Natural Language Processing with Machine Learning
Yogesh Haribhau Kulkarni

Introduction

- Natural Language Processing (NLP) and Machine Learning (ML) form the foundation for modern intelligent systems.
- Retrieval Augmented Generation (RAG) leverages NLP pipelines, embeddings, and ML for knowledge-grounded responses.
- This presentation explores:
  - Text preprocessing and representation
  - spaCy-based NLP components
  - ML techniques for classification and clustering
  - Embedding-based search and semantic understanding
  - Sentiment analysis applications

(Ref: Stanford NLP Group)

Text Preprocessing

- Clean and normalize raw text data for consistent downstream analysis.
- Common steps include lowercasing, punctuation & digit removal, and whitespace normalization.
- Ensures reliable tokenization and model training.
- Helps retain semantic integrity by focusing on meaningful linguistic content.

(Ref: Text Cleaning Steps ResearchGate)

Tokenization

- Tokenization splits text into smaller linguistic units (words, subwords, or sentences).
- spaCy provides efficient rule-based and statistical tokenizers.
- Basis for vectorization, tagging, and parsing.

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Retrieval-Augmented Generation is powerful!")
tokens = [token.text for token in doc]
print(tokens)
# ['Retrieval', '-', 'Augmented', 'Generation', 'is', 'powerful', '!']
```

Lowercasing & Cleaning

- Lowercasing removes case sensitivity in NLP models.
- Cleaning eliminates noise: URLs, HTML tags, punctuation, numbers.
- Helps search and RAG systems achieve higher recall and consistency.

```
import re
text = "Visit https://openai.com for <b>AI Research!</b>"
cleaned = re.sub(r"http\S+|<.*?>", "", text).lower()
print(cleaned)
# 'visit  for ai research!'
```

Stop Word Removal

- Removes high-frequency words that add little meaning.
- Reduces dimensionality and improves efficiency of embeddings.
- Example stop words: “the”, “is”, “in”, “on”.

```
from spacy.lang.en.stop_words import STOP_WORDS
tokens = ["Retrieval", "Augmented", "Generation", "is", "powerful"]
filtered = [w for w in tokens if w.lower() not in STOP_WORDS]
print(filtered)
# ['Retrieval', 'Augmented', 'Generation', 'powerful']
```

Stemming & Lemmatization

- Stemming removes suffixes using heuristic rules.
- Lemmatization uses grammar-based transformations to return dictionary form.
- Reduces vocabulary size and enhances text matching in retrieval.

```
from nltk.stem import WordNetLemmatizer
lem = WordNetLemmatizer()
print(lem.lemmatize("running", "v"))  # 'run'
print(lem.lemmatize("better", "a"))   # 'good'
```

Regular Expressions

- Enable pattern-based extraction from text.
- Examples: extract emails, phone numbers, or URLs.
- Essential for data cleaning, anonymization, and text mining.

```
import re
text = "Contact us at info@company.com or sales@domain.org"
emails = re.findall(r"\b[A-Za-z0-9._%+-]+@[A-Za-z]+\.[A-Z|a-z]{2,}\b", text)
print(emails)
# ['info@company.com', 'sales@domain.org']
```

Project: Text Preprocessing Pipeline

- Modular preprocessing pipeline for flexible text workflows.
- Each function handles one cleaning step.
- Easily extendable for lemmatization, stemming, or entity masking.

```
class TextPipeline:
    def __init__(self, steps):
        self.steps = steps  # list of functions

    def run(self, text):
        for fn in self.steps:
            text = fn(text)
        return text

pipeline = TextPipeline([basic_clean])
```

```
print(pipeline.run("Hello World! This is 2025."))
```

Project: Document Cleaner

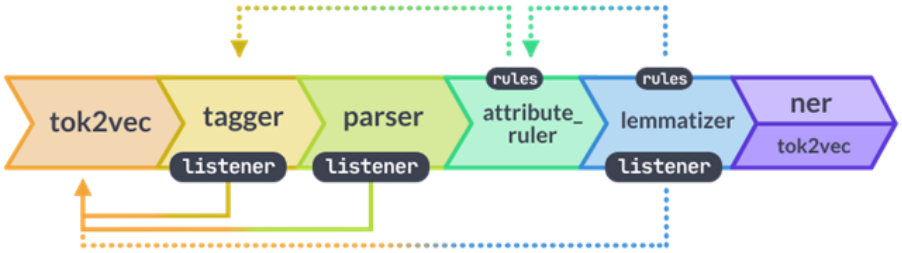
- Extracts text from PDFs, HTML, and Word documents.
- Normalizes whitespace, removes headers/footers, handles encoding.
- Outputs clean text ready for embedding or indexing.

```
from bs4 import BeautifulSoup
from PyPDF2 import PdfReader

def clean_document(path):
    if path.endswith(".pdf"):
        text = "".join([page.extract_text() for page in PdfReader(path).pages])
    elif path.endswith(".html"):
        html = open(path).read()
        text = BeautifulSoup(html, "html.parser").get_text()
    else:
        text = open(path).read()
    return re.sub(r"\s+", " ", text.strip())
```

spaCy Pipeline Basics

- spaCy's pipeline includes tokenization, tagging, parsing, and entity recognition.
- Designed for efficiency and extensibility.
- Supports custom components for preprocessing and postprocessing.



(Ref: spaCy Docs)

Part-of-Speech Tagging

- Assigns syntactic roles: noun, verb, adjective, etc.
- Enables grammar-aware retrieval, information extraction, and text summarization.

```
doc = nlp("NLP models enhance communication.")
for token in doc:
    print(token.text, token.pos_)
# Output: NLP NOUN, models NOUN, enhance VERB, communication NOUN
```

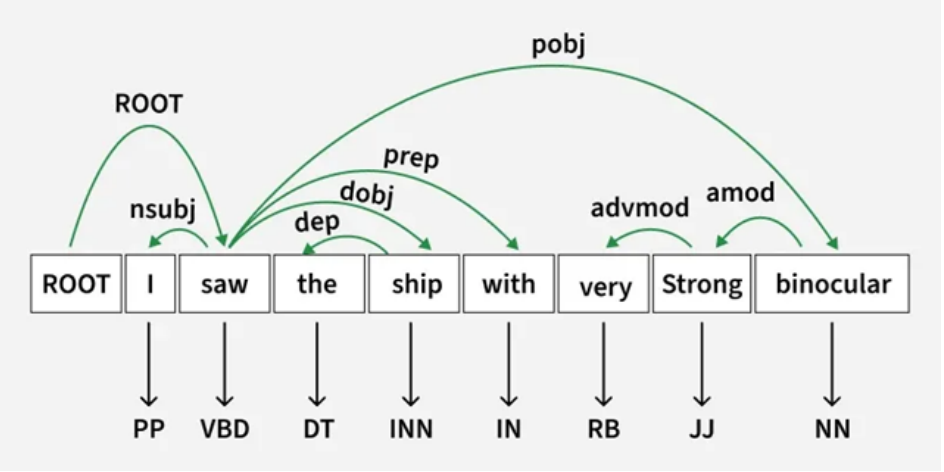
Named Entity Recognition (NER)

- Identifies named entities: PERSON, ORG, LOC, DATE, MONEY, etc.
- Crucial for knowledge extraction and linking to external databases.

```
doc = nlp("OpenAI was founded in San Francisco in 2015.")
for ent in doc.ents:
    print(ent.text, ent.label_)
# OpenAI ORG, San Francisco GPE, 2015 DATE
```

Dependency Parsing

- Analyzes grammatical structure of sentences.
- Reveals subject-object relationships.
- Basis for relation extraction and semantic role labeling.



(Ref: Dependency Parsing with NLTK - GeeksforGeeks)

Text Similarity

- Measures semantic closeness between two texts.
- Useful in RAG for ranking documents by contextual relevance.

```
doc1 = nlp("AI improves healthcare")
doc2 = nlp("Artificial intelligence helps medicine")
print(doc1.similarity(doc2))
# Output: similarity score ~0.85
```

Project: Named Entity Extractor

- Extracts and exports named entities from text.
- Can be integrated with visualization tools or stored in databases.

```
def extract_entities(text):
    nlp = spacy.load("en_core_web_sm")
    doc = nlp(text)
    return {ent.text: ent.label_ for ent in doc.ents}

print(extract_entities("Elon Musk leads SpaceX and Tesla."))
```

Project: Text Similarity Engine

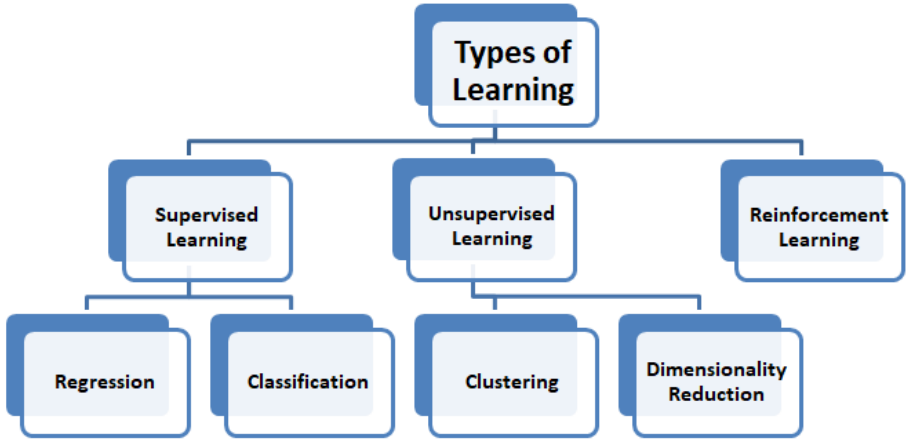
- Uses embeddings to compute cosine similarity between texts.
- Enables semantic document search and clustering.

```
from sklearn.metrics.pairwise import cosine_similarity

def rank_docs(query, docs, model):
    q_vec = model.encode([query])
    d_vecs = model.encode(docs)
    scores = cosine_similarity(q_vec, d_vecs)[0]
    ranked = sorted(zip(docs, scores), key=lambda x: x[1], reverse=True)
    return ranked[:3]
```

Types of Machine Learning

- **Supervised**: learns from labeled data (e.g., text classification).
- **Unsupervised**: discovers hidden structure (e.g., clustering).
- **Reinforcement**: learns from feedback/reward.
- NLP tasks map naturally to these paradigms.



(Ref: https://www.studytrigger.com/article/types-of-learning-in-machine-learning/)

Classification

- Predicts categorical labels (spam, positive/negative sentiment).
- Logistic Regression is a strong baseline for text classification.

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

vec = TfidfVectorizer()
X_train = vec.fit_transform(train_texts)
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
preds = model.predict(vec.transform(test_texts))
```

Clustering

- Groups similar documents based on content.
- Common methods: KMeans, DBSCAN, Agglomerative.
- Supports topic discovery and unsupervised organization.

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

km = KMeans(n_clusters=5, random_state=42)
km.fit(embeddings)
labels = km.labels_

pca = PCA(n_components=2)
reduced = pca.fit_transform(embeddings)
plt.scatter(reduced[:,0], reduced[:,1], c=labels)
```

```
plt.show()
```

Project: Iris Flower Classifier

- Classic ML dataset demonstration.
- Illustrates feature scaling, model training, and evaluation.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
model = LogisticRegression(max_iter=300)
model.fit(X_train, y_train)
print("Accuracy:", accuracy_score(y_test, model.predict(X_test)))
```

Project: Document Clusterer

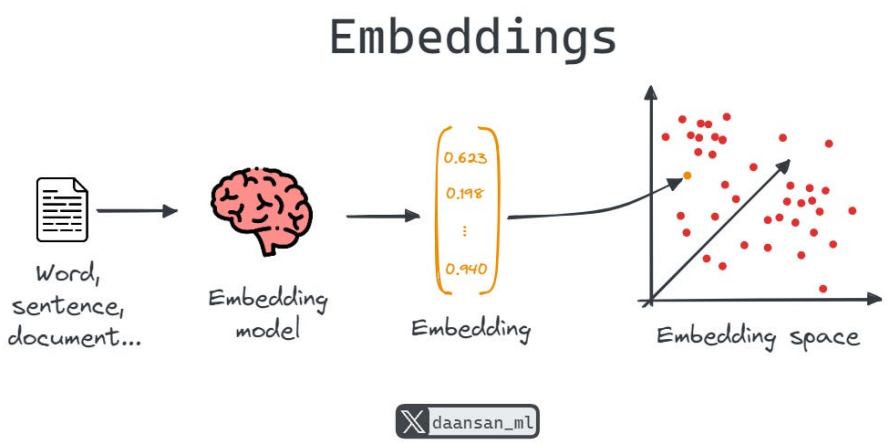
- Embeds documents using spaCy or sentence-transformers.
- Applies KMeans for unsupervised grouping.
- Visualizes using t-SNE or PCA for semantic clusters.

```
import spacy
from sklearn.cluster import KMeans
nlp = spacy.load("en_core_web_md")

docs = ["AI in healthcare", "Quantum computing basics", "AI in finance"]
vectors = [nlp(d).vector for d in docs]
labels = KMeans(n_clusters=2).fit_predict(vectors)
print(list(zip(docs, labels)))
```

Word Embeddings Concepts

- Represent words as dense numerical vectors capturing context.
- Models: Word2Vec, GloVe, FastText, Transformer embeddings.
- Enable semantic understanding and transfer learning.



(Ref: https://mlpills.substack.com/p/issue-58-embeddings-in-nlp)

Project: Semantic Search Engine

- Retrieve documents semantically, not just by keyword match.
- Uses embedding models and cosine similarity for ranking.

```
from sentence_transformers import SentenceTransformer, util
model = SentenceTransformer("all-MiniLM-L6-v2")

def semantic_search(query, docs):
    q_emb = model.encode(query, convert_to_tensor=True)
    d_emb = model.encode(docs, convert_to_tensor=True)
    scores = util.cos_sim(q_emb, d_emb)
    return sorted(zip(docs, scores[0]), key=lambda x: float(x[1]),
     reverse=True)[:3]
```

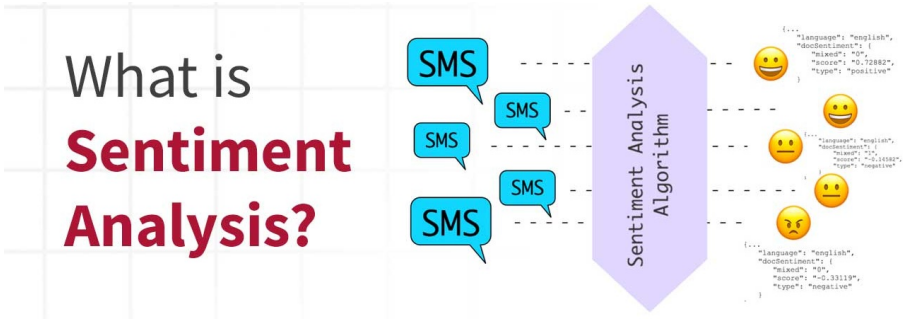
Project: Word Analogy Solver

- Demonstrates vector arithmetic: $king - man + woman \approx queen$
- Captures latent gender, tense, and semantic relations.

```
from gensim.models import KeyedVectors
model = KeyedVectors.load_word2vec_format("GoogleNews-vectors.bin", binary=True)
result = model.most_similar(positive=["king", "woman"], negative=["man"],
     topn=1)
print(result)  # [('queen', 0.75)]
```

Sentiment Analysis Basics

- Detects emotional polarity (positive, neutral, negative).
- Used in reviews, feedback, and social media monitoring.
- Can be rule-based, ML-based, or transformer-based.



(Ref: Sentiment Analysis — Engati)

Project: Review Sentiment Analyzer

- Train a sentiment classifier using Naive Bayes.
- Vectorize text with TF-IDF.
- Evaluate accuracy using test set.

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

X = vec.fit_transform(reviews)
model = MultinomialNB()
model.fit(X, labels)
preds = model.predict(vec.transform(test_reviews))
print("Accuracy:", accuracy_score(test_labels, preds))
```

Project: Tweet Sentiment Dashboard

- Streams tweets and classifies their sentiment in real-time.
- Visualizes sentiment trends dynamically using dashboards.

```
import tweepy, pandas as pd
from textblob import TextBlob

def classify_sentiment(text):
    polarity = TextBlob(text).sentiment.polarity
    return "positive" if polarity > 0 else "negative" if polarity < 0 else
     "neutral"

# TODO: integrate with dashboard for visualization
```

Conclusions

- Robust NLP preprocessing and embeddings are critical for effective information retrieval.
- spaCy and scikit-learn streamline NLP + ML workflows.
- Vector-based understanding enables semantic and contextual search.
- Combining these components forms the backbone of Retrieval-Augmented Generation.
- Future directions:
  - Fine-tuning LLMs for domain tasks
  - Efficient vector database retrieval (FAISS, Milvus)
  - Integrating sentiment and entity-level reasoning

References

- Explosion AI, *spaCy Documentation*, https://spacy.io/
- Pedregosa et al., *Scikit-learn: Machine Learning in Python*, JMLR, 2011.
- Mikolov et al., *Efficient Estimation of Word Representations in Vector Space*, arXiv, 2013.
- Bird et al., *Natural Language Processing with Python*, O'Reilly, 2009.
- OpenAI, *Retrieval-Augmented Generation (RAG) Overview*, 2024.
- Manning & Schütze, *Foundations of Statistical Natural Language Processing*, MIT Press.
- Jurafsky & Martin, *Speech and Language Processing*, 3rd Ed. Draft, 2023.