cct | College Dublin
Computing • IT • Business

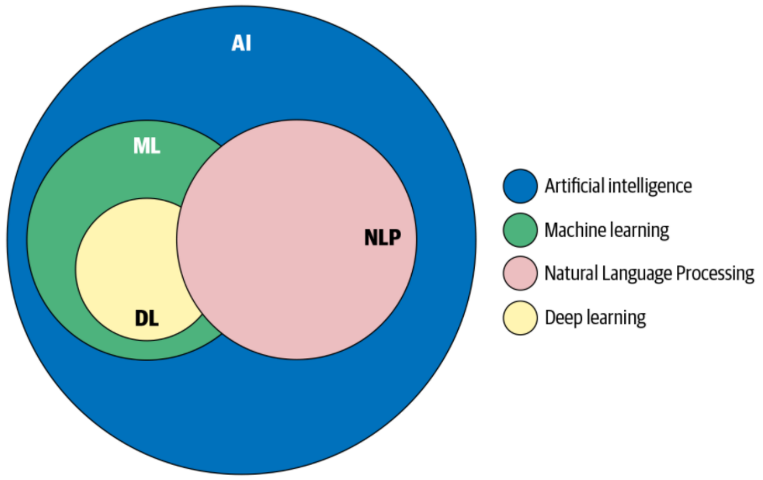
Machine Learning for Data Analysis
MSc in Data Analytics
CCT College Dublin

NLP and Topic Modelling
Week 8

Lecturer: Dr. Muhammad Iqbal*
Email: miqbal@cct.ie

Agenda

- Introduction to **Natural Language Processing (NLP)**
- NLP Pipeline
- Topic modelling
- Latent Dirichlet Allocation
- Latent Dirichlet Allocation: Example
- Decomposing Text Documents with LDA
- Sentiment Analysis
- Implementation of Sentiment Analysis



How NLP, ML, and DL are related

Introduction to NLP

- Let's imagine a hypothetical person called John Doe is the CEO of a fast-growing technology company. John wakes up every morning, has this conversation with his personal assistant.
- ***John:*** "How is the weather today?"
- ***Digital assistant:*** "It is 37 degrees centigrade outside with no rain today."
- ***John:*** "What does my schedule look like?"
- ***Digital assistant:*** "You have a strategy meeting at 4 p.m. and an all-hands at 5:30 p.m. Based on today's traffic situation, it is recommended you leave for the office by 8:15 a.m."
- *John inquires of the assistant about his fashion choices as he gets dressed:*
- ***John:*** "What should I wear today?"
- ***Digital assistant:*** "White seems like a good choice."
- You may have used Apple Siri, Google Home, or Amazon Alexa to accomplish similar tasks.
- The assistants that we have interacted with do not speak a programming language; instead, they speak English, the language in which we all communicate.

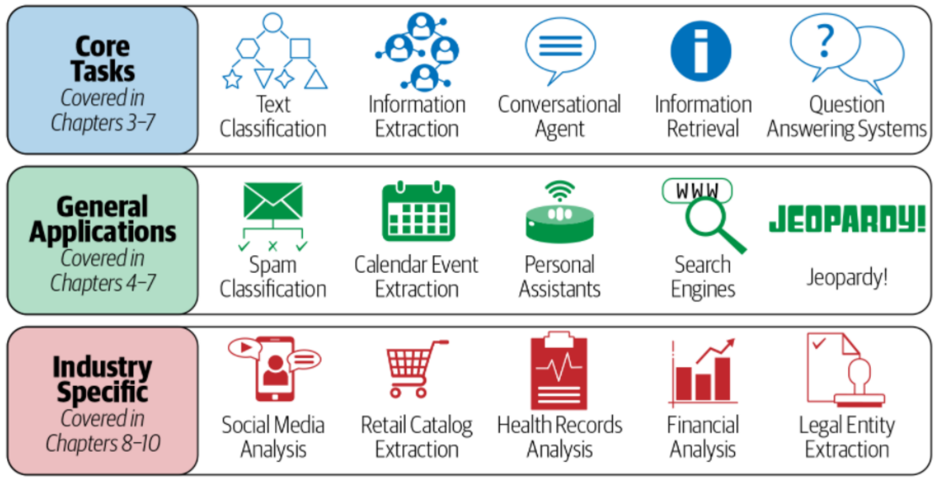
Introduction to NLP

- Smart assistants such as Amazon Alexa, Google Home, or Apple Siri to do similar things are commonly used in this world. We talk to these assistants in our natural language rather than programming language.

- This natural language has been the primary medium of communication between humans since time immemorial.

- Computers can process data in a binary format, such as **0s** and **1s**. While we can represent language data in binary.

- **How do we make machines to understand the language?**



NLP tasks and applications

- This is where **Natural Language Processing (NLP)** comes in. It is an area of computer science that deals with methods to **<u>analyze</u>**, **<u>model</u>**, and **<u>understand</u>** human language.

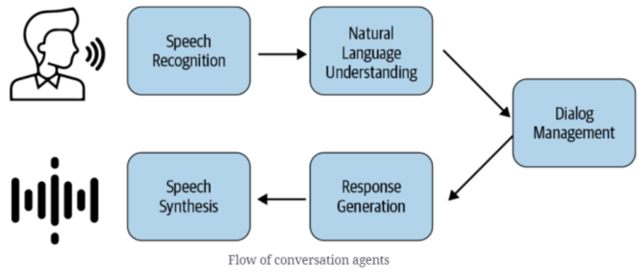
- Every intelligent application involving human language has some **NLP** behind it.

- A given **NLP** problem may be solved using dozens of alternative approaches because of the open-ended nature of **NLP**.

Introduction to NLP

- In today's area of internet and online services, data is generating at incredible **speed** and **amount**.
- **Data analysts, Engineers, and Scientists** deal with relational data or tabular data columns, which may contain numerical or categorical data.
- The data has a variety of structures such as **text**, **image**, **audio**, and **video**. Online activities such as articles, website text, blog posts, social media posts are generating unstructured textual data.
- **Corporate** and **Business** need to analyze textual data to understand customer activities, opinion, and feedback to successfully derive their business. To compete with big textual data, text analytics is evolving at a faster rate than ever before.
- **Text Analytics has lots of applications in today's online world. By analyzing tweets on Twitter, we can find trending news and peoples reaction on a particular event. Amazon can understand user feedback or review on the specific product.**

NLP Tasks



- There is a collection of fundamental tasks that appear frequently across various NLP projects.
  - **Topic modelling**
  - **Text Classification**
  - Information Extraction
  - Information Retrieval
  - Conversational agent
  - Text summarization
  - Question answering
  - Machine translation

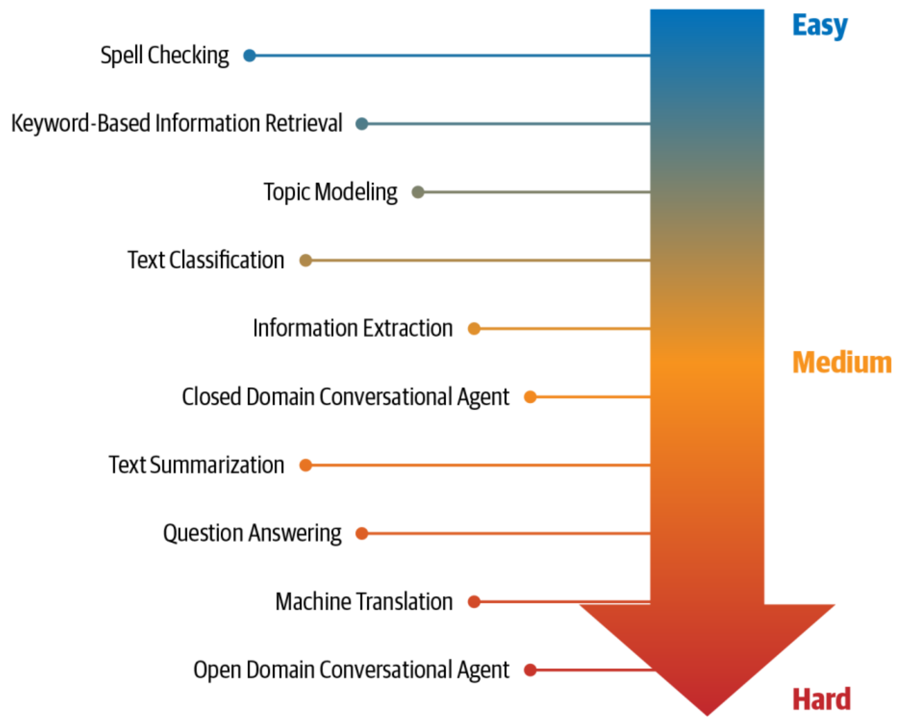
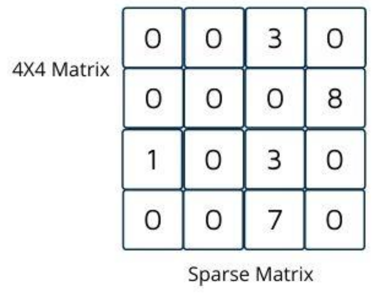


Figure shows a depiction of these tasks based on their relative difficulty in terms of developing comprehensive solutions.

Learning from Text



0	0	3	0
0	0	0	8
1	0	3	0
0	0	7	0

- Most machine learning applications in the text domain work with the **bag-of-words** representation in which the **words** are treated as **dimensions** with values corresponding to **word frequencies**.

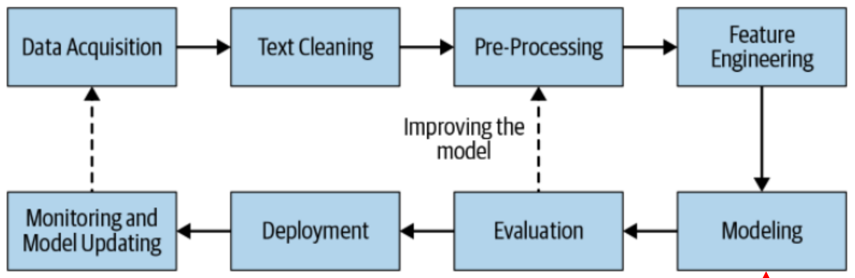
- A **data set** corresponds to a collection of documents, which is also referred to as a ***corpus***. The complete and distinct set of words used to define the **corpus** is referred to as the ***lexicon***.

- **Dimensions** are also referred to as **Terms** or **Features**. Some applications of text work with a binary representation in which the presence of a term in a document corresponds to a value of **1** and **0**.

- Other applications use a normalized function of the word frequencies as the values of the dimensions. In each of these cases, the dimensionality of data is very large, and may be of the order of **10^5** or even **10^6**.

- Furthermore, most values of the dimensions are **0s**, and only a few dimensions take on positive values. In other words, text is a ***high-dimensional, sparse, and non-negative*** representation.

NLP Pipeline



Generic NLP pipeline

- The first step in the process of developing any **NLP system** is to collect data relevant to the given task. We convert the text into **canonical form** after cleaning and pre-processing.
- **Feature Engineering** is the process of using domain knowledge to extract features (characteristics, properties and attributes) from raw data.
- In **modeling and evaluation** phases, we build one or more models and compare and contrast them using a relevant evaluation metric(s).
- Once the best model among the ones evaluated is chosen, we move toward deploying this model in production.
- In the real world, the process may not always be linear as it's shown in the pipeline in Figure.
- It involves going back and forth between individual steps (e.g., between feature extraction and modeling, evaluation, and so on). There are loops in between, most commonly going from **evaluation** to **pre-processing**, **feature engineering**, **modeling**, and back to **evaluation**.
- In addition, there is an overall loop involving monitoring and data collection, but this loop can be seen at the project level.

Bag-of-words Model

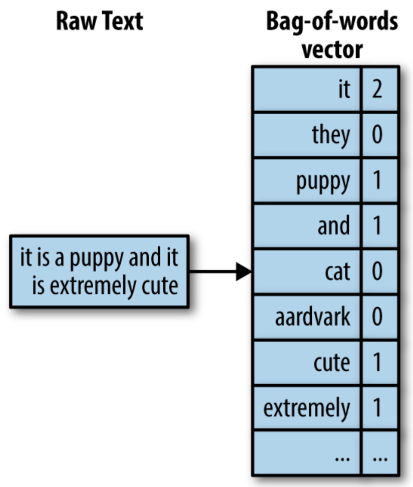
- We have to turn categorical data, such as text or words, into a numeric form before we can pass them on to a machine learning algorithm for the training.

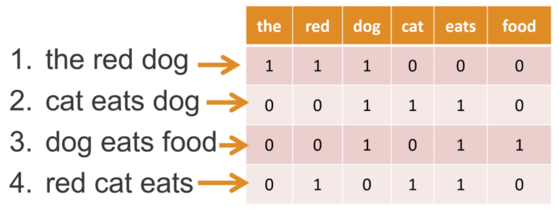
- We introduce the ***bag-of-words***, which allows us to represent text as numeric feature vectors. The idea behind the ***bag-of-words model*** is quite simple and can be summarized as follows

  1. We create a vocabulary of unique tokens, for example, words: from the entire set of documents.

  2. We construct a feature vector from each document that contains the counts of how each word occurs in the particular document.

- Since the unique words in each document represent only a small subset of all the words in the bag-of-words vocabulary, the feature vectors will mostly consist of zeros, which is why we call them sparse.





	the	red	dog	cat	eats	food
1. the red dog →	1	1	1	0	0	0
2. cat eats dog →	0	0	1	1	1	0
3. dog eats food →	0	0	1	0	1	1
4. red cat eats →	0	1	0	1	1	0

Feature Generation

Bag of Words

- In the **Text Classification**, we have a set of texts and their respective labels. But we can't directly use text for our model. We need to convert these text into some numbers or vectors of numbers.

- **Bag-of-words Model (BoW)** is the simplest way of extracting features from the text. **BoW** converts text into the matrix of occurrence of words within a document. This model concerns about whether the given words occurred or not in the document.

- **Example:** There are three documents

- **Doc 1:** I love dogs. **Doc 2:** I hate dogs and knitting. **Doc 3:** Knitting is my hobby and passion.

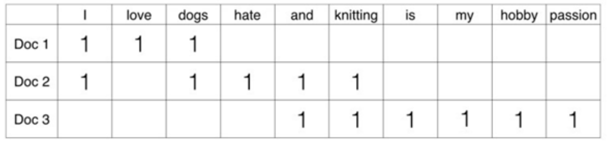
- We can create a matrix of document and words by counting the occurrence of words in the given document. This matrix is known as **Document-Term Matrix (DTM).**

	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	1	1	1							
Doc 2	1		1	1	1	1				
Doc 3					1	1	1	1	1	1

```
from sklearn.feature_extraction.text import CountVectorizer
```

Feature Generation

TF-IDF



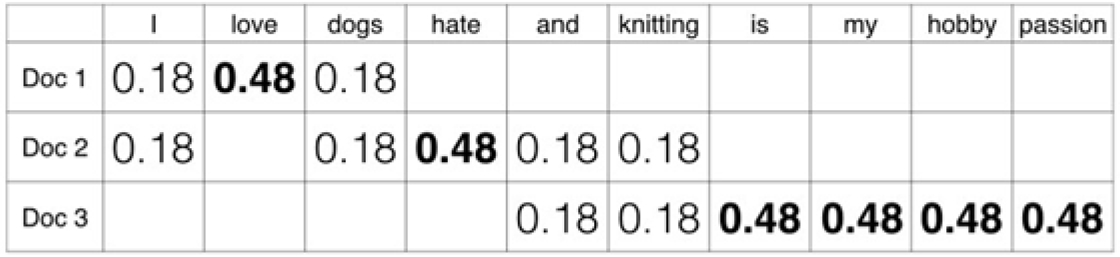
	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	1	1	1							
Doc 2	1		1	1	1	1				
Doc 3					1	1	1	1	1	1

- In **Term Frequency (TF)**, we count the number of words occurred in each document. The main issue with this Term Frequency is that it will give more weight to longer documents. Term frequency is basically the output of the **BoW model**.

- **IDF (Inverse Document Frequency)** measures the amount of information a given word provides across the document. **IDF** is the logarithmically scaled inverse ratio of the number of documents that contain the word and the total number of documents.

$$\mathrm{idf}(\mathrm{W}) = \log \frac{\#(\mathrm{documents})}{\#(\mathrm{documents\ containing\ word\ W})}$$

- **TF-IDF (Term Frequency-Inverse Document Frequency)** normalizes the document term matrix. It is the product of **TF** and **IDF**. Word with high **tf-idf** in a document, it is most of the times occurred in given documents and must be absent in the other documents. So, the words must be a signature word.



	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	0.18	**0.48**	0.18							
Doc 2	0.18		0.18	**0.48**	0.18	0.18				
Doc 3					0.18	0.18	**0.48**	**0.48**	**0.48**	**0.48**

What is Topic Modeling?

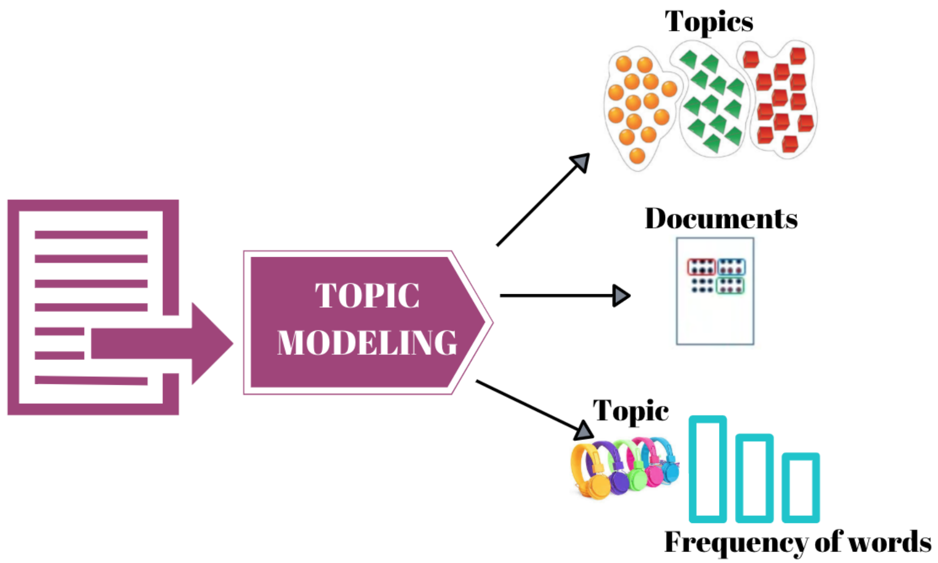
- **Topic modeling** is an unsupervised technique that intends to analyze large volumes of text data by assigning topics to the documents and segregate the documents into groups based on the assigned topics.

- In the case of topic modeling, the text data do not have any labels attached to it. Rather, topic modeling tries to group the documents into based on similar topics.

- A typical example of topic modeling is clustering a large number of newspaper articles that belong to the same category.

- In other words, cluster documents that have the same topic. It is important to mention that it is extremely difficult to evaluate the performance of topic modeling since there are no right answers.

- It depends upon the user to find similar characteristics between the documents of one cluster and assign it an appropriate label or topic.



Latent Dirichlet Allocation

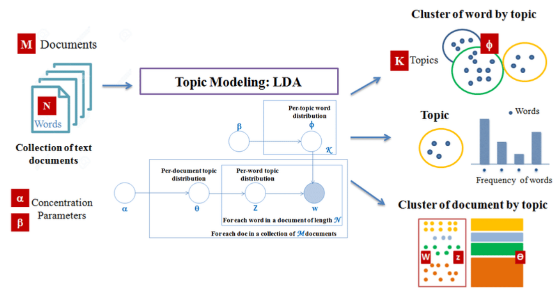
Decomposing Text Documents

Latent Dirichlet Allocation (LDA)

- **LDA** is a generative probabilistic model that tries to find groups of words that appear frequently together across different documents.

- These frequently appearing words represent our topics, assuming that each document is a mixture of different words.

- The input to an **LDA** is the bag-of-words model. We can take a bag-of-words matrix as input, **LDA** decomposes it into two new matrices as
  - **A document to topic matrix**
  - **A word to topic matrix**



M Documents
N Words
Collection of text documents
α β Concentration Parameters
Topic Modeling: LDA
Per-topic word distribution
β
ϕ
K
Per-document topic distribution
Per-word topic distribution
α
θ
z
w
For each word in a document of length N
For each doc in a collection of M documents
Cluster of word by topic
K Topics
ϕ
Topic
Words
Frequency of words
Cluster of document by topic
W
z
θ

- **LDA** decomposes the **bag-of-words** matrix in such a way that if we multiply those two matrices together, we would be able to reproduce the **input**, the **bag-of-words** matrix, with the lowest possible error.

- In practice, we are interested in those topics that **LDA** found in the **bag-of-words** matrix. For **LDA**, we must define the number of topics beforehand.

Latent Dirichlet Allocation (LDA)

Example

- We consider the following 5 documents:
    - **Document 1:** "cat dog cat"
    - **Document 2:** "dog bird"
    - **Document 3:** "fish bird fish"
    - **Document 4:** "apple banana"
    - **Document 5:** "banana apple banana"

- **Step 1: Preprocessing**
- Tokenize each document: split the documents into individual words.
- Create a vocabulary: collect all unique words in the corpus.
- **Vocabulary: ["cat", "dog", "bird", "fish", "apple", "banana"]**

Latent Dirichlet Allocation (LDA)

Example

- **Step 2:** Create a Document-Term Matrix (DTM)
- Create a matrix where each row represents a document, and each column represents a word. The entries are the word frequencies in each document.

– Document 1: "cat dog cat"

– Document 2: "dog bird"

– Document 3: "fish bird fish"

– Document 4: "apple banana"

– Document 5: "banana apple banana"

	cat	dog	bird	fish	apple	banana
Doc 1	2	1	0	0	0	0
Doc 2	0	1	1	0	0	0
Doc 3	0	0	1	2	0	0
Doc 4	0	0	0	0	1	1
Doc 5	0	0	0	0	2	1

Latent Dirichlet Allocation (LDA)

Example

- **Step 3: Apply LDA**
- Suppose we consider two topics (K = 2) and running the LDA algorithm on the DTM, we get two matrices as mentioned
- **Document-Topic Matrix (A document to topic matrix):** This matrix represents the probability of each document belonging to each topic as shown in left Table.

	Topic 1	Topic 2
Doc 1	0.2	0.8
Doc 2	0.9	0.1
Doc 3	0.3	0.7
Doc 4	0.8	0.2
Doc 5	0.1	0.9

	Topic 1	Topic 2
cat	0.8	0.2
Dog	0.2	0.8
Bird	0.7	0.3
Fish	0.1	0.9
apple	0.9	0.1
banana	0.4	0.6

- The **LDA** method calculates these probabilities by looking at the co-occurrence patterns of terms in topics and documents as shown in Table on right.
- They show the probability that a word associated with a particular topic will appear in a document. These matrices have values between 0 and 1, which add up to 1 for every word or page.

LDA

Scikit-learn

- We use the **Latent Dirichlet Allocation** class implemented in **scikit-learn** to decompose the movie review dataset and categorize it into different topics.

- In the following example, we restrict the analysis to 10 different topics.

- We are going to load the dataset into a pandas DataFrame using the local **movie_data.csv** file of the movie .

```
>>> import pandas as pd
>>> df = pd.read_csv('movie_data.csv', encoding='utf-8')
```

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> count = CountVectorizer(stop_words='english',
...                         max_df=.1,
...                         max_features=5000)
>>> X = count.fit_transform(df['review'].values)
```

```
>>> from sklearn.decomposition import LatentDirichletAllocation
>>> lda = LatentDirichletAllocation(n_topics=10,
...                                 random_state=123,
...                                 learning_method='batch')
>>> X_topics = lda.fit_transform(X)
```

- We set the maximum document frequency of words to be considered to **10 percent** (max_df = 0.1) to exclude words that occur too frequently across documents. The rationale behind the removal of frequently occurring words is that these might be common words appearing across all documents and are less likely associated with a specific topic category of a given document.

- We limited the number of words to be considered to the most frequently occurring 5,000 words (max_features = 5000), to limit the dimensionality of this dataset, so that it improves the inference performed by **LDA**.

LDA

Scikit-learn Coding

- The code example demonstrates how to fit a **LatentDirichletAllocation** estimator to the bag-of-words matrix and infer 10 different topics from the documents.

```
>>> from sklearn.decomposition import LatentDirichletAllocation
>>> lda = LatentDirichletAllocation(n_topics=10,
...                                 random_state=123,
...                                 learning_method='batch')
>>> X_topics = lda.fit_transform(X)
```

- By setting learning_method = 'batch', we allow the **lda** estimator do its estimation based on all available training data (the bag-of-words matrix) in one iteration, which is slower than the alternative 'online' learning method but can lead to more accurate results (setting learning_method = 'online' is analogous to online or mini-batch learning).

- After fitting **LDA**, we have access to the **components_** attribute of the **lda** instance, which stores a matrix containing the word importance (here, 5000) for each of the 10 topics in increasing order.

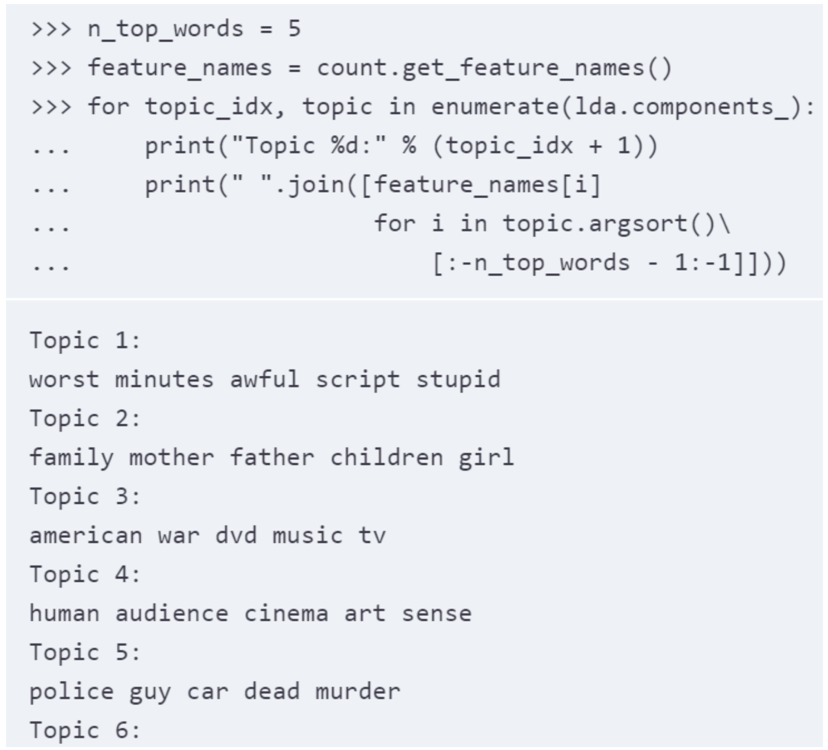
```
>>> lda.components_.shape
(10, 5000)
```

LDA

Scikit-learn Coding

- To analyze the results, we print five most important words for each of **10 topics**. The word importance values are ranked in increasing order.

- To print **top five words**, we need to sort the topic array in reverse order. Based on reading the five most important words for each topic, we may guess that **LDA** identified the following topics as
  - Generally bad movies (not really a topic category)
  - Movies about families
  - War movies
  - Art movies
  - Crime movies
  - Horror movies
  - Comedy movies
  - Movies somehow related to TV shows
  - Movies based on books
  - Action movies



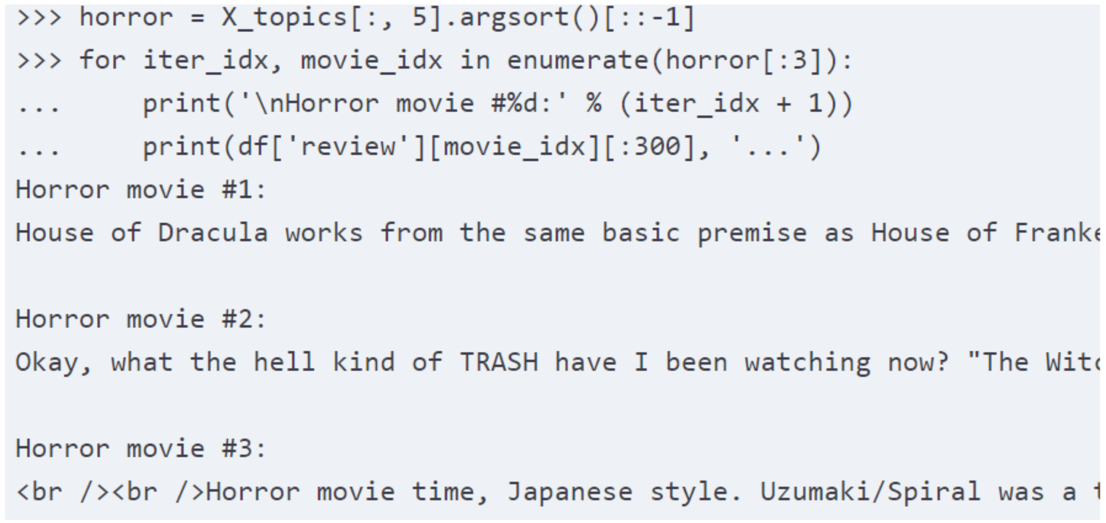
```
>>> n_top_words = 5
>>> feature_names = count.get_feature_names()
>>> for topic_idx, topic in enumerate(lda.components_):
...     print("Topic %d:" % (topic_idx + 1))
...     print(" ".join([feature_names[i]
...                     for i in topic.argsort()\
...                         [:-n_top_words - 1:-1]]))
```

```
Topic 1:
worst minutes awful script stupid
Topic 2:
family mother father children girl
Topic 3:
american war dvd music tv
Topic 4:
human audience cinema art sense
Topic 5:
police guy car dead murder
Topic 6:
```

LDA

Scikit-learn Coding

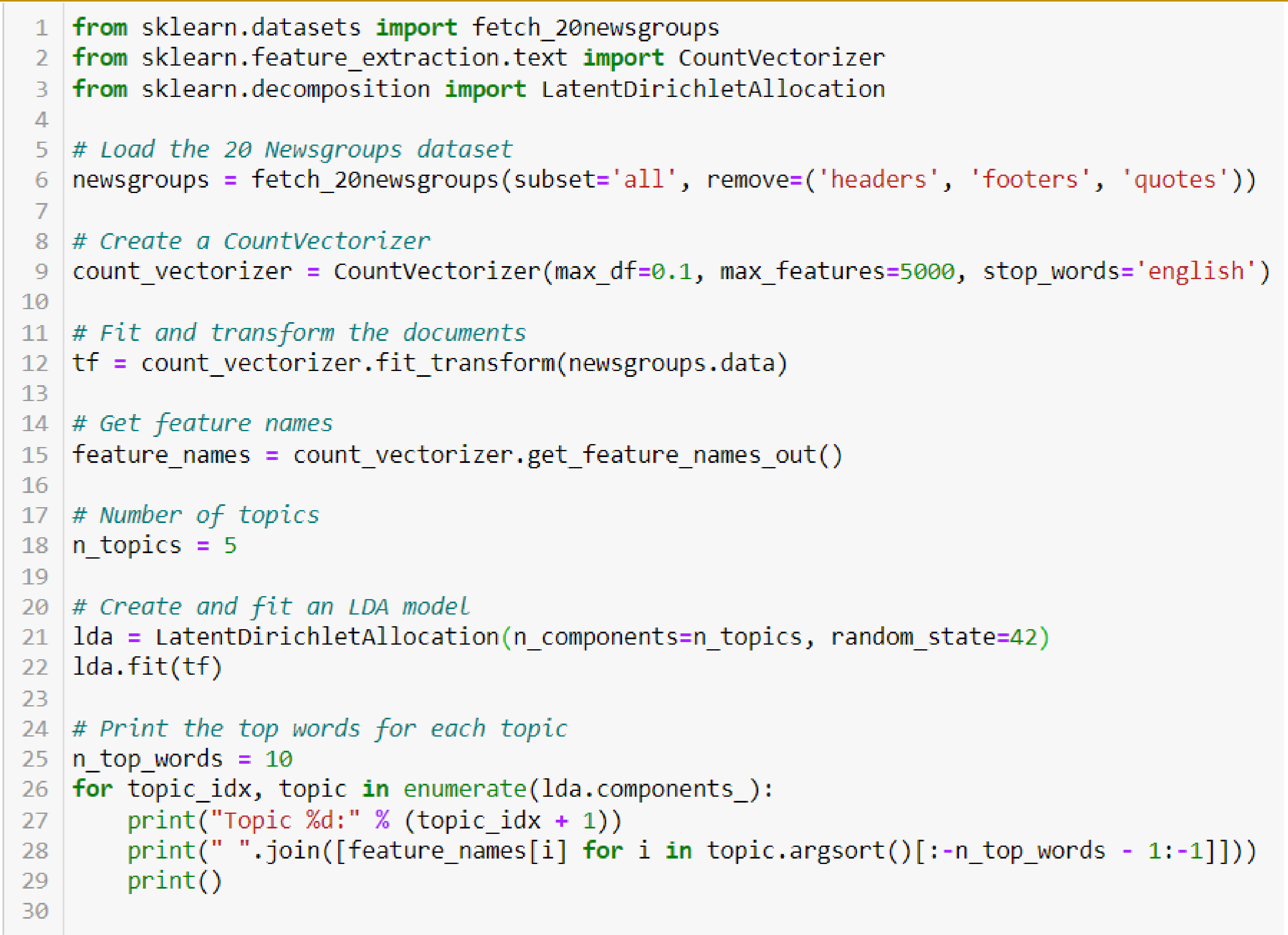
- To confirm that the categories make sense based on the reviews, we plot three movies from the horror movie category (horror movies belong to category 6 at index position 5)



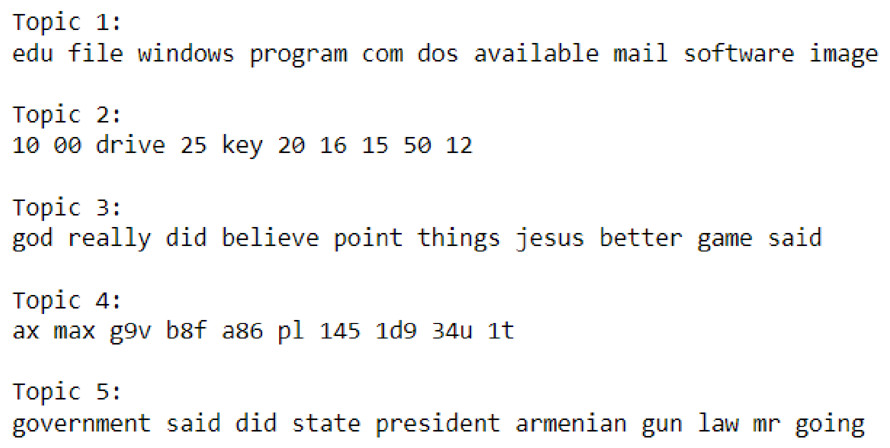
```
>>> horror = X_topics[:, 5].argsort()[::-1]
>>> for iter_idx, movie_idx in enumerate(horror[:3]):
...     print('\nHorror movie #%d:' % (iter_idx + 1))
...     print(df['review'][movie_idx][:300], '...')
Horror movie #1:
House of Dracula works from the same basic premise as House of Franke

Horror movie #2:
Okay, what the hell kind of TRASH have I been watching now? "The Wit

Horror movie #3:
<br /><br />Horror movie time, Japanese style. Uzumaki/Spiral was a
```

- Using the code example, we printed the first 300 characters from the top three horror movies, and we can see their reviews.
- We don't know which exact movie they belong to, sound like reviews of horror movies (however, one might argue that Horror movie #2 could also be a good fit for topic category 1: Generally bad movies).

LDA
Scikit-learn Coding



```python
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

# Load the 20 Newsgroups dataset
newsgroups = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))

# Create a CountVectorizer
count_vectorizer = CountVectorizer(max_df=0.1, max_features=5000, stop_words='english')

# Fit and transform the documents
tf = count_vectorizer.fit_transform(newsgroups.data)

# Get feature names
feature_names = count_vectorizer.get_feature_names_out()

# Number of topics
n_topics = 5

# Create and fit an LDA model
lda = LatentDirichletAllocation(n_components=n_topics, random_state=42)
lda.fit(tf)

# Print the top words for each topic
n_top_words = 10
for topic_idx, topic in enumerate(lda.components_):
    print("Topic %d:" % (topic_idx + 1))
    print(" ".join([feature_names[i] for i in topic.argsort()[:-n_top_words - 1:-1]]))
    print()

```

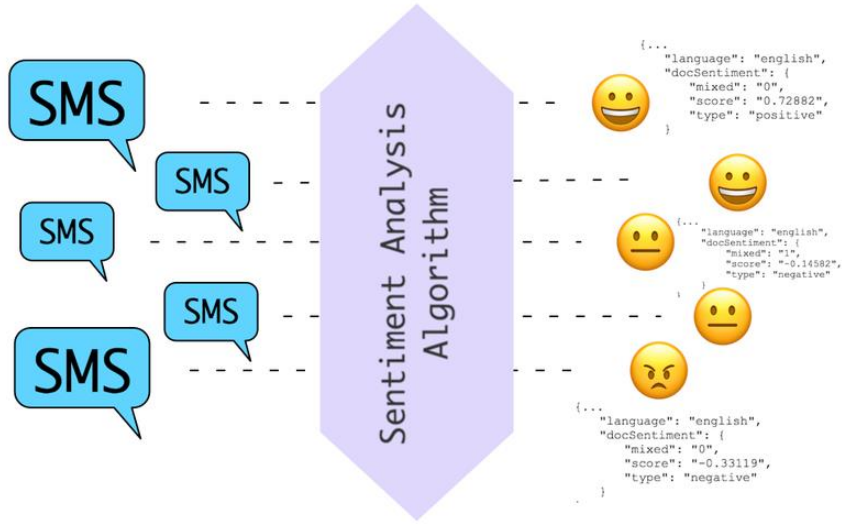


```
Topic 1:
edu file windows program com dos available mail software image

Topic 2:
10 00 drive 25 key 20 16 15 50 12

Topic 3:
god really did believe point things jesus better game said

Topic 4:
ax max g9v b8f a86 pl 145 1d9 34u 1t

Topic 5:
government said did state president armenian gun law mr going
```

Sentiment Analysis

- In this internet and social media age, people's opinions, reviews, and recommendations have become a valuable resource for political science and businesses.
- We delve into a subfield of **Natural Language Processing (NLP)** called sentiment analysis and learn how to use machine learning algorithms to classify documents based on their polarity: the attitude of the writer.
- We are going to work with a dataset of 50,000 movie reviews from the **Internet Movie Database (IMDb)** and build a predictor that can distinguish between positive and negative reviews.
- **Sentiment Analysis** also called opinion mining, is a popular subdiscipline of the broader field of NLP.
- It is concerned with analyzing the polarity of documents. ***A popular task in sentiment analysis is the classification of documents based on the expressed opinions or emotions of the authors with regard to a particular topic.***

We are able to collect and analyze such data most efficiently.



```
{...
   "language": "english",
   "docSentiment": {
      "mixed": "0",
      "score": "0.72882",
      "type": "positive"
   }
```

```
{...
   "language": "english",
   "docSentiment": {
      "mixed": "1",
      "score": "-0.14582",
      "type": "negative"
   }
```

```
{...
   "language": "english",
   "docSentiment": {
      "mixed": "0",
      "score": "-0.33119",
      "type": "negative"
   }
```

VADER sentiment analysis with NLTK

- For the English language, NLTK provides an already trained model called **Valence Aware Dictionary and sEntiment Reasoner (VADER)** which works in a slightly different way and adopts a rule engine together with a lexicon to infer the sentiment intensity of a piece of text.
- The NLTK version uses the **SentimentIntensityAnalyzer class** and can immediately be used to have a polarity sentiment measure made up of four components:
    1. Positive factor
    2. Negative factor
    3. Neutral factor
    4. Compound factor
- The first three don't need any explanation, while the last one is a particular measure (a normalized overall score), which is computed as follows:

$$Compound = \frac{\sum_i Sentiment(w_i)}{\sqrt{(\sum_i Sentiment(w_i))^2 + \alpha}}$$

- Here, Sentiment(w_i) is the score valence of the word w_i and α is a normalization coefficient that should approximate the maximum expected value (the default value set in NLTK is 15).

Resources/ References

- Python Machine Learning, Sebastian Raschka, Vahid Mirjalili, Packt Publishing, September 2017.
- Practical Natural Language Processing, Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta, Harshit Surana, June 2020, O'Reilly Media, Inc., ISBN: 9781492054054.
- Introduction to Machine Learning with Python A Guide for Data Scientists, Andreas C. Müller and Sarah Guido, Copyright © 2017, O'Reilly.
- Machine Learning with Python Cookbook, Chris Albon, Publisher: O'Reilly Media, Inc., Release Date: March 2018, ISBN: 9781491989388.
- https://www.datacamp.com/community/tutorials/text-analytics-beginners-nltk