

Machine learning

Lecture 9 - NLP

08.05.2025

Toker Gilat

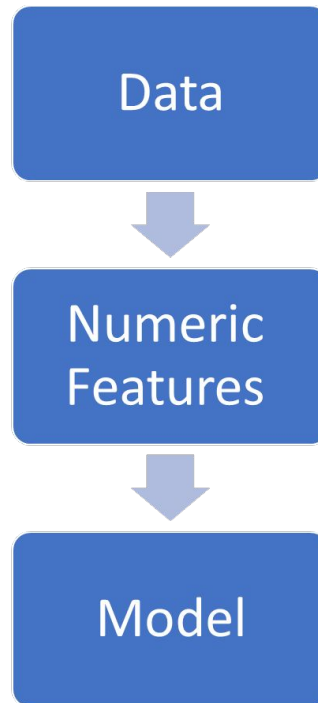


NLP

Natural language processing

ML pipeline

purchase_time	sales_id	customer_id	currency	amount_paid	device	has_discount
2018-01-01 8:48:00	466	1234	USD	\$98.23	mobile app	FALSE
2018-01-01 9:15:00	467	3485	USD	\$34.35	desktop	TRUE
2018-01-01 15:12:00	468	875	USD	\$34.35	mobile web	FALSE
2018-01-01 16:48:00	469	875	USD	\$34.35	desktop	FALSE
2018-01-01 18:09:00	470	5246	USD	\$34.35	mobile web	FALSE
2018-01-01 18:33:00	471	9872	USD	\$34.35	mobile web	FALSE
2018-01-02 7:25:00	472	2498	USD	\$34.35	desktop	FALSE
2018-01-02 8:01:00	473	7892	USD	\$34.35	desktop	FALSE



ML pipeline

	Review_ID	Rating	Year_Month	Reviewer_Location	Review_Text
0	670772142	4	2019-4	Australia	If you've ever been to Disneyland anywhere you...
1	670682799	4	2019-5	Philippines	Its been a while since d last time we visit HK...
2	670623270	4	2019-4	United Arab Emirates	Thanks God it wasn't too hot or too humid wh...
3	670607911	4	2019-4	Australia	HK Disneyland is a great compact park. Unfortu...
4	670607296	4	2019-4	United Kingdom	the location is not in the city, took around 1...

Data



Textual
Features



Model

What is NLP?

NLP is a discipline at the intersection of computer science, linguistics, and artificial intelligence that focuses on the interaction between computers and human (natural) language. It enables machines to understand, process, and generate text or speech.

Computers process data as numbers — bits, vectors, and matrices. But human language is messy, ambiguous, context-dependent, and full of nuance.

The fundamental goal of NLP is to bridge the gap between how humans communicate and how machines compute.

Alan Turing, in 1950, proposed a test of machine intelligence: **if a computer could engage in conversation indistinguishable from a human, it could be considered intelligent.**

<https://www.investopedia.com/terms/t/turing-test.asp#:~:text=The%20Turing%20Test%20is%20a, it%20has%20demonstrated%20human%20intelligence.>

Nlp Application



Nlp Application



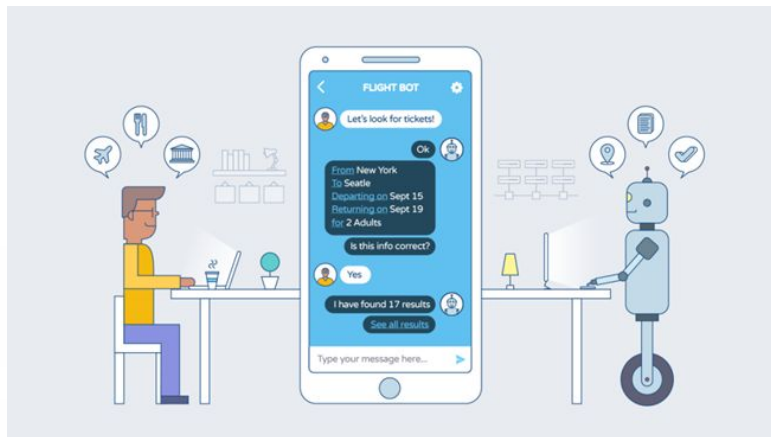
Nlp Application



**EMAIL SPAM
DETECTION**



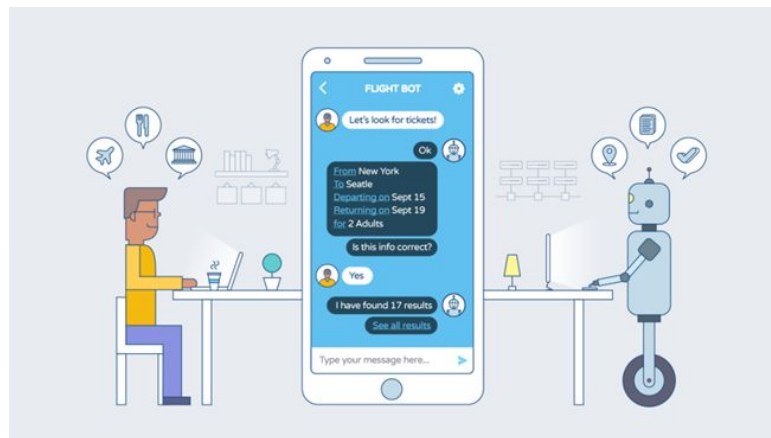
Nlp Application



EMAIL SPAM DETECTION



Nlp Application



**EMAIL SPAM
DETECTION**



Why NLP important?

NLP makes it possible for computers to understand human language while still retaining the ability to process data much faster than humans.

- Ubiquity of Text and Language
- Actionable Insights from Text Data
- Revolutionizing Industries



NLP- The Path to Machine Understanding

Humans learn a language by associating sounds with meanings and later progressing to reading and writing. **Computers** process numbers, not words.


Goal: Convert text input into numerical representations (vectors) that machines can analyze.

1. Text Preprocessing: Clean and prepare the text for analysis (Tokenization, Removing stop words, Lowercasing, Stemming/Lemmatization)
2. Text Encoding Convert text into vectorized features using techniques like Bag of Words (BoW), TF-IDF, Word Embeddings

NLP- The Path to Machine Understanding

Humans learn a language by associating sounds with meanings and later progressing to reading and writing. **Computers** process numbers, not words.

Goal: Convert text input into numerical representations (vectors) that machines can analyze.

1. Text Preprocessing: Clean and prepare the text for analysis (Tokenization,  Removing stop words, Lowercasing, Stemming/Lemmatization)
2. Text Encoding Convert text into vectorized features using techniques like Bag of Words (BoW), TF-IDF, Word Embeddings

Text Preprocessing

Text Preprocessing



Text Preprocessing

Why Preprocessing Is Essential?

Preprocessing prepares raw text for analysis or modeling by removing noise, standardizing format, and extracting meaningful features.

Why it Matter?

1. **Clean Data:** Removes noise, such as unnecessary characters and irrelevant details, improving clarity and focus.
Impact: For example, URLs (<https://example.com>) and emojis (🥰) add no predictive value to most models but increase dimensionality.
2. **Standardize:** Ensures uniformity across all text inputs, making patterns easier to detect.
Impact: Without standardization, words like "Amazing" and "amazing" would be treated as different, introducing inconsistency.
3. **Reduce Complexity:** Reduces the number of features by eliminating redundant information.
Impact: Leads to more efficient processing and improved model performance by focusing on meaningful components. e.g Remove Stopwords.

Text Preprocessing

Common Preprocessing Steps

Key Steps in Text Preprocessing:

1. **Lowercasing:** Convert all text to lowercase to ensure uniformity (e.g., "Hello" == "hello").
2. **Removing Punctuation:** Eliminate punctuation marks that may not add value to text analysis.
3. **Tokenization:** Split text into individual words or phrases (tokens).
Example: "The quick brown fox" → ['The', 'quick', 'brown', 'fox'].
4. **Stopword Removal:** Remove common, uninformative words (e.g., "the," "is," "and").
5. **Stemming and Lemmatization:**

Aspect	Stemming	Lemmatization
Approach	Heuristic rules	Dictionary + morphological analysis
Output	Root word (may be invalid)	Base form (valid word)
Example	"running" → "run", "flies" → "fli"	"running" → "run", "better" → "good"
Speed	Fast	Slower, more accurate

Text Preprocessing

Tokenization

Tokenization is the process of splitting a sequence of text into smaller units, known as tokens.

Tokens can represent words, characters, subwords, or phrases (n-grams). These tokens are the building blocks for text analysis and machine learning models.

- Serves as the first step in methods like BOW, TF-IDF, and embeddings.

Types of Tokenization:

- **Word Tokenization:** Splitting text into individual words.
- **Character Tokenization:** Splitting text into individual characters.
- **Subword Tokenization:** Splitting text into meaningful subwords or fragments. Commonly used in modern NLP models like BERT and GPT:

```
"unhappiness" → ['un', 'happi', 'ness']  
"autonomous" → ['auto', '##nomous']
```
- **n-gram Tokenization:** Splitting text into sequences of n consecutive tokens (words or characters).

Text Preprocessing

Tokenization

```
from nltk.tokenize import word_tokenize

text = "The new movie is amazing!"
word_tokens = word_tokenize(text)
print(word_tokens)
```

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
subword_tokens = tokenizer.tokenize("unbelievable")
print(subword_tokens)
```

```
text = "hello"
char_tokens = list(text)
print(char_tokens)
```

```
from sklearn.feature_extraction.text import CountVectorizer

text = ["The new movie is amazing"]
vectorizer = CountVectorizer(ngram_range=(2, 2)) # Bi-grams
ngrams = vectorizer.fit_transform(text)
print(vectorizer.get_feature_names_out())
```

Libraries That Handle Tokenization: NLTK, SpaCy, Hugging Face Transformers, SentencePiece

Text Preprocessing

Tokenization

Type	Input	Tokens
Word	"I love NLP"	['I', 'love', 'NLP']
Character	"hello"	['h', 'e', 'l', 'l', 'o']
Subword	"unbelievable"	['un', 'believ', 'able']
n-grams (Bi)	"The new movie"	['The new', 'new movie']
Sentence	"The movie is great. I loved it."	['The movie is great.', 'I loved it.']

Text Preprocessing

Stopword Removal

Stopwords are common words that don't carry significant meaning (e.g., "is," "the," "and").

Why Remove Them? They can add noise and inflate dimensionality.

Example: "The cat is on the mat" → "cat mat"

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

text = "This is an example sentence showing off stopwords removal."
stop_words = set(stopwords.words("english"))
tokens = word_tokenize(text)
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
print(filtered_tokens)
```

Text Preprocessing

Stemming

Removes word suffixes to reduce words to their root form. It operates on the principle of removing common suffixes like "ing", "ly", "es", "s", etc.

Example:

"running" -> "run"

"happiness" -> "happi"

"flies" -> "fli"

*** Stemming doesn't always produce actual words, and the stems may not always be the root form of the word.**

```
import nltk
nltk.download('punkt') #For more information: https://www.nltk.org/data.html

tokenized_sen = nltk.word_tokenize('Hi how are you doing?')
>> ['Hi', 'how', 'are', 'you', 'doing', '?']

ps = nltk.stem.PorterStemmer()
stemmed_sen = [ps.stem(word) for word in tokenized_sen]
>> ['hi', 'how', 'are', 'you', 'do', '?']
```

Text Preprocessing

Lemmatization

Maps words to their dictionary base form.

Lemmatization is a more sophisticated process that involves using a dictionary and morphological analysis of words. **It reduces words to their base or dictionary form, known as the lemma, but ensures that the result is an actual word.**

Example:

"running" -> "run"

"better" -> "good"

Lemmatization is slower but considers word context and grammar.

Common Lemmatization Libraries:

- WordNet Lemmatizer: Part of the NLTK library in Python, which uses the WordNet lexical database to perform lemmatization.
- SpaCy Lemmatizer: A more modern NLP library that provides efficient lemmatization

Text Preprocessing

Lemmatization

```
from nltk.stem import PorterStemmer, WordNetLemmatizer

text = "running flies better"
tokens = text.split()


stemmer = PorterStemmer()
print("Stemming:", [stemmer.stem(token) for token in tokens])

lemmatizer = WordNetLemmatizer()
print("Lemmatization:", [lemmatizer.lemmatize(token) for token in tokens])
```

NLP- The Path to Machine Understanding

Humans learn a language by associating sounds with meanings and later progressing to reading and writing. **Computers** process numbers, not words.

Goal: Convert text input into numerical representations (vectors) that machines can analyze.

1. Text Preprocessing: Clean and prepare the text for analysis (Tokenization, Removing stop words, Lowercasing, Stemming/Lemmatization)
2. Text Encoding Convert text into vectorized features using techniques like  Bag of Words (BoW), TF-IDF, Word Embeddings

Convert text input into vector features

Bag Of Words (BoW)

BoW is one of the simplest and most widely used techniques to represent text data numerically in machine learning. It treats a document as a collection of words, **disregarding grammar and word order while focusing on word frequency**. Each unique word in the document set is a feature, and the **model creates a vector for each document with word counts**.

Step 1: Collect Data, Tokenization:

Document 1: It was the best of times,
Document 2: it was the worst of times,
Document 3: it was the age of wisdom,

Step 2: Design the Vocabulary

{“it”, “was”, “the”, “best”, “of”, “times”, “worst”, “age”, “wisdom”} - Each word is a feature or dimension

Step 3: Create Document Vectors

"It was the best of times" = [1, 1, 1, 1, 1, 1, 0, 0, 0]
"it was the worst of times" = [1, 1, 1, 0, 1, 1, 1, 0, 0]
"it was the age of wisdom" = [1, 1, 1, 0, 1, 0, 0, 1, 1]

Convert text input into vector features

Bag Of Words (BoW)

Code

```
from sklearn.feature_extraction.text import CountVectorizer
# Multiple documents
text = ["It was the best of times", "it was the worst of times", "it was the age of wisdom",
        "it was the age of foolishness"]
# create the transform
vectorizer = CountVectorizer()
# tokenize and build vocab
vectorizer.fit(text)
# summarize
print(sorted(vectorizer.vocabulary_))['age', 'best', 'foolishness', 'it', 'of', 'the', 'times', 'was', 'wisdom', 'worst']
```

```
# encode document
vector = vectorizer.transform(text)
# summarize encoded vector
print(vector.shape)
print(vector.toarray())
```

(4, 10)

```
[[0 1 0 1 1 1 1 1 0 0]
 [0 0 0 1 1 1 1 1 0 1]
 [1 0 0 1 1 1 0 1 1 0]
 [1 0 1 1 1 1 0 1 0 0]]
```

Convert text input into vector features

Bag Of Words (BoW)

Advantages of BoW:

- **Simplicity:** Easy to implement and understand.
- **Effective for Basic Tasks:** Works well for simple text classification tasks when used with traditional ML algorithms.

Limitations Disadvantages of BoW:

- **High Dimensionality:** Vocabulary size can grow significantly with large datasets.
- **Sparsity:** Most documents contain only a small subset of the total vocabulary, leading to sparse matrices.
- **Ignores Context:** Does not capture relationships between words or their order.
The boy is happy he is not going.
The boy is not happy he is going.
- **Out of Vocabulary (OOV):** if the text includes only words that are not from the vocabulary, we will receive a 0 vector.

Convert text input into vector features

TF-IDF

TF-IDF is a numerical statistic that reflects the significance of a word within a document relative to a collection of documents, known as a corpus. TF-IDF improves upon BoW by weighting terms based on their importance in a document relative to the entire corpus.

Components of TF-IDF:

Term Frequency (TF) - Measures how often a term appears in a document.

$$TF(w_i, D_i) = \frac{\text{number of times } w_i \text{ appears in } D_i}{\text{number of words in } D_i}$$

Inverse Document Frequency (IDF) - Penalizes words that occur frequently across all documents.

$$IDF(t, D) = \log \left(\frac{\text{Total number of documents } |D|}{1 + \text{Number of documents where term } t \text{ appears}} \right)$$

* Why **log**? Preventing Over-penalization of Common Words, Smoothing the Range of Values, Information Scaling

$$TFIDF(w_i, D_i) = TF(w_i, D_i) * IDF(w_i)$$

Convert text input into vector features

TF-IDF

Step 1: Collect Data, Tokenization:

Document 1: It was the best of times,

Document 2: it was the worst of times,

Document 3: it was the age of wisdom,

Step 2: Design the Vocabulary

{“it”, “was”, “the”, “best”, “of”, “times”, “worst”, “age”, “wisdom”} - Each word is a feature or dimension

Step 3: Find TF

Word	Frequency	Total Words	TF
it	1	6	$\frac{1}{6} = 0.167$
was	1	6	$\frac{1}{6} = 0.167$
the	1	6	$\frac{1}{6} = 0.167$
best	1	6	$\frac{1}{6} = 0.167$
of	1	6	$\frac{1}{6} = 0.167$
times	1	6	$\frac{1}{6} = 0.167$

Convert text input into vector features

TF-IDF

Step 4: Find IDF

Word	Documents Containing Term	IDF Formula
it	3	$\log \frac{3}{1+3} = \log \frac{3}{4} \approx -0.125$
was	3	$\log \frac{3}{1+3} = \log \frac{3}{4} \approx -0.125$
the	3	$\log \frac{3}{1+3} = \log \frac{3}{4} \approx -0.125$
best	1	$\log \frac{3}{1+1} = \log \frac{3}{2} \approx 0.176$
worst	1	$\log \frac{3}{1+1} = \log \frac{3}{2} \approx 0.176$
age	1	$\log \frac{3}{1+1} = \log \frac{3}{2} \approx 0.176$
wisdom	1	$\log \frac{3}{1+1} = \log \frac{3}{2} \approx 0.176$
of	3	$\log \frac{3}{1+3} = \log \frac{3}{4} \approx -0.125$
times	3	$\log \frac{3}{1+3} = \log \frac{3}{4} \approx -0.125$

Convert text input into vector features

TF-IDF

Step 5: Create Document Vectors

Word	TF	IDF	TF-IDF
it	0.167	-0.125	$0.167 \times -0.125 = -0.021$
was	0.167	-0.125	$0.167 \times -0.125 = -0.021$
the	0.167	-0.125	$0.167 \times -0.125 = -0.021$
best	0.167	0.176	$0.167 \times 0.176 = 0.029$
of	0.167	-0.125	$0.167 \times -0.125 = -0.021$
times	0.167	-0.125	$0.167 \times -0.125 = -0.021$

" It was the best of times" = [0.021, -0.021, -0.021, 0.029, -0.021, -0.021, 0.000, 0.000, 0.000]

"it was the worst of times" = [0.021, -0.021, -0.021, 0.000, -0.021, -0.021, 0.029, 0.000, 0.000]

"it was the age of wisdom" = [-0.021, -0.021, -0.021, 0.000, -0.021, 0.000, 0.000, 0.029, 0.029]

Convert text input into vector features

TF-IDF

Code

```
from sklearn.feature_extraction.text import TfidfVectorizer
# list of text documents
text = ["It was the best of times", "it was the worst of times", "it was the age of wisdom",
        "it was the age of foolishness"]
# create the transform
vectorizer = TfidfVectorizer()
# tokenize and build vocab
vectorizer.fit(text)
# summarize
print(sorted(vectorizer.vocabulary_))
# encode document
vector = vectorizer.transform([text[0]])
```

```
print(vectorizer.idf_)
[1.51082562, 1.91629073, 1.91629073, 1, 1, 1, 1.51082562, 1, 1.91629073, 1.91629073]
```


Convert text input into vector features

TF-IDF

Advantages of TF-IDF:

- **Simplicity:** Straightforward to implement and understand.
- **Effective for Basic Tasks:** Works well for traditional machine learning tasks like text classification, clustering, and document retrieval.
- **Highlights Important Words:** Helps focus on terms that are unique to a document by penalizing commonly occurring words, such as "the" or "and."
- **Interpretable:** Provides clear numerical importance for each term in the document, making it easier to explain results.

Limitations Disadvantages of TF-IDF:

- **High Dimensionality:** Vocabulary size can grow significantly with large datasets.
- **Sparsity:** Most documents contain only a small subset of the total vocabulary
- **Ignores Semantic Context:** Does not capture relationships between words or their order.
- **Out of Vocabulary (OOV):** Words not present in the training data vocabulary are ignored, leading to zero-vector representations for documents containing unseen words.

Hands on !



Open Google colab notebook -
NLP Notebook

Common Pitfalls in Machine Learning and Best Practices

Overfitting

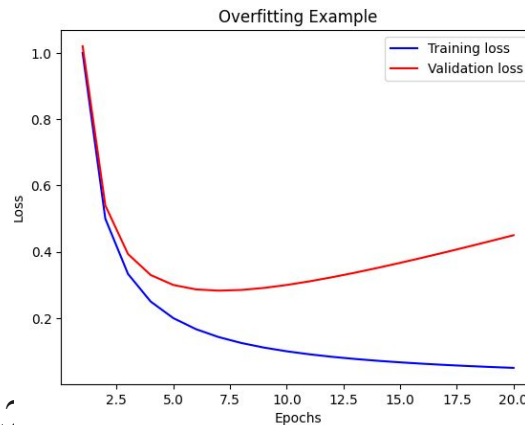
Definition: Overfitting occurs when a model learns not only the patterns in the training data **but also the noise**, leading to **poor generalization** on unseen data.

Indicators: High accuracy on training data but low accuracy on validation/test data.

Solutions:

- Simplify the model architecture.
- Increase the amount of training data.
- Use cross-validation to detect overfitting.
- Use regularization techniques (e.g., L1, L2)

<https://medium.com/@nerdjock/lesson-18-machine-learning-regularization-techniques-l1-lasso-and-l2-ridge-regularization-b9dc312c71fe>



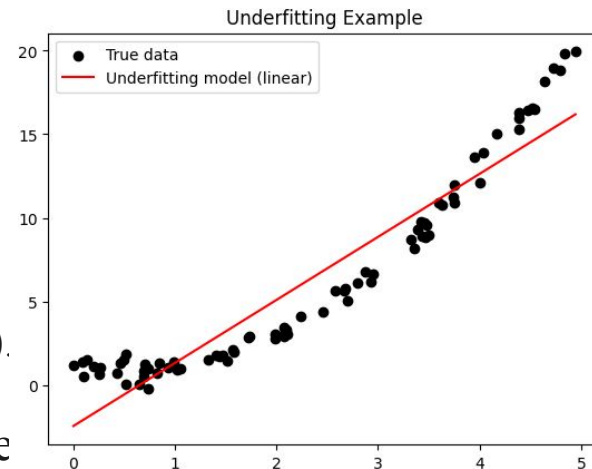
Underfitting

Definition: Underfitting occurs when a model is too simple to capture the underlying patterns in the data, resulting in poor performance on both training and test data.

Indicators: Low accuracy on both training and validation/test datasets.

Solutions:

- Increase model complexity (increase parameters).
- Use a more sophisticated algorithm.
- Ensure sufficient training time and better feature e



Data Leakage

Definition: Data leakage happens when information from outside the training dataset is used to create the model, leading to overly optimistic performance during evaluation.

Technical Causes of Data Leakage:

- **Incorrect preprocessing** before data splitting (e.g., normalizing on the entire dataset before splitting → **test data influences scaling**)
- Using **future** information that won't be available at inference (e.g., timestamped features like outcome, revenue, diagnoses)
- Label leakage via engineered features (e.g., aggregations that **accidentally include test samples**)

Prevention:

- Always split data into train/validation/test before any transformation
- Carefully inspect features to ensure no data from the target is included.

Imbalanced Dataset

Definition: An imbalanced dataset occurs when the distribution of classes in a classification task is skewed — some classes appear much more frequently than others.

The Challenge: Despite high accuracy, a model may completely ignore the minority class — the class we often care about the most (fraud, disease, malfunction).

Solutions:

- Evaluate performance using metrics like precision, recall, F1-score, AUC-ROC.
- Use resampling techniques (e.g., oversample minority, undersample majority).
- Apply algorithms designed for imbalanced data (e.g., SMOTE:

$$x_{\text{new}} = x_{\text{original}} + \lambda \cdot (x_{\text{neighbor}} - x_{\text{original}})$$

Bias-Variance Tradeoff

Definition: The bias-variance tradeoff highlights the balance between underfitting (high bias) and overfitting (high variance).

Key Points:

- Bias: Errors due to incorrect assumptions in the model.
- Variance: Errors due to sensitivity to small fluctuations in the training data.

Ideal Model: Achieves a balance where both bias and variance are minimized to optimize generalization.

