



ECOM SCHOOL

המכללה למקצועות הדיגיטל וההייטק

Last lecture reminder



We learned about:

- SpaCy - Part of Speech (POS)
- SpaCy - Named Entity Recognition (NER)
- SpaCy - NER Customization
- SpaCy - Sentence Segmentation & Segmentation Rule
- Text Classification - Introduction
- Text Classification - Feature Extraction (Bag of Words & TF-IDF)

Naive Bias - Introduction

Naive Bayes → Naive Bayes is a highly effective supervised learning model, particularly in the context of Natural Language Processing (NLP). Its effectiveness stems from its simplicity and efficiency, making it well-suited for tasks such as text classification, spam detection, sentiment analysis, and document categorization.

Naive Bayes is a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

Bayes' Theorem → Bayes' theorem calculates the probability of a new events occurring given some past events occurrence.

Mathematically, it's represented by the following formula:

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



Naive Bias - Introduction

Let's explain the formula →

- **$P(A|B)$** → The probability of event A happening given event B been happened.
- **$P(B|A)$** → The probability of event B happening given event A been happened.
- **$P(A)$** → The probability of event A happening.
- **$P(B)$** → The probability of event B happening.

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

So according to the Bayes' theorem, in case we have the probability of $B|A \rightarrow P(B|A)$, we can calculate the probability of $A|B \rightarrow P(A|B)$.

Naive Bias - Simple Example

Let's now take a look on a real life example to better understand how the Bayes' theorem works.

Let's say we are living in a city which each building has its own fire alarm system.

Once in a while the system produce false alarm meaning that smoke was detected but there is no actual fire in the building. For example → Smoke that coming from the oven can make such a false alarm.

Now let's consider the known probabilities that we have for the fire system:

- Real fires occur only 1% of the time.
- Smoke alarms are been activated 10% of the time.
- The smoke alarm is triggered on real fires 95% of the time (meaning the 5% of real fire cases will not cause the smoke alarm to be triggered)

According to the following data, in case the fire alarm was triggered what is the probability that there is an actual real fire situation and it's not a false alarm?



Naive Bias - Simple Example

So what we have here in terms of probability events are:

- Event A \rightarrow Actual fire event
- Event B \rightarrow Smoke alarm been triggered
- $P(A / B) \rightarrow$ The probability of actual fire event given smoke alarm was triggered
- $P(B / A) \rightarrow$ The probability of smoke alarm been triggered event given an actual fire

Now, let's put the probability data that we know in the Bayes formula:

- $P(A) = 1/100 \rightarrow$ Probability of actual fire event is 1%
- $P(B) = 1/10 \rightarrow$ Probability of smoke alarm been triggered 10%
- $P(B / A) = P(\text{Alarm} / \text{Fire}) = 95/100 \rightarrow$ The smoke alarm is triggered on real fires 95% of the time

$$P(A / B) = P(\text{Fire} / \text{Alarm}) = (P(B / A) * P(A)) / P(B) \Rightarrow (0.95 * 0.01) / 0.1 = 0.095 \Rightarrow 9.5\%$$

We got that the probability of an actual fire event given the smoke alarm was triggered is 0.095

Class Exercise - Naive Bias

Instructions:

Imagine we have a medical test for a rare disease, with the following probabilities:

- The disease is quite rare and only occurs in 0.5% (0.005) of the population.
- The test is quite reliable, it correctly identifies those with the disease 99% (0.99) of the time.
- However, the test also produces false positives 2% (0.02) of the time (meaning 2% of the people who do not have the disease will also test positive).

Given this information, we want to calculate the probability that a person has the disease given that they tested positive. Show your calculations in the answer.

Class Exercise Solution - Naive Bias



Naive Bias In NLP

As mentioned, Naive Bias can be very useful in NLP classification. NLP classification task is basically classify a text file given the provided tokens (words).

So we can think about the Naive Bias formula in NLP classification task as the following:

$$\mathbf{x} = (x_1, \dots, x_n)$$

$$P(A | B) = \frac{P(B | A) \cdot P(A)}{P(B)} \quad \Rightarrow \quad p(C_k | \mathbf{x}) = \frac{p(C_k) p(\mathbf{x} | C_k)}{p(\mathbf{x})}$$

Given a vector of features (tokens) \mathbf{x} , we want to calculate the probability of belonging the class K .

After some mathematics actions on the formula we will get the following formula:

$$p(C_k | x_1, \dots, x_n) \propto p(C_k) \prod_{i=1}^n p(x_i | C_k)$$

The probability of belonging to class K given x_1, x_2, \dots, x_n features

Proportional symbol

The probability of belong to class K multiply by the multiplication of the probabilities of getting x_i given belonging to class K

9

Naive Bias In NLP

Why Naive Bias algorithms are useful in NLP classification tasks?

- **Simplified Assumption** → Naive Bayes operates on the assumption that the features (words in NLP) are conditionally independent given the class. While this assumption is not always true in practice, it simplifies the computation and often still yields good performance in text classification tasks.
- **Efficiency** → Naive Bayes is computationally efficient both in terms of training and prediction making. This makes it suitable for scenarios where rapid classification decisions are crucial, and for applications involving large datasets.
- **Handling of High-Dimensional Data** → Text data is inherently high-dimensional as every unique word can be considered a feature. Naive Bayes can handle high-dimensional data without much difficulty, making it well-suited for text classification.
- **Robust to Irrelevant Features** → Naive Bayes is relatively robust to irrelevant features because irrelevant features do not significantly impact the overall probability calculation due to the independence assumption.




Naive Bias In NLP - Simple Example


Let's take a look on a simple NLP classification problem and see how Naive Bias can help us:

Consider that we have 35 different movies reviews, each review contain the actual review text and a label that classified this review as positive or negative.

We want to train a Naive Bias model that will be able to predict for future reviews if they are positive or negative reviews.

First we will split our reviews for the 2 different classes (positive and negative) and calculate the probability to belong to each class:


$$P(\text{pos}) = 25/35$$


$$P(\text{neg}) = 10/35$$

Naive Bias In NLP - Simple Example

Next, we will perform count vectorization on each class:



10	2	8	4
movie	actor	great	film



8	10	0	2
movie	actor	great	film

Next, we will calculate the probability of get the specific token given the review is belong to that class:

$$P(\text{movie}|\text{pos}) = 10/24 = 0.42$$

$$P(\text{actor}|\text{pos}) = 2/24 = 0.08$$

$$P(\text{great}|\text{pos}) = 8/24 = 0.33$$

$$P(\text{film}|\text{pos}) = 4/24 = 0.17$$



0.42	0.08	0.33	0.17
10	2	8	4
movie	actor	great	film



Naive Bias In NLP - Simple Example

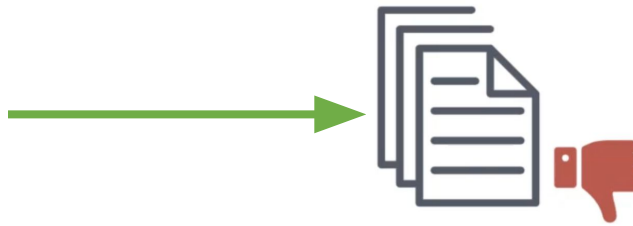
We will perform the same calculation on the negative class as well:

$$P(\text{movie}|\text{neg}) = 8/20 = 0.4$$

$$P(\text{actor}|\text{neg}) = 10/20 = 0.5$$

$$P(\text{great}|\text{neg}) = 0/20 = 0$$

$$P(\text{film}|\text{neg}) = 2/20 = 0.1$$



0.4	0.5	0	0.1
8	10	0	2
movie	actor	great	film

Now let's say we got a new movie review → “movie actor”, we want to predict if this review is a positive or not.

So basically what we want is to calculate the probability of belong to class “positive” or “negative” given the feature vector “movie actor”. $P(\text{positive} / \text{“movie actor”})$

According to the Naive Bias formula:

$$P(\text{positive} / \text{“movie actor”}) = P(\text{positive}) * P(\text{“movie”} / \text{positive}) * P(\text{“actor”} / \text{positive})$$

$$= (0.71) * (0.42) * (0.08) = 0.024$$

$$P(\text{negative} / \text{“movie actor”}) = P(\text{negative}) * P(\text{“movie”} / \text{negative}) * P(\text{“actor”} / \text{negative})$$

$$= (10 / 35) * (0.4) * (0.5) = 0.057$$

Naive Bias In NLP - Simple Example

What we left to do is to choose the class with the higher probability score and this will be our prediction.

So in our example the predicted class for the review “movie actor” will be **Negative**.

Now let's consider another review → “**great movie**”.

If we will calculate the probability that “great movie” will belong to the negative class we will get 0.

This is Because according to the count vectorization the word “great” appears 0 times in negative reviews:

$$P(\text{negative} / \text{“great movie”}) = (10 / 35) * 0 * 0.4 = 0$$



0.4	0.5	0	0.1
8	10	0	2
movie	actor	great	film

So how can we fix this issue?

We can fix it by using an **alpha smoothing parameter** which just mean adding an alpha value

(in our case add 1) to all the counts in the count vectorization. This way we won't have any token with 0 counts.



Naive Bias In NLP - Python Example

Now that we understand how Naive Bias is working, let's see an actual Python example.

For the example we will use the **'airline_tweets.csv'** file. This file contain data about customers tweets in the social network twitter (X now), the tweet id, the tweet sentiment predicate by the airline company, the tweet text, etc...

Our mission is to use supervised learning Naive Bias model to get the raw tweet text and predict if that tweet sentiment is positive, natural or negative.

First, we want to examine the data provided:

```
In [2]: df = pd.read_csv('/Users/ben.meir/Downloads/airline_tweets.csv')
df.head()
```

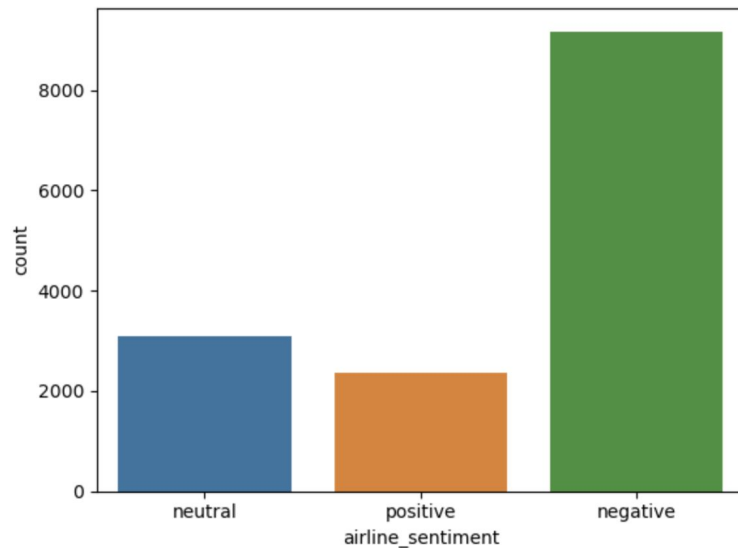
Out[2]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name
0	570306133677760513	neutral	1.0000	NaN	NaN	Virgin America	NaN	cairdin
1	570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	NaN	jnardino
2	570301083672813571	neutral	0.6837	NaN	NaN	Virgin America	NaN	yvonnalynn
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virgin America	NaN	jnardino
4	570300817074462722	negative	1.0000	Can't Tell	1.0000	Virgin America	NaN	jnardino

Naive Bias In NLP - Python Example

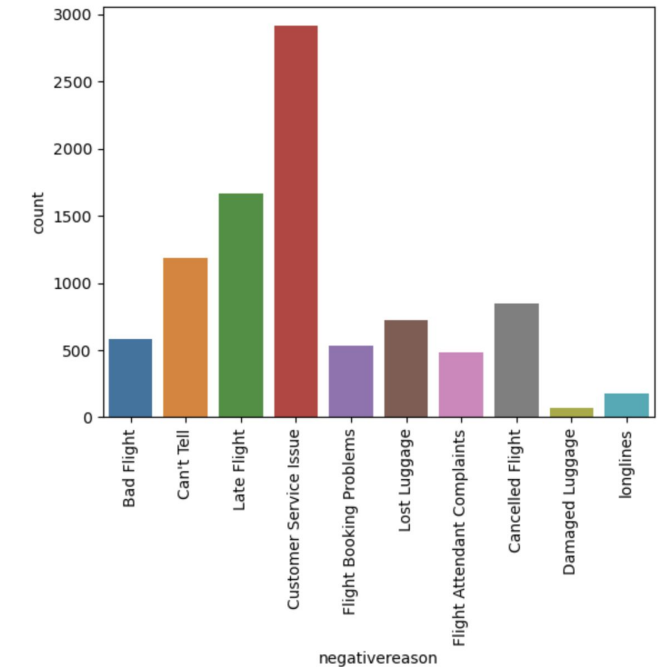
Next, we want to perform some data exploration:

```
In [3]: sns.countplot(data=df, x='airline_sentiment')  
plt.show()
```



Count how many data points we have for each label (tweet sentiment).
We can see that most of the tweets are with negative sentiment

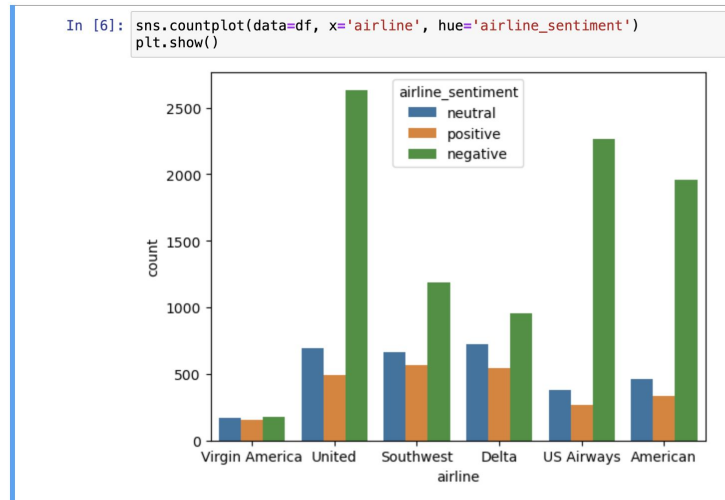
```
In [4]: sns.countplot(data=df, x='negative_reason')  
plt.xticks(rotation=90)  
plt.show()
```



Count what are the different reasons the customers gave for their negative tweets

Naive Bias In NLP - Python Example

We can also explore the tweet sentiment on the different airline companies:



We can see that the most negative sentiment tweets coming from the United airline company

Now let's start preparing our data for training, in this case we want to use only the raw tweet text data and the label will be the 'airline_sentiment' column.

```
In [7]: data = df[['airline_sentiment', 'text']]
data.head()
```

Out[7]:

	airline_sentiment	text
0	neutral	@VirginAmerica What @dhepburn said.
1	positive	@VirginAmerica plus you've added commercials t...
2	neutral	@VirginAmerica I didn't today... Must mean I n...
3	negative	@VirginAmerica it's really aggressive to blast...
4	negative	@VirginAmerica and it's a really big bad thing...

Naive Bias In NLP - Python Example

Now, let's separate our dataset into features and labels and perform train / test split. The reason we want to perform the train / test split before the vectorization is to avoid data leakage.

```
In [14]: from sklearn.model_selection import train_test_split

X = data['text']
y = data['airline_sentiment']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Next, let's perform the TF-IDF vectorization on the 'text' feature column:

```
In [15]: from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english')

tfidf_vectorizer.fit(X_train)
X_train_tfidf = tfidf_vectorizer.transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

The 'stop_words' parameter is used to specify a list of words (stop words) that should be ignored (excluded) during the vectorization process because we don't want to give them any importance. Stop words examples are 'is', 'the', etc...

We are doing the fit only on the training dataset and the transform on both the training and the test datasets.

Naive Bias In NLP - Python Example

Once we have our dataset ready, let's create an instance of the Naive Bias model we want to train.

In this case we will use the **MultinomialNB** model.

MultinomialNB model → MultinomialNB is an implementation of the Naive Bayes algorithm for classification tasks, which is particularly suitable for discrete data. This classifier is commonly used for text classification problems like spam detection or sentiment analysis.

```
In [17]: from sklearn.naive_bayes import MultinomialNB  
naive_bias = MultinomialNB()  
naive_bias.fit(X_train_tfidf, y_train)
```

```
Out[17]: ▼ MultinomialNB  
MultinomialNB()
```

We train the MultinomialNB model

```
In [18]: from sklearn.linear_model import LogisticRegression  
logistic_reg = LogisticRegression(max_iter=1000)  
logistic_reg.fit(X_train_tfidf, y_train)
```

```
Out[18]: ▼ LogisticRegression  
LogisticRegression(max_iter=1000)
```

We also train another classifier model (logistic regression) for comparison

Naive Bias In NLP - Python Example

Finally, let's examine our models accuracy and compare them to each other, for that we will build a dedicated Python function that will print for each model its performance metrics.

```
In [ ]: from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

def print_metrics(model):
    predictions = model.predict(X_test_tfidf)
    print(accuracy_score(y_test, predictions))
    print()
    print(classification_report(y_test, predictions))
    print()
    print(confusion_matrix(y_test, predictions))
```

```
In [20]: print_metrics(naive_bias)
```

0.6935336976320583

	precision	recall	f1-score	support
negative	0.68	0.99	0.81	2814
neutral	0.75	0.14	0.23	884
positive	0.94	0.18	0.30	694
accuracy			0.69	4392
macro avg	0.79	0.44	0.45	4392
weighted avg	0.74	0.69	0.61	4392

```
[[2798  16   0]
 [ 754 122   8]
 [ 544  24 126]]
```

```
In [21]: print_metrics(logistic_reg)
```

0.7893897996357013

	precision	recall	f1-score	support
negative	0.81	0.93	0.87	2814
neutral	0.65	0.48	0.55	884
positive	0.80	0.60	0.69	694
accuracy			0.79	4392
macro avg	0.76	0.67	0.70	4392
weighted avg	0.78	0.79	0.78	4392

```
[[2626 135  53]
 [ 409 426  49]
 [ 188  91 415]]
```

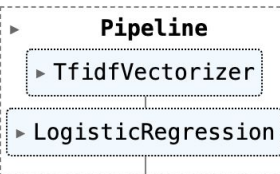
Naive Bias In NLP - Python Example

After comparing the models performance we clearly can see that the logistic regression model is produce much more accurate predictions. This is not a surprise because this model is much more complex than the Naive Bias model so this result can be expected.

What we can do now is creating a pipeline for future prediction with our preferred model:

```
In [25]: from sklearn.pipeline import Pipeline  
pipeline = Pipeline([('tfidf', TfidfVectorizer()), ('logistic_reg', LogisticRegression())])  
pipeline.fit(X, y)
```

```
Out[25]:
```



```
  Pipeline  
  ├── TfidfVectorizer  
  └── LogisticRegression
```

```
In [34]: print(pipeline.predict(["good flight"]))  
print(pipeline.predict(["bad flight"]))  
print(pipeline.predict(["ok flight"]))  
  
['positive']  
['negative']  
['neutral']
```

NLP - Topic Modeling

Until now we discussed on supervised learning types of tasks in NLP such as text sentiment and classification. But not all text files will come with predefined label, sometimes we will need to handle unlabeled text as well. The most common task in unlabeled text files is to summarize them and extract relevant subjects and topics. In case we have unlabeled text documents and we want to summarize them or extract the main topics for each text we can use topic modeling.

Topic modeling → Topic modeling is a type of statistical model used in natural language processing (NLP) to uncover the abstract "topics" that occur in a collection of documents. It is a form of unsupervised machine learning that helps in discovering the hidden thematic structure in large sets of documents. By applying topic modeling we can summarize large text files and extract themes and subjects from the text itself.

NLP - Topic Modeling

Topic → In the context of NLP, a topic is a collection of words that frequently occur together. These words are semantically linked and represent a coherent theme or concept. For example, a topic could consist of words like "apple," "banana," and "orange," representing the concept of "fruits."

Topic modeling does not require labeled data. Instead, it finds patterns and structures within the data itself. This is why topic modeling belongs to unsupervised learning.

Topic modeling can be applied to a variety of NLP tasks:

- **Content Recommendation** → Recommending articles or documents based on identified topics.
- **Document Clustering** → Grouping similar documents based on their topic distributions.
- **Trend Analysis** → Tracking how certain topics evolve over time in a collection of documents.
- **Information Retrieval** → Enhancing search engines by understanding the topic structure of documents.



NLP - Topic Modeling

Key steps in topic modeling:

- **Preprocessing the text** → Common NLP preprocessing text tasks such as tokenization, removal of stop words, stemming / lemmatization and vectorization (count vectorization or TF-IDF).
- **Choosing the Number of Topics** → Same as with other clustering problems, choosing the number of topics will determine the number of different topics the model will try to associate each text file into. We won't be able to get the topic label from the model but we could try to understand it ourselves after the association.
- **Training the model** → In topic modeling we can select from variety of different models. The most common models in this field are **LDA (Latent Dirichlet Allocation)** and **NMF (Non-Negative Matrix Factorization)**.
- **Evaluating and Interpreting Topics** → Once the model complete the training process it will provide us different topics and different words that been associated to each topic.

The model will also provide, for each text file, the probability to belong to each topic.



Topic Modeling - LDA Model

Latent Dirichlet Allocation (LDA) → LDA is a generative probabilistic model that allows sets of observations (in this case, documents) to be explained by unobserved groups (topics) that explain why some parts of the data are similar. It assumes that documents are mixtures of topics and that topics are mixtures of words. The main goal of LDA is to find the mixture of topics in each document and the mixture of words in each topic.

In the context of an unsupervised learning clustering task, we can think of topics identified by LDA as clusters. For each document, the model outputs a probability distribution indicating the likelihood of the document belonging to each topic (cluster).

Additionally, the model constructs these topics based on the probability distribution of all the words in the entire corpus belonging to each topic. The words with the highest probability to belong to each topic can help us understand and potentially label the topics.

For example → If a topic contains high probabilities for words like 'computer,' 'desktop,' 'chair,' 'worker,' and 'store,' we might interpret this topic as related to an "office environment" or "workspace."

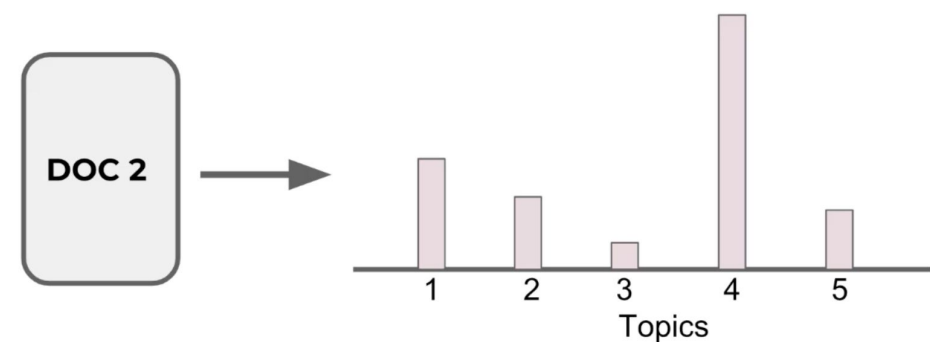
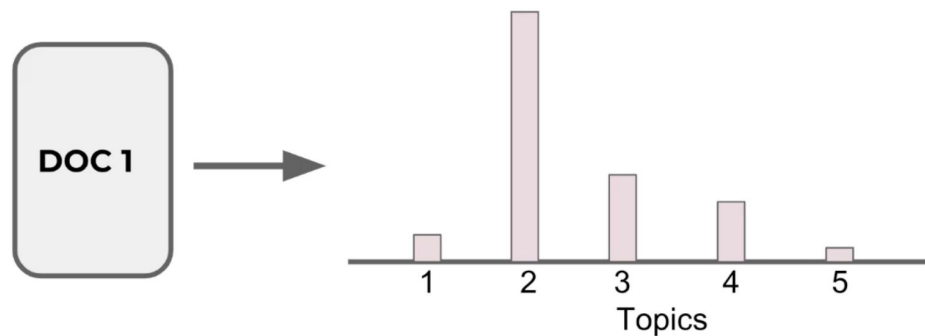
Topic Modeling - LDA Model

The main different between LDA and clusters in traditional clustering tasks is that while clustering typically assigns each instance to a single cluster, LDA shows a probabilistic mix of topics for each document, reflecting the reality that documents often cover multiple themes.

Visually, it can be explain like this:

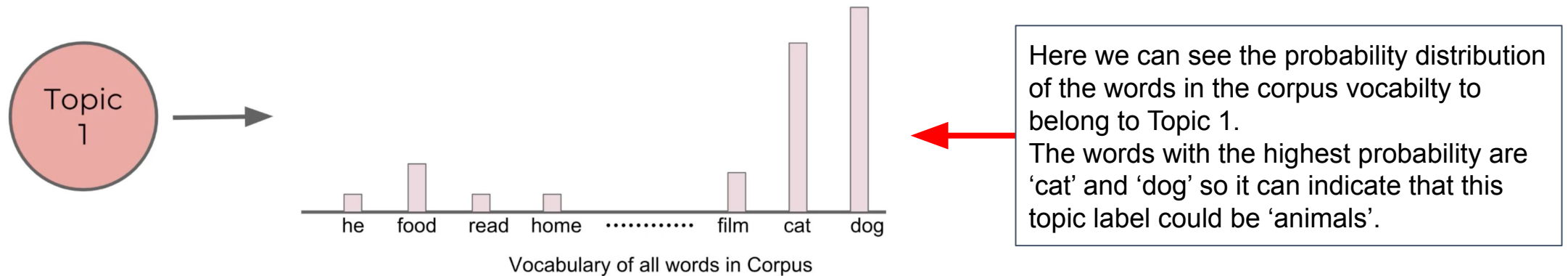
Let's say we run LDA on 2 documents (Doc 1 & Doc 2) and selected the number of topics as 5.

The LDA model result will be the probability distribution of all 5 topics for each document.



Topic Modeling - LDA Model

In addition, the model will provide the words probability distribution for each topic.



Now let's discuss how the LDA model manage to create those topics and calculate the probability of each topic.

The LDA model is using the **Gibbs sampling is a Markov Chain Monte Carlo (MCMC)** which is a mathematical technique for calculating joint probability.

The LDA model is iterate on each word for each document and first calculate the probability of the word to belong to a given topic. Next the model calculate the probability of the topic to belong to the document and finally it perform joint probability calculation to get the final probability of the word to belong to the topic when the topic belong to the document.

Topic Modeling - LDA Model

The LDA model is an iterative model which mean it use iterations and backpropagation to find the final probability distribution of each word in each topic and the final probability distribution of each topic in each document.

LDA model step by step:

- **Parameters selection** → Selecting the K value for number of topics and select the number of iterations (optional).
- **Random Initialization** → The model start by randomly assigning each word in the corpus to one of the k topics.
- **Iterative Refinement** → The model Iteratively update topic assignments for each word in each document by calculating the joint probability of each word to belong to each topic and each topic to belong to each document. In case it find topic with greater probability the model will re-assign the word to the relevant topic accordingly.
- **Model prediction** → The result of the training process will be for each documents a list of topics and the corresponding probability and for each topic a list of words and the corresponding probability.

LDA Model - Python Example

For this example we will use the 'npr.csv' file. This file contains raw text documents on different topics. The text itself is unlabeled text, making it suitable for performing topic modeling.

```
In [2]: df = pd.read_csv('/Users/ben.meir/Downloads/npr.csv')
df.head()
```

Out [2]:

	Article
0	In the Washington of 2016, even when the polic...
1	Donald Trump has used Twitter — his prefe...
2	Donald Trump is unabashedly praising Russian...
3	Updated at 2:50 p. m. ET, Russian President VI...
4	From photography, illustration and video, to d...

Let's also take a look on a specific document to check its length:

```
In [4]: df['Article'][0]
```

```
Out[4]: 'In the Washington of 2016, even when the policy can be bipartisan, the politics cannot. And in that sense, this year shows little sign of ending on Dec. 31. When President Obama moved to sanction Russia over its alleged interference in the U. S. election just concluded, some Republicans who had long called for similar or more severe measures could scarcely bring themselves to approve. House Speaker Paul Ryan called the Obama measures "appropriate" but also "overdue" and "a prime example of this administration's ineffective foreign policy that has left America weaker in the eyes of the world." Other GOP leaders sounded much the same theme. "[We have] been urging President Obama for years to take strong action to deter Russia's worldwide aggression, including its operations," wrote Rep. Devin Nunes, chairman of the House Intelligence Committee. "Now with just a few weeks left in office, the president has suddenly decided that some stronger measures are indeed warranted." Appearing on CNN, frequent Obama critic Trent F. F. ranks, called for "much tougher" actions and said three times that Obama had "finally found his tongue." Meanwhile, at and on Fox News, various spokesmen for Trump said Obama's real target was not the Russians at all but the man poised to take over the White House in less than three weeks. They spoke of Obama trying to "tie Trump's hands
```

LDA Model - Python Example

Now, let's execute feature extraction with count vectorization:

We will also determine the '**max_df**' and '**min_df**' parameters in the count vectorizer.

- **min_df** → Considering in the count vectorizer only words that above the min threshold (in our case words that appear in less than 2 data points will excluded).
- **max_df** → Considering in the count vectorizer only words that below the max threshold (in our case words that appear in more than 90% of the data points will excluded).

```
In [6]: from sklearn.feature_extraction.text import CountVectorizer  
  
X = df['Article']  
count_vectorizer = CountVectorizer(max_df=0.9, min_df=2, stop_words='english')  
  
X_counts = count_vectorizer.fit_transform(X)
```

Next, we will train our LDA model:

```
In [7]: from sklearn.decomposition import LatentDirichletAllocation  
  
lda_model = LatentDirichletAllocation(n_components=7, random_state=101)  
lda_model.fit(X_counts)
```

```
Out[7]:  
▼ LatentDirichletAllocation  
LatentDirichletAllocation(n_components=7, random_state=101)
```

n_components → Determine the number of topics
random_state → Determine the seed for the first random allocation of words to topics

LDA Model - Python Example

We can see all the different tokens in the count vectorizer using the `get_feature_names_out()` method:

```
In [17]: print(len(count_vectorizer.get_feature_names_out()))
         print(type(count_vectorizer.get_feature_names_out()))

54777
<class 'numpy.ndarray'>
```

We have 54,777 different tokens in the count vectorizers

We can also see the LDA probability distribution of each token in each topic:

```
In [19]: lda_model.components_.shape

Out[19]: (7, 54777)
```

We got a matrix with 7 rows (for 7 topics) and 54,777 columns (for each token in the count vectorizer)

```
In [22]: lda_model.components_
Out[22]: array([[4.27796767e+01, 1.44544830e+02, 3.14271569e+00, ...,
                6.14050308e+00, 1.14282922e+00, 1.44872383e-01],
               [2.91400381e+00, 2.87280928e+02, 1.42857150e-01, ...,
                1.42859888e-01, 1.42857145e-01, 1.42923441e-01],
               [8.33697665e+00, 1.34435536e+03, 1.42884916e-01, ...,
                1.42971829e-01, 1.42881992e-01, 1.42881876e-01],
               ...,
               [5.39601321e+00, 7.31995969e+02, 1.42857160e-01, ...,
                1.43946968e-01, 1.43033590e-01, 1.43235733e-01],
               [2.51607733e+01, 1.04552784e+03, 1.42857150e-01, ...,
                1.42876367e-01, 1.42857145e-01, 1.42895237e-01],
               [5.18310384e+00, 2.45000957e+02, 1.42970786e-01, ...,
                1.42857157e-01, 1.14260382e+00, 2.14033418e+00]])
```

The probability of each token to belong to each topic according to the LDA model calculation

LDA Model - Python Example

What we will want to do next is to see the top 10 tokens with the highest probability to belong to each topic.

Those tokens will provide us insight on how we should label the topic.

Because the LDA model is not providing us this topic to token match we will need to construct it manually.

np.argsort() → `argsort()` is a function in the NumPy library, used primarily for sorting and ordered indexing of arrays. It returns the indices that would sort an array. Let's break down how it works and its application.

For example → Executing `argsort()` on the array `[10, 2, 5]` will return `[1, 2, 0]` because those are the indexes of the values that sorted the array from lowest to highest

```
In [23]: array = np.array([10,2,5])  
         np.argsort(array)
```

```
Out[23]: array([1, 2, 0])
```



LDA Model - Python Example

So we are going to use the `np.argsort()` method to find the top 10 probability tokens for each topic.

By executing the following code we will get the 10 tokens with the highest probability for the first topic:

```
In [13]: np.argsort(first_topic)[-10:]  
Out[13]: array([53211, 39848, 33390, 27439, 49183, 49459, 36283, 42993, 26752,  
                28659])
```

Because `argsort()` by default sorting from lowest to highest we can select the highest 10 values by using Python index slice

What we want next is to match the indexes to the token in the token list and execute this to all topics:

```
In [14]: for topic_index, topic in enumerate(lda_model.components_):  
        topic_token_array = []  
        for index in topic.argsort()[-10:]:  
            topic_token_array.append(count_vectorizer.get_feature_names_out()[index])  
        print()  
        print(f"Topic number {topic_index} - Top 10 tokens:")  
        print(topic_token_array)  
        print()  
  
Topic number 0 - Top 10 tokens:  
['way', 'really', 'new', 'know', 'think', 'time', 'people', 'says', 'just', 'like']  
  
Topic number 1 - Top 10 tokens:  
['party', 'election', 'republican', 'people', 'state', 'campaign', 'president', 'clinton', 'said', 'trump']  
  
Topic number 2 - Top 10 tokens:  
['federal', 'new', 'care', 'people', 'house', 'health', 'says', 'president', 'said', 'trump']  
  
Topic number 3 - Top 10 tokens:  
['years', 'percent', 'new', 'food', 'school', 'just', 'health', 'like', 'people', 'says']  
  
Topic number 4 - Top 10 tokens:  
['told', 'men', 'day', 'team', 'world', 'national', 'people', 'said', 'says', 'women']  
  
Topic number 5 - Top 10 tokens:  
['told', 'state', 'city', 'reports', 'country', 'government', 'people', 'police', 'says', 'said']  
  
Topic number 6 - Top 10 tokens:  
['space', 'research', 'drugs', 'cancer', 'time', 'human', 'patients', 'new', 'science', 'drug']
```

The top words of each topic can help us determine the topic label. For example topic 1 can get label that related to election / politics

LDA Model - Python Example

Finally, we can create a new column in the original data frame with the topic index prediction from LDA model:

```
In [15]: topic_results = lda_model.transform(X_counts)
         topic_results[0]
```

```
Out[15]: array([2.25540846e-04, 2.89303875e-01, 5.90198905e-01, 2.25305200e-04,
                2.25357393e-04, 1.19595709e-01, 2.25306729e-04])
```

The probability of the first document to belong to each topic

```
In [17]: df['topic'] = topic_results.argmax(axis=1)
         df.head()
```

```
Out[17]:
```

	Article	topic
0	In the Washington of 2016, even when the polic...	2
1	Donald Trump has used Twitter — his prefe...	5
2	Donald Trump is unabashedly praising Russian...	1
3	Updated at 2:50 p. m. ET, Russian President Vl...	2
4	From photography, illustration and video, to d...	3

We are getting the topic with the highest probability for each document and populate the new 'topic' column with the matched topic index

