(*) validating the [K] value

(i) kneelocator
(ii) silhautee scoring

NLP

- What about features that are completely text?

→ We can convert those text wit to continuous values by using [OneHotEncoding]

→ But Letter wise OHE is not viable,

So, we've to represent the data using vectors

- If we start OHE for letter wise we'll have

[[0,0,0,1,0], [----]] very big.

(A) Roadmap

- Token Text Preprocessing: Tokenization, Lemmatization, Stemming, Stopwords
- Text Preprocessing II (Vectors repr): Bag of words, TFIDF, Unigram, bigrams

[Input -> vectors] Better [Advanced]

- Text Preprocessing III : word2vec, Avg Word2vec
- RNN, LSTM, GRU,
- Text Preprocessing : Word embedding (uses word2vec)
- Transformer, Bert

→ Accuracy increases from bottom to top.

Deep Learning : RNN, LSTM, GRU
↳ Tensorflow, Pytorch

BERT, Transformer : NLTK, SPacy libs.

Tokenization in NLP

(i) Corpus : Paragraph
(ii) Documents : sentences
(iii) vocabulary : Unique words [dictionary]
(iv) words : all the words in the corpus

Tokenization is process to convert Corpus, ~~documents~~ to tokens i.e sentences.

[sent1. sent2]

so,

[sent1
sent2] into token

- Split on the basis of full stops.

Then again, apply Tokenization to convert to [words] tokens.

- once we've all the words we'll find the unique words i-e Vocabulary for the corpus [prompt]

Text Preprocessing

- NLTK library [Advanced RegEx, Pattern Matching]

- sent_tokenize (corpus) ~~document~~

Uses "Punkt" model for tokenization.

- word_tokenize (document)

Converts every characters, words to separate token.
[, word, ;, !, ...]

Then we clean to make actual words.

- word punct_tokenize
- Punctuations are also considered tokens.

- TreeBankWordTokenizer()

→ considers [.] full stops with the word itself.

e.g I am good.

So, ["I",
"am",
"good."]

(*) Stemming

Process of reducing a word to it's [word stem] that affixes to suffixes and prefixes or to the [roots] of words unknown as a [lemma.]

In data, we can have same words with different verbs such as

eat, eaten, eat, eating, eats

whose root word is [eat]
So, we can only use [eat] instead of every single word.

- PorterStemmer.
  Stem.PorterStemmer

- stem(word) => stemword

e.g. "eating" => eat
"History" => histori (disadv, for some words not accurate)

- RegexpStemmer()
  -> can use RegEx for each word to generate the stem.

  RegexStemmer('ing$') checks if end is ing and give stem.

- Snowball stemmer
  - Better than Porter stemmer

  - stem import SnowballStemmer

  SnowballStemmer('english').stem(word)

  -> still not efficient. for some words

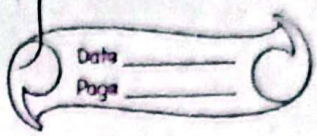
  Use "Lemmatization"

Lemmatization

- similar to stemming. The output after lemmatization is lemma which is root word rather than stem.

  nltk.WordNetLemmatizer()
  -> wrapper around WordNet Dataset.

→ Lemmatization preserves the meaning



- WordNetLema(). lemmatizer("word", Pos
  noun verb adj adverb
  Pos = n, v, a, r

we can find for noun, adverb, verbs etc.

→ WordNetLemma takes a lot of time.
can be used for chatbots, text summarization,

⊛ Stopwords

In a data, words like I, the, have, and, to does not play a big role. So we need to remove such words.

Stopwords
→ Every language has stop words
e.g Nepali → [हो, छ, थ्यो, पनि]
we need to remove such words.

→ we can download or have our own set of stopwords.

Download and remove those words.

⊛ Steps
- Tokenization
- Stopword [if not in stopword apply lemma]
- Lemmatization, stemming, snowball

- Converts words to lowercase and the apply the stopword, lemma, stemming

(*) Parts of Speech Tagging

- Each parts of speech has a representation in short (Abbreviation)

nltk.post_tag(word) => Repts of Part of Speech. Tuple.

For ex,

[(I, PRP), (three, CD), (vision, NNS)]

- PRP => Personal Pronoun.
- NNP => Proper noun.

(*) Name Entity Recognition

Earlier we've got Parts of speech but we can also find the category or entity of each word. Perfect example of category.

For e.g:

"The Eiffel tower was built by Bisut"

Person: Bisut
Place: India
Date:
Money:

- ne_chunk(nltk.Post_tag(word))

We can draw and visualize as well.

How to Approach a Dataset / Problem?

- Identify the Problem i.e Summarization, sentiment, Classification, Translation.

(i) start with Cleaning (Tokenization to word)
- Use Regex for data cleaning [Rows]
- Lower case

(ii) ~~Tokenization~~ Stemming, Lemmatization, Stop words, Parts of speech, Name entity.

(iii) Join and prepare the data for Embedding [vector repr]
- Transformer, BERT [word embeddings]
- word2vec
- Bag of words, TF-IDF
- one HotEncoding [very less category] [Not eff]

(iv) one we've vector representation, we train with ML, Deep Learning Algos.

{word2vec, Bag of words} → [Gensim Lib]

Text to Vector

- OneHotEncoding
  -> unique words in the dataset -> Sparse matrix
  -> Length will be length of unique x total words.

e.g if, Total unique words are 3
then,
To represent a sentence with 4 words.

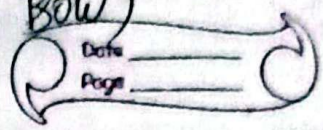
[[0,1,0], [1,0,0], [0,0,1], [0,1,0]]
↳ sent 1.

3x4 => 12 i.e 12 values for sent of 4 words.

-> Not efficient.
-> Sparse leads to overfitting.
-> No semanting meaning.

- Bag of Words
  -> Lower all words
  -> Stop words [Reduce words]
  -> Calculate vocubulary (unique)
  -> Calculate the frequency of vocubulary.
  -> Sort on Descending order
  ->

→ Increase the count (Non Binary BoW)
→ Boolean for (Binary)



eg

He is a good boy good → good boy good
She is a good girl → good girl

* vocab

good → 3
boy → 2
girl → 2

* vector repr

	good	boy	girl	
S1	2	1	0	→ vectors.
S2	1	0	1	

cont..

→ This approach has less values, represents more with less

→ ~~Issues~~
→ Does not have fixed size inputs.

- Ordering of the word is changing.
- Does not considers any new words while testing i-e out of vocab problem
- No capture semantic meaning [which is the most imp word?]

- N-grams (How to build vocabulary)

eg) The food is good
The food is not good

Vocab: [food not good]

S_1	1	0	1	vectors should be very diff, but they aren't [solve with ngrams]
S_2	1	1	1	

so, what if while building vocabulary we take combination of vocabs.

i.e Bigrams, Trigrams
Two words Three words => Combination

Then vocab becomes

	food	not	good	food good	food not	not good
S_1	1	0	1	1	0	0
S_2	1	1	1	0	1	1

Now, if we compare this vectors they are very different. This shows semantic meaning.

Sklearn -> n-grams (1,1) -> unigrams
-> (1,2) -> unigram, bigram
-> (1,3) -> unigram, bigram, trigram.

- feature_extraction.text import CountVectorizer.
CountVectorize (n_gram_range = (1,2)) bigram

- TF-IDF [Term frequency - Inverse Document Frequency]

(i) TF = $\frac{\text{No. of repeation of words in sentence}}{\text{total words in sentence.}}$

(ii) IDF = $\log_e \left(\frac{\text{No. of sentence}}{\text{No. of sentence containing the words}} \right)$

* TF

	S_1	S_2
good	1/2	1/2
boy	1/2	0/2
girl	0	1/2

* IDF

good $\log_e(2/2) = 0$
boy $\log_e(2/1)$
girl $\log_e(2/1)$

we multiply TF * IDF

	good	boy	girl	
S_1	1/2 x 0 = 0	1/2 x $\log_e(2/1)$	0	vector
S_2	0	0	1/2 $\log_e(3/2)$	vector

-> Train this vector

Advs

- Fixed size
- Word importance is being captured [Repeated words have less value]

Dis

- Out of vocabulary problem

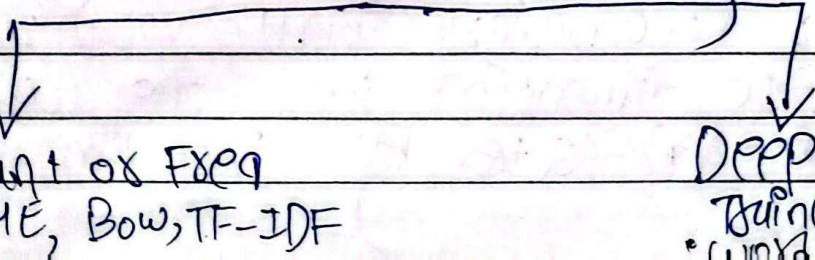
.text import TfidfVectorization.

Word Embedding
- Representation of text in vector.
- Text is encoded into vector.

-> Words with close meaning are closer in the vector space
-> Words with diff meaning are far in the vector space.

e.g
Happy ≈ excited [so closer vector space]
Angry ≠ Happy [Far and opposite space]

-> Distance is calculated, cosine similarity.



→ Deep Learning has better accuracy.

Word2Vec (Gensim library)

Two types (Based on Training)
→ CBow (continuous Bag of words)
→ Skipgram

- Text to vector
- Uses Neural Networks to learn word associations from a large corpus of text.
- Once trained model can detect synonymous words or suggest additional words for a particular sentence.
- Represents each distinct words with a particular list of numbers called a vector.

(*) Intuition

→ First we've a set of dimensions or features [category] that word belongs to.

Say 300 dimension, such as Gender, Food, Animal, Location etc.

→ Then our vocabulary (input's vocab would be of 300 dimension i.e one word would be represented by 300 dim vector.

- Relation of word with the Dimension (category)
→ How strong the xpsn is.

Date
Page

e.g

(Dim)	Vocabs				
Category	Boy	Girl	Mi	You	Mango
Gender	-1	1	.	.	
Royal	0.01				
Age	0.03				
Food					
Animal					

- These all values comes from the trained word2vec model.

Now,
After we've vector xepresentation,

$$[\vec{KING} - \vec{MAN} + \vec{QUEEN}] = \text{GIRL}$$

↳ closes to QUEEN.

(A) Cosine Similarity

- Distance based on point i.e Euclidean Dist

- Distance based on Angle

i.e $d = 1 - \text{cosine}(\theta)$
$= 1 - \cos(45)$
$= 0.29$

$[0.29 < 1]$

But,
$d = 1 - \cos(0)$
$= 0$ 1

So, the vector with angle 45 is closer to another vector.

① The goal is to first find the total feature representation i.e Dimension

② Then create vectors for each word in that Dimensions to understand the where each value is the relatio in the vectors shows it's relationship with n^{th} Dimension as Feature.

For e.g if any word has

[0.25, -1, 0, 1, -0.25] => cat

↳ Dog ↳ Plant ↳ Animal

(*) Ways to Train Word2Vec

- Train from scratch
- Train Pretrained model

(i) (BoW [Continuous Bag of Words]

- Dataset [Large] [Input, output] [supervised]

We decide Window_size = k [Feature Representation (Dimension)

We take k word token, and find the middle token.

Middle token becomes output (target)
other token becomes input.

For K=5

e.g Dataset

ineuron company is related to AI

Input	output
[ineuron, company, related to]	Is
company is related to AI	related.

Then, we [oneHot] Encode the inputs and outputs.

Data 1 = [[1,0,0,0,0,0], [0,1,0,0,0,0], [0,0,0,1,0,0], [0,0,0,0,1,0,]]

output = [0,0,1,0,0,0]

Then we fed into the ANN

So, Now sending 1st word of 1st sentence is

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

• we've fixed input i.e - window - 1

So, CBOW looks like below.

[words]

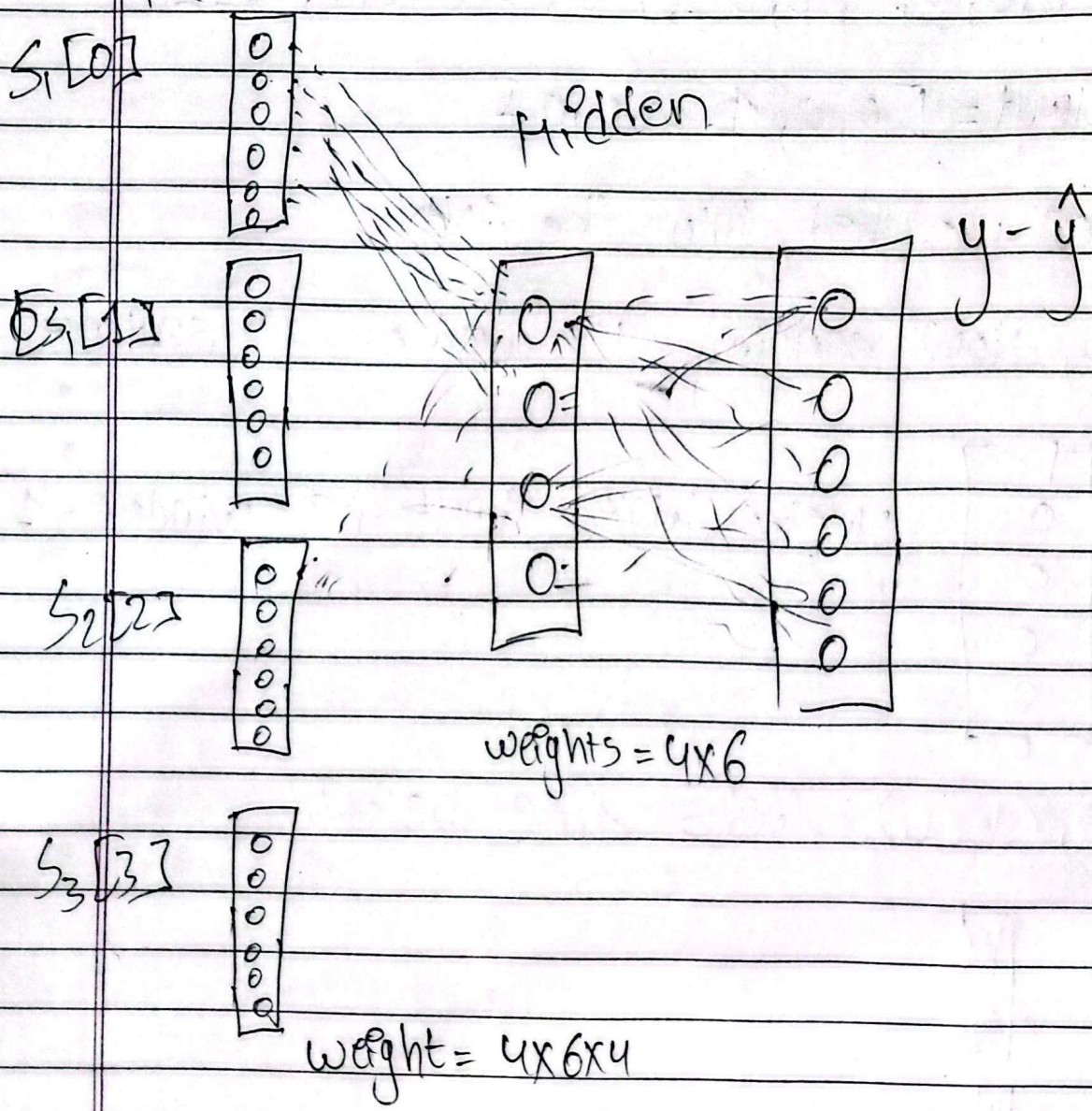
For, W-1 input we need len(voc) for each word.

So, If W = 5 and len(voc) = 6 then,

input for CBow model would require

4x6 neurons ⇒ 24

i.e Input



- Hidden would be of size [W-1]
- output would be of len(voc)

- Then, y-ŷ, Loss, Backprop.
- Update weight and Bias

Then our model is ready.

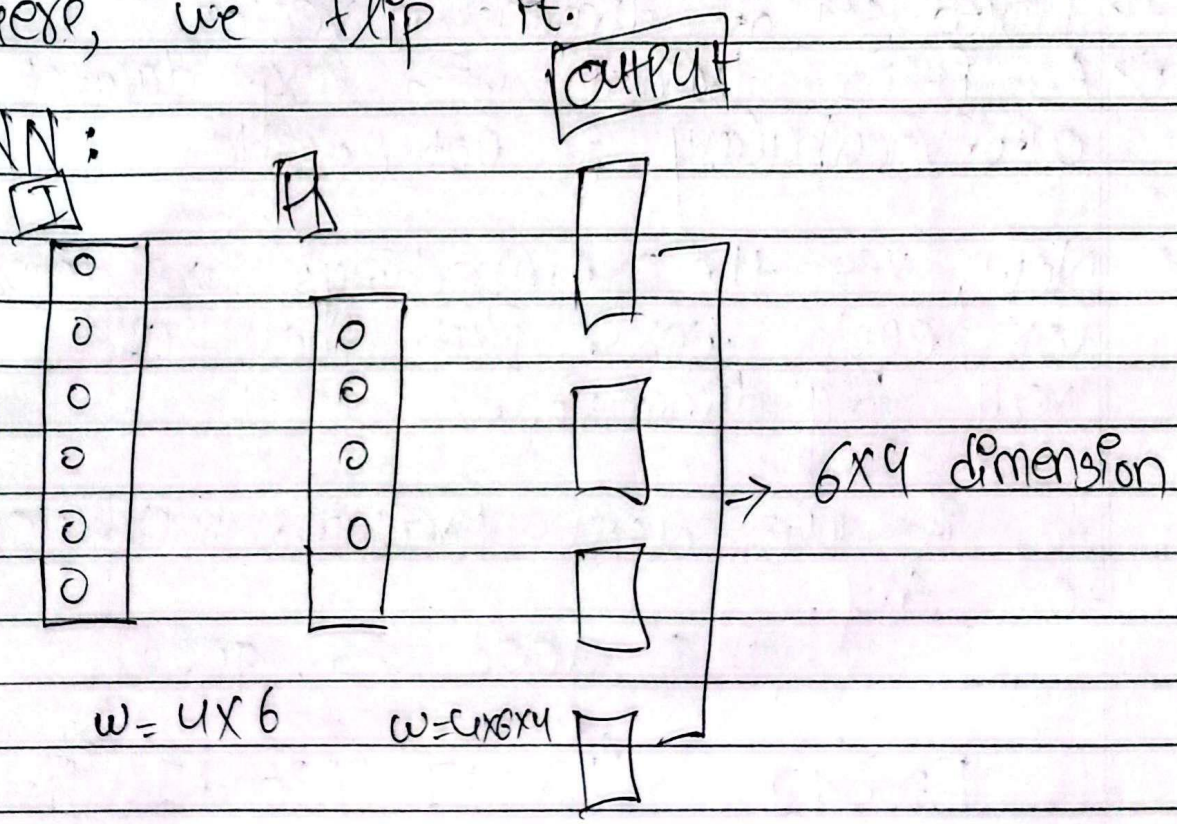
(*) Note:
→ Feature Representation = window-size
→ More window, better the model.

(ii) Skip Gram

~~Everything~~ · while data prep in CBow we take middle as output and rest as input.

Here, we flip it.

NN:



- Output => Softmax] => Activation
- Hidden => ReLU

- Training is same.

(*) When to use CBow and SkipGram?

- small dataset → CBow
- Large Dataset → SkipGram

Googles word2vec
→ Trained with 3B dataset
→ Feature Repr of 300 dim.

(*) Advantage of word2vec
→ Not sparse matrix → Dense Matrix
→ Semantic is captured
→ Inference is easier [Cosine]
→ Fixed set of dimension vector.
→ As it is trained with large dataset outofvocabulary is not possible.

Now, yes the embeddings are generated for each word. But they are very Big dimension.

So, if we use Pretrained word2vec for say, "The food is good"

[300d] [300d] [300d] [300d]

To represent a sentence with 4 words we need 4x300 values i.e 1200 inputs required for our own training.

We need to represent a complete sentence or document in less dimension.
i.e say 300 only.

one way is to take average of all the vector row wise and take average.

So, 300 dim will reps the sentence

This is known as Avg Word2Vec.

We can build Neural Network with input of only [300] neurons.

④ Pretrained word2vec

gensim.models import Word2Vec

.most_similar("word") => Gives the top similar.

once we've all the embeddings, we choose the machine learning or deep learning model to train the data.

Based on classification or Regression choose model.