Yes	Yes
13	Holiday	No	No	No
14	Weekend	Yes	Yes	Yes
15	Holiday	Yes	Yes	Yes

Naive_Bayes_Dataset

Frequency Table

Based on this dataset containing three input types of *Day*, *Discount* and *Free Delivery*, we will populate frequency tables for each attribute

Frequency Table		Buy	
		Yes	No
Discount	Yes	19	1
	No	5	5

Frequency Table		Buy	
		Yes	No
Free Delivery	Yes	21	2
	No	3	4

Frequency Table		Buy	
		Yes	No
Day	Weekday	9	2
	Weekend	7	1
	Holiday	8	3

Frequency Table

Based on the 3 inputs Day, Discount and Free Delivery, populate the frequencies tables for each attribute

The diagram illustrates three frequency tables for attributes Discount, Free Delivery, and Day, each with a 'Buy' column. Dashed blue boxes and arrows indicate the relationship between these attributes and the event 'Buy' (A) and the independent variables (B).

Frequency Table		Buy	
		Yes	No
Discount	Yes	19	1
	No	5	5

Frequency Table		Buy	
		Yes	No
Free Delivery	Yes	21	2
	No	3	4

Frequency Table		Buy	
		Yes	No
Day	Weekday	9	2
	Weekend	7	1
	Holiday	8	3

For the Bayes theorem, let the event Buy be A and the independent variables, Discount, Free Delivery and Day be B

Likelihood Table

Now let us calculate the Likelihood table for one of the variable, *Day* which includes *Weekday*, *Weekend* and *Holiday*

Frequency Table		Buy		
		Yes	No	
Day	Weekday	9	2	11
	Weekend	7	1	8
	Holiday	8	3	11
		24	6	30

Likelihood Table		Buy		
		Yes	No	
Day	Weekday	9/24	2/6	11/30
	Weekend	7/24	1/6	8/30
	Holiday	8/24	3/6	11/30
		24/30	6/30	

$$P(B) = P(\text{Weekday}) \\ = 11/30 = 0.37$$

$$P(A) = P(\text{No Buy}) \\ = 6/30 = 0.2$$

$$P(B|A) \\ = P(\text{Weekday} | \text{No Buy}) \\ = 2/6 = 0.33$$

Prob of weekday is calculated from total sample space ie no of days. This prob can act as sample space for calculating the prob of buying or not buying. 11/30 is the prob of getting a week day.

Likelihood Table

Based on this likelihood table, we will calculate conditional probabilities as below

Frequency Table		Buy		
		Yes	No	
Day	Weekday	9	2	11
	Weekend	7	1	8
	Holiday	8	3	11
		24	6	30

Likelihood Table		Buy		
		Yes	No	
Day	Weekday	9/24	2/6	11/30
	Weekend	7/24	1/6	8/30
	Holiday	8/24	3/6	11/30
		24/30	6/30	

$$P(B) = P(\text{Weekday}) = 11/30 = 0.367$$

$$P(A) = P(\text{Buy}) = 24/30 = 0.8$$

$$P(B|A) = P(\text{Weekday} | \text{Buy}) = 9/24 = 0.375$$

If A equals Buy, then

$$\begin{aligned} P(A|B) &= P(\text{Buy} | \text{Weekday}) \\ &= P(\text{Weekday} | \text{Buy}) * P(\text{Buy}) / P(\text{Weekday}) \\ &= (0.375 * 0.8) / 0.367 = 0.817 \end{aligned}$$

As the Probability(Buy | Weekday) is more than Probability(No Buy | Weekday), we can conclude that a customer will most likely buy the product on a Weekday

Constructing Likelihood Tables From the Frequency Tables

Frequency Table		Buy	
		Yes	No
Day	Weekday	3	7
	Weekend	8	2
	Holiday	9	1

Likelihood Table		Buy		
		Yes	No	
Day	Weekday	9/24	2/6	11/30
	Weekend	7/24	1/6	8/30
	Holiday	8/24	3/6	11/30
		24/30	6/30	

Frequency Table		Buy	
		Yes	No
Discount	Yes	19	1
	No	5	5

Frequency Table		Buy		
		Yes	No	
Discount	Yes	19/24	1/6	20/30
	No	5/24	5/6	10/30
		24/30	6/30	

Frequency Table		Buy	
		Yes	No
Free Delivery	Yes	21	2
	No	3	4

Frequency Table		Buy		
		Yes	No	
Free Delivery	Yes	21/24	2/6	23/30
	No	3/24	4/6	7/30
		24/30	6/30	

Naïve Bayes Classifier

Naive Bayes Classifier is based on Bayes' Theorem which gives the conditional probability of an event A given B

Bayes Theorem

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)}$$

where:

$P(A|B)$ = Conditional Probability of A given B

$P(B|A)$ = Conditional Probability of A given B

$P(A)$ = Probability of event A

$P(B)$ = Probability of event A

No Purchase

Likelihood Tables

Likelihood Table		Buy		
		Yes	No	
Day	Weekday	9/24	2/6	11/30
	Weekend	7/24	1/6	8/30
	Holiday	8/24	3/6	11/30
		24/30	6/30	

Frequency Table		Buy		
		Yes	No	
Discount	Yes	19/24	1/6	20/30
	No	5/24	5/6	10/30
		24/30	6/30	

Frequency Table		Buy		
		Yes	No	
Free Delivery	Yes	21/24	2/6	23/30
	No	3/24	4/6	7/30
		24/30	6/30	

Calculating Conditional Probability of purchase on the following combination of day, discount and free delivery:

Where B equals:

- Day = **Holiday**
- Discount = **Yes**
- Free Delivery = **Yes**

Let A = **No Buy**

$P(A|B) = P(\text{No Buy} \mid \text{Discount} = \text{Yes}, \text{Free Delivery} = \text{Yes}, \text{Day} = \text{Holiday})$

$$= \frac{P(\text{Discount} = \text{Yes} \mid \text{No}) * P(\text{Free Delivery} = \text{Yes} \mid \text{No}) * P(\text{Day} = \text{Holiday} \mid \text{No}) * P(\text{No Buy})}{P(\text{Discount} = \text{Yes}) * P(\text{Free Delivery} = \text{Yes}) * P(\text{Day} = \text{Holiday})}$$

$$= \frac{(1/6) * (2/6) * (3/6) * (6/30)}{(20/30) * (23/30) * (11/30)}$$

$$= 0.178$$

Naïve Bayes

Naïve Bayes Classifier

Likelihood Tables

Likelihood Table		Buy		
		Yes	No	
Day	Weekday	9/24	2/6	11/30
	Weekend	7/24	1/6	8/30
	Holiday	8/24	3/6	11/30
		24/30	6/30	

Frequency Table		Buy		
		Yes	No	
Discount	Yes	19/24	1/6	20/30
	No	5/24	5/6	10/30
		24/30	6/30	

Frequency Table		Buy		
		Yes	No	
Free Delivery	Yes	21/24	2/6	23/30
	No	3/24	4/6	7/30

Calculating Conditional Probability of purchase on the following combination of day, discount and free delivery:

Where B equals:

- Day = **Holiday**
- Discount = **Yes**
- Free Delivery = **Yes**

Let A = **Buy**

$P(A|B) = P(\text{Yes Buy} \mid \text{Discount} = \text{Yes}, \text{Free Delivery} = \text{Yes}, \text{Day} = \text{Holiday})$

$$= \frac{P(\text{Discount} = \text{Yes} \mid \text{Yes}) * P(\text{Free Delivery} = \text{Yes} \mid \text{Yes}) * P(\text{Day} = \text{Holiday} \mid \text{Yes}) * P(\text{Yes Buy})}{P(\text{Discount} = \text{Yes}) * P(\text{Free Delivery} = \text{Yes}) * P(\text{Day} = \text{Holiday})}$$

$$= \frac{(19/24) * (21/24) * (8/24) * (24/30)}{(20/30) * (23/30) * (11/30)}$$

$$= 0.986$$

Naïve
Bayes

Conditional Probabilities: Normalizing

$$\begin{aligned}\text{SUM OF PROBABILITIES} \\ &= 0.986 + 0.178 = 1.164\end{aligned}$$

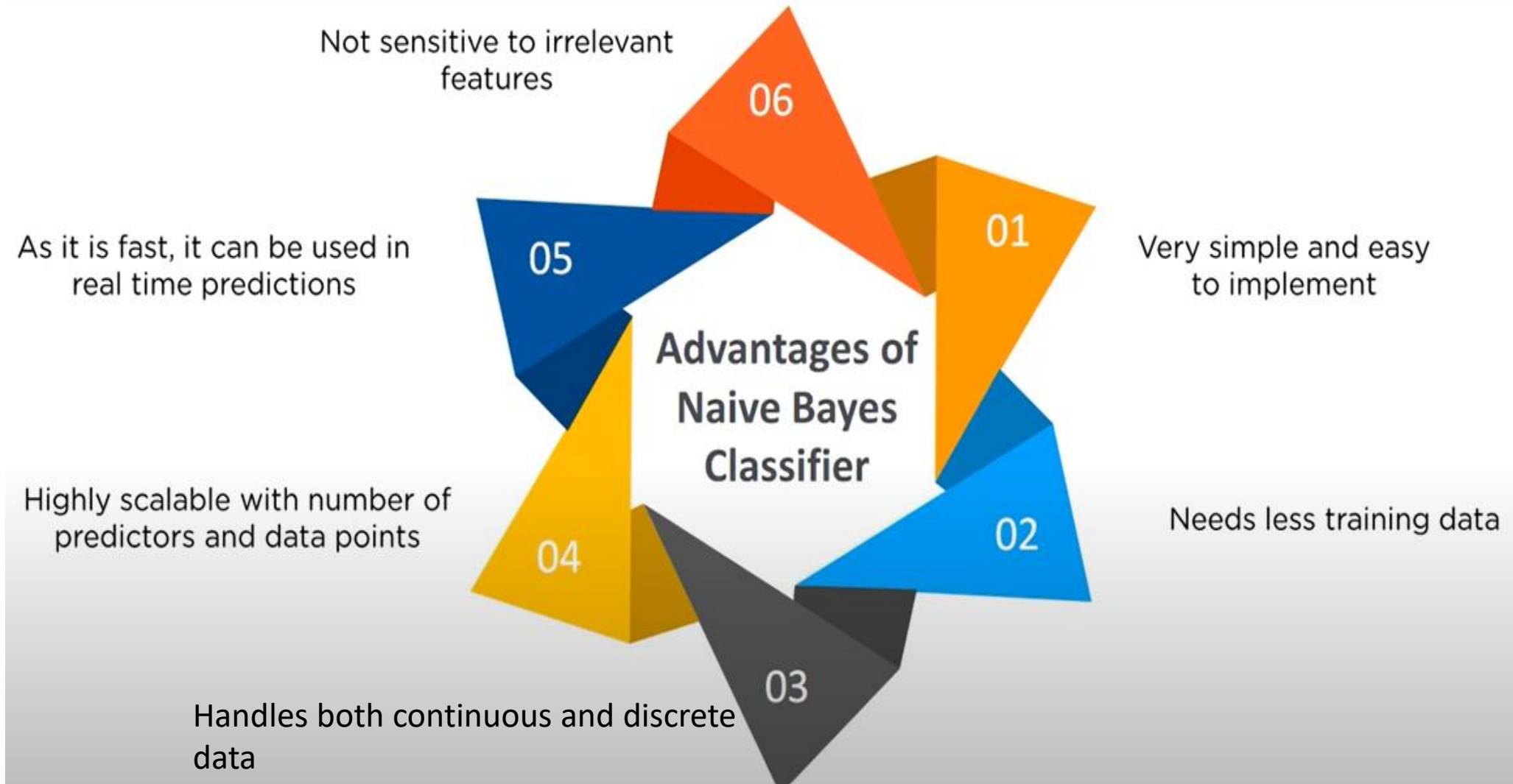
$$\begin{aligned}\text{LIKELIHOOD OF PURCHASE} \\ &= 0.986 / 1.164 = 84.71 \%\end{aligned}$$

$$\begin{aligned}\text{LIKELIHOOD OF NO PURCHASE} \\ &= 0.178 / 1.164 = 15.29 \%\end{aligned}$$

$$\begin{aligned}\text{PROBABILITY OF PURCHASE} &= 0.986 \\ \text{PROBABILITY OF NO PURCHASE} &= 0.178\end{aligned}$$

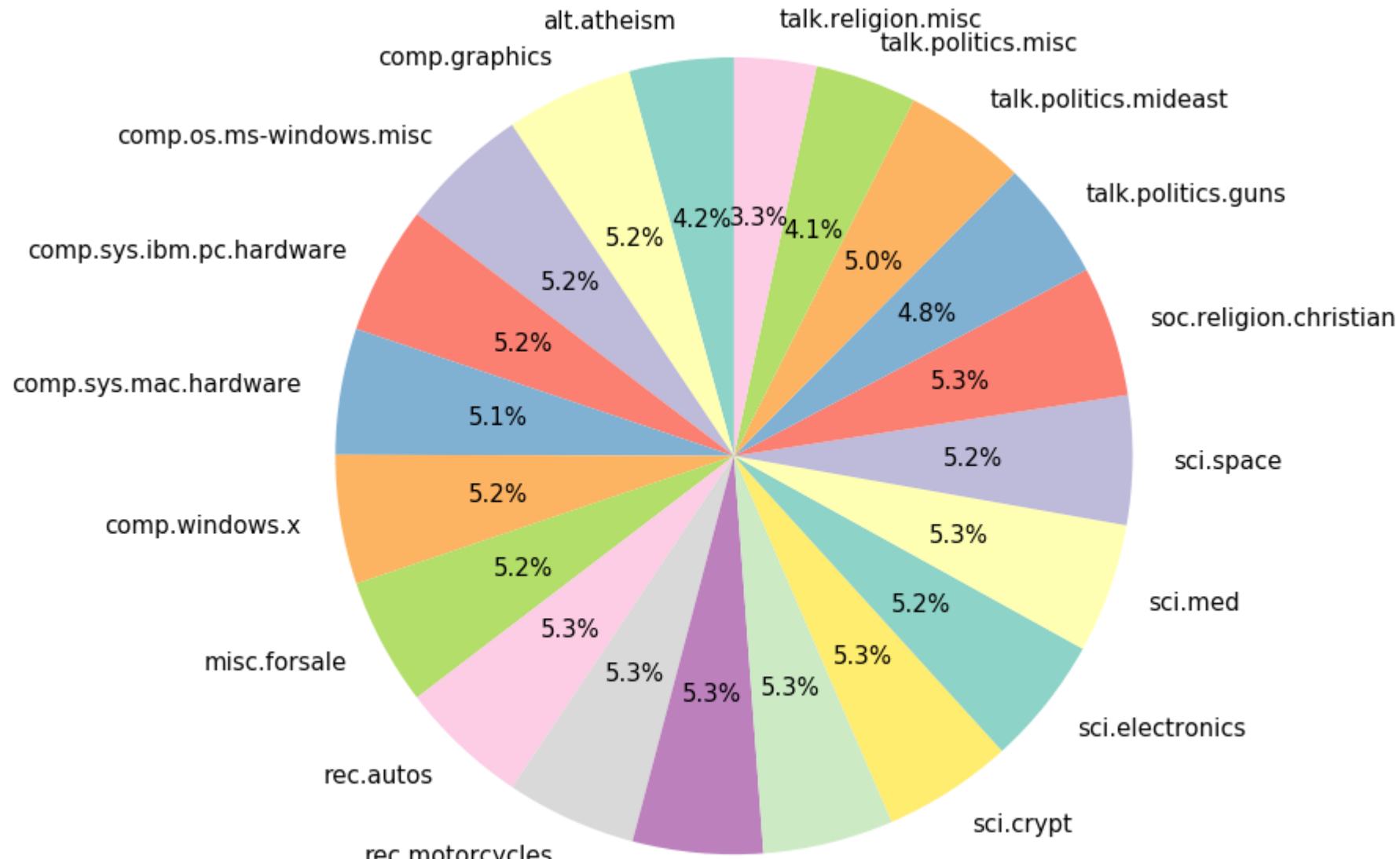
AS 84.71% is greater than 15.29%, we can conclude that an average customer will buy on a holiday with discount and free delivery.

Advantages



“20 Newsgroup” corpus

- available in [scikit-learn](#), get the data with `fetch_20newsgroups()`:



Use Case – Text Classification

To perform text classification of News Headlines and classify news into different topics for a News Website

Google News

SECTIONS

- Top Stories
- India
- World
- Business
- Technology
- Entertainment
- Sport
- Science
- Health

Manage sections

World

Easter 2018: Here's how the Easter Sunday date is determined

The Indian Express · 17m ago

RELATED COVERAGE



Check out Easter festivities at the Wigwam
ABC15 Arizona

MORE ABOUT

Ever wonder why the day Jesus Christ died is called 'Good' Friday?
India Today · 1h ago

Good Friday

Jesus

Good Friday 2018: Christians take out processions, observe fast on day marking crucifixion of Jesus Christ [Photos]

Easter

Firstpost · 1h ago

Easter is the greatest mystery told

In-depth · The Australian · 57m ago

View full coverage →

Related

Gaza Strip

Cambridge
Analytica

Malala Yousafzai

Israel

Pakistan

Hamas

Facebook

Syria

Palestinians

Good Friday

NEWS
GROUPS

NEWS SUB
GROUPS

Use Case – Text Classification

Implementation

- `import numpy as np`
- `import matplotlib.pyplot as plt`
- `import seaborn as sns`
- `from sklearn.datasets import fetch_20newsgroups`
- `data=fetch_20newsgroups()`
- `data.target_names`

Use Case – Text Classification

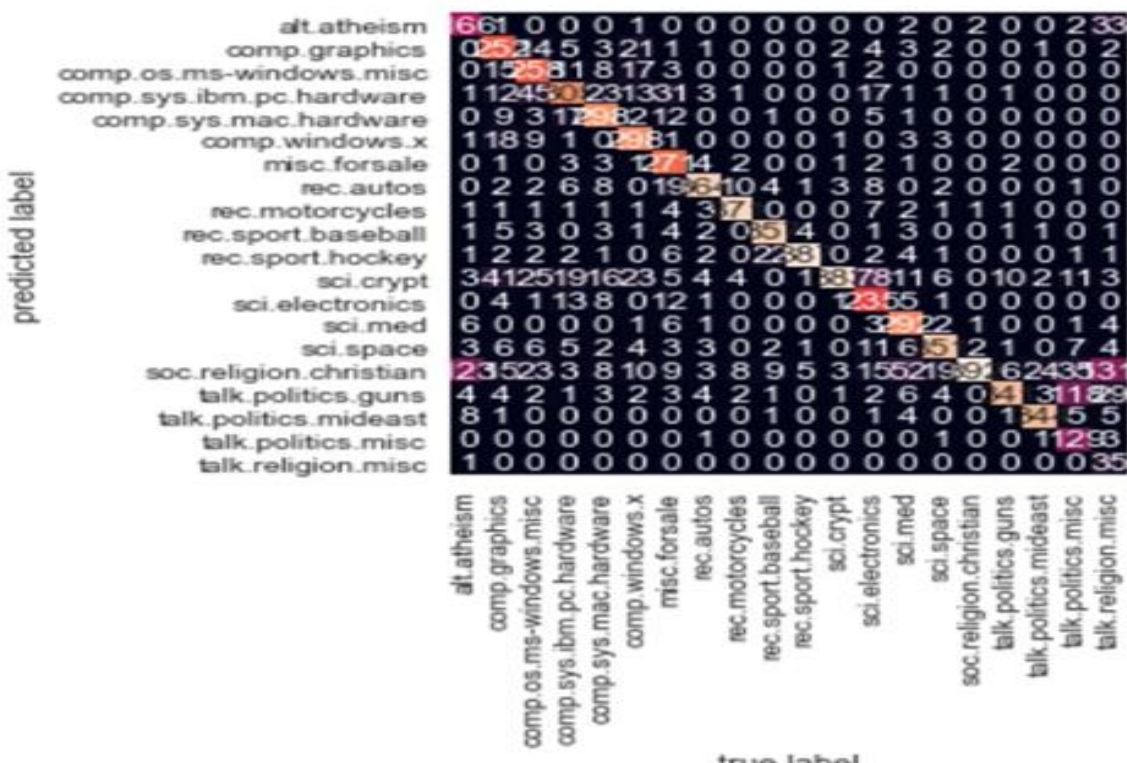
- `categories = ['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos',`
- `'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med',`
- `'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast',`
- `'talk.politics.misc', 'talk.religion.misc']`
- `train=fetch_20newsgroups(subset='train', categories=categories)`
- `test=fetch_20newsgroups(subset='test', categories=categories)`
- `#print(train.data[5])`
- `print(len(train.data))`

Applying Naïve Bayes

- `#importing packages`
- `from sklearn.feature_extraction.text import TfidfVectorizer`
- `from sklearn.naive_bayes import MultinomialNB`
- `from sklearn.pipeline import make_pipeline`
- `#creating a model`
- `model=make_pipeline(TfidfVectorizer(),MultinomialNB())`
- `#training the model`
- `model.fit(train.data, train.target)`
- `labels=model.predict(test.data)`

Confusion Matrix : Heat Map

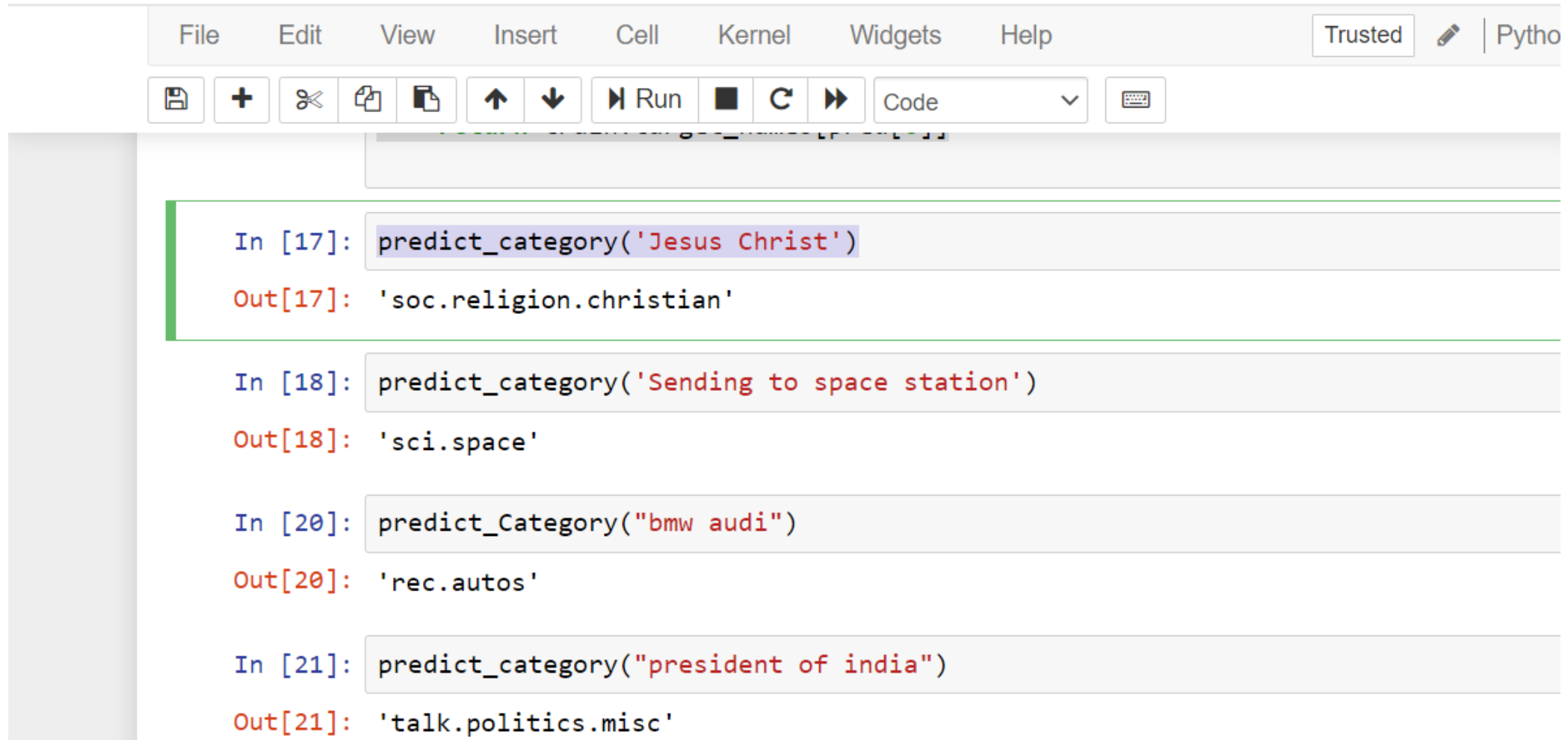
- from sklearn.metrics import confusion_matrix
 - mat= confusion_matrix(test.target, labels)
 - sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
 - xticklabels=train.target_names,yticklabels=train.target_names)
 - #plotting heatmap of confusion
 - plt.xlabel('true label')
 - plt.ylabel("predicted label")
-
- | | alt.atheism | 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 | ... |
|--------------------------|-------------|---------------|-------------------------|--------------------------|-----------------------|----------------|--------------|-----------|-----------------|--------------------|------------------|-----------|-----|
| alt.atheism | 661 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| comp.graphics | 0 | 25 | 4 | 5 | 3 | 2 | 1 | 1 | 0 | 0 | 0 | 0 | ... |
| comp.os.ms-windows.misc | 0 | 15 | 3 | 8 | 1 | 8 | 17 | 3 | 0 | 0 | 0 | 0 | ... |
| comp.sys.ibm.pc.hardware | 1 | 12 | 4 | 5 | 0 | 23 | 13 | 3 | 1 | 0 | 0 | 0 | ... |
| comp.sys.mac.hardware | 0 | 9 | 3 | 17 | 3 | 8 | 2 | 12 | 0 | 0 | 1 | 0 | ... |
| comp.windows.x | 1 | 18 | 9 | 1 | 0 | 2 | 8 | 1 | 0 | 0 | 0 | 0 | ... |
| misc.forsale | 0 | 1 | 0 | 3 | 3 | 1 | 2 | 1 | 4 | 2 | 0 | 0 | ... |
| rec.autos | 0 | 2 | 2 | 6 | 8 | 0 | 19 | 3 | 10 | 4 | 1 | 3 | ... |
| rec.motorcycles | 1 | 1 | 1 | 1 | 1 | 4 | 3 | 7 | 0 | 0 | 0 | 7 | ... |
| rec.sport.baseball | 1 | 5 | 3 | 0 | 3 | 1 | 4 | 2 | 0 | 5 | 4 | 0 | ... |
| rec.sport.hockey | 1 | 2 | 2 | 1 | 0 | 6 | 2 | 0 | 2 | 3 | 8 | 0 | ... |
| sci.crypt | 3 | 4 | 12 | 5 | 19 | 16 | 23 | 5 | 4 | 4 | 0 | 1 | ... |



Predict Category

- #predict category on new data based on trained model
- def predict_category(s,train=train, model=model):
 - pred=model.predict([s])
 - return train.target_names[pred[0]]

Predicting the categories from text



The screenshot displays a Jupyter Notebook interface. At the top is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu bar are buttons for 'Trusted' and a Python logo. Below the menu bar is a toolbar containing icons for saving, adding new cells, cutting, copying, pasting, undo, redo, running the current cell, and a dropdown menu currently set to 'Code'. The notebook area contains four code cells, each with an input prompt (In [n]:) and an output (Out[n]:).

```
In [17]: predict_category('Jesus Christ')
Out[17]: 'soc.religion.christian'

In [18]: predict_category('Sending to space station')
Out[18]: 'sci.space'

In [20]: predict_Category("bmw audi")
Out[20]: 'rec.autos'

In [21]: predict_category("president of india")
Out[21]: 'talk.politics.misc'
```

We were able to correctly classify texts into different groups based on which category they belong to using Naive Bayes Classifier

```
predict_category('Jesus Christ')
```

RELIGION

```
predict_category('Sending load to International Space Station ISS')
```

SPACE

```
predict_category('Suzuki Hayabusa is a very fast motorcycle')
```

MOTORCYCLES

```
predict_category('Audi is better than BMW')
```

AUTOS

```
predict_category('President of India')
```

POLITICS

Text Vectorization and Transformation Pipelines

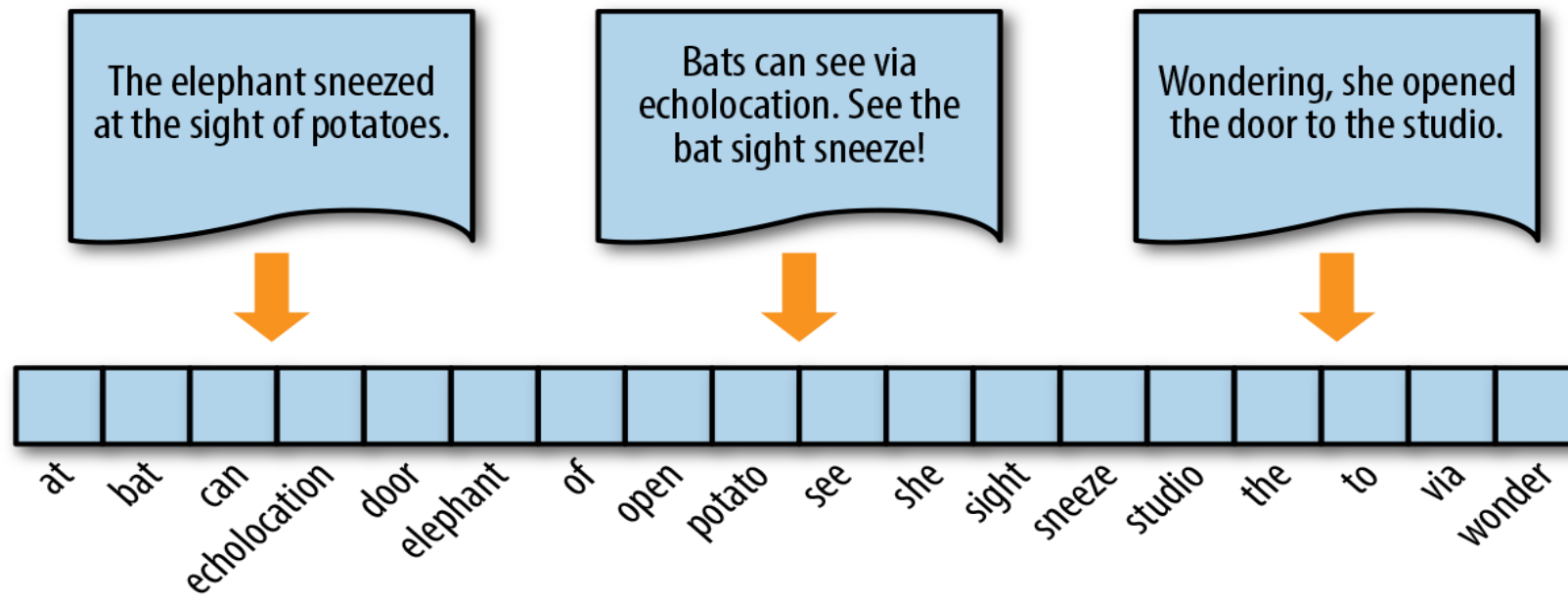
- In order to perform machine learning on text, we need to transform our documents into vector representations such that we can apply numeric machine learning.
- This process is called *feature extraction* or more simply, *vectorization*, and is an essential first step toward language-aware analysis.
- In text analysis, instances are entire documents or utterances, which can vary in length from quotes or tweets to entire books, but whose vectors are always of a uniform length. Each property of the vector representation is a *feature*.
- The features (attributes and content) of a document describe a multidimensional feature space on which machine learning methods can be applied.

Semantic Space

- Sequence of words are points that occupy a high-dimensional semantic *space*.
- Points in space can be close together or far apart, tightly clustered or evenly distributed.
- Semantic space is therefore mapped in such a way where documents with similar meanings are closer together and those that are different are farther apart.
- By encoding similarity as distance, we can begin to derive the primary components of documents and draw decision boundaries in our semantic space.

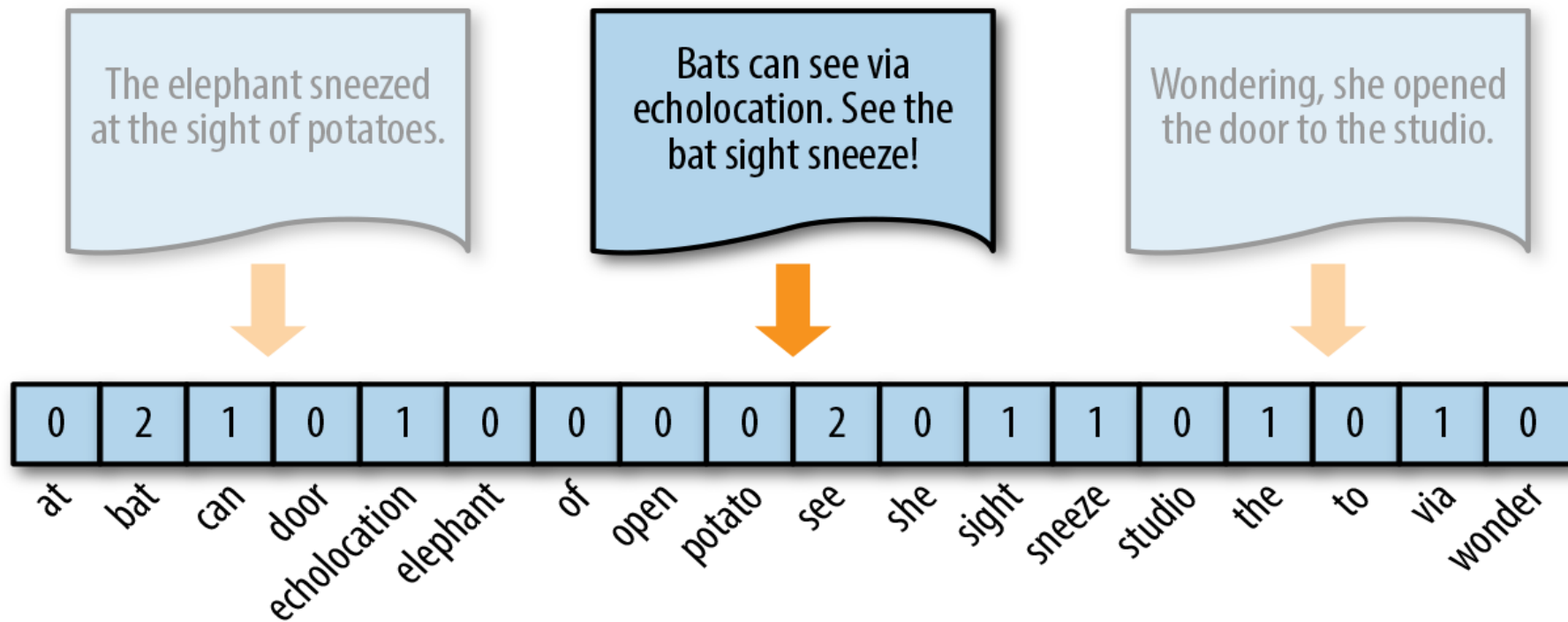
Bag of Words (Words in Space)

- The simplest encoding of semantic space is the *bag-of-words* model.
- To vectorize a corpus with a bag-of-words (BOW) approach, we represent **every document from the corpus as a vector** whose length is equal to the vocabulary of the corpus.
- We can simplify the computation by sorting token positions of the vector into alphabetical order.



Frequency Vectors

- The simplest vector encoding model is to simply fill in the vector with the frequency of each word as it appears in the document.
- In this encoding scheme, each document is represented as the multiset of the tokens that compose it and the value for each word position in the vector is its count.

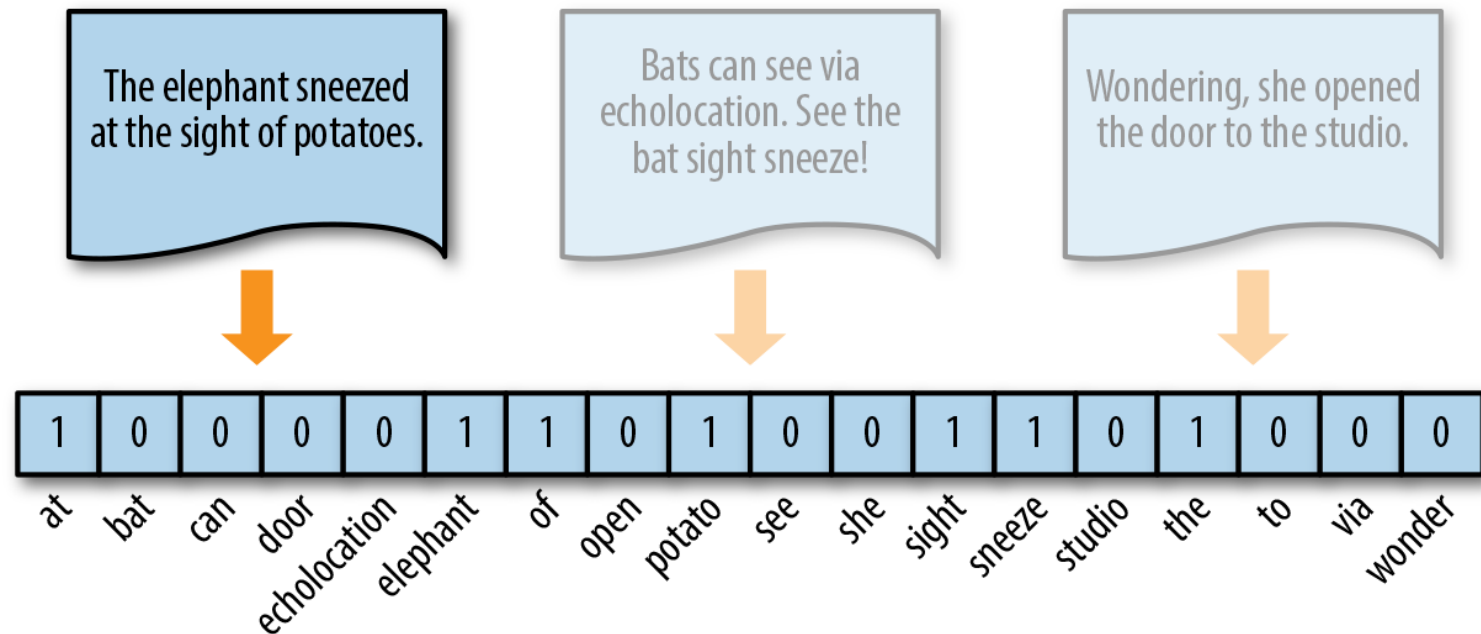


Vectorization

- NLTK, Scikit-Learn, or Gensim are the popular techniques of vectorization.
- The choice of a specific vectorization technique is largely driven by the problem space and by the requirements of the application.
- For instance, NLTK offers many methods that are especially well-suited to text data, but is a big dependency.
- Scikit-Learn was not designed with text in mind, but does offer a robust API and many other conveniences particularly useful in an applied context.
- Gensim can serialize dictionaries and references in matrix market format, making it more flexible for multiple platforms.

One-Hot Encoding

- One-hot encoding is a boolean vector encoding method that marks a particular vector index with a value of true (1) if the token exists in the document and false (0) if it does not. In other words, each element of a one-hot encoded vector reflects either the presence or absence of the token.
- It is a solution to frequency-based encoding methods which suffer from the long tail, or Zipfian distribution, that characterizes natural language. It also results in a significant impact on linear models that expect normally distributed features.



Why is TF-IDF used in Machine Learning?

- Machine learning with natural language is faced with one major hurdle – its algorithms usually deal with numbers, and natural language is, well, text. So we need to transform that text into numbers, otherwise known as text vectorization.
- TF-IDF algorithms help them decipher words by allocating them a numerical value or vector. This has been revolutionary for machine learning, especially in fields related to NLP such as text analysis.
- Once you've transformed words into numbers, in a way that's machine learning algorithms can understand, the TF-IDF score can be fed to algorithms such as Naive Bayes and Support Vector Machines, greatly improving the results of more basic methods like word counts.

Term Frequency–Inverse Document Frequency (TF-IDF)

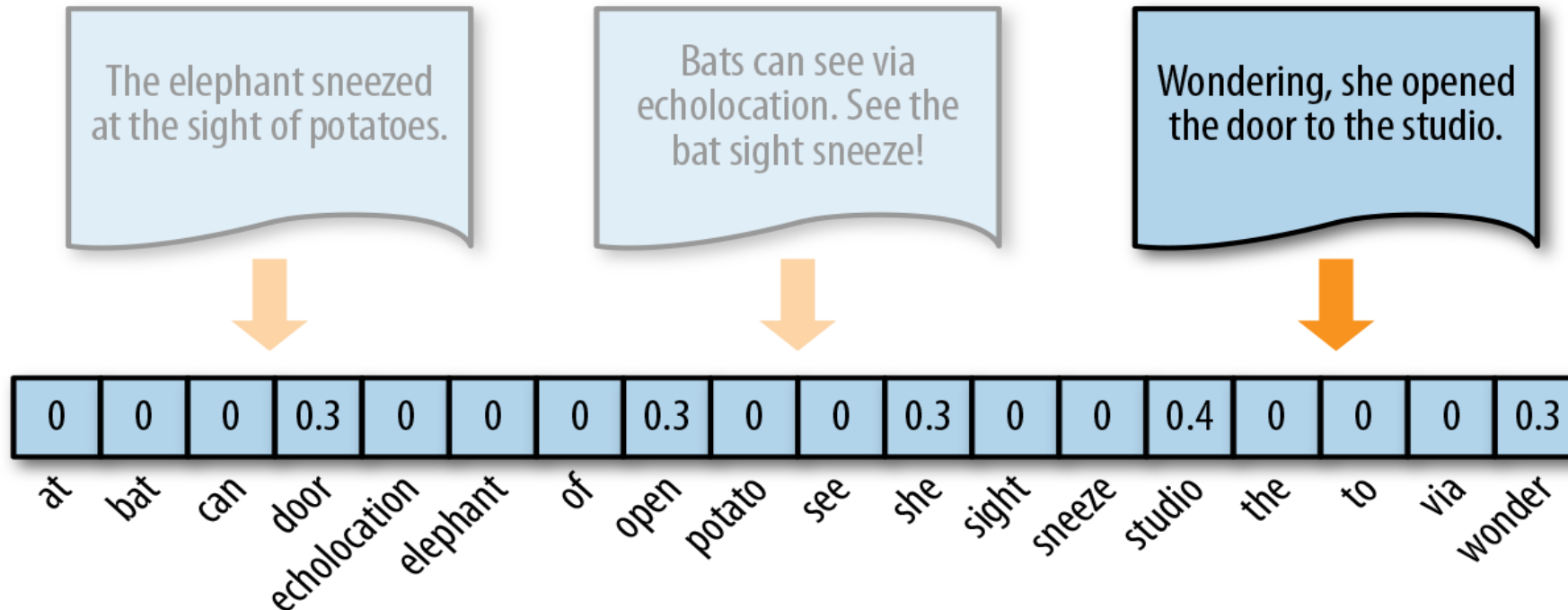
- The bag-of-words representations that we have explored so far only describe a document in a standalone fashion, not taking into account the context of the corpus.
- A better approach would be to consider the relative frequency or rareness of tokens in the document against their frequency in other documents.
- TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.
- It has many uses, most importantly in automated text analysis, and is very useful for scoring words in machine learning algorithms for Natural Language Processing (NLP).
-

Term Frequency–Inverse Document Frequency (TF-IDF)

- It works by increasing proportionally to the number of times a word appears in a document, but is offset by the number of documents that contain the word. So, words that are common in every document, such as this, what, and if, rank low even though they may appear many times, since they don't mean much to that document in particular.
- For example, in a **corpus of sports text**, tokens such as “umpire,” “base,” and “dugout” appear more frequently in documents that discuss baseball, while other tokens that appear frequently throughout the corpus, like “run,” “score,” and “play,” are less important.

TF-IDF

- TF-IDF encoding normalizes the frequency of tokens in a document with respect to the rest of the corpus. Eg the token studio has a higher relevance to this document since it only appears there.



- TF-IDF is computed on a per-term basis, such that the relevance of a token to a document is measured by the scaled frequency of the appearance of the term in the document, normalized by the inverse of the scaled frequency of the term in the entire corpus.

Calculating TF

Term Frequency (tf): gives us the frequency of the word in each document in the corpus.

It is the ratio of number of times the word appears in a document compared to the total number of words in that document. It increases as the number of occurrences of that word within the document increases.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Calculating IDF

- **Inverse Data Frequency (idf):** used to calculate the weight of rare words across all documents in the corpus. The words that occur rarely in the corpus have a high IDF score. It is given by the equation below.

$$idf(w) = \log\left(\frac{N}{df_t}\right)$$

- Combining Tf and IDF, we get the TF-IDF score (w) for a word in a document in the corpus. It is the product of tf and idf:

$$w_{i,j} = tf_{i,j} \times \log \left(\frac{N}{df_i} \right)$$

tf_{ij} = number of occurrences of i in j

df_i = number of documents containing i

N = total number of documents

Because the ratio of the idf log function is greater or equal to 1, the TF-IDF score is always greater than or equal to zero. We interpret the score to mean that the closer the TF-IDF score of a term is to 1, the more informative that term is to that document. The closer the score is to zero, the less informative that term is.

An introduction to TF-IDF

- Let's take an example to get a clearer understanding.
- Sentence 1 : The car is driven on the road.
- Sentence 2: The truck is driven on the highway.
- In this example, each sentence is a separate document.
- We will now calculate the TF-IDF for the above two documents, which represent our corpus.

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	$\log(2/2) = 0$	0	0
Car	1/7	0	$\log(2/1) = 0.3$	0.043	0
Truck	0	1/7	$\log(2/1) = 0.3$	0	0.043
Is	1/7	1/7	$\log(2/2) = 0$	0	0
Driven	1/7	1/7	$\log(2/2) = 0$	0	0
On	1/7	1/7	$\log(2/2) = 0$	0	0
The	1/7	1/7	$\log(2/2) = 0$	0	0
Road	1/7	0	$\log(2/1) = 0.3$	0.043	0
Highway	0	1/7	$\log(2/1) = 0.3$	0	0.043

From the above table, we can see that TF-IDF of common words was zero, which shows they are not significant. On the other hand, the TF-IDF of “car”, “truck”, “road”, and “highway” are non-zero. These words have more significance.

Applications of TF-IDF

- **Information retrieval**
- TF-IDF was invented for document search and can be used to deliver results that are most relevant to what you're searching for.
- Imagine you have a search engine and somebody looks for LeBron. The results will be displayed in order of relevance. That's to say the most relevant sports articles will be ranked higher because TF-IDF gives the word LeBron a higher score.
- It's likely that every search engine you have ever encountered uses TF-IDF scores in its algorithm.

Keyword Extraction

- TF-IDF is also useful for **extracting keywords** from text. How? The highest scoring words of a document are the most relevant to that document, and therefore they can be considered *keywords* for that document. Pretty straightforward.
- Text vectorization transforms text within documents into numbers, so TF-IDF algorithms can rank articles in order of relevance.

In Scikit-Learn

- Scikit-Learn provides a transformer called the `TfidfVectorizer` in the module called `feature_extraction.text` for vectorizing documents with TF–IDF scores.
- It uses the `CountVectorizer` estimator to produce the bag-of-words encoding to count occurrences of tokens, followed by a `TfidfTransformer`, which normalizes these occurrence counts by the inverse document frequency.
- The input for a `TfidfVectorizer` is expected to be a sequence of filenames, file-like objects, or strings that contain a collection of raw documents, similar to that of the `CountVectorizer`.
- The vectorizer returns a sparse matrix representation in the form of `((doc, term), tfidf)` where each key is a document and term pair and the value is the TF–IDF score.

In Scikit-Learn

- `from sklearn.feature_extraction.text import TfidfVectorizer`
- `tfidf = TfidfVectorizer()`
- `corpus = tfidf.fit_transform(corpus)`

In Scikit-Learn

- `from nltk.text import TextCollection`
- `def vectorize(corpus):`
 - `corpus = [tokenize(doc) for doc in corpus]`
 - `texts = TextCollection(corpus)`
- `for doc in corpus:`
 - `yield {`
 - `term: texts.tf_idf(term, doc)`
 - `for term in doc`
 - `}`

With NLTK

- NLTK expects features as a dict object whose keys are the names of the features and whose values are boolean or numeric. To encode our documents in this way, we'll create a vectorize function that creates a dictionary whose keys are the tokens in the document and whose values are the number of times that token appears in the document.
- The defaultdict object allows us to specify what the dictionary will return for a key that hasn't been assigned to it yet. By setting `defaultdict(int)` we are specifying that a should be returned, thus creating a simple counting dictionary. We can map this function to every item in the corpus using the last line of code, creating an iterable of vectorized documents.

vector encoding—frequency, one-hot, TF–IDF, and distributed representations

```
import nltk
import string
def tokenize(text):
    stem = nltk.stem.SnowballStemmer('english')
    text = text.lower()
    for token in nltk.word_tokenize(text):
        if token in string.punctuation: continue
    yield stem.stem(token)
corpus = [ "The elephant sneezed at the sight of potatoes.",
           "Bats can see via echolocation. See the bat sight sneeze!",
           "Wondering, she opened the door to the studio.", ]
```

With NLTK

```
from collections import defaultdict
```

```
def vectorize(doc):  
    features = defaultdict(int)  
    for token in tokenize(doc):  
        features[token] += 1  
    return features
```

```
vectors = map(vectorize, corpus)  
list(vectors)
```

In Scikit-Learn

- The `CountVectorizer` transformer from the `sklearn.feature_extraction` model has its own internal tokenization and normalization methods.
- The `fit` method of the vectorizer expects an iterable or list of strings or file objects, and creates a dictionary of the vocabulary on the corpus.
- When `transform` is called, each individual document is transformed into a sparse array whose index tuple is the row (the document ID) and the token ID from the dictionary, and whose value is the count:

```
from sklearn.feature_extraction.text
```

```
import CountVectorizer
```

```
vectorizer = CountVectorizer()
```

```
vectors = vectorizer.fit_transform(corpus)
```

With NLTK

- To vectorize text in this way with NLTK, we use the TextCollection class, a wrapper for a list of texts or a corpus consisting of one or more texts. This class provides support for counting, concordancing, collocation discovery, and more importantly, computing tf_idf.
- Because TF–IDF requires the entire corpus, our new version of vectorize does not accept a single document, but rather all documents. After applying our tokenization function and creating the text collection, the function goes through each document in the corpus and yields a dictionary whose keys are the terms and whose values are the TF–IDF score for the term in that particular document.

Text vectorization methods

Vectorization Method	Function	Good For	Considerations
Frequency	Counts term frequencies	Bayesian models	Most frequent words not always most informative
One-Hot Encoding	Binarizes term occurrence (0, 1)	Neural networks	All words equidistant, so normalization extra important
TF-IDF	Normalizes term frequencies across documents	General purpose	Moderately frequent terms may not be representative of document topics
Distributed Representations	Context-based, continuous term similarity encoding	Modeling more complex relationships	Performance intensive; difficult to scale without additional tools (e.g., Tensorflow)

- <https://www.oreilly.com/library/view/applied-text-analysis/9781491963036/ch04.html>