

TOPIC MODELING

Prof. Marco Viviani

marco.viviani@unimib.it



Introduction

- **Topic Modeling** is an **unsupervised machine learning technique** aimed at:
 - Scanning a set of documents, detecting **word** and **phrase patterns** within them;
 - Automatically **clustering word groups** and similar expressions that best characterize a set of documents.

Topic Modeling

- Topic Modeling provides **collections of words** that make sense together, which are interpreted as **topics**.

Example: Five topics from a twenty-five topic model fit on Enron e-mails. Example topics concern financial transactions, natural gas, the California utilities, federal regulation, and planning meetings. We provide the five most probable words from each topic (each topic is a distribution over all words).

Topic	Terms
3	trading financial trade product price
6	gas capacity deal pipeline contract
9	state california davis power utilities
14	ferc issue order party case
22	group meeting team process plan

Text Clustering and Topic Modeling

What's the difference?

- In Text Clustering, the basic idea is to **group documents** into different clusters based on a suitable similarity measure (or distance).
- In Topic Modeling, the basic idea is to **group words** into different clusters, where:
 - Each **word** in the cluster is likely to occur “more” (have a probability of occurrence) for the given **topic**;
 - Different **topics** have their respective clusters of **words** along with corresponding probabilities;
 - Different **topics** may share some words and a **document** can have more than one **topic** associated with it.

Text Clustering and Topic Modeling

What's the difference?



Topic Modeling

T_1	T_2	...	T_m
Word ₁₁ , pr ₁₁	Word ₂₁ , pr ₂₁	...	Word _{m1} , pr _{m1}
Word ₁₂ , pr ₁₂	Word ₂₂ , pr ₂₂	...	Word _{m2} , pr _{m2}
.	.	.	.
.	.	.	.
.	.	.	.
Word _{1k} , pr _{1k}	Word _{2k} , pr _{2k}	...	Word _{mk} , pr _{mk}

Document Clustering

C_1	C_2	...	C_m
D_a	D_i	...	D_{c_1}
D_b	D_j	...	D_{c_2}
.	.	.	.
.	.	.	.
.	.	.	.
D_c	D_k	...	D_s

An example of Topic Modeling

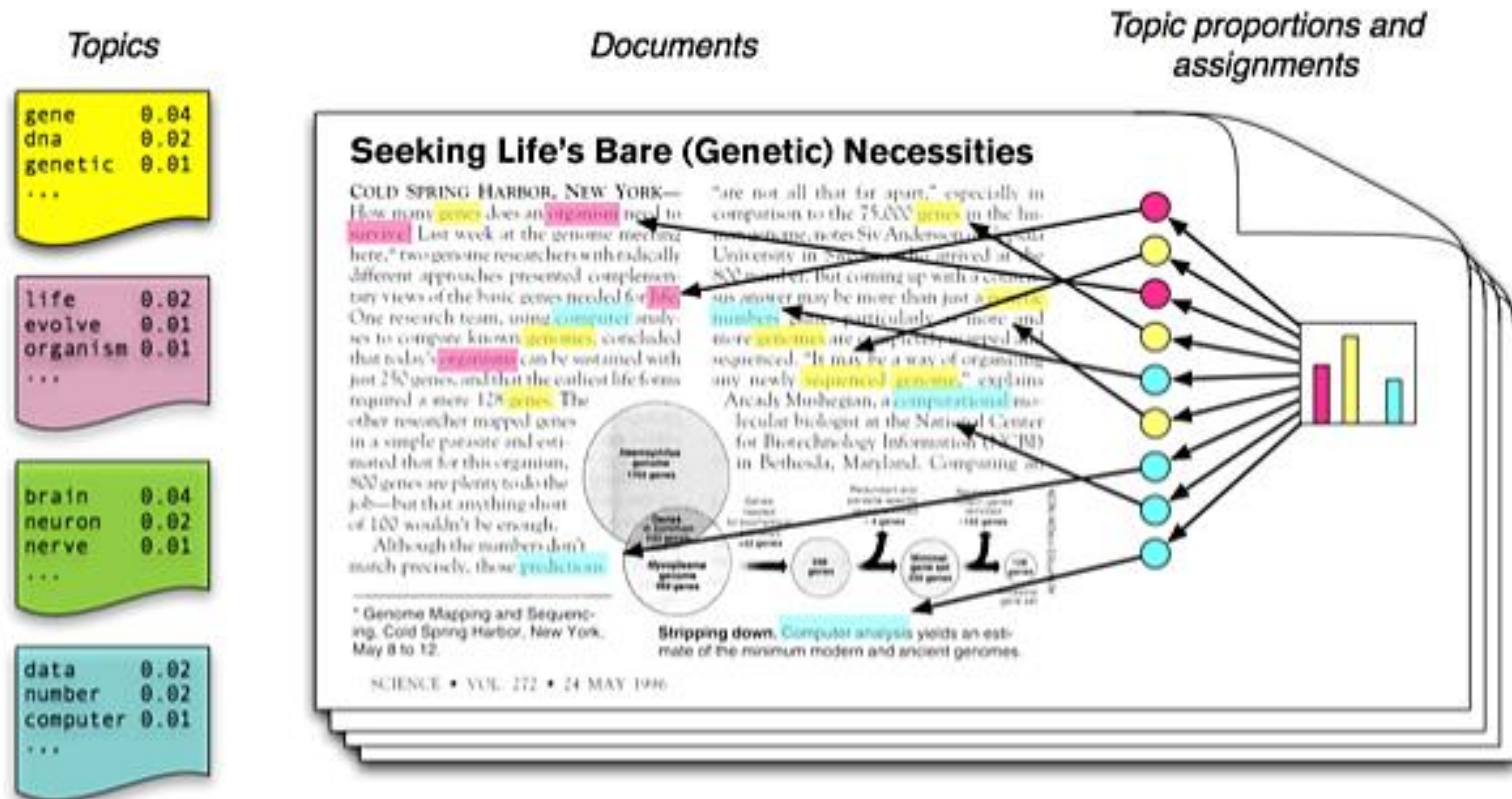


Figure source: Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM*, 55(4), 77-84.

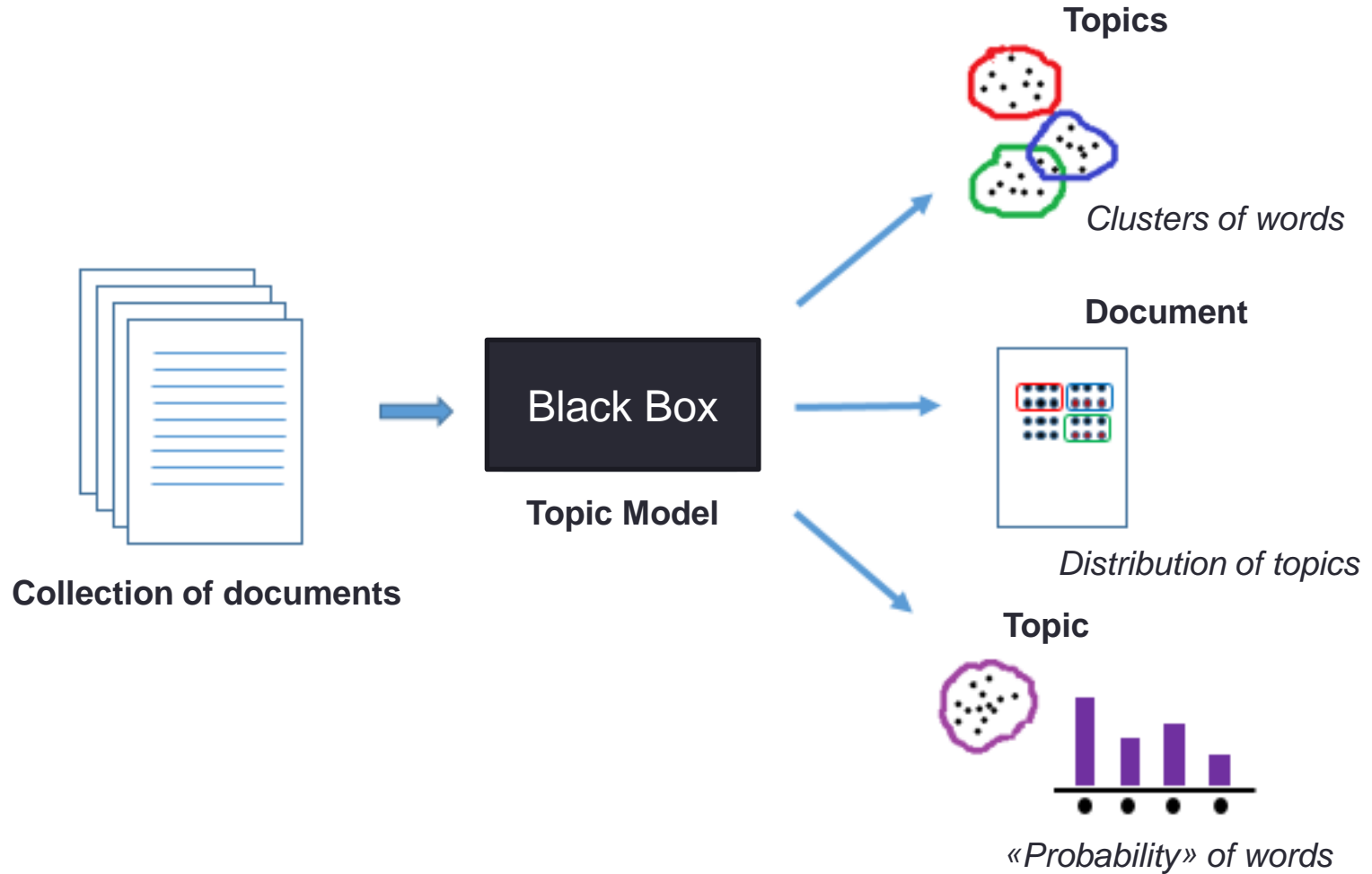
Another example of Topic Modeling

"Manipulating facial expressions and body movements in videos has become so advanced that most people struggle to tell the difference between fake and real. A fake video of Barack Obama went viral last year where you see the former President addressing the camera. If you turn off the sound, you will not even realize it's a fake video!"

	Topic 1
	Topic 2
	Topic 3

- There are **three topics** (or concepts) – Topic 1, Topic 2, and Topic 3.
- The most **dominant topic** in the above example is Topic 2, which indicates that this piece of text is primarily about fake videos.

From Documents to Topics



Main techniques for Topic Modeling

- **Latent Semantic Analysis (LSA)**

- The core idea is to take the Document-Term matrix and **decompose** it into a separate Document-Topic matrix and a Topic-Term matrix.

- **Latent Dirichlet Allocation (LDA)**

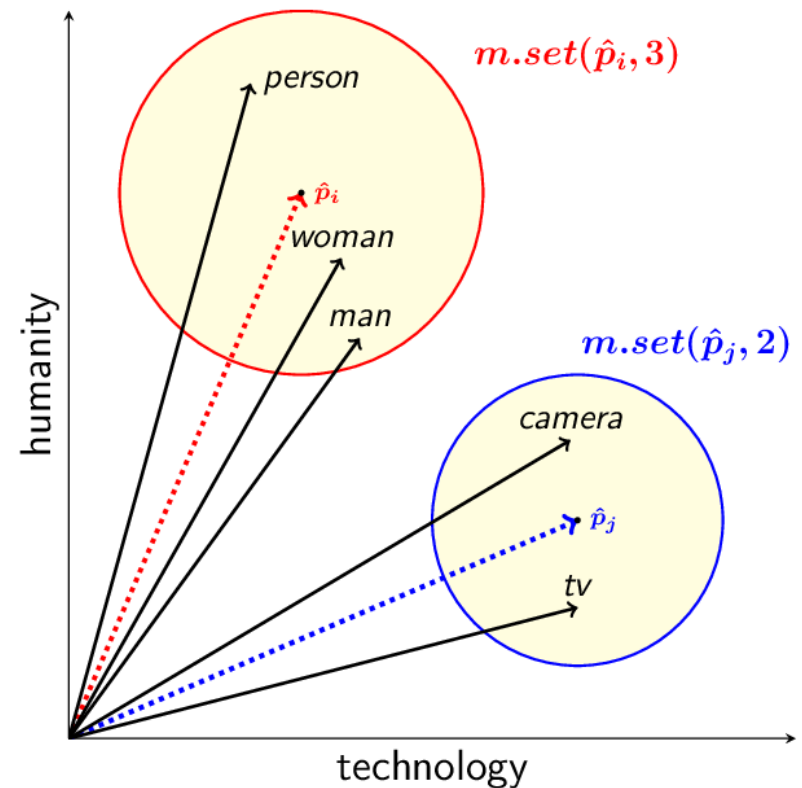
- Each **document** is considered a mixture of topics and each **word** in a document is considered randomly drawn from document's topics;
- The **topics** are considered hidden (latent) and must be uncovered via analyzing joint distribution to compute the conditional distribution of hidden variables (topics) given the observed variables, words in documents.

- Disregarding the approach, the **output** of a topic modeling algorithm is a list of topics with associated clusters of words (and their probabilities).

LATENT SEMANTIC ANALYSIS (LSA)

Latent Semantic Analysis (LSA)

- **Latent Semantic Analysis (LSA)** is one of the simplest Topic Modeling methods.
- It is based on the **distributional hypothesis**:
 - The **semantics** of words can be grasped by looking at the **contexts** the words appear in;
 - Under this hypothesis, the **semantics** of two words will be **similar** if they tend to occur in **similar contexts**.



Latent Semantic Analysis (LSA)

- LSA computes **how frequently** words occur in the documents – and the whole corpus – and assumes that similar documents will contain approximately the same distribution of word frequencies for certain words.
- In this case, syntactic information (e.g., word order) and semantic information (e.g., the multiplicity of meanings of a given word) are ignored and each document is treated as a **bag of words**.

Latent Semantic Analysis (LSA)

- The standard method for computing word frequencies is **TF-IDF**.
- Once TF-IDF frequencies have been computed, we can create a **Document-Term** matrix that contains the TF-IDF values for each term in a given document.

Latent Semantic Analysis (LSA)

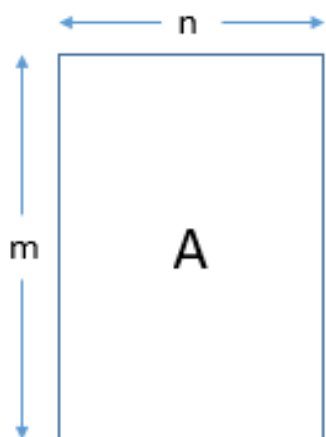
Document-Term Matrix	Lebron	Senate	Celtics	Sprain	Cancer
Document 1	0.4	0.01	0.2	0	0
Document 2	0	0.9	0	0	0.02
Document 3	0	0	0	0.2	0.3
Document 4	0	0	0	0.2	0.3

Latent Semantic Analysis (LSA)

- The Document-Term matrix can be **decomposed** into the product of 3 matrices (USV^T) by using **Singular Value Decomposition (SVD)**.
- The U matrix is known as the **Document-Topic** matrix and the V^T matrix is known as the **Topic-Term** matrix.
- **Linear algebra** guarantees that the S matrix will be diagonal, and LSA will consider each singular value, i.e., each of the numbers in the main diagonal of matrix S , as a **potential topic** found in the documents.

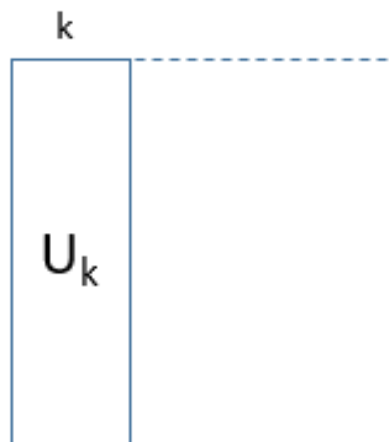
Latent Semantic Analysis (LSA)

Document-Term matrix

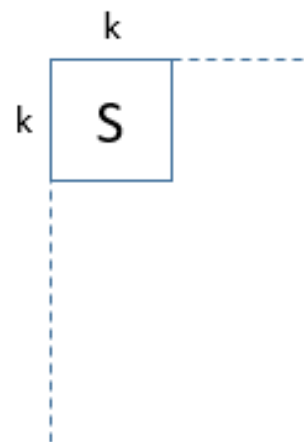


=

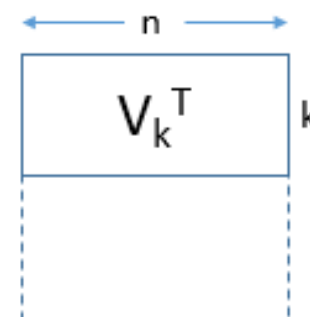
Document-Topic matrix



Topic-Topic matrix




Topic-Term matrix




Latent Semantic Analysis (LSA)

	Term1	Term2	Term3	Term4
Doc1				
Doc2				
Doc3				
Doc4				



$m \times m$ matrix


Topic distribution across documents



	Topic1	Topic2
Doc1		
Doc2		
Doc3		
Doc4		

$m \times n$ singular matrix


Topic importance



	Topic1	Topic2
Topic1		
Topic2		

$n \times n$
diagonal
matrix

Word assignment to topics

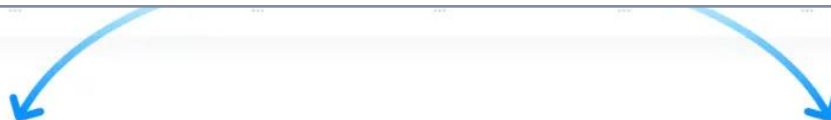


	Term1	Term2	Term3	Term4
Topic1				
Topic2				

$n \times m$
singular
matrix

Latent Semantic Analysis (LSA)

Document-Term Matrix	Lebron	Senate	Celtics	Sprain	Cancer
Document 1	0.4	0.01	0.2	0	0
Document 2	0	0.9	0	0	0.02
Document 3	0	0	0	0.2	0.3
Document 4	0	0	0	0.2	0.3



Document-Topic Matrix	T1	T2	T3	T4
Document 1	0.8	0.2	0	0
Document 2	0	0.7	0	0
Document 3	0.1	0	0	0
Document 4	0.6	0	0.2	0.2

Topic-Term Matrix	Lebron	Senate	Celtisc	Sprain	Cancer
T1	0.8	0	0.9	0.6	0
T2	0.1	0.7	0.1	0	0
T3	0.1	0.3	0	0.4	0.7
...

Implementation of LSA in Python

```
#IMPORTING DATA
```

```
import pandas as pd  
pd.set_option("display.max_colwidth", 200)
```

```
from sklearn.datasets import fetch_20newsgroups
```

```
dataset = fetch_20newsgroups(shuffle = True,  
random_state=1, remove=('headers', 'footers', 'quotes'))
```

```
documents = dataset.data
```

```
#print(len(documents))
```

```
#print(dataset.target_names)
```

WARNING:
NOT to be used for
Projects !!!

Implementation of LSA in Python

11,314

```
['alt.atheism', 'comp.graphics', 'comp.os.ms-  
windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.har  
dware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.mo  
torcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.cr  
ypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion  
.christian', 'talk.politics.guns', 'talk.politics.mideast', '  
talk.politics.misc', 'talk.religion.misc']
```

Implementation of LSA in Python

```
#BUILDING THE MATRIX
```

```
from sklearn.feature_extraction.text import  
TfidfVectorizer
```

```
vectorizer = TfidfVectorizer(stop_words='english',  
max_features= 1000, # keep top 1000 terms  
max_df = 0.5,  
smooth_idf = True)
```

```
X = vectorizer.fit_transform(news_df['clean_doc'])
```

Implementation of LSA in Python

```
#PERFORMING TOPIC MODELING
```

```
from sklearn.decomposition import TruncatedSVD
```

```
# SVD represent documents and terms in vectors
```

```
svd_model = TruncatedSVD(n_components = 20, algorithm =  
'randomized', n_iter = 100, random_state = 122)
```

```
svd_model.fit(X)
```

```
#print(len(svd_model.components_))
```

Implementation of LSA in Python

```
#PRINTING TOPICS
```

```
terms = vectorizer.get_feature_names()
```

```
for i, comp in enumerate(svd_model.components_):  
    terms_comp = zip(terms, comp)  
    sorted_terms = sorted(terms_comp, key= lambda x:x[1],  
reverse=True)[:7]  
    print("Topic "+str(i)+" : ")  
    for t in sorted_terms:  
        print(t[0])  
        print(" ")
```

Implementation of LSA in Python

Topic 0: like know people think good time thanks
Topic 1: thanks windows card drive mail file advance
Topic 2: game team year games season players good
Topic 3: drive scsi disk hard card drives problem
Topic 4: windows file window files program using problem
Topic 5: government chip mail space information encryption data
Topic 6: like bike know chip sounds looks look
Topic 7: card sale video offer monitor price jesus
Topic 8: know card chip video government people clipper
Topic 9: good know time bike jesus problem work
Topic 10: think chip good thanks clipper need encryption
Topic 11: thanks right problem good bike time window
Topic 12: good people windows know file sale files
Topic 13: space think know nasa problem year israel
Topic 14: space good card people time nasa thanks
Topic 15: people problem window time game want bike
Topic 16: time bike right windows file need really
Topic 17: time problem file think israel long mail
Topic 18: file need card files problem right good
Topic 19: problem file thanks used space chip sale

Some of LSA's Drawbacks

- Results that can be justified on the mathematical level, may have **no interpretable meaning** in natural language.
- LSA can only **partially capture polysemy**.
 - This is not always a problem due to words having a predominant sense throughout a corpus (i.e., not all meanings are equally likely).
- Limitations of the bag of words (BOW) model → **unordered** collection of words. Possible solutions:
 - **Multi-gram dictionary** can be used to find direct and indirect association;
 - **Higher-order co-occurrences** among terms.

PROBABILISTIC LSA (pLSA)

High-level Description of pLSA

- pLSA uses a **probabilistic method** instead of SVD to tackle the problem.
- The core idea is to **find a probabilistic model with latent topics** that can generate the data we **observe** in our Document-Term matrix.
- In particular, we want a model $P(D, W)$ such that for any document $d \in D$ and word $w \in W$, $P(d, w)$ corresponds to that entry in the Document-Term matrix.

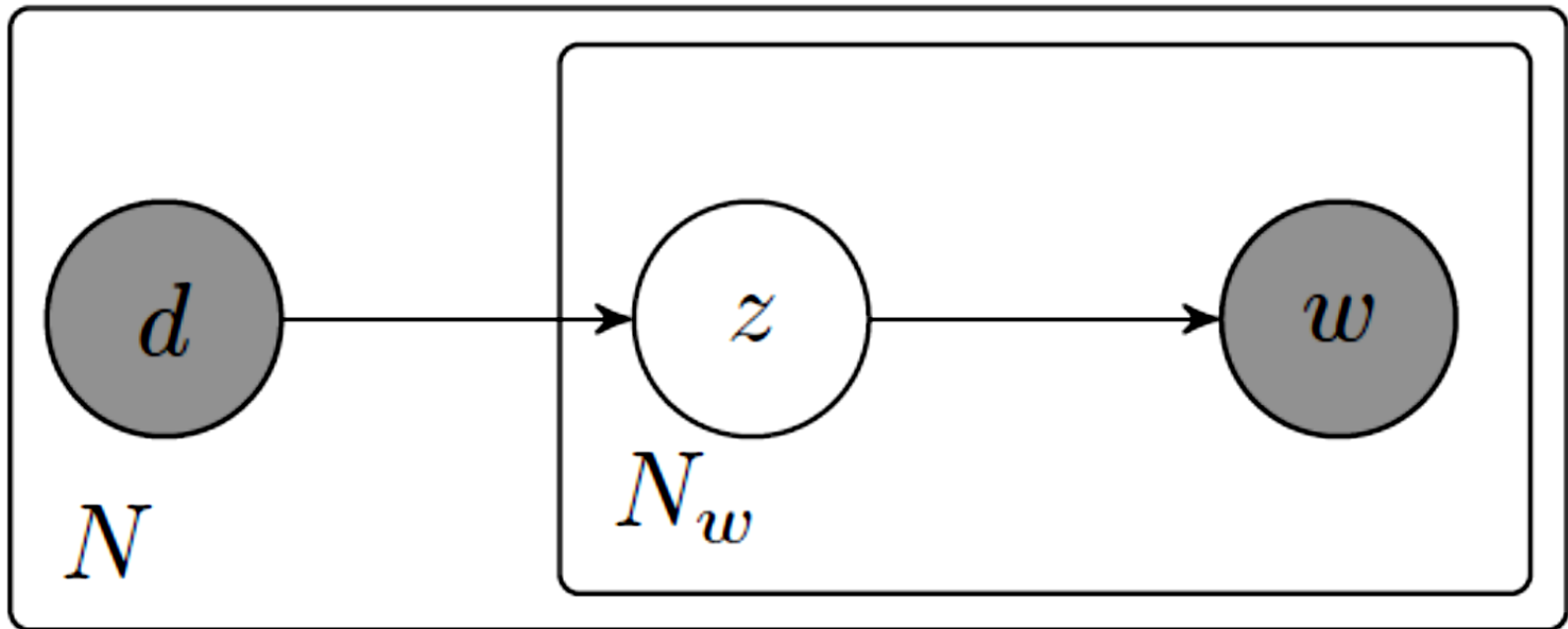
High-level Description of pLSA

- pLSA considers that our data can be expressed in terms of **3 sets of variables**:
 - **Documents**: $d \in D = \{d_1, \dots, d_N\}$ observed variables. Let N be their number, defined by the size of our given corpus.
 - **Words**: $w \in W = \{w_1, \dots, w_M\}$ observed variables. Let M be the number of distinct words from the corpus.
 - **Topics**: $z \in Z = \{z_1, \dots, z_K\}$ latent (or hidden) variables. Their number, K , has to be specified a priori.
- $P(D, W) = \prod_{(d,w)} P(d, w)$

For more details:

http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV1011/oneata.pdf

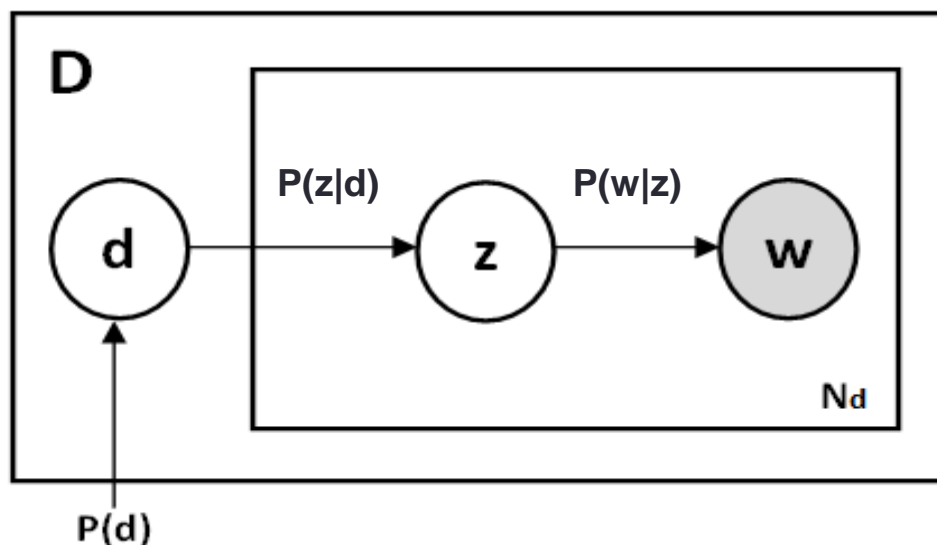
High-level Description of pLSA



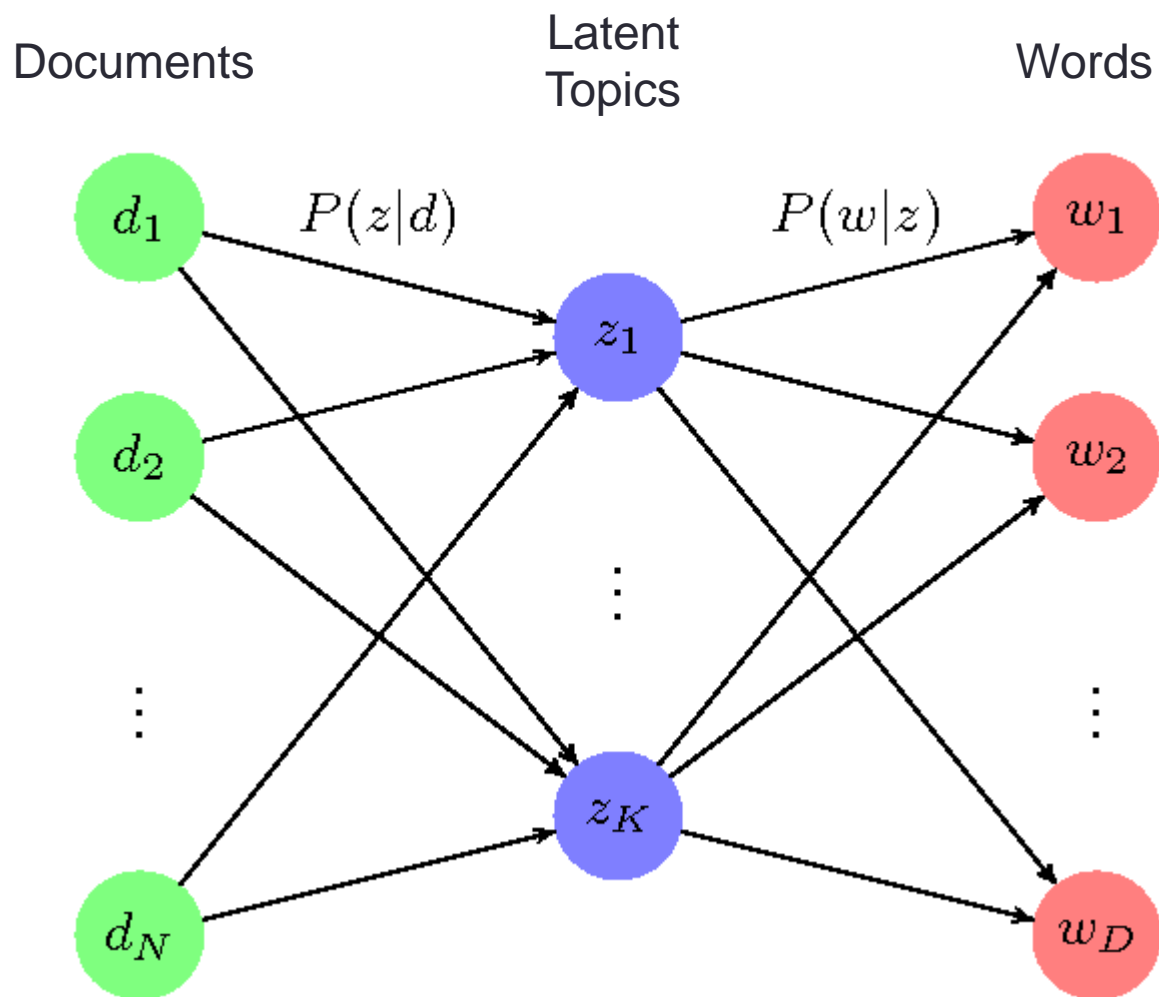
High-level Description of pLSA

Probability theory

- $P(d, w)$ → under conditional independence and using the Bayesian rule:
 - $P(w, d) = \sum_{z \in Z} P(z)P(d|z)P(w|z)$
 - $P(w, d) = P(d) \sum_{z \in Z} P(z|d)P(w|z)$



High-level Description of pLSA



High-level Description of pLSA

- If we reason in terms of **matrix decomposition**:

$$P(w, d) = \sum_{z \in Z} \underbrace{P(z)}_{\text{blue}} \underbrace{P(d|z)}_{\text{red}} \underbrace{P(w|z)}_{\text{yellow}}$$

$$A \approx \underbrace{U_k}_{\text{red}} \underbrace{S_k}_{\text{blue}} \underbrace{V_k^T}_{\text{yellow}}$$

pLSA: Syntesis (1)

- **Document-Term Matrix**
 - Imagine to have a **large collection of documents**.
 - Each document can be represented as a **bag of words**.
 - We create a **matrix** where rows represent documents, columns represent unique words, and the cells contain the **frequency of each word (TF)** in the corresponding document.
- Latent Semantic Analysis (LSA)
 - It analyzes the relationships between terms and documents by performing SVD on the Document-Term Matrix.
 - This helps identify latent (hidden) semantic patterns (drawbacks discussed before).
- **pLSA, probabilistic aspect**
 - In pLSA, the idea is to introduce a probabilistic model to the latent structure.
 - Instead of treating the relationships between terms and documents as fixed values, pLSA introduces probabilities.
 - It assumes that there are latent (hidden) variables governing the generation of terms within a document.

pLSA: Syntesis (2)

- **Generative model**

- pLSA assumes that documents are generated by a **mixture of latent topics**, and each topic is associated with a **probability distribution over terms**.
- The process of generating a document involves choosing a topic according to a probability distribution and then selecting words from the corresponding topic's distribution.

- **Learning parameters**

- The goal in pLSA is to learn the parameters of the model, which include the **probability distributions over terms for each topic** and the **probability of each document belonging to a particular topic**.
- This is typically done using the **Expectation-Maximization (EM)** algorithm.
 - A statistical method used for finding **maximum likelihood** estimates of parameters in models with latent variables.

Implementation of pLSA in Python

- Exercise.
- It can be part of your projects.

Probabilistic LSA (References)

- Hofmann, T. (1999, August). Probabilistic latent semantic indexing. In Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval (pp. 50-57).
- Hofmann, T. (2013). Probabilistic latent semantic analysis. arXiv preprint arXiv:1301.6705.
- http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV1011/oneata.pdf

Probabilistic LSA (Some details) (1)

From

$$P(w, d) = \sum_{z \in Z} P(z)P(d|z)P(w|z)$$

to?

$$P(w, d) = P(d) \sum_{z \in Z} P(z|d)P(w|z)$$

Probabilistic LSA (Some details) (2)

$$P(w, d) = \sum_{z \in Z} P(z) P(d|z) P(w|z) =$$

$$= \sum_{z \in Z} P(z) \frac{P(z|d)P(d)}{P(z)} P(w|z) =$$

$$= P(d) \sum_{z \in Z} P(z|d) P(w|z)$$

Bayes' Theorem

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

LATENT DIRICHLET ALLOCATION (LDA)

High-level Description of LDA

- **Latent Dirichlet Allocation** (LDA) is a Bayesian version of pLSA.
 - LDA treats documents as **bags of words**;
 - It is designed to discover topics based on **term frequencies**.
 - LDA assumes documents are produced from a **mixture of topics**;
 - LDA categorizes documents by topic via a **generative probabilistic model**;
 - Distribution of topics in a document and the distribution of words in topics are **Dirichlet distributions**.

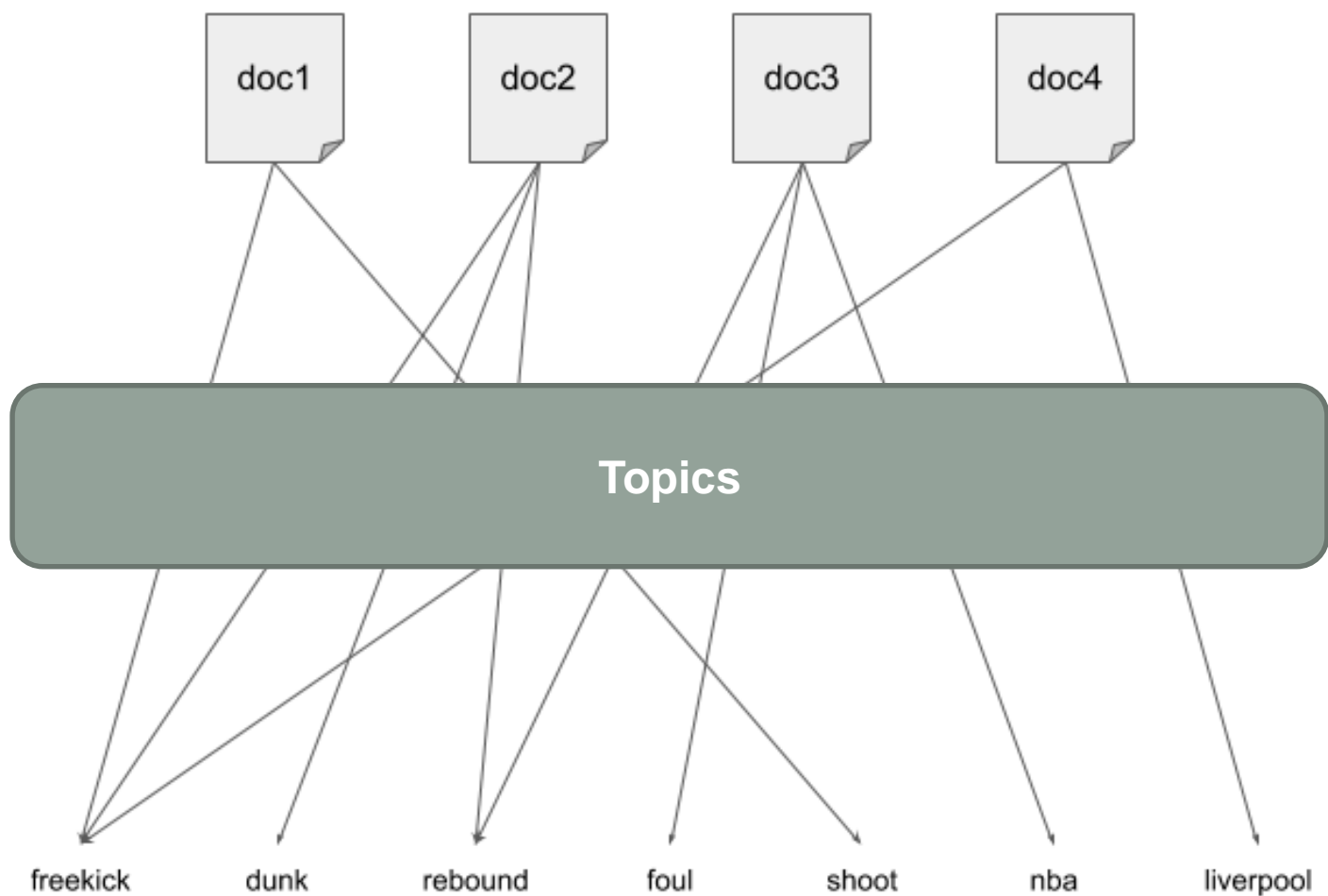
High-level Description of LDA

- The **idea** is that:
 - **Words** are generated from **topics**;
 - Each **document** has a particular probability of using particular **topics** to generate a given **word**;
 - We seek to find which **topics** given **documents** are likely to use to generate **words**.

High-level Description of LDA

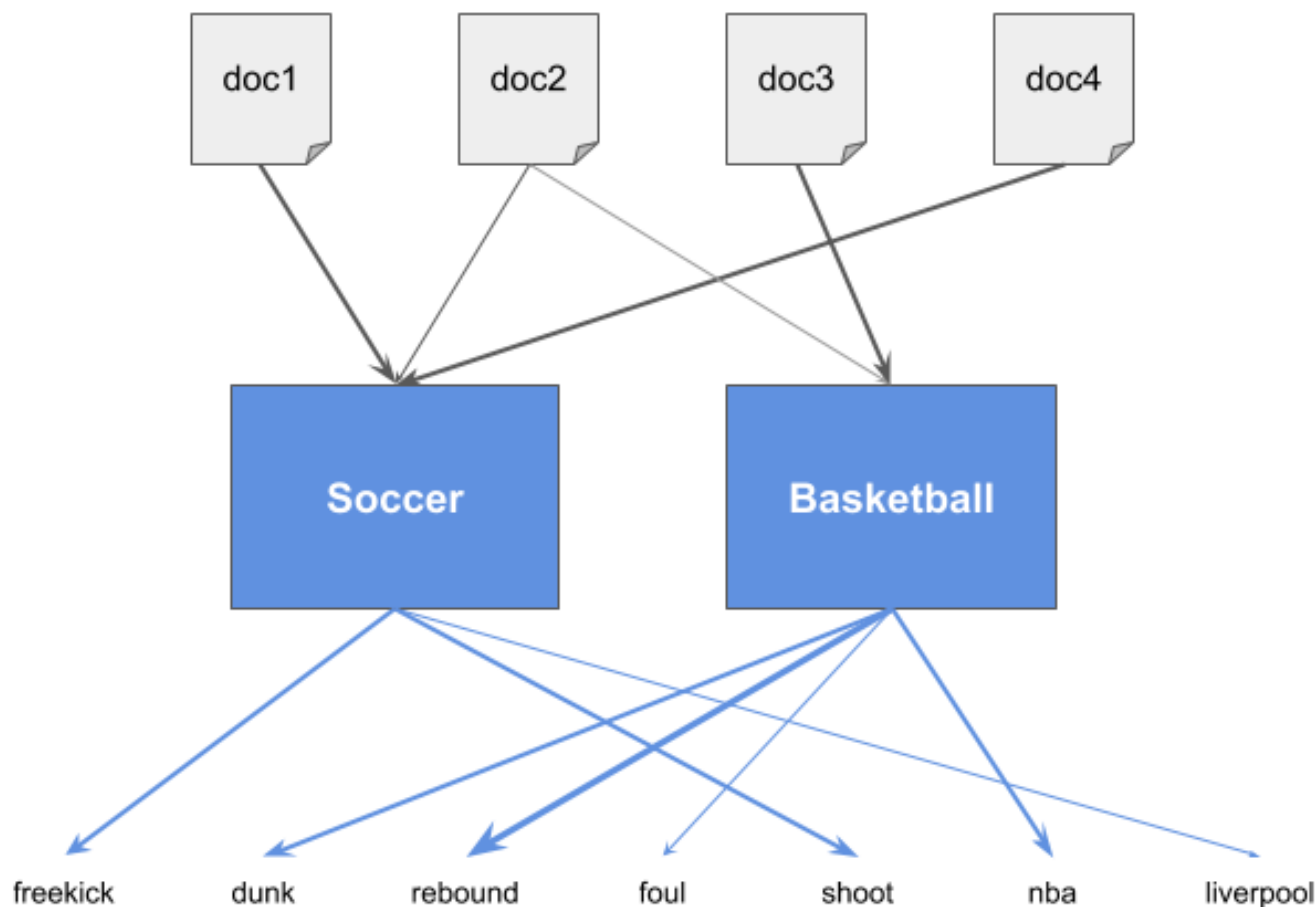
- INPUT: We start with a **corpus** of M documents and choose how many k topics we want to discover out of this corpus.
- OUTPUT: the **topic model**, and the M documents expressed as a combination of the k topics.
- OPERATION: the algorithm finds the **weight of connections** between documents and topics and between topics and words.

High-level Description of LDA



High-level Description of LDA

- For $k = 2$, an LDA model could look like this:



High-level Description of LDA

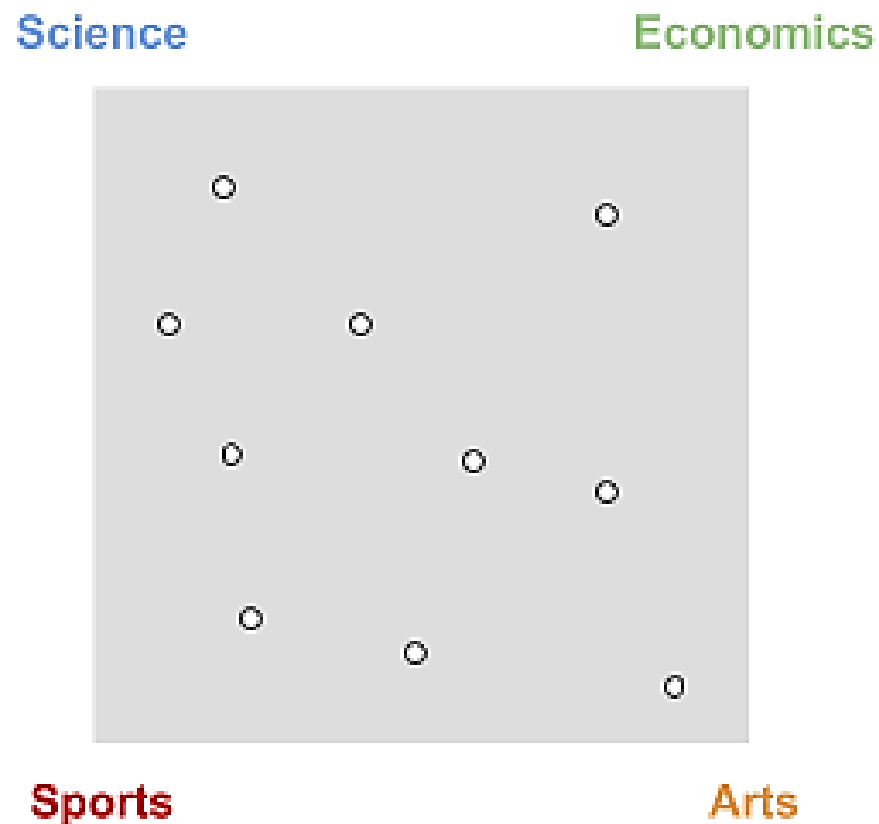
- The algorithm created an **intermediate layer with topics** and figured out the weights between documents and topics and between topics and words.
- **Documents are no longer connected to words but to topics.**
- In the previous example, each topic was **named for clarity**, but in real life, we would not know exactly what they represent.
 - We would have topics 1, 2, ..., up to k , that's all.

Why Dirichlet Distributions?

- Dirichlet distributions encode the **intuition** that documents are related to a few topics.
- In **practical terms**, this results in better disambiguation of words and a more precise assignment of documents to topics.
- Let us suppose to have four topics:
 - **Science**
 - **Sports**
 - **Arts**
 - **Economics**

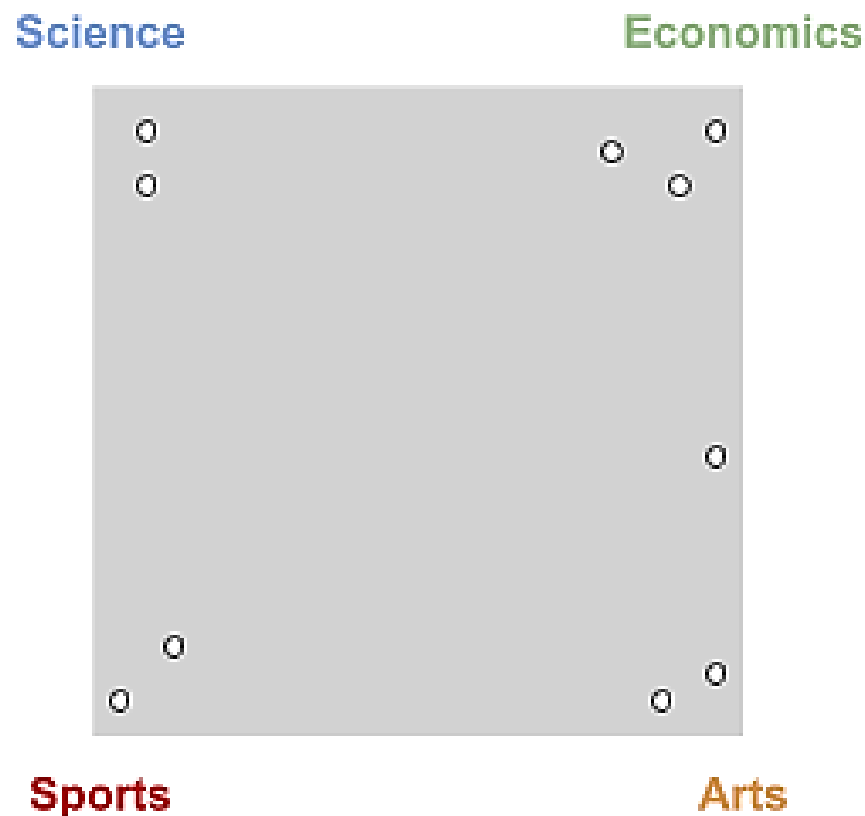
Why Dirichlet Distributions?

- In a **random distribution**, documents would be evenly distributed across the four topics:



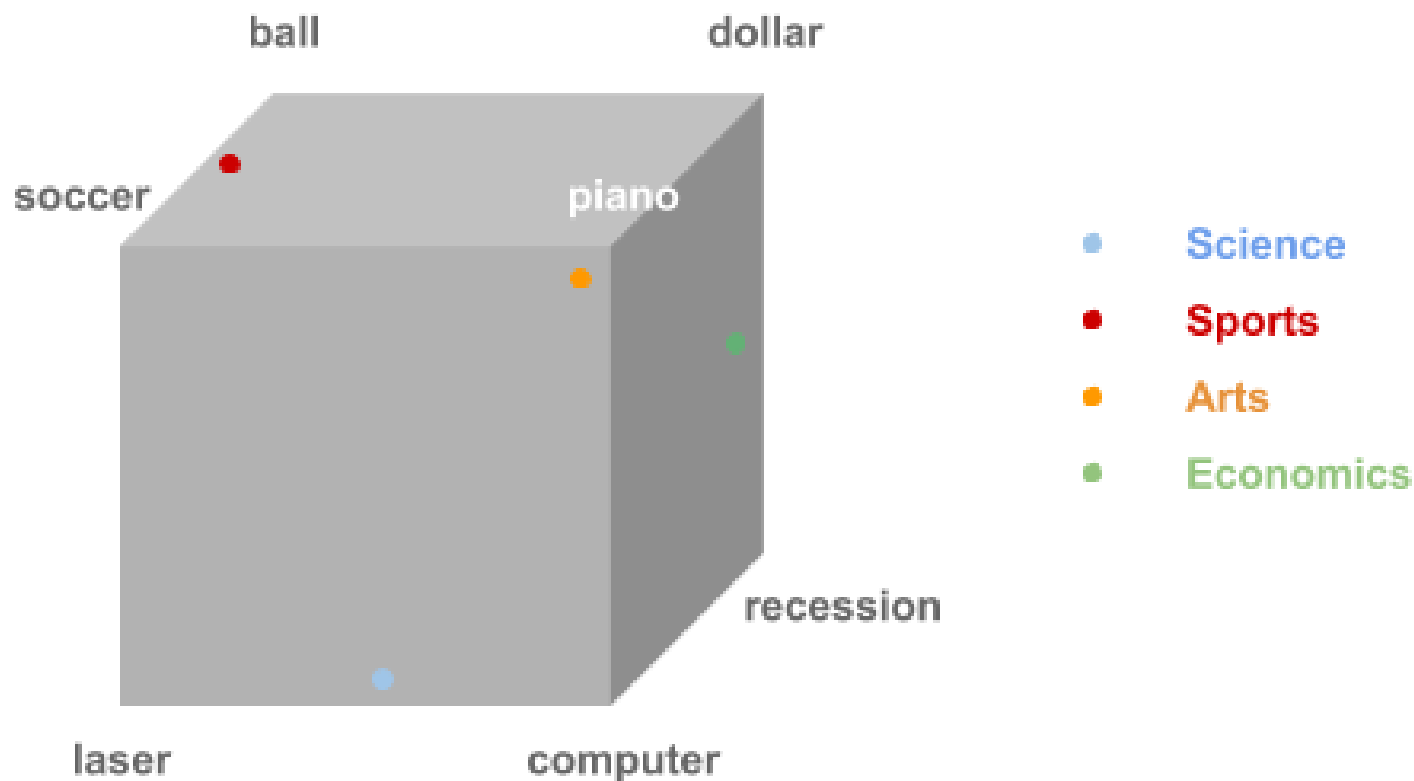
Why Dirichlet Distributions?

- In **real life**, however, we know they are more sparsely distributed, like this:



What are Dirichlet Distributions?

- This also happens between topics and words:



What are Dirichlet Distributions?

- A Dirichlet distribution $Dir(p)$ is **a way to model a probability function** (PF) which gives probabilities for discrete random variables.
- **Example:** rolling a die
 - It is a **discrete random variable**: The result is unpredictable, and the values can be 1, 2, 3, 4, 5, or 6.
 - For a **fair die**, a PF would give these probabilities: [0.16, 0.16, 0.16, 0.16, 0.16, 0.16].
 - For a **biased die**, a PF could return these probabilities: [0.25, 0.15, 0.15, 0.15, 0.15, 0.15], where obtaining a one is higher than the other sides.

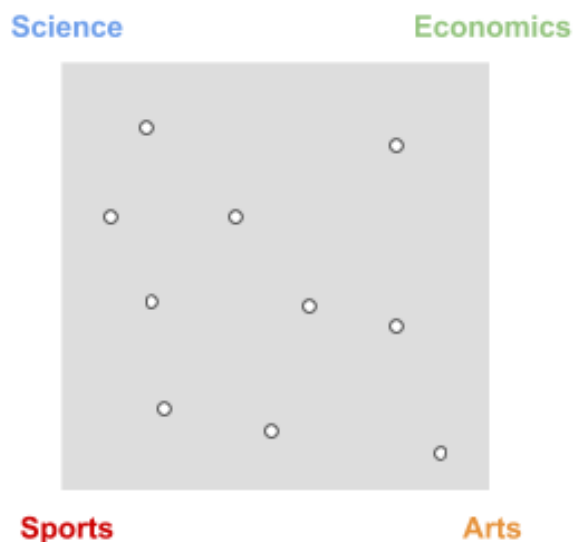
What are Dirichlet Distributions?

- In the example with documents, topics, and words, we have **two PFs**:
 - θ_d : the probability of topic k occurring in document d ;
 - φ_k : the probability of word w occurring in topic k .
- The p parameter in $Dir(p)$ is named **concentration parameter**, and rules the trend of the distribution to be:
 - uniform ($p = 1$)
 - concentrated ($p > 1$)
 - sparse ($p < 1$)

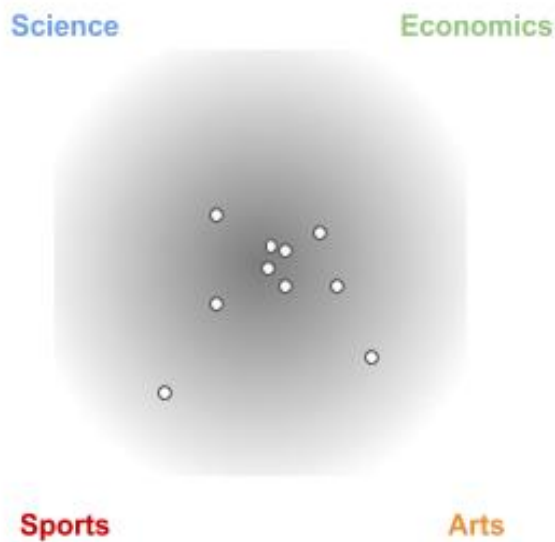
What are Dirichlet Distributions?

- When we consider **document and topics**, we denote $p = \alpha$.

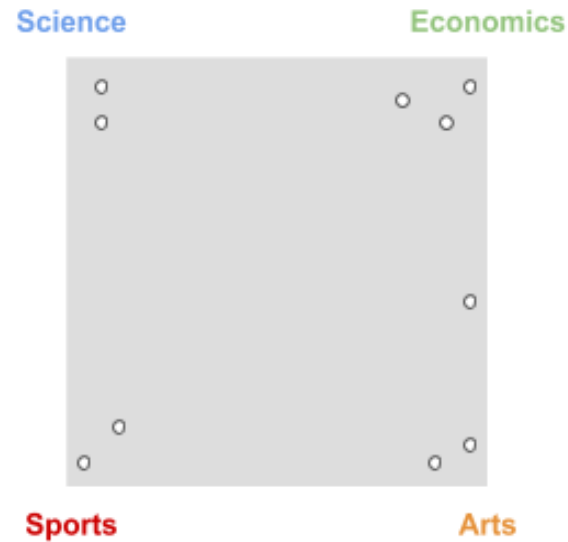
distribution with $\alpha = 1$



distribution with $\alpha > 1$



distribution with $\alpha < 1$



What are Dirichlet Distributions?

- When we consider **topics and words**, we denote $p = \beta$.
- By using concentration parameters $\alpha, \beta < 1$, these probabilities will be closer to the real world.
- In other words, they follow Dirichlet distributions:

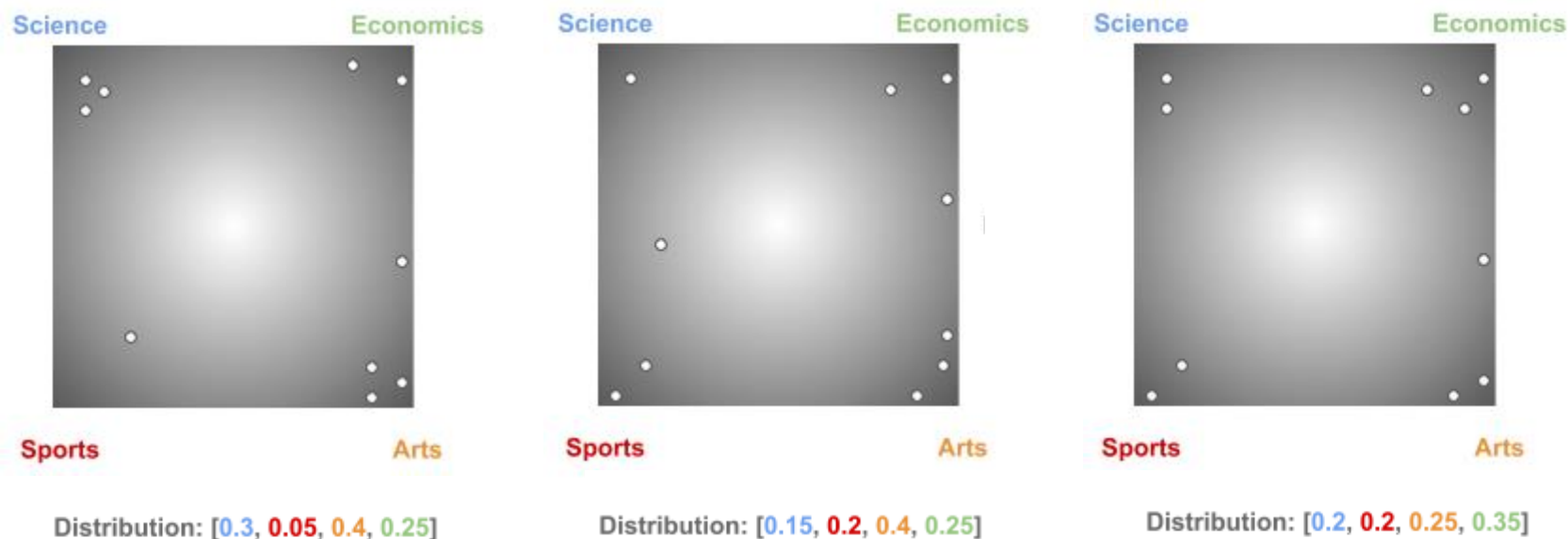
$$\theta_d \approx \text{Dir}(\alpha)$$

$$\varphi_k \approx \text{Dir}(\beta)$$

where α and β rule each distribution, and both have values < 1 .

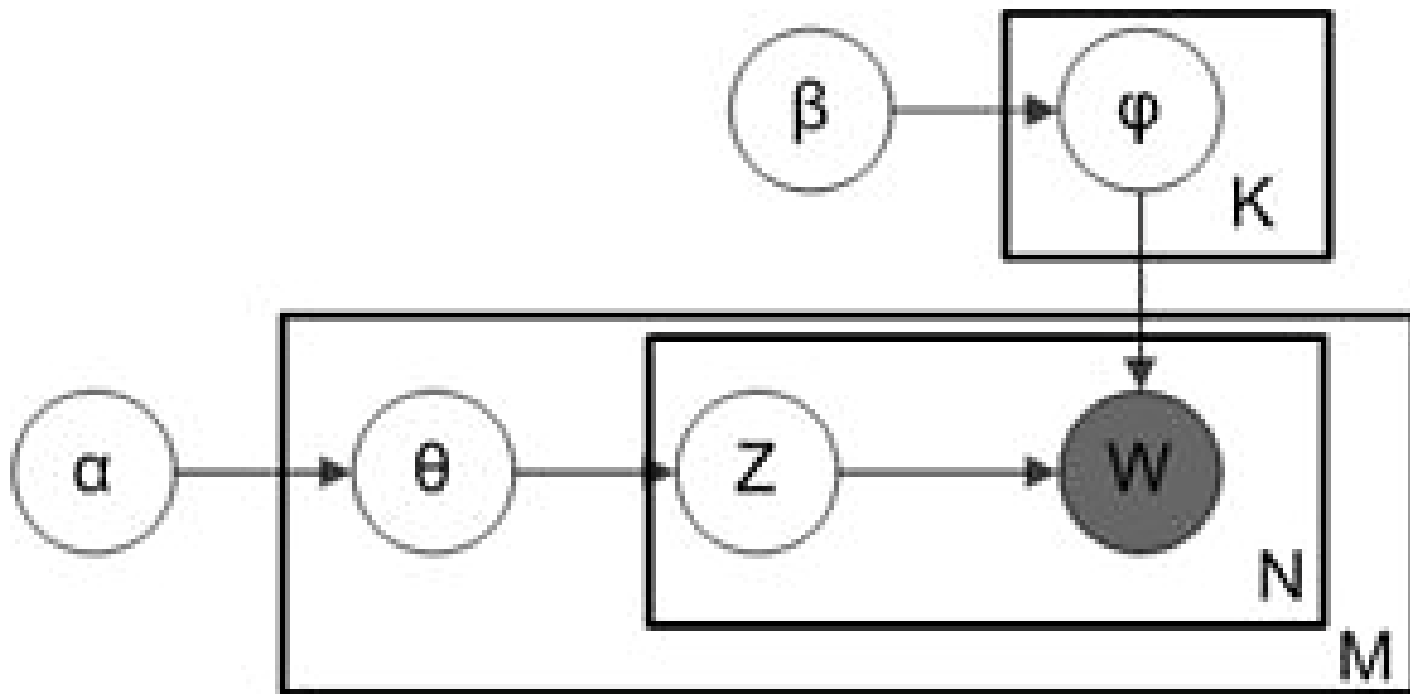
What are Dirichlet Distributions?

- **IMPORTANT.** Using the same concentration parameter, e.g., α , we obtain many different distributions of documents over topics



- They **get adjusted** during the training process to make the model better.

High-level Description of LDA



Parameters of LDA (1)

- **Alpha and Beta Hyperparameters**

- Alpha represents document-topic density.
 - Higher the value of alpha, documents are composed of more topics and lower the value of alpha, documents contain fewer topics.
- Beta represents topic-word density.
 - Higher the beta, topics are composed of a large number of words in the corpus, and with the lower value of beta, they are composed of few words.

- **Number of Topics**

- Selected randomly.
- By using the Kullback-Leibler (KL) Divergence Score.

Parameters of LDA (2)

- **Number of Topic Terms**

- Generally decided according to the requirement.
 - If the problem statement talks about extracting themes or concepts, it is recommended to choose a higher number;
 - If problem statement talks about extracting features or terms, a low number is recommended.

- **Number of Iterations**

- Maximum number of iterations allowed to LDA algorithm for convergence.

LSA (and pLSA) vs LDA

- The main difference between LSA and LDA is that LDA assumes that the distribution of topics in a document and the distribution of words in topics are **Dirichlet distributions**.
- **LSA does not assume any distribution** and, therefore, leads to more opaque vector representations of topics and documents.
- LDA typically works better than pLSA because it can **generalize to new documents** easily.
 - In pLSA, the document probability is a fixed point in the dataset.
 - If we have not seen a document, we do not have that data point.
 - In LDA, the dataset serves as training data for the Dirichlet distribution of document-topic distributions.
 - If we have not seen a document, we can easily sample from the Dirichlet distribution and move forward from there.

Implementation of LDA in Python

```
#IMPORTING DATA - AS IN THE LSA EXAMPLE
```

```
#PREPROCESSING - LDA requires some basic pre-processing of text data
```

```
def tokenize_lemma_stopwords(text):
```

```
    tokens = nltk.tokenize.word_tokenize(text.lower())
```

```
    # split string into words (tokens)
```

```
    tokens = [t for t in tokens if t.isalpha()]
```

```
    # keep strings with only alphabets
```

```
    tokens = [wordnet_lemmatizer.lemmatize(t) for t in tokens]
```

```
    # put words into base form
```

```
    tokens = [stemmer.stem(t) for t in tokens]
```

```
    tokens = [t for t in tokens if len(t) > 2]
```

```
    # remove short words, they're probably not useful
```

```
    tokens = [t for t in tokens if t not in stopwords]
```

```
    # remove stopwords
```

```
    return tokens
```

```
def dataCleaning(data):
```

```
    data["content"] = data["content"].apply(tokenize_lemma_stopwords)
```

```
    return data
```

Implementation of LDA in Python

```
#Convert pre-processed tokens into a dictionary with word index  
and it's count in the corpus
```

```
#We can use gensim package to create this dictionary then to  
create bag-of-words
```

```
dictionary = gensim.corpora.Dictionary(X)
```

```
dictionary.filter_extremes(no_below=5, no_above=0.5,  
keep_n=100000)
```

```
# filter words that occurs in less than 5 documents and words  
that occurs in more than 50% of total documents
```

```
# keep top 100000 frequent words
```

```
bow_corpus = [dictionary.doc2bow(doc) for doc in X]
```

```
# create bag-of-words ==> list(index, count) for words in  
dictionary
```


Implementation of LDA in Python

```
from pprint import pprint
pprint(lda_model.print_topics())

[(0, # Seems to be Computer and Technology
  '0.014*"key" + 0.007*"chip" + 0.006*"encryption" + 0.006*"system" + '
  '0.005*"clipper" + 0.005*"article" + 0.004*"university" + '
  '0.004*"information" + 0.004*"government" + 0.004*"time"'),
 (1, # Seems to be Science and Technology
  '0.008*"drive" + 0.007*"university" + 0.007*"window" + 0.007*"system" + '
  '0.006*"doe" + 0.005*"card" + 0.005*"thanks" + 0.005*"space" + '
  '0.004*"article" + 0.004*"computer"'),
 (2, # Seems to be politics
  '0.010*"people" + 0.006*"gun" + 0.006*"armenian" + 0.005*"time" + '
  '0.005*"article" + 0.005*"then" + 0.005*"israel" + 0.004*"war" + '
  '0.004*"government" + 0.004*"israeli"'),
 (3, # Seems to be sports
  '0.013*"game" + 0.011*"team" + 0.008*"article" + 0.007*"university" + '
  '0.006*"player" + 0.006*"time" + 0.005*"play" + 0.005*"season" + '
  '0.004*"hockey" + 0.004*"win"'),
 (4, # Seems to be religion
  '0.018*"god" + 0.011*"people" + 0.008*"doe" + 0.008*"christian" + '
  '0.007*"jesus" + 0.006*"believe" + 0.006*"then" + 0.006*"article" + '
  '0.005*"life" + 0.005*"time"')]
```

EVALUATING TOPIC MODELING

Evaluation Approaches

- **Eye Balling Models**

- Top- n words
- Topics/Documents

- **Intrinsic Evaluation Metrics**

- Capturing model semantics
- Topics interpretability

Internal coherence of topic models

- **Human Judgements**

- Quantitative methods for evaluating human judgement

- **Extrinsic Evaluation Metrics/Evaluation at task**

- Is the model good at performing predefined tasks, such as classification?

Intrinsic Evaluation Metrics

- **Intrinsic Evaluation metrics** that best describe the performance of a topic model:
 - Perplexity
 - Coherence
 - Diversity

Measure Details

- **Perplexity** is a measure of uncertainty, meaning lower the perplexity better the model.
- **Coherence** is the measure of semantic similarity between top words in our topic. Higher the coherence better the model performance.
- **Diversity** evaluates whether topics are diverse and not redundant.

Perplexity

- **Perplexity** is a statistical measure of how well a probability model predicts a sample.
- It aims to capture how “surprised” a model is of new data it has not seen before.
- This metric is measuring “how probable” some new unseen data is given the model that was learned earlier.

#COMPUTING PERPLEXITY

```
print('Perplexity: ',  
lda_model.log_perplexity(bow_corpus))
```

Coherence

- **Topic Coherence** measures score a single topic by measuring the degree of semantic similarity between high scoring words in the topic.
- These measurements help distinguish between topics that are semantically interpretable topics and topics that are artifacts of statistical inference.

#COMPUTING COHERENCE

```
coherence_model_lda =  
models.CoherenceModel(model=lda_model, texts=X,  
dictionary=dictionary, coherence='c_v')
```

```
coherence_lda = coherence_model_lda.get_coherence()  
print('Coherence Score: ', coherence_lda)
```

Distinct Coherence measures

- **c_v**
 - based on a sliding window, one-set segmentation of the top words and an indirect confirmation measure that uses normalized pointwise mutual information (NPMI) and the cosine similarity.
- **c_p**
 - based on a sliding window, one-preceding segmentation of the top words and the confirmation measure of Fitelson's coherence.
- **c_uci**
 - based on a sliding window and the pointwise mutual information (PMI) of all word pairs of the given top words.

Distinct Coherence measures

- **c_umass**
 - based on document cooccurrence counts, a one-preceding segmentation and a logarithmic conditional probability as confirmation measure.
- **c_npmi**
 - enhanced version of the `c_uci` coherence using the normalized pointwise mutual information (NPMI).
- **c_a**
 - based on a context window, a pairwise comparison of the top words and an indirect confirmation measure that uses normalized pointwise mutual information (NPMI) and the cosine similarity.

Diversity

- **Metrics for diversity** could include measuring the cosine similarity between topic vectors or quantifying the spread of topics across documents.
- Redundant topics might occur when the model identifies similar topics with slight variations.

#COMPUTING DIVERSITY

```
doc_topic_matrix =  
lda_model.get_document_topics(corpus)  
doc_topic_array = np.array([np.array(doc_topic)[: , 1]  
for doc_topic in doc_topic_matrix])  
cosine_sim_matrix = cosine_similarity(doc_topic_array)  
  
topic_diversity = 1 - np.mean(np.max(cosine_sim_matrix,  
axis=1))  
print("Topic Diversity: {topic_diversity}")
```


A (VERY) BRIEF INTRODUCTION TO TOPIC CLASSIFICATION

Topic Modeling VS Topic Classification

- **Topic Modeling** is an **unsupervised machine learning** technique (i.e., it does not require training).
 - If there is not the possibility to priorly analyze texts (to label it), or if the aim is not looking for a fine-grained analysis, topic modeling algorithms are indicated.
- **Topic Classification** is a **supervised machine learning** technique, i.e., it needs training before being able to automatically analyze texts.
 - If there is a list of predefined topics for a set of texts, and the aim is to gain accurate insights, topic classification is more suitable.

Topic Classification Approaches

- **Rule-based systems**
 - Human-based
- **Machine learning systems**
 - Automatic supervised approaches
- **Hybrid systems**
 - A mix of the previous two

Rule-Based Systems

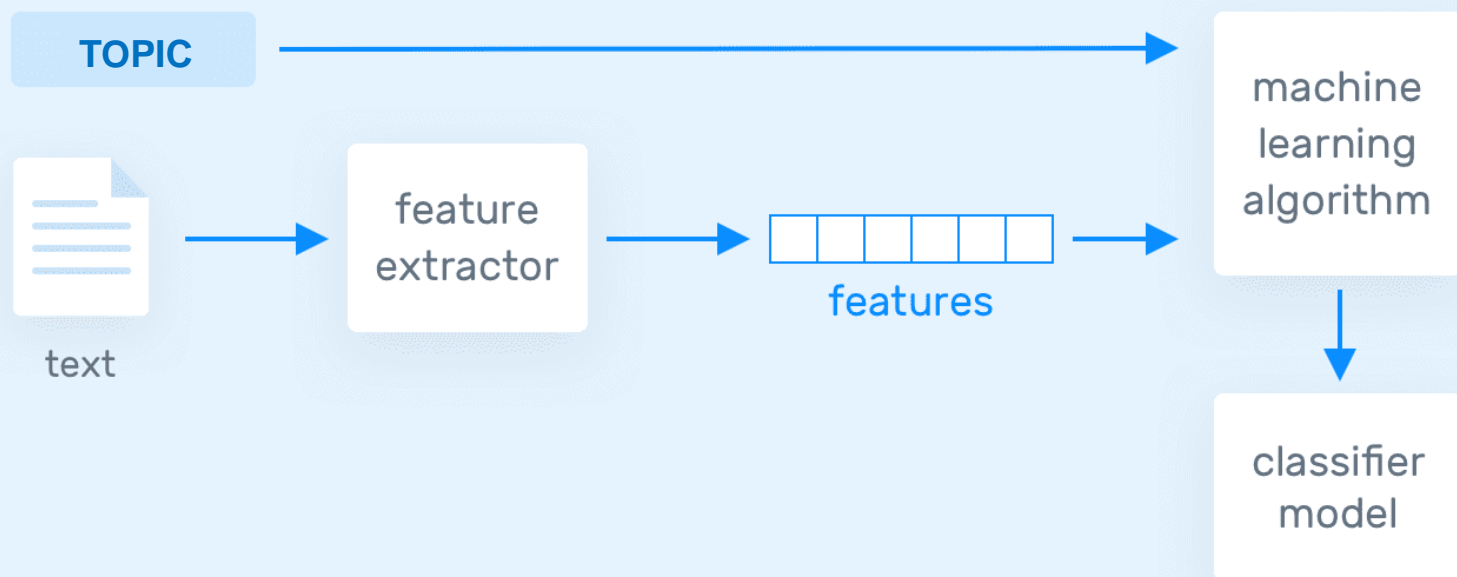
- They work by directly programming a set of **hand-made rules**, based on the content of the documents that a human **expert** has read.
- Each one of these rules is made up of a **pattern** and a **prediction**. Since we are focusing on topic analysis, the prediction will be the topic.
- **Downsides:**
 - Too complex for someone without expert knowledge;
 - Require constant analysis and testing to ensure they are functioning in the correct way;
 - When adding new rules, existing rules are altered;
 - In short, these systems are **high-maintenance** and **unscalable**.

Machine Learning Systems

- A topic classification machine learning model needs to be fed examples of text and a list of predefined tags, known as **training data**.
- Once the text is transformed into **vectors** and the training data is tagged with the expected tags, this information is fed to an algorithm to create the **classification model**.

Machine Learning Systems

(a) Training

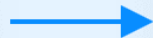


Machine Learning Systems

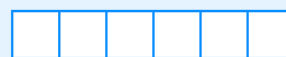
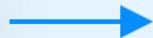
(b) Prediction



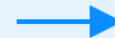
text



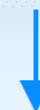
feature
extractor



features



classifier
model



TOPIC

Machine Learning Systems

- **Naive Bayes**

- Family of that deliver good results even when dealing with **small amounts of data**, say between 1,000 and 10,000 texts;
- It works by correlating the probability of words appearing in a text with the probability of that text being about a certain topic.

- **Support Vector Machines (SVM)**

- Slightly more complex than Naive Bayes;
- They often deliver better results than NB for topic classification;
- **Downside**: they require complex programming and require more computing resources.
 - It is possible to speed up the training process of an SVM by optimizing the algorithm by feature selection, in addition to running an optimized linear kernel such as scikit-learn's Linear SVC.

Machine Learning Systems

- **Deep Learning**
 - Topic Classification benefit from Deep Learning;
 - It employs two main deep learning architectures:
 - Convolutional Neural Networks (CNN);
 - Recurrent Neural Networks (RNN).
 - **Downside:** They require much more training data than traditional machine learning algorithms.
 - Instead of, for example, 1,000 training samples, it is necessary to have millions of samples.

Hybrid Systems

- These are simply **combinations of machine learning classifiers and rule-based systems**, which improve results as you fine-tune rules.
- You can use these to rules to tweak topics that have been incorrectly modeled by the machine learning classifier.

Metrics and Evaluation

- As in many other classification tasks, in Topic Classification it is necessary to **test the actual label** (topic) for a specific text and **compare it to the predicted label** (topic).
- With the results, it is possible to compute the following (well-known) **evaluation metrics**:
 - **Accuracy**: the percentage of texts that were assigned the correct topic;
 - **Precision**: the percentage of texts the classifier tagged correctly out of the total number of texts it predicted for each topic;
 - **Recall**: the percentage of texts the model predicted for each topic out of the total number of texts it should have predicted for that topic;
 - **F1 Score**: the average of both Precision and Recall.