Natural language process and understand

What we learn

1. Word –Embedding - Bag of Word (BOW) , Word2Vec- C-Bow , Skip -gram , N-gram , Glove , Fast-Text , TF-IDf
2. Empirical Laws of Language - Zip’s Law and Heaps Law
3. Pos Tagging
4. Hidden Markov Model – Initial , Transition , Emission , Probability
5. Name Entity Recognition
6. Viterbi Algorithm
7. Baum-Welch algorithm
8. Custom Feature
9. POS Tagging Cheat-sheet

Word –Embedding

- **What is Text Representation?**
- Text representation means converting **text into numbers** so that machines can understand it.
- **Why needed?**
- Computers do not understand words
- ML models work only with numbers
- To extract **meaning, patterns, and relationships**
- **Example**
- Text: "I love NLP"
  Converted into numbers → model can learn from it

1. Bag of Word (BOW)

- **What is BOW?**
- BOW represents text using **word frequency**.
- **How it works**
- Create vocabulary
- Count occurrence of each word
- **Example**
- Sentence:
  I love NLP and I love AI
- Vocabulary: [I, love, NLP, and, AI]
  Vector: [1, 2, 1, 1, 1]
- **Why use BoW?**
- Simple
- Fast
- Good baseline model
- **Where used**
- Spam detection
- Fake news detection
- Sentiment analysis (basic)
- **Limitations**
- ⛓️ No word order
  No meaning/context
  Large feature size

2. N-gram

- **What is N-gram?**
- Sequence of **N continuous words**.
- **Types**
- Unigram → single word
- Bigram → two words
- Trigram → three words
- **Example**
- Sentence: "I love NLP"
- Unigram: I, love, NLP
- Bigram: I love, love NLP
- **Why use N-gram?**
- Captures **word order**
- Improves meaning over BoW
- **Where used**
- Language modeling
- Search engines
- Text prediction
- **Limitations**
- ❌ Large vocabulary
  Still limited semantics

3. Word2Vec

- **What is Word2Vec?**
- Word2Vec is a technique that converts **words into numerical vectors** in such a way that **similar-meaning words get similar vectors**. **Why Word2Vec?**
- Words with similar meaning → similar vectors.
- **Example**
- king − man + woman ≈ queen

- **Why Word2Vec is Needed?**
- **BoW and N-gram:**
- Do not understand meaning
- Treat words independently
- **Word2Vec:**
- Captures **semantic relationships**
- Understands how words are used **in context**
- **Example:**
  “king” is closer to “queen” than to “apple” in vector space.

- **Where used**
- Recommendation systems
- Chatbots
- Semantic search
- **Two Architectures of Word2Vec**
- Word2Vec has **two models**:

1. CBOW (Continuous Bag of Words) - CBOW predicts the **target word using context words**.

- Input: context words
- Output: target word
- **Example sentence:**
  I love NLP
- **Input → I, NLP**
  Output → love
- ✓ Fast
  ❌ Weak for rare words
-

2. Skip-Gram - Skip-Gram predicts **context words from a target word**.

- Input: target word
- Output: context words
- **Input → love**
  Output → I, NLP
- ✓ Better for rare words
  ❌ Slower than CBOW

4. GloVe

- **What is GloVe?**
- GloVe learns word vectors using **global word co-occurrence statistics**.
- **Why GloVe?**
- Combines local + global context
- Better word relationships
- **Example**
- Paris : France :: Rome : Italy
- **Where used**
- NLP research
- Pretrained embeddings
- **Limitations**
- ❌ Fixed vocabulary
  ❌ Cannot handle new words

5. Fast-Text

- **What is FastText?**
- FastText creates embeddings using **sub-word information**.
- **How it works**
- Breaks word into character n-grams.
- **Example**
- Word: playing
  Sub-words: play, lay, aying
- **Why FastText?**
- Handles spelling mistakes
- Supports rare and new words
- **Where used**
- Social media text
- Indian languages
- Noisy datasets
- **Limitations**
- ❌ Larger model size

6. TF-IDF (Term Frequency – Inverse Document Frequency)

- **What is TF-IDF?**
- TF-IDF is a text representation technique that converts words into numbers by measuring **how important a word is in a document compared to the entire collection (corpus)**.
- 👉 It improves **Bag of Words** by reducing the importance of very common words.

- **Why TF-IDF is Needed?**
- Common words like *“is, the, and”* appear everywhere
- BoW gives them high weight (which is wrong)
- TF-IDF reduces weight of common words and highlights **important words**

- **How TF-IDF Works in Practice**
- Build vocabulary
- Calculate TF for each word
- Calculate IDF using full corpus
- Multiply TF and IDF
- Convert text into numerical vectors

- **Where TF-IDF is Used**
- Search engines
- Document classification
- Fake news detection
- Resume screening
- Text similarity

- # TF-IDF Formula
- **1. Term Frequency (TF)**
- Measures how often a word appears in a document.
- $\mathbf{TF}(t, d) = \frac{\textit{Number of times term } t \textit{ appears in document } d}{\textit{Total number of terms in document } d}$
- **.2Inverse Document Frequency (IDF)**
- Measures how rare or important a word is across all documents.
- $\mathbf{IDF}(t) = \boldsymbol{log}\left(\frac{N}{\boldsymbol{DF}(t)}\right)$
- Where:
- N= Total number of documents
- $\mathrm{DF}(t)$ =Number of documents containing term t
- *(Sometimes written as* $\log\left(\frac{N}{1+DF(t)}\right)$*to avoid division by zero)*
- **3. TF-IDF (Final Formula)**
- $\mathrm{TF{-}IDF}(t, d) = TF(t, d) \times IDF(t)$
- **Simple Example**
- Documents:
- D1: “I love NLP”
- D2: “I love AI”
- Word **“love”**:
- Appears in both documents → low IDF
- TF-IDF value → low
- Word **“NLP”**:
- Appears only in D1 → high IDF
- TF-IDF value → high
- 👉 So **“NLP” is more important than “love”**

- Comparison Table

Technique	Meaning	Word Order	Context	Best Use
BoW	Word frequency	❌	❌	Simple ML models
TF-IDF	Importance-weighted frequency	❌	❌	Document ranking, text classification
N-gram	Word sequence	✓	❌	Search, autocomplete
CBOW	Context → word	✓	✓	Fast training on large data
Skip-Gram	Word → context	✓	✓	Learning rare words
GloVe	Global word statistics	✓	✓	Pretrained embeddings

Empirical Laws of Language

- **What are Empirical Laws in NLP?**
- **What**
- Empirical laws are **patterns observed from real text data**, not rules made by humans.

- **Why important**
- They help us:
- Understand language behavior
- Design efficient NLP systems
- Handle large vocabularies

1. Zip's Law

- **What is Zipf's Law?**
- **Definition**
- Zipf's Law states that:
- *The frequency of a word is inversely proportional to its rank in the frequency table.*

- **Simple Meaning**
- Few words appear **very frequently**
- Many words appear **rarely**

Zipf's Law Equation

- $f(r) = \frac{C}{r^s}$
- Where:
- $f(r)$ =frequency of word with rank r
- r= rank of the word
- C= constant
- $s \approx 1$
- **Simplified form**
- $f(r) \propto \frac{1}{r}$

- Zip’s Law Example

Rank	Word	Frequency
1	the	very high
2	is	high
10	data	medium
1000	entropy	low

➢ Top few words dominate the text.

- **Why Zipf's Law Happens (How it Works)**
- Speakers reuse common words
- Rare words are used for specific meanings
- Language balances **effort vs clarity**
- This natural behavior produces Zipf distribution.
-

- **Slide 6: Why Zipf's Law is Needed / Used**
- Explains why stopwords dominate text
- Helps in **stopword removal**
- Helps in **feature selection**
- Improves **search & compression**
-

- **Slide 7: Where Zipf's Law is Used**
- Information Retrieval
- Search engines
- NLP preprocessing
- Language modeling
- Text compression
-

- **Slide 8: Limitations of Zipf's Law**
- ❌ Does not capture meaning
  ❌ Not exact for all corpora
  ❌ Works statistically, not logically

2. Heap's Law

- **Heaps' Law Equation**
- $V(N) = k \times N^{\beta}$
- Where:
- $V(N)$ =vocabulary size
- N= total number of words
- k= constant
- $0.4 \leq \beta \leq 0.6$

- **Heap's Law Example**

Total Words (N)	**Vocabulary (V)**
1,000	400
10,000	2,000
1,000,000	50,000

➢ New words appear slower as data grows.

- **How Heaps’ Law Works**
- Early text → many new words
- Later text → mostly repeated words
- Vocabulary growth slows naturally
-

- **Slide 13: Why Heaps’ Law is Needed / Used**
- Predict memory requirements
- Estimate vocabulary size
- Plan indexing and storage
- Optimize NLP pipelines
-

- **Slide 14: Where Heaps’ Law is Used**
- Search engines
- Large corpus processing
- NLP system design
- Tokenizer and embedding planning

Relationship Between Zipf & Heaps

Zipf's Law	**Heaps' Law**
Frequency vs Rank	Vocabulary vs Corpus size
Explains word frequency	Explains vocab growth
Local distribution	Global growth

- Both describe **natural language behavior**

POS Tagging (Part Of Speech)

- **What is POS Tagging?**
- **Definition**
- POS Tagging is the process of **assigning grammatical labels** (noun, verb, adjective, etc.) to each word in a sentence.
- **Simple Meaning**
- POS tagging tells:
- *Which word is doing what job in a sentence.*

Why POS Tagging is Needed?

- Words can have **multiple meanings**
- Same word can behave differently in different sentences
- **Example**
- *"I **book** a ticket"* → **book (verb)**
- *"Read this **book**"* → **book (noun)**
- 👉 POS tagging removes ambiguity.

Why We Use POS Tagging?

- To understand **sentence structure**
- To improve **meaning extraction**
- To support higher-level NLP tasks
- **Without POS tagging:**
  Computer sees words only
- **With POS tagging:**
  Computer understands grammar + role

- **How POS Tagging Works (Basic Idea)**
- POS tagging works by:
- Reading words in sequence
- Using grammar rules or trained models
- Assigning the most likely tag based on **context**
- *A word's tag depends on surrounding words.*

POS Tagging Approaches

- **1. Rule-Based**
- Uses grammar rules
- Manual and limited
- **2. Statistical**
- Uses probability
- Learns from labeled data

Machine Learning / Deep Learning

- Uses HMM, CRF, LSTM, Transformers
- Most accurate (industry standard)

POS Tagging Equation (Statistical View)

- The goal is to find the **most probable tag sequence**:
- $\hat{T} = \arg\max_{T} \quad P(T \mid W)$
- **Where:**
- W = sequence of words
- T = sequence of POS tags
- Using Bayes' Rule:
- $P(T \mid W) \propto P(W \mid T) \times P(T)$

- POS Tagging Example

Sentence
“I love NLP”

Word	**POS Tag**
I	PRON
love	Verb
NLP	Noun

- **POS Tag Set (Common Tags)**

Tag	**Meaning**
NOUN	Name/thing
VERB	Action
ADJ	Description
ADV	Manner
PRON	Pronoun
PREP	Preposition

- **What Effect POS Tagging Has (Impact)**
- **Before POS Tagging**
- Flat text
- No grammatical understanding
- **After POS Tagging**
- Clear sentence structure
- Better meaning extraction
- Improved NLP accuracy
-

Where POS Tagging is Used

- Information extraction
- Named Entity Recognition (NER)
- Machine translation
- Question answering
- Text summarization
- Chatbots

- **Why Industry Uses POS Tagging**
- Improves feature quality
- Helps resolve word ambiguity
- Boosts downstream model performance
- POS tagging is a **foundation step** in NLP pipelines.
-

Limitations of POS Tagging

- ❌ Ambiguous words
  ❌ Domain-specific errors
  ❌ Needs labelled data
  ❌ Accuracy drops in informal text

Hidden Markov Model (HMM)

- **What is HMM?**

Definition

- A Hidden Markov Model (HMM) is a **statistical model** where:
- The system has **hidden states** (not directly visible)
- The states **generate observable data** (what we see)
- Probabilities govern **state transitions** and **emissions**
- "We see data, but the system behind it is hidden."

Simple Meaning

- Hidden states → true labels / causes
- Observed states → visible outputs / measurements
- **Example:** Weather (hidden) vs umbrella use (observed)

Why HMM is Needed / Used

- Some processes are **sequential / time-dependent**
- Hidden states affect observable data
- Used to **predict sequences**, like:
  - POS tagging in NLP
  - Speech recognition
  - Stock price modeling
  - DNA sequence analysis

- Components of HMM

Component	Meaning
States (S)	Hidden states (e.g., Rainy, Sunny)
Observations (O)	Observable data (e.g., Umbrella, No umbrella)
Initial Probabilities (π)	Probability of starting in a state
Transition Probabilities (A)	Probability of moving from one state to another
Emission Probabilities (B)	Probability of a state producing an observation

- HMM Probabilities (Equations)

- **1. Initial Probability**
- $\boldsymbol{\pi_i = P(q_1 = S_i)}$
- Probability that first state is S_i
- **.2Transition Probability**
- $\boldsymbol{a_{ij} = P(q_{t+1} = S_j \mid q_t = S_i)}$
- Probability of going from state $S_i$to S_j
- **.3Emission Probability**
- $\boldsymbol{b_j(k) = P(O_t = v_k \mid q_t = S_j)}$
- Probability of observing symbol $v_k$from state S_j

HMM Equation (Sequence Probability)

- For a state sequence $Q = q_1, q_2, \ldots, q_T$and observations $O = o_1, o_2, \ldots, o_T$:
- $\boldsymbol{P(O, Q) = \pi_{q_1} \cdot b_{q_1}(o_1) \cdot \prod_{t=2}^{T} a_{q_{t-1}q_t} \cdot b_{q_t}(o_t)}$
- Combines **initial**, **transition**, and **emission** probabilities
- Shows **joint probability** of a state sequence and observations

- **Simple Example**
- **Scenario**
- Hidden states: Weather → {Sunny, Rainy}
- Observations: Umbrella → {Yes, No}

Probabilities	Value
Initial (π)	P(Sunny)=0.6, P(Rainy)=0.4
Transition (A)	Sunny→Sunny=0.7, Sunny→Rainy=0.3, Rainy→Sunny=0.4, Rainy→Rainy=0.6
Emission (B)	Sunny→Umbrella=0.1, Sunny→No Umbrella=0.9, Rainy→Umbrella=0.8, Rainy→No Umbrella=0.2

How HMM Works

- **Initialization** → Assign initial state probabilities
- **Recursion** → Use **transition + emission probabilities** to compute sequences
- **Termination** → Pick the **most probable state sequence** (Viterbi algorithm)

Why HMM is Used

- Models **hidden sequences**
- Handles **time-series / sequential data**
- Works with **probabilistic predictions**
- Can **fill missing data** or infer hidden states

Effect of HMM on Data

- Converts **observations → hidden states**
- Handles **ambiguity in sequences**
- Predicts **patterns over time**
- Improves **accuracy in sequential NLP tasks** like:
    - POS tagging
    - Named Entity Recognition
    - Speech recognition
    - ➢ **When to Apply HMM**
- Sequential / time-series data
- Hidden information affects observable data
- Applications:
    - POS tagging in NLP
    - Speech recognition
    - Bioinformatics (DNA sequences)
    - Stock market trends

Named Entity Recognition (NER)

What is NER?

- **Named Entity Recognition (NER)** is an NLP technique used to **identify and classify important entities** in text into predefined categories.
- 📌 **Common Entity Types**
- **PERSON** → names of people
- **ORG** → organizations
- **LOC / GPE** → locations, countries, cities
- **DATE / TIME** → dates, time
- **MONEY** → currency, prices
- **PERCENT** → percentages

Why NER? (Why Needed)

- NER helps machines **understand "who", "what", "where", and "when"** from raw text.
- Without NER → text is just words
  With NER → text becomes **structured information**

Why We Use NER

- Extract key information automatically
- Convert unstructured text → structured data
- Improve downstream NLP tasks
- Reduce manual data extraction

Where NER is Used (Applications)

- 📰 **News analysis** – people, places, organizations
- 💬 **Chatbots & Virtual Assistants**
- 🏦 **Finance** – company names, money, transactions
- 🏥 **Healthcare** – disease, drug names
- ⚖️ **Legal documents** – names, laws, dates
- 🔍 **Search engines** – entity-based search

How NER Works (Simple Flow)

Raw Text

↓

Tokenization

↓

POS Tagging

↓

NER Model

↓

Labelled Entities

How NER is Implemented

- NER is treated as a **sequence labeling problem**
- Each word is assigned a tag like:
- **B-PER** → Beginning of Person
- **I-PER** → Inside Person
- **O** → Outside entity

NER Tagging Example
Sentence:
"Virat Kohli plays for India and RCB."

Word	NER Tag
Virat	B-PER
Kohli	I-PER
Plays	o
For	o
India	B-LOC
And	o
RCB	B-ORG

NER Model Types

- **Rule-based** (dictionary + patterns)
- **Statistical**
  - HMM
  - CRF
- **Deep Learning**
  - BiLSTM + CRF
  - Transformers (BERT, RoBERTa)
- 👉 **Industry uses Transformer-based NER**

NER Mathematical View (High Level)

- NER finds the **best label sequence** for words:
- $\hat{Y} = \arg\max_{Y} \quad P(Y \mid X)$
- Where:
- **X** = input words
- **Y** = entity labels

- **Effect of Using NER on Data**
- ✅ Text becomes structured
  ✅ Easier querying & analytics
  ✅ Better accuracy in QA systems
  ✅ Improves intent & context understanding

NER Example (Real Life)

- **Input:**
- *“Apple acquired Beats for $3 billion in 2014.”*
- **Output:**
- **Apple** → ORG
- **Beats** → ORG
- **$3 billion** → MONEY
- **2014** → DATE

Limitations of NER

- Ambiguity (Apple = fruit or company)
- Domain dependency
- Needs large labeled data
- Struggles with unseen entities

NER Tools & Libraries

- **spaCy** ⭐ (industry standard)
- **NLTK**
- **HuggingFace Transformers**
- **Stanford NER**

Viterbi Algorithm

- **1. Viterbi Algorithm**

What is Viterbi Algorithm?

- The **Viterbi Algorithm** is a **dynamic programming algorithm** used to find the **most probable sequence of hidden states** in a **Hidden Markov Model (HMM)**.
- 📌 In simple words:
- **"Given observations, Viterbi finds the best hidden path."**

Why Viterbi Algorithm? (Why Needed)

- HMM has **hidden states**
- We observe only **outputs**, not states
- Many possible state sequences exist
  👉 Viterbi efficiently finds the **best one**

What Problem It Solves

- Decoding problem in HMM
- Finds:
- $\arg\max P\,(\textit{State Sequence} \mid \textit{Observations})$

Where Viterbi is Used

- POS tagging
- Speech recognition
- Named Entity Recognition
- Bioinformatics (gene prediction)
- OCR (handwriting recognition)

How Viterbi Works (Step-by-Step)

- **Initialization**
- **Recursion** (dynamic programming)
- **Termination**
- **Backtracking** (best path)

Viterbi Equation

- $\boldsymbol{\delta_t(j) = \max_i \left[\delta_{t-1}(i) \cdot a_{ij}\right] \cdot b_j(o_t)}$
- Where:
- $\delta_t(j)$ =max probability till time t
- a_{ij} =transition probability
- $b_j(o_t)$ =emission probability
- o_t =observed word

Simple Example (POS Tagging)

- Sentence:
- **“I eat fish”**
- Hidden states: **Noun (N), Verb (V)**
  Observed words: *I, eat, fish*
- Viterbi finds:
- I → N
- eat → V
- fish → N

- **✓ Best tag sequence**

Baum–Welch Algorithm

What is Baum–Welch Algorithm?

- The **Baum–Welch Algorithm** is a **training algorithm** used to **learn HMM parameters** when hidden states are unknown.
- 📌 In simple words:
- **“Baum–Welch teaches the HMM.”**

- **Why Baum–Welch? (Why Needed)**
- We don’t know:
  - Transition probabilities
  - Emission probabilities
- Data is **unlabeled**
  👉 Baum–Welch learns parameters automatically

What Problem It Solves

- **Learning problem** in HMM
- Estimates:
  - Initial probabilities
  - Transition matrix
  - Emission matrix

Where Baum–Welch is Used

- Speech recognition
- POS tagging (unsupervised)
- Biological sequence modeling
- Time-series pattern learning

How Baum–Welch Works

- It is based on **Expectation–Maximization (EM)**
- **Step 1: Expectation (E-step)**
- Compute expected counts of states
- **Step 2: Maximization (M-step)**
- Update HMM parameters
- Repeat until convergence.

Baum–Welch Equations (High-Level)

- Forward probability:
- $\boldsymbol{\alpha_t(i) = P(o_1 , . , . , . o_t q_t = i)}$
- Backward probability:
- $\boldsymbol{\beta_t(i) = P(o_{t+1} , . , . . o_T \mid q_t = i)}$
- State probability:
- $\boldsymbol{\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O)}}$

Simple Example

- Observed sentence:
- **“time flies like arrow”**
- No tags given.
- Baum–Welch:
- Learns which words behave like **verbs**, **nouns**
- Automatically adjusts probabilities

Effect of Using Baum–Welch

- ✅ Trains HMM without labeled data
  ✅ Improves model accuracy
  ✅ Adapts to new datasets

➢ **Baum–Welch learns the parameters of an HMM from unlabeled data**.

- Viterbi vs Baum–Welch (Quick Table)

Feature	Viterbi	Baum–Welch
Purpose	Decoding	Training
Finds	Best state sequence	Best model performance
Data	Observed + train HMM	Only observeb
Output	State path	Transition & emission
Type	Dynamic programming	EM algorithm

- The **Expectation-Maximization (EM)** algorithm is a powerful iterative optimization technique used to **estimate unknown parameters in probabilistic models**, particularly **when the data is incomplete**, **noisy or contains hidden (latent)** variables.

- **Two Step**
  1. E-step
  2. M-Step

Custom Feature (Feature Engineering)

What is a Custom Feature?

- A **custom feature** is a **manually designed input variable** created using **domain knowledge** to help a machine learning or NLP model perform better.
- 📌 In simple words:
- **Any feature that is not directly present in raw data and is created by us is a custom feature.**

Example (Very Simple)

- **Raw Text:**
- "Virat Kohli scored 100 runs"
- **Custom Features:**
- Word length
- Is word capitalized?
- Is number present?
- Is word a player name?
- Position of word in sentence
- These are **custom features**.

Are Custom Features Used in Industry?

YES, they are used — but selectively

Model Type	Custom Features
Traditional ML (SVM, CRF, LR)	✅ Highly used
Rule-based systems	✅ Required
Deep Learning (LSTM)	⚠ Sometimes
Transformers (BERT, GPT)	❌ Rare

- **Why Industry Uses Custom Features**
- Improve accuracy on small datasets
- Add domain knowledge
- Reduce training data requirement
- Increase interpretability
- Help rule-based + ML hybrid systems

- **Why Industry Sometimes Avoids Custom Features**
- Time-consuming
- Hard to generalize
- Not scalable
- Deep models learn features automatically
- 📌 That's why **modern NLP prefers representation learning**, not heavy feature engineering.

- **Why Custom Features Are Still Important**
- In **low-resource data**
- In **domain-specific NLP** (legal, medical)
- In **structured NLP tasks** (NER, POS with CRF)
- In **production systems needing explainability**

What Can Be a Custom Feature?

Common Custom Features in NLP

Feature Type	Example
Lexical	Word, prefix, suffix
Orthographic	Capitalization, digits
Syntactic	POS tag
Position	Word index
Semantic	Gazetteer match
Contextual	Previous/next word

Thank You

By – Deepak Suthar