📘 NATURAL LANGUAGE PROCESSING (NLP) WITH MACHINE LEARNING – MASTER NOTES

1️⃣ WHAT IS NLP?

✅ DEFINITION

Natural Language Processing (NLP) is a branch of Artificial Intelligence (AI) and Machine Learning (ML) that enables computers to understand, interpret, analyze, and generate human language. It bridges human communication and computer understanding.

🎯 GOALS OF NLP

✔ Convert unstructured text \u2192 structured numeric data

✔ Enable ML models for classification, prediction, clustering

✔ Capture context, semantics, and syntax

💡 APPLICATIONS

📧 Text Classification \u2192 Spam detection, sentiment analysis

🌍 Machine Translation \u2192 English \u2192 French

📰 Summarization \u2192 Auto-summarize articles

🤖 Chatbots & Assistants \u2192 Siri, Alexa, Google Assistant

🔍 Question Answering \u2192 Search engines, customer support

2️⃣ KEY TERMS

Term	Definition	Example
📂 Corpus	Collection of text	100 movie reviews = corpus
📜 Document	Single piece of text	Sentence, paragraph, or article

📖 Vocabulary	All unique words in corpus	"I love pizza" + "I love pasta" \u2192 {I, love, pizza, pasta}

3️⃣ TEXT PREPROCESSING

🧹 DEFINITION

Preprocessing = Cleaning and standardizing text for ML models.

🔑 STEPS

1️⃣ Lowercasing: "I Love NLP" \u2192 "i love nlp"

2️⃣ Tokenization: "i love nlp" \u2192 ["i", "love", "nlp"]

3️⃣ Stopword Removal: Remove common words \u2192 ["love", "nlp"]

4️⃣ Stemming & Lemmatization:

🔹 Stemming: "running" \u2192 "run"

🔹 Lemmatization: "better" \u2192 "good"

5️⃣ Vectorization: Convert text \u2192 numbers (see below)

4️⃣ FEATURE EXTRACTION / VECTORIZATION TECHNIQUES

1️⃣ ONE-HOT ENCODING (OHE)

✅ Definition

Converts text or categorical data into a binary vector. Each unique word gets 1 in its index and 0 elsewhere.

📌 Example

Vocabulary = {I, love, NLP}

"love NLP" \u2192 [0, 1, 1]

✔ Pros: Simple, easy

✖ Cons: Sparse, no semantics

2️⃣ BAG OF WORDS (BOW)

✅ Definition

Represents text as a vector of word counts. Each position = frequency of a word in the document.

📌 Example

Vocabulary = {I, love, NLP, fun}

"I love NLP NLP" \u2192 [1, 1, 2, 0]

✔ Pros: Simple, fast

✖ Cons: Ignores word order, context, semantics

3️⃣ TF-IDF (TERM FREQUENCY \U2013 INVERSE DOCUMENT FREQUENCY)

✅ Definition

Weighs words by importance: frequent in document but rare in corpus.

$$ TF{-}IDF(t,d) = TF(t,d) imes \log\frac{N}{DF(t)} $$

TF(t,d): Term frequency in document

DF(t): Number of documents containing term

N: Total documents

✔ Pros: Highlights key words

✖ Cons: Sparse for large vocab

4️⃣ WORD EMBEDDINGS

✅ Definition

Dense vector representations capturing semantic meaning. Words with similar meanings are close in vector space.

Examples: Word2Vec, GloVe, FastText, BERT embeddings

📌 Example

"king" \u2013 "man" + "woman" \u2245 "queen"

✔ Pros: Captures meaning, context

✖ Cons: Requires large corpus

5️⃣ N-GRAMS

✅ Definition

Contiguous sequence of n words \u2192 partial context.

Unigram: "I", "love", "NLP"

Bigram: "I love", "love NLP"

Trigram: "I love NLP"

✔ Pros: Adds semantic meaning

✖ Cons: High dimensionality, OOV problem

5️⃣ PYTHON EXAMPLES

🖊 BAG OF WORDS

```
from sklearn.feature_extraction.text import CountVectorizer
docs = ["I love NLP", "NLP is fun"]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(docs).toarray()
print(vectorizer.get_feature_names_out())
print(X)
```

🖊 TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer
docs = ["I love NLP", "NLP is fun"]
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(docs).toarray()
```

```
print(vectorizer.get_feature_names_out())
print(X)
```

🖊 WORD2VEC

```
from gensim.models import Word2Vec
sentences = [["I", "love", "NLP"], ["NLP", "is", "fun"]]
model = Word2Vec(sentences, vector_size=5, window=2, min_count=1)
print(model.wv['NLP'])
```

6️⃣ TEXT CLASSIFICATION WITH ML

✔ Algorithms: Na\u00efve Bayes, Logistic Regression, SVM, Random Forest

✔ Metrics: Accuracy, Precision, Recall, F1-score

\u25c0 Example (TF-IDF + Na\u00efve Bayes):

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

docs = ["I love NLP", "NLP is fun", "I hate spam"]
labels = [1, 1, 0]

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(docs)

model = MultinomialNB()
model.fit(X, labels)
print(model.predict(vectorizer.transform(["I love spam"])))
```

7️⃣ TOPIC MODELING (UNSUPERVISED)

✔ Algorithms: LDA, NMF

🔹 Example:

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

docs = ["I love NLP", "NLP is fun", "I hate spam"]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(docs)

lda = LatentDirichletAllocation(n_components=2, random_state=0)
lda.fit(X)
print(lda.components_)
```

8️⃣ SEQUENCE MODELS (RNN, LSTM)

✔ RNN: Maintains hidden state

✔ LSTM: Handles long-term dependencies

✔ Use Cases: Next-word prediction, sentiment analysis, chatbots

9️⃣ QUICK COMPARISON TABLE

Technique	Type	✅ Pros	❌ Cons	Use Cases
🔲 One-Hot Encoding	Feature Extraction	Simple	Sparse, no semantics	Small datasets
📦 BoW	Feature Extraction	Simple, fast	Ignores context, sparse	Text classification

📊 TF-IDF	Feature Extraction	Highlights important words	Sparse for large vocab	Document retrieval
🔗 Word Embeddings	Feature Extraction	Captures semantic meaning & context	Requires large corpus	Similarity, chatbots
🔍 N-grams	Feature Extraction	Adds some context	High dimension, OOV	Text classification
📑 LDA	Unsupervised	Finds hidden topics	Needs preprocessing	Topic modeling
🤖 ML Classifiers	Supervised	Predicts labels	Needs labeled data	Spam detection, sentiment
🔁 RNN/LSTM	Sequence Modeling	Handles sequential dependencies	Computationally expensive	Text generation, chatbots