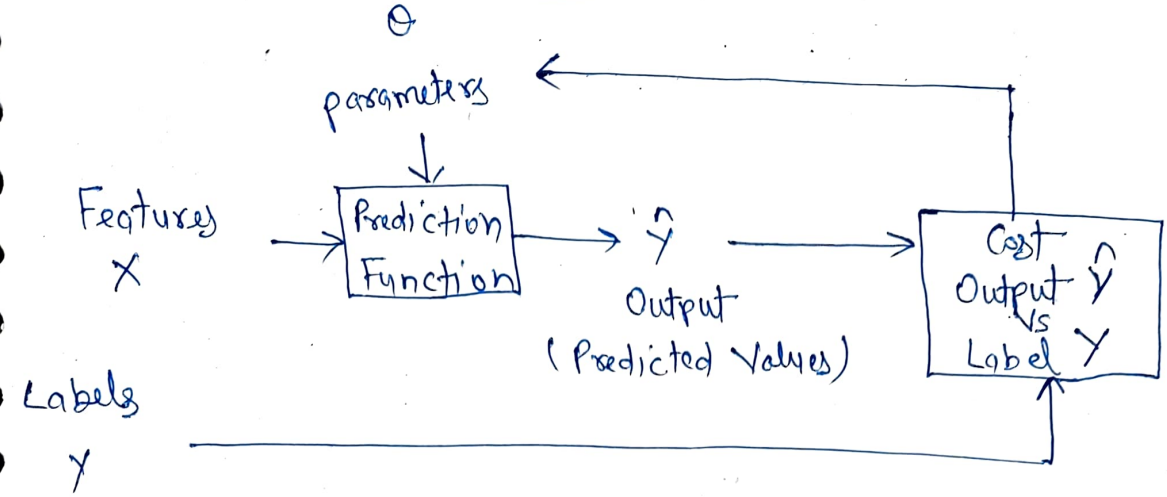
NLP



Tweet: I am happy because I am learning NLP

Logistic Regression → Positive 1
→ Negative 0

Vocabulary & Feature Extraction

Tweets
[tweet1, tweet2 -- tweetm]

I am happy because I am learning NLP

I hated the movie

V = [I, am, happy, because --- hated, the movie]

I am happy because I am learning NLP

[I am happy because NLP -- hated the movie]

[1, 1, 1, 1, 1 .... 0, 0, 0]

A lot of zeros, sparse representation.

I am happy I am learning NLP

All zeros

[1, 1, 1, 1, 1, 1 -- 0, 0, 0]

1 ← |V| → Large Training Time

[θ0, θ1, θ2 -- θn] → Large Prediction Time

n=|V|

Corpus: Positive & negative counts

I am happy because I am learning NLP } +ve class
I am happy

I am sad, I am not learning NLP } -ve class
I am sad

Vocabulary
I
am
happy
because
learning
NLP
sad
not

Positive Table

Vocab	Post Freq.
I	3
am	3
happy	2
because	1
learning	1
NLP	1
sad	0
not	0

Negative Table

Vocab	Post Freq.
I	3
am	3
happy	0
because	0
learning	1
NLP	1
sad	2
not	1

Encode the words using frequency of the class

Feature Extraction

freqs: dictionary mapping from (word, class) to frequency

$$X_m = \left[1, \sum_w freqs(w,1), \sum_w freqs(w,0)\right]$$

I am sad, I am not learning NLP.

$X_m = [1, 8, 11]$

Now Vector size reduced to 3.

Preprocessing: Stop words & punctuation

@joshua_bengio and @Geoffrey_Hinton are tuning a ~~great~~ GREAT AI model at https://www.umontreal.ca !!!

Stop words	Punctuation
and	,
is	.
are	:
at	;
has	!
for	"
a	'

tuning GREAT AI model

Stemming & lowercasing

tun ← tune, tuned, tuning

GREAT, Great, great → great

Preprocessed tweet:

[tun, great, ai, model]

General Overview:

I am Happy Because i am learning NLP @ AI

↓ Preprocessing

[, happy, learn, nlp]

Bias ← [1, 4, 2] → Sum of Negative frequencies

↓

Sum of positive frequencies

X': Set of tweets

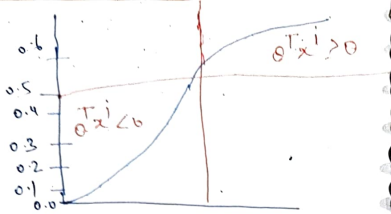
Our X will look like

$$\begin{bmatrix} 1 & X_1^{(1)} & X_2^{(1)} \\ 1 & X_1^{(2)} & X_2^{(2)} \\ & \vdots & \\ 1 & X_1^{(m)} & X_2^{(m)} \end{bmatrix}$$

[1, 40, 20]
⇔ [1, 20, 50]
[1, 5, 35]

ogistic Regression overview

$$h(x^{(i)}, \theta) = \frac{1}{1 + e^{-\theta^T x^{(i)}}}$$



@ Yoshua Bengio @ Geoffrey Hinton are training a GREAT model.

↓

[fun, ai, great, model]

$$x^{(i)} = \begin{bmatrix} 1 \\ 3476 \\ 245 \end{bmatrix} \qquad \theta = \begin{bmatrix} -0.0003 \\ -0.00150 \\ -0.00120 \end{bmatrix}$$

Naive Bayes classifier.

Corpus of tweets

+	+	+	+	+
+	+	+	+	+
+	+	+	−	−
−	−	−	−	−

A → Positive tweet

$P(A) = 13/20 = 0.65$

Tweets containing the words happy

+	+	+	+	+
+	+	+	+	+
+	+	+ ha	− PP	− X
−	−	−	−	−

$P(\text{Negative}) = 1 - P(\text{positive}) = 1 - 0.65 = 0.35$

Tweets containing the words happy

		h	a PP	X

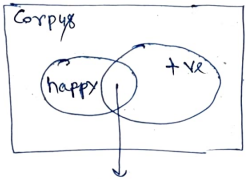
B → Tweets containing happy

$P(\text{happy}) = N_{\text{happy}}/N$

$P(B) = 4/20 = 0.20$

Probability of intersection

$P(A \cap B) = P(A, B) = 3/20 = 0.15$



Conditional Probabilities:

+ h	+ a	+ ha	PPY

$P(A|B) = P(\text{Positive} | \text{happy})$

$P(A|B) = 3/4 = 0.75$

$P(B|A) = P(\text{happy} | \text{Positive}) = 3/13 = 0.231$

$$P(\text{Positive} | \text{happy}) = \frac{P(\text{Positive} \cap \text{happy})}{P(\text{happy})}$$

Baye's Rule:

$$P(\text{Positive} | \text{happy}) = \frac{P(\text{Positive} \cap \text{happy})}{P(\text{happy})}$$

$$P(\text{happy} | \text{Positive}) = \frac{P(\text{happy} \cap \text{Positive})}{P(\text{Positive})}$$

Summary:

Conditional Probabilities → Bayes' Rule

$$P(X/Y) = \frac{P(Y/X) \times P(X)}{P(Y)}$$

Naive Bayes for Sentiment Analysis

Corpus.

+ve tweets

I am happy because I am learning NLP.
I am happy, not sad

-ve tweets

I am sad, I am not learning NLP.
I am sad, not happy

Words	Pos	Neg	P(wi/class) Pos	Neg
I	3	3	3/13 = 0.24	0.25
am	3	3	0.24	0.25
happy	2	1	0.15	0.08
because	1	0	0.08	0.0
learning	1	1	0.08	0.08
NLP	1	1	0.08	0.08
sad	1	2	0.08	0.17
not	1	2	0.08	0.17
	13	12	1	1 ← Sum

Naive Bayes:

Tweet: I am happy today, I am learning.

$$\prod_{i=1}^{m} \frac{P(w_i/Pos)}{P(w_i/Neg)} = \frac{0.14}{0.10} = 1.4 > 1$$

$P(I/+ve) = P(I/Pos)$

$P(+ve/w) \propto P(w/Positive)\,P(+ve)$

$$\frac{0.26}{0.20} * \frac{0.20}{0.20} \times \frac{0.14}{0.10} \times \frac{0.26}{0.20} \times \frac{0.26}{0.20} \times \frac{0.10}{0.10}$$

$$= \frac{P(w/Positive)\,P(+ve)}{P(w)}$$

NB: Based on Bayes Theorem widely used for text classification:

Spam detection, sentiment analysis, topic classification, document filtering.

Naive → assume that features (words) are conditionally independent given the class.

$$P(c/x) = \frac{P(x/c)\,P(c)}{P(x)}$$

c: class, x = Document

P(c): Prior probability of class

P(x/c): likelihood

P(c/x): posterior probability

P(x) is constant for all the class

$$P(c/x) \propto P(c) \prod_{i=1}^{n} P(w_i/c)$$

aive Independence Assumption

· for a document $x = \{w_1, w_2 \dots w_n\}$

$$P(X|C) = \prod_{i=1}^{n} P(w_i|c)$$

Interpretation

The presence of one word does not affect the probability of another word given the class.

Steps:

1) Text Preprocessing (tokenization, stopword removal)
2) Feature Extraction (Bag of words / word counts)
3) Compute:
   - class Prior $P(c)$
   - word likelihood $P(w|c)$
4. Apply Bayes Rule
5. choose class with maximum posterior probability.

Example:

Document	Text	Class
D_1	free money offer	Spam
D_2	free prize money	Spam
D_3	meeting schedule today	Ham
D_4	Project meeting today	Ham

Vocab = { free, money, offer, prize, meeting, schedule, today, Project }

Vocab Size = 8

Class prior ⇒ $P(Spam) = \frac{2}{4} = 0.5$

$P(Ham) = \frac{2}{4} = 0.5$

Word Counts per class:

word	Count	
free	2	Spam Documents
money	2	
offer	1	
prize	1	

Words	Count	
meeting	2	Ham Documents
today	2	
schedule	1	
project	1	

Total words in Spam = 6

Total words in Ham = 6

Likelihood with Laplace Smoothing:

$$P(w|c) = \frac{\text{Count}(w,c) + 1}{\text{total words in } C + |V|}$$

Ex: $P(\text{free}|\text{Spam}) = \frac{2+1}{6+8} = 3/14$

$P(\text{free}|\text{Ham}) = \frac{0+1}{6+8} = 1/14$

Test Document: free money

Spam Probability:

$P(\text{Spam}|X) \propto P(\text{Spam}) \times P(\text{free}|\text{Spam}) \times P(\text{money}|\text{Spam})$

$= 0.5 \times \frac{3}{14} \times \frac{3}{14} = 0.0229$

$P(\text{Ham}|X) \propto 0.5 \times \frac{1}{14} \times \frac{1}{14} = 0.00255$

Final Prediction: $P(\text{Spam}|X) > P(\text{Ham}|X)$

· Spam (✓)

Advantage

Simple & easy to implement
Very fast
works well with high dimensional data
performs well with small datasets
Strong baseline for NLP tasks.

Limitation:

- Independence assumption is unrealistic
- Can't capture word order semantics
- poor performance when feature are correlated
- lower accuracy compared DL model.

Hidden Markov model (HMM)

HMM is a probabilistic sequence model where

- System is assumed to be a markov process (current state depends only on the previous state)
- The states are hidden, but
- Each hidden state emit on observable symbol.

In NLP:

- Hidden States: linguistic labels (POS tags, NER tags, phonems)
- Observations → words (or characters)

Why HMM in NLP?

Natural language is

- Sequential
- Context dependent
- Uncertain

HMMs model:

- Sequential dependency: via transition probabilities.
- Word-label uncertainty: via emission probabilities.

Core Assumptions of HMM:-

1. Markov Assumption

$$P(S_t / S_{t-1} S_{t-2} \cdots) = P(S_t / S_{t-1})$$

2. Output Independence Assumption

$$P(O_t / S_t S_{t-1} \cdots) = P(O_t / S_t)$$

Component of Hmm :

An Hmm is defined by $\lambda \Rightarrow (A, B, \pi)$

(a) State (S)

Hidden labels:

Example (POS tagging). $S = \{ \text{Noun}(N), \text{Verb}(V) \}$

(b) Observation (O)

observed words $O = \{ \text{dog}, \text{barks} \}$

(c) Transition Probabilities (A) : $A_{ij} = P(S_j / S_i)$

(d) Emission Probabilities (B):

$$B_j(o) = P(o / S_j)$$

(e) Initial Probabilities (π)

$$\pi_i \Rightarrow P(S_i = i)$$

Application of HMM in NLP.

Task	Hidden State	observation
POS Tagging	POS Tags	words
NER	Entity Tag	words
Speech Recognition	Phonems	Acoustic Signal
Text Segmentation	Segment label	characters
language modeling	latent state	words

Time/NN Flies/VB

Fruit/NN flies/NNS

Time/NN like/IN arrow/~~IN~~ NN.

State = {NN, VB, NNS, IN}

observation (words) = { Time, flies, Fruit, like, arrow}

Intial Probability $P(NN) = 1.0$

Transition Probability :-

NN → VB : 1 $P(VB/NN) = \frac{1}{3}$

NN → NNS : 1 $P(NNS/NN) = 1/3$

NN → IN : 1 $P(IN/NN) = 1/3$

Emission Probability

Tag	word	Probability
NN	Time	2/4
NN	Fruit	1/4
NN	arrow	1/4
VB	Flies	1
NNS	flies	1
IN	like	1

word 1 : Time

$V_1(NN) = P(NN) \times P(Time/NN) = 1.0 \times 0.5$

$= 0.5$

ord 2: fly

NN → VB $0.5 \times \frac{1}{3} \times 1 = 0.1667$

NN → NNS $0.5 \times \frac{1}{3} \times 1 = 0.1667$

Both path tie

Time/NN Flies/VB

Time/NN fly/NNS

Q. 2 Given Training corpus:

I/PRP like/VB dogs/NNS

You/PRP like/VB cats/NNS

1) Find the state & observation
2) Compute Initial & transition probability
3) Compute the probability of sentence "I like dogs"

Soln: States & observations

states {PRP, VB, NNS}

observations {I, You, like, dogs, cats}

Probability Estimation

P(PRP) = 1

Transition Probability

Transition	Prob
PRP → VB	1
VB → NNS	1

Emission Probability

Tag	word	Probability
PRP	I	0.5
PRP	You	0.5
VB	like	1
NNS	dogs	0.5
NNS	Cats	0.5

Sentence Prob Calculation

⇒: I like dogs

$P = P(PRP) \times P(I|PRP) \times P(VB|PRP) \times P(like|VB) \times P(NNS|VB) \times P(dogs|NNS)$

$= 1 \times 0.5 \times 1 \times 1 \times 1 \times 0.5$

$= 0.25$

imitation of HMM

- Strong independence assumptions (output depends only on current state).
- Limited context modelling due to first order markov assumption.
- Poor handling of long range dependencies
- Data sparsity & zero probability problem.
- Generative nature, model P(words | tags) instead of P(tags/words)

Modern Solutions:

- LSTM / GRU → Capture long term dependencies
- Transformer-based models (BERT, ROBERTA)
  ⇒ Context representations
- Hybrid HMM – Neural models – useful for low-resource languages.

Training Corpus:

	DET	NOUN	VERB
1.	the	dog	barks
	DET	NOUN	VERB
2.	a	dog	runs
	DET	NOUN	VERB
3.	the	cat	runs

Test: the dog runs

Tag Set: DET, NOUN, VERB

Vocab: the, a, dog, cat, barks, runs

Frequency Counts:

Tag Counts: DET = 3, NOUN = 3, VERB = 3

Transition Count:

START → DET = 3, DET → NOUN = 3, NOUN → VERB = 3

Emissions:

DET: the = 2, a = 1

NOUN: dog = 2, cat = 1

VERB = barks = 1, runs = 2

Laplacian Smoothing:

Number of tags = 3, Vocab size = 6

$P(\text{NOUN}/\text{DET}) = \frac{3+1}{3+3} = 4/6$

$P(\text{DET}/\text{DET}) = 0+1/3+3 = 1/6$

Emission Prob:

$P(\text{the} \mid \text{DET}) = \frac{2+1}{3+6} = 3/9$

$P(\text{dog} \mid \text{Noun}) = \frac{2+1}{3+6} = 3/9$

$P(\text{runs} \mid \text{VERB}) = \frac{2+1}{3+6} = 3/9$

Viterbi (Normd Probability)

Sentence: the dog runs.

Intidization: $\text{DET} = 4/6 \times 3/9 = 0.222$

Recursion:

$\text{Noun} = 0.2222 \times \frac{4}{6} \times \frac{3}{9} = 0.0494$

$\text{VERB} = 0.0494 \times \frac{4}{6} \times \frac{3}{9} = 0.0110$

Best path = DET → Noun → VERB

Viterbi (log probability)

$\ln(4/6) = -0.405$

$\ln(3/9) = -1.099$

DET = −1.504

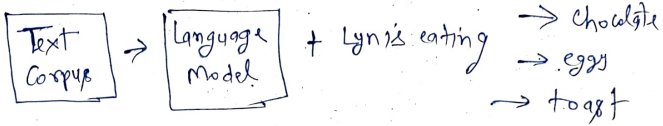
Noun = −3.008

VERB = −4.512

Language Model

N-gram based language model:

- Create a language model from text corpus to
- estimate the probability of word sequences
- estimate the prob of a word following a sequence of words.
- Apply this concept to autocomplete a sentence



other applications:

- Speech Recognition
- Spelling Correction
- Augmentative Communication

N-Gram:

An N-gram is a sequence of N words.

Corpus: I am happy because I am learning

Unigram: {I, am, happy, because, learning}

Bigram: {I am, am happy, happy because}

Trigrams: {I am happy, am happy because.}

Sequence Notation:

Corpus: This is great ... Teachers drinks tea. $m = 500$

$w_1 \; w_2 \qquad\qquad w_{500}$

$$w_1^m = w_1 w_2 \dots w_m$$

Unigram Probability:

Corpus: I am happy because I am learning.

Size of corpus $\Rightarrow m = 7$

$$P(I) = 2/7 \qquad P(\text{happy}) = 1/7$$

Bigram Prob

$$P(am|I) = \frac{C(I\ am)}{C(I)} = 2/2 = 1$$

$$P(\text{happy}|I) = \frac{C(I\ \text{happy})}{C(I)} = 0/2 = 0$$

$$P(y|x) = \frac{C(x,y)}{\sum_w C(x,w)} = \frac{C(x,y)}{C(x)}$$

Trigram Prob

$$P(\text{happy}|I\ am) = \frac{C(I\ am\ \text{happy})}{C(I\ am)}$$

$$P(w_3|w_1^2) = \frac{C(w_1^2 w_3)}{C(w_1^2)}$$

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1} w_n)}{C(w_{n-1})}$$

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like Sam </s>

Calculate the prob of $P(am/I)$ using the corpus.

$$P(w_n/w_{n-1}) = \frac{Count(w_{n-1}, w_n)}{Count(w_{n-1})}$$

$P(am/I) = 2/3 = 0.67$

Prob of sentence <s> I am Sam <s> according to bigram model.

Decomposition

$P(I/<s>) \times P(am/I) \times P(Sam/am) \times P(</s>/Sam)$

$P(Sentence) = \frac{2}{3} \times \frac{2}{3} \times \frac{1}{2} \times \frac{1}{3} = \frac{4}{54} = 0.074$

(3) Calculate $P(Sam/do)$ using Add-1 Laplacian Smoothing.

$$P = \frac{Count(w_{n-1}, w_n)+1}{Count(w_{n-1})+V}$$

$$P(Sam/do) = \frac{0+1}{1+6} = \frac{1}{7}$$

Perplexity: a standard metric used to evaluate how well a language model predict a sample.

Ex: S = I like Sam

$P(I/<s>) = 0.5$
$P(like/I) = 0.1$
$P(Sam/like) = 0.2$
$P(</s>/Sam) = 0.4$

Perplexity is the inverse probability of test set, normalized by the no of words, for a setence W with N words

$$PP(w) = P(w_1, w_2, \ldots w_N)^{\frac{-1}{N}}$$

$$PP = \sqrt[N]{\frac{1}{P(w_1/w_0) \times P(w_2/w_1) \ldots P(w_N/w_{N-1})}}$$

Note: Usually N include end of setence token </s> not <s> (start token)

For a sentence $W = w_1 w_2 \dots w_N$, the perplexity of a language model is defined as:

$$PP(W) = [P(W)]^{\frac{-1}{N}}$$

or equivalent in log form

$$PP(W) = \exp\left(\frac{-1}{N} \sum_{i=1}^{N} \log P(w_i / w_{i-(N-1)} \dots w_{i-1})\right)$$

Lower Perplexity $\Rightarrow$ better language model.

Ex: Sentence : <s> I love NLP </s>

No of words (including end token)

$N = 4$

Given Bigram Probability

Bigram	Prob
$P(I \mid <s>)$	0.5
$P(love \mid I)$	0.4
$P(NLP \mid love)$	0.1
$P(</s> \mid NLP)$	0.2

P(Sentence)

$P(W) = 0.5 \times 0.4 \times 0.1 \times 0.2$

$= 0.004$

Perplexity Calculation

$$PP(W) = (0.004)^{\frac{-1}{4}}$$

$$= \sqrt[4]{250} = 3.97$$

log-Prob version (Ref in practice)

$$\log P(W) = \log 0.50 + \log 0.40 + \log 0.10 + \log 0.20$$

$$= -(0.6931) + (-0.9163) + (-2.3026) + (-1.6094)$$

$$= -5.5214$$

$$PP(W) = \exp\left(-\frac{-5.5214}{4}\right)$$

$$= \exp(1.38035) = 3.97$$

Perplexity = 3.97

Sentence

<s><s> I love NLP </s>

<s><s> I love AI </s>

N = 4+4 = 8

Given trigram prob

Trigram — Prob

P(I | <s> <s>)

Basic Word Representations:

- integer
- One hot vectors
- Word embeddings

Word	Number
a	1
able	2
about	3
⋮	⋮
hand	615
happy	621
zebra	1000

happy

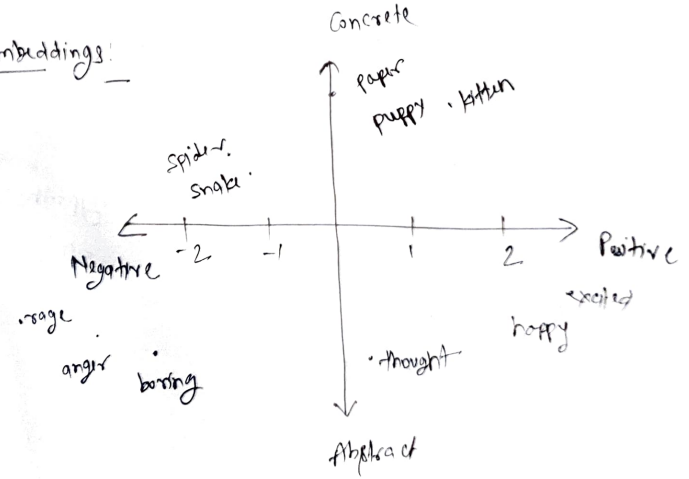
1	0	a
2	0	able
3	0	about
⋮	⋮	⋮
615	0	hand
621	1	happy
1000	0	zebra

⇒ One hot encoding

Pros: Simple & require no implied ordering.

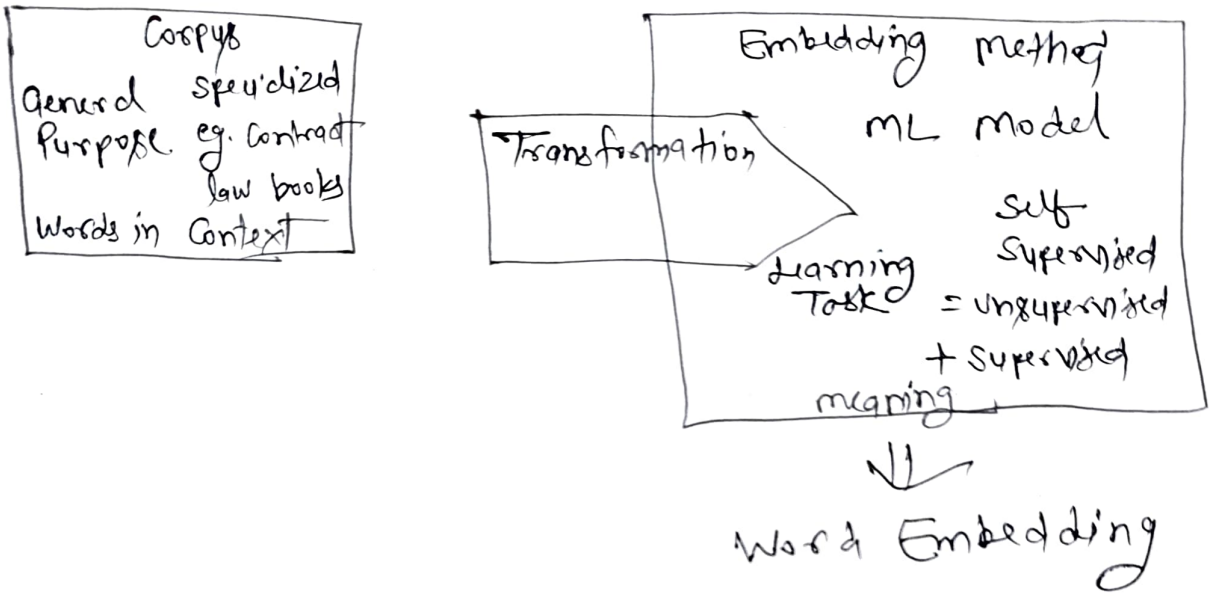
Cons: one hot vectors: huge & encode no meaning.

Embeddings:



Word Embedding Process

Hyperparameters
Word Embedding Size



Classical method

. Word2Vec (Google 2013)

Continuous bag of word (CBOW) the model learn to predict the centre word given some context words.

• Word 2 Vec operates on Distributional hypothesis words that appear in similar context share similar meaning

The cat sit on the mat

The dog sits on the mat

CBOW

input [The _ sits on]

. Target : Cat