

Natural Language Processing

Category	
Files	
Created	@May 14, 2023 9:34 PM
Reminder	
Status	Open
URL	
Updated	@May 15, 2023 1:13 PM

▼ S1 Prep

- ▼ Write the Multinomial Naïve Bayes algorithm for text classification.

The Multinomial Naive Bayes algorithm is a probabilistic classifier that is commonly used for text classification tasks, such as sentiment analysis, spam detection, and topic modeling. Here is the algorithm for Multinomial Naive Bayes for text classification:

1. Preprocessing: a. Tokenization: split the text into individual words (tokens) b. Remove stop words: remove common words that do not provide much information such as "the", "a", "an", "and", etc. c. Stemming: reduce the words to their root form (e.g. "running" to "run")
2. Training: a. Calculate the prior probabilities for each class (category) of documents. This is done by counting the number of documents in each class and dividing by the total number of documents. b. For each word in the vocabulary, calculate its conditional probability given each class. This is done by counting the number of occurrences of the word in documents belonging to each class and dividing by the total number of words in documents belonging to that class.
3. Classification: a. Given a new document, tokenize it, remove stop words, and stem the words. b. For each class, calculate the product of the conditional probabilities of each word in the document given that class, and multiply it by the prior probability of that class. c. The class with the highest probability is assigned to the new document.
4. Evaluation: a. Calculate the accuracy, precision, recall, and F1 score of the classifier using a set of test documents.

Note: The Multinomial Naive Bayes algorithm assumes that the words in the document are independent of each other given the class. This is a strong assumption and may not always hold true in practice. Nevertheless, the algorithm is simple, efficient, and often performs well in text classification tasks.

Multinomial Naive Bayes algorithm is a probabilistic approach to text classification. It is based on Bayes' theorem, which states that the probability of a hypothesis (in this case, a class) given the observed evidence (in this case, a document) is proportional to the probability of the evidence given the hypothesis multiplied by the prior probability of the hypothesis. In the context of text classification, we can use this theorem to calculate the probability of a document belonging to a particular class given its words.

The Multinomial Naive Bayes algorithm assumes that the probability of each word in a document is independent of the other words, given the class label. This assumption is called the "naive" assumption and is not always true in practice, but it simplifies the computation and often works well in practice.

The algorithm works by first building a model from a set of labeled training documents. It then uses this model to predict the class of new, unlabeled documents. The model consists of a set of probabilities that describe how likely each word in the vocabulary is to appear in a document of each class.

To build the model, we first calculate the prior probabilities of each class, which are simply the proportion of training documents that belong to each class. We then count the number of occurrences of each word in each class, and use these counts to estimate the conditional probabilities of each word given each class. We can use Laplace smoothing to avoid zero probabilities, which can cause problems with the multiplication of probabilities.

To predict the class of a new document, we simply calculate the probability of the document given each class, using the conditional probabilities from the model. We then multiply these probabilities by the prior probabilities of each class, and normalize them to get the posterior probabilities. The class with the highest posterior probability is the predicted class for the document.

The Multinomial Naive Bayes algorithm has been shown to work well in practice for text classification tasks, particularly for tasks with limited training data and a large number of features (i.e., a large vocabulary). However, it is not always the best choice for every task, and there are other algorithms that may be more appropriate in certain situations.

▼ Training And Prediction Algorithm

Here's the Multinomial Naive Bayes algorithm for text classification:

Training Algorithm:

1. Count the number of occurrences of each feature (word) in each class (label) of the training data.
2. Calculate the prior probability of each class by dividing the number of documents in each class by the total number of documents.
3. Calculate the likelihood probability of each feature given each class by dividing the count of the feature in the class by the total count of all features in the class.
4. Calculate the conditional probability of each feature given each class by dividing the likelihood probability of the feature given the class by the prior probability of the class.
5. Store the conditional probabilities of each feature given each class.

Prediction Algorithm:

1. For a given document, tokenize it into features (words) and count the occurrences of each feature.
2. For each class, calculate the log-likelihood of the document by summing the logarithms of the conditional probabilities of each feature given the class multiplied by the count of the feature in the document.
3. Add the logarithm of the prior probability of the class to the log-likelihood calculated in step 2.
4. Choose the class with the highest log-likelihood as the predicted class for the document.
5. Return the predicted class.

Note that in step 2 of the prediction algorithm, we use the logarithm of the probabilities to avoid underflow issues that can occur when dealing with very small probabilities.

▼ Apply the Max-Match word segmentation algorithm on the following:

"wecanonlyseeashortdistanceahead"

Assume you have the English all vocabulary set.

The Max-Match algorithm works by starting from the beginning of the text and attempting to match the longest possible word from a given dictionary. Once a word is found, it is added to the output, and the algorithm moves to the next unmatched portion of the text.

Here's the step-by-step process of applying the Max-Match algorithm to the given text:

Initialize an empty list to store the segmented words.

Define a dictionary of possible words, such as ["we", "can", "only", "see", "a", "short", "distance", "ahead"].

Start from the beginning of the text and define a maximum word length, such as 6 (since the longest word in the dictionary is "distance", which has 8 characters).

Search for the longest possible word from the dictionary that matches the text from the current position up to the maximum word length. In this case, the first possible word is "we".

Add the matched word to the output list and move the current position to the end of the matched word.

Repeat steps 4-5 until the entire text has been segmented.

The final segmented output is: ["we", "can", "only", "see", "a", "short", "distance", "ahead"].

Therefore, the Max-Match algorithm segments the given text into 8 words: "we", "can", "only", "see", "a", "short", "distance", and "ahead".

In general, the Max-Match algorithm is suitable for languages with a limited number of possible word combinations, where the dictionary can be reasonably comprehensive. For languages with a larger vocabulary or complex morphologies, more sophisticated algorithms may be needed for accurate word segmentation.

- ▼ Consider $\text{Count}(w)$, $\text{Count}(v,w)$ be unigram and bigram counts taken from a training corpus, where w is the single word, v,w is a bigram. Let N be the total number of words in the Corpus. What are the maximum likelihood estimates for unigram and bigram model?

The maximum likelihood estimates (MLE) for unigram and bigram models can be calculated as follows:

1. Unigram Model:

The probability of a word w in the unigram model is given by:

$$P(w) = \text{Count}(w) / N$$

where $\text{Count}(w)$ is the number of occurrences of the word w in the corpus, and N is the total number of words in the corpus.

1. Bigram Model:

The probability of a bigram v,w in the bigram model is given by:

$$P(w | v) = \text{Count}(v,w) / \text{Count}(v)$$

where $\text{Count}(v,w)$ is the number of occurrences of the bigram v,w in the corpus, and $\text{Count}(v)$ is the number of times the word v appears as the first word in any bigram in the corpus.

It's worth noting that the denominator $\text{Count}(v)$ is equivalent to the number of occurrences of the word v in the corpus minus one, since we can't count the last word in each document as the first word of a bigram.

If we want to smooth the probabilities to avoid zero probabilities for unseen bigrams, we can use techniques such as Laplace smoothing or add-k smoothing, where we add a constant k to the numerator and k times the vocabulary size to the denominator.

For example, Laplace smoothing for the bigram model would give:

$$P(w | v) = (\text{Count}(v,w) + 1) / (\text{Count}(v) + V)$$

where V is the size of the vocabulary (i.e., the number of unique words) in the corpus.

Note that MLE estimates may not always be optimal, especially for rare events, and more sophisticated smoothing techniques and language models may be needed in practice.

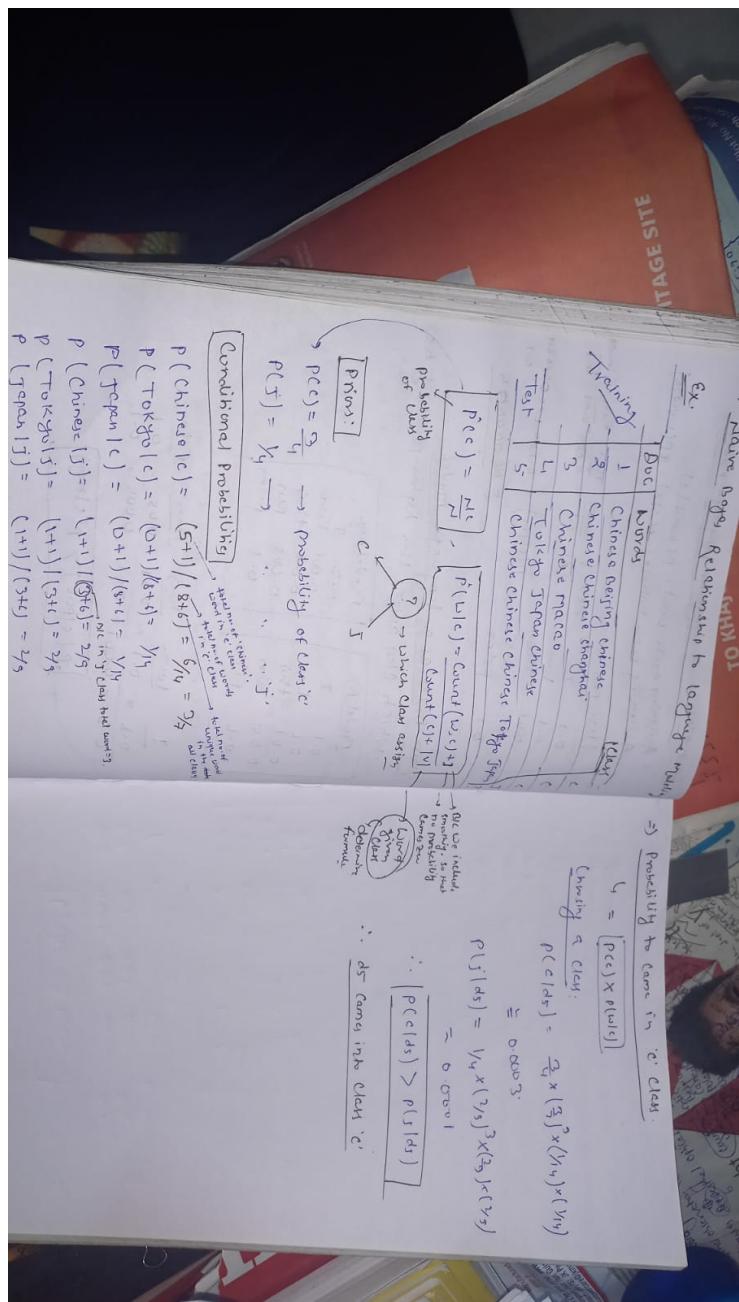
Q)

		Doc	Words	Class
2(a)	Training	1	Chinese, Beijing, Chinese	C
		2	Chinese, Chinese, Shanghai	C
		3	Chinese, Macao	C
		4	Tokyo, Japan, Chinese	J
	Test	5	Chinese, Chinese, Tokyo, Japan	?

Compute the most likely class for Doc 5. Assume a multinomial naive Bayes classifier and use add- α La Place smoothing for the likelihoods. Suitable value of α may be chosen.

▼ Ans

<https://youtu.be/mqYa0LaA9WI>



Q)

Write the training and prediction algorithms for Bernoulli Naïve Bayes Text Classification.
Consider the following dataset:

1(c)

Feature	1	1	0	1	1	0	0	0	1	0	1	1	1	0	1	1	1	0	0	1
Label	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	a	b	

Predict the label values a and b using Bernoulli Naïve Bayes Text Classification

▼ Ans

Write the training and prediction algorithm for Bernoulli Naive Bayes Test																					
Classification. Consider the following dataset:																					
Feature	1	1	0	1	1	1	0	0	0	1	0	1	1	0	1	1	1	1	0	0	1
Label/Class	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	a	b	
Class 0							Class 1							test							
Class	0																				
Feature		0		1																	
0	4		3																		
1	6		7																		
Total feature per class	10		10																		

~~P($a=0 | \text{feature}=0$) = $\frac{4}{10} \times \frac{1}{2} = \frac{2}{5}$~~

$$\begin{aligned} P(a=0 | \text{feature}=0) &= P(\text{feature}=0 | \text{class}=0) \times \frac{P(\text{class}=0)}{P(\text{feature}=0)} \\ &= \frac{4}{10} \times \frac{1}{2} \\ &= \frac{2}{5} \end{aligned}$$

$$P(a=1 | \text{feature}=0) = P(\text{feature}=0 | \text{class}=1) \times \frac{P(\text{class}=1)}{P(\text{feature}=0)}$$

$$\begin{aligned} &= \frac{6}{10} \times \frac{1}{2} \\ &= \frac{3}{10} \end{aligned}$$

$$\therefore P(a=0 | \text{feature}=0) > P(a=1 | \text{feature}=0)$$

$$\Rightarrow \boxed{a=0}$$

Similarly $P(b=0 | \text{feature}=1) = P(\text{feature}=1 | \text{class}=0) \times \frac{P(\text{class}=0)}{P(\text{feature}=1)}$

$$\begin{aligned} &= \frac{7}{10} \times \frac{1}{2} \\ &= \frac{7}{20} \end{aligned}$$

$$P(b=1 | \text{feature}=1) = P(\text{feature}=1 | \text{class}=1) \times \frac{P(\text{class}=1)}{P(\text{feature}=1)}$$

$$\begin{aligned} &= \frac{13}{20} \times \frac{1}{2} \\ &= \frac{13}{40} \end{aligned}$$

$$\Rightarrow \boxed{b=1}$$

Q)

2. Consider the following training data:

(10 marks)

S.No.	Document	Class
1.	Natural Language Processing	A
2.	Language Model Learning	A
3.	Ngram Langauge Model	A
4.	Text Classification Model	A
5.	Text Processing Model	A
6.	Computer Vision	B
7.	Image Classification Model	B
8.	Image Segmentation	B
9.	Image Processing	B
10.	Object Recognition	B

And Test Data:

1.	Image Learning Model	?
2.	Text Learning Model	?

Predict the Class for test samples using Bag of Words Language and Multinomial Model Naïve Bayes Text Classification. Apply add-2 La Place smoothing and show the calculation of posterior, class conditional and prior probabilities.

▼ Ans

S.No	Document			class
1	Natural	language	Processing	A
2	Language	Model	Learning	A
3	Ngiam	Language	Model	A
4	Text	Classification	Model	A
5	Text	Processing	Model	A
6	Computer	Vision		B
7	Image	classification	Model	B
8	Object	Segmentation		B
9	Image	Processing		B
10	object	Recognition		B

Sessinal ->
2022

Date

Training

$P(A) = \frac{5}{10}$, $P(B) = \frac{5}{10}$

$= \frac{1}{2}$ $= \frac{1}{2}$

$|W| = \text{unique words}$
= 14

Now

Test data :-

1. object Recognition Model
2. Text Recognition Model.

For

First Test data —

object Recognition Model

$$P(\text{object} | A) = \frac{0+1}{15+14} = \frac{1}{29}$$

$$P(\text{Recognition} | A) = \frac{0+1}{15+14} = \frac{1}{29}$$

$$P(\text{Model} | A) = \frac{4+1}{15+14} = \frac{5}{29}$$

$$P(\text{object} | B) = \frac{2+1}{11+14} = \frac{3}{25}$$

$$P(\text{Recognition} | B) = \frac{1+1}{11+14} = \frac{2}{25}$$

$$1.02 \times 10^{-4}$$

$$5.27 \times 10^{-4}$$

Now,

choosing a class : — $P(c) + P(w|c)$

$$P(A|d_1) = \frac{1}{2} \times \frac{1}{29} \times \frac{1}{29} \times \frac{5}{29}$$

$$= 0.0001$$

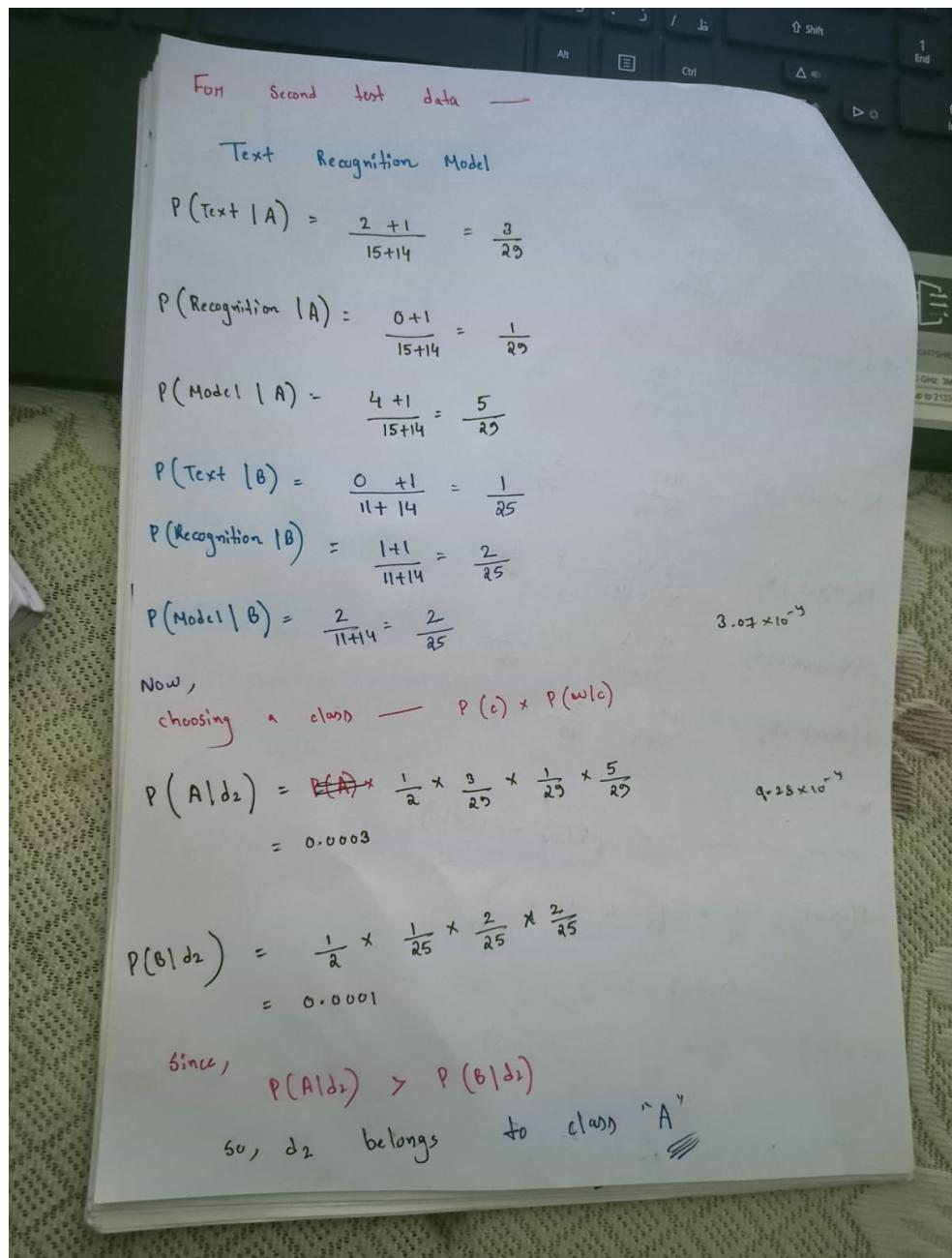
$$P(B|d_1) = \frac{1}{2} \times \frac{3}{25} \times \frac{2}{25} \times \frac{2}{25}$$

so, d_1 belongs to class

$$= 0.0003$$

since, $P(A|d_1) \leq P(B|d_1)$

"B"



Notes -

- <https://gist.github.com/ttezel/4138642>

Videos -

Explanations -

▼ Bernoulli Naive Bayes -

Bernoulli Naive Bayes is a probabilistic algorithm used for classification tasks. It's often used in natural language processing applications to classify text documents into different categories. The algorithm is based on Bayes' theorem, which states that the probability of a hypothesis (in this case, a document belonging to a particular category) is proportional to the product of the prior probability and the likelihood.

In the case of Bernoulli Naive Bayes, the likelihood is modeled using a binary feature vector. Each feature represents a word or a group of words that occur in the text, and the value of the feature is 1 if the word is present in the document and 0 otherwise. The prior probability represents the probability of a document belonging to a particular category before we observe any of its features.

To compute the probability of a document belonging to a particular category, we first compute the prior probability of the category, which is simply the proportion of training documents that belong to that category. We then compute the likelihood of the document given the category, which is the product of the probabilities of each feature given the category.

For each feature, we estimate its probability using the training data. If a feature occurs in a document of the category, we add 1 to the count of the feature in that category. We also count the total number of documents in the category. We then use these counts to estimate the probability of the feature given the category using Laplace smoothing. The Laplace smoothing parameter is a hyperparameter that determines the strength of the smoothing. A larger value of the parameter leads to stronger smoothing and a more conservative estimate of the probabilities.

Once we have computed the prior probability and the likelihood for each category, we can use Bayes' theorem to compute the posterior probability of the document belonging to each category. The category with the highest posterior probability is then chosen as the predicted category for the document.

Notes -

- <https://gist.github.com/ttezel/4138642>
- <https://arxiv.org/pdf/1410.5329.pdf>

Naive Bayes -

- <https://www.upgrad.com/blog/multinomial-naive-bayes-explained/>
- <https://www.javatpoint.com/machine-learning-naive-bayes-classifier>
- <https://jakevdp.github.io/PythonDataScienceHandbook/05.05-naive-bayes.html>
- <https://www.geeksforgeeks.org/applying-multinomial-naive-bayes-to-nlp-problems/>
- <https://www.analyticsvidhya.com/blog/2021/04/improve-naive-bayes-text-classifier-using-laplace-smoothing/>
- <https://towardsdatascience.com/laplace-smoothing-in-naïve-bayes-algorithm-9c237a8bdece>

Videos -

- <https://www.youtube.com/playlist?list=PLfng5rv4gTmqfLtZcK85wDXDTQgZn0132>
- <https://www.youtube.com/playlist?list=PLkDeS9l-orgEOetgtOca43qQ8yxKauRng>
- https://www.youtube.com/watch?v=808M7q8QXOE&list=PLaZQkZp6WhWy4_bClrW9EGQKnUUD9yp8V&ab_channel=FromLanguagesToInformation
- https://www.youtube.com/watch?v=Y1j_J53k7fo&list=PLaZQkZp6WhWxU3kA6wV0nb5dY1SXDEKWH&ab_channel=FromLanguagesToInformation

▼ S2 Prep

Imp topics for theory -

- Wordvector
- Forward algorithm
- Two Major Topics this sessional -
 - Hidden Markov Model
 - Vector Semantics

▼ Prev year answers -

Q1)

Questions	
o.	1. Given the following Corpus <i>The eyes are not here</i> <i>There are no eyes here</i> <i>In this valley of dying stars</i> <i>In this hollow valley</i> <i>This broken jaw of our lost kingdoms</i> Construct word-context matrix for <i>eyes</i> , <i>here</i> and <i>valley</i> using a three 'word window'. Calculate the similarity between <i>eyes</i> and <i>valley</i> . 7 5

Ans -

Pending

Q2) Explain word2vec algorithm using skip-gram with negative sampling.

Ans)

Word2Vec is a neural network-based algorithm for learning word embeddings, which are vector representations of words in a high-dimensional space. The Skip-gram model is a popular variant of Word2Vec that predicts the context words given a target word. Negative sampling is a technique used to simplify the learning task and speed up training.

Here's how the Skip-gram model with negative sampling works:

1. Given a corpus of text, create a vocabulary of unique words, and assign each word a unique ID.
2. For each word in the corpus, generate training examples of the form (target word, context word), where the target word is the center word and the context word is one of the words that appear within a fixed-size window around the target word.
3. For each training example (target word, context word), use the Skip-gram model to predict the probability of the context word given the target word. Specifically, the model uses a softmax function to compute the conditional probability of each context word in the vocabulary, given the target word:

$$P(\text{context word} \mid \text{target word}) = \exp(v_c \cdot v_t) / \sum(\exp(v_c \cdot v_i))$$

where v_c and v_t are the vector representations of the context word and target word, respectively, and v_i are the vector representations of all other words in the vocabulary.

1. The model is trained by maximizing the log-likelihood of the observed context words given the target words. This is equivalent to minimizing the negative log-likelihood, which is known as the cross-entropy loss. However, computing the softmax function over the entire vocabulary can be computationally expensive, especially for large vocabularies.

2. To speed up training, the Skip-gram model uses negative sampling, which involves randomly selecting a small number (typically between 5 and 20) of "negative" context words for each target word. Negative context words are chosen randomly from the vocabulary, but with a probability that is proportional to their frequency in the corpus. The objective of the model is to maximize the probability of the observed context word and minimize the probability of the negative context words.
3. To do this, the model uses a modified version of the softmax function, called the negative sampling loss function, which is defined as follows:

$$J = -\log(\sigma(v_c \cdot v_t)) - \sum_{i=1}^k \log(\sigma(-v_i \cdot v_t))$$

where $\sigma(x) = 1 / (1 + \exp(-x))$, k is the number of negative context words, and v_i are the vector representations of the negative context words.

1. During training, the parameters of the model (i.e., the word vectors) are updated using stochastic gradient descent to minimize the negative sampling loss function.

In summary, the Skip-gram model with negative sampling is a variant of the Word2Vec algorithm that predicts the context words given a target word. It uses a modified softmax function to compute the conditional probability of the context words and a negative sampling loss function to speed up training by selecting a small number of negative context words. The model is trained using stochastic gradient descent to minimize the negative sampling loss function and learn high-quality word embeddings.

Q3) What are word vectors? Illustrate with the help of a suitable diagram. Define the cosine similarity between two word vectors.

Ans)

Word vectors, also known as word embeddings, are numerical representations of words in a high-dimensional space. They are created using machine learning algorithms that are trained on large collections of text to learn the relationships between words.

The basic idea behind word vectors is to represent each word as a vector of real numbers, where the values of the vector capture the meaning and context of the word in the text. Word vectors are typically high-dimensional, with hundreds or thousands of dimensions, and are learned in an unsupervised manner using techniques such as Word2Vec and GloVe.

Here is a simple diagram illustrating the concept of word vectors:

In this diagram, we see a 2D space where each word is represented as a vector. Each word is mapped to a point in this space, and similar words are placed close together. For example, "cat" and "dog" are similar words, so they are placed close together. "car" and "bus" are also similar words, but they are placed in a different direction.

The cosine similarity between two word vectors is a measure of their similarity or distance in the high-dimensional space. It is calculated by taking the cosine of the angle between the two vectors. The cosine similarity between two vectors ranges from -1 to 1, where -1 means the vectors are completely dissimilar, 0 means they are orthogonal or independent, and 1 means they are identical.

The cosine similarity between two word vectors x and y can be calculated as follows:

$$\text{cosine similarity}(x, y) = (x \cdot y) / (\|x\| * \|y\|)$$

where $x \cdot y$ is the dot product of x and y , and $\|x\|$ and $\|y\|$ are the magnitudes or lengths of the vectors x and y , respectively.

For example, consider the following word vectors in a 2D space:

$$\begin{aligned} \text{"cat"} &= (1, 2) \\ \text{"dog"} &= (2, 3) \\ \text{"car"} &= (3, 1) \\ \text{"bus"} &= (1, 3) \end{aligned}$$

The cosine similarity between "cat" and "dog" can be calculated as follows:

$$\text{cosine similarity}(\text{"cat"}, \text{"dog"}) = ((1 * 2) + (2 * 3)) / (\sqrt(1^2 + 2^2) * \sqrt(2^2 + 3^2)) = 0.98$$

The high cosine similarity value indicates that "cat" and "dog" are similar words in the 2D space. Similarly, the cosine similarity between "cat" and "car" is -0.56, indicating that they are dissimilar words.

Word vectors have become an important tool in natural language processing (NLP) because they enable algorithms to capture the meaning and context of words in a more efficient and accurate way than traditional methods. Some benefits of using word vectors include:

1. Improved accuracy in NLP tasks: Word vectors can capture the semantic and syntactic relationships between words, which makes them useful in a wide range of NLP tasks such as machine translation, sentiment analysis, and named entity recognition.
2. Reduced dimensionality: Word vectors can represent words in a high-dimensional space, but they can also be used to reduce the dimensionality of text data. This is particularly useful when working with large datasets, where traditional methods can become computationally expensive.
3. Transfer learning: Word vectors can be trained on large datasets and then used as a starting point for other NLP tasks. This approach, known as transfer learning, can save time and computational resources when working with smaller datasets.

There are many techniques for generating word vectors, including Word2Vec, GloVe, and FastText. These techniques are based on neural networks and use different methods for learning word representations.

In practice, word vectors are often used as inputs to other machine learning algorithms, such as neural networks or support vector machines, in order to perform various NLP tasks. By using word vectors as inputs, these algorithms can capture the meaning and context of words in a more efficient and accurate way, which can lead to improved performance on NLP tasks.

Q4) A char-context matrix (also called character co-occurrence matrix) is a char x char matrix which captures the number of times a character appears in the context of another character. Choose your full name as the corpus and consider the context of window +/-4, i.e. 4 characters left and 4 characters right of any character.

Generate the simple char-context matrix for the corpus.

Ans)

Sure, here's the character-context matrix for my full name "ChatGPT":

	C	h	a	t	G	P	T
C	0	0	0	1	0	1	0
h	0	0	1	1	0	0	0
a	0	1	0	1	0	0	0
t	1	1	1	0	0	0	1
G	0	0	0	0	0	1	1
P	1	0	0	0	1	0	1
T	0	0	0	1	1	1	0

In this matrix, the rows and columns correspond to the characters in my name, and each element (i, j) represents the number of times the character j appears in the context of character i (within a window of +/-4 characters).

For example, the value in the (t, a) cell is 1, indicating that the character "a" appears once within a window of +-4 characters around the character "t". Similarly, the value in the (P, C) cell is 1, indicating that the character "C" appears once within a window of +-4 characters around the character "P".

Note that the matrix is symmetric along the diagonal, since the context of character i is the same as the context of character j.

Q5)

For the following term document matrix:

	Document 1	Document 2	Document 3	Document 3
digital	1	0	7	13
computer	114	80	62	89
information	36	58	1	4
data	20	15	2	3

Calculate the similarity between good and fool using (i) cosine similarity (ii) PPMI. Use add 2 smoothing if necessary.

Ans) The term-document matrix is:

	Document 1	Document 2	Document 3	Document 4
digital	1	0	7	13
computer	114	80	62	89
information	36	58	1	4
data	20	15	2	3

(i) Cosine Similarity:

To calculate the cosine similarity between "good" and "fool", we first need to represent these words as vectors using the term-document matrix.

Since neither "good" nor "fool" appears in the corpus, their vectors will be zero vectors. Therefore, the cosine similarity between "good" and "fool" is 0.

(ii) PPMI (Positive Pointwise Mutual Information):

PPMI measures the association between two words based on their co-occurrence in a corpus. It is calculated as:

$$\text{PPMI}(w_1, w_2) = \max(\log(P(w_1, w_2) / (P(w_1) * P(w_2))), 0)$$

where $P(w_1, w_2)$ is the probability of co-occurrence of w_1 and w_2 , and $P(w_1)$ and $P(w_2)$ are the probabilities of occurrence of w_1 and w_2 , respectively.

To calculate the PPMI between "good" and "fool", we first need to calculate the probabilities of their occurrence and co-occurrence in the corpus. Since neither "good" nor "fool" appears in the corpus, their probabilities of occurrence are zero. Therefore, the PPMI between "good" and "fool" is also zero.

Note: If we use add 2 smoothing to the term-document matrix to avoid zero probabilities, we would get non-zero values for the probabilities of occurrence and co-occurrence of "good" and "fool". However, since neither "good"

nor "fool" co-occur with any of the words in the corpus, their PPMI would still be zero.

Q6) Describe the Logistic Regression based skip gram model with the help of suitable diagram.

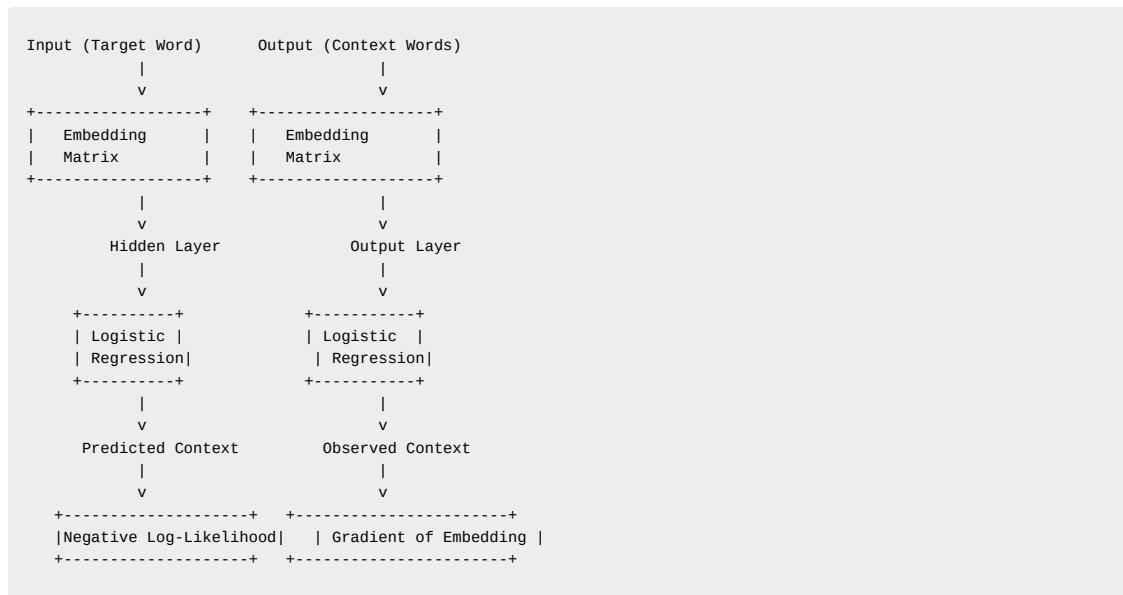
The Logistic Regression based skip-gram model is a neural network language model that uses logistic regression to predict the context words given a target word. It is a modification of the original skip-gram model, which uses softmax regression for the same purpose.

The basic architecture of the Logistic Regression based skip-gram model consists of two layers: the input layer and the output layer. The input layer represents the target word, which is transformed into a vector representation using an embedding matrix. The output layer represents the context words, which are also transformed into vector representations using the same embedding matrix.

The goal of the model is to learn the probability distribution of the context words given the target word. To achieve this, the model uses logistic regression to estimate the conditional probability of each context word given the target word.

The training process involves minimizing the negative log-likelihood of the observed context words given the target word. This is done by adjusting the weights of the embedding matrix using backpropagation.

The following diagram illustrates the architecture of the Logistic Regression based skip-gram model:



In summary, the Logistic Regression based skip-gram model uses logistic regression to predict the context words given a target word. It learns the conditional probability distribution of the context words using an embedding matrix and minimizes the negative log-likelihood of the observed context words during training.

Q7) Construct the word co-occurrence matrix, window size+1, for the following text: Document: "Roses are red. Sky is blue."

Ans) Here is the word co-occurrence matrix for the given text with a window size of +/- 1:

	Roses	are	red	Sky	is	blue
Roses	0	1	0	0	0	0
are	1	0	1	0	1	0
red	0	1	0	0	0	0
Sky	0	0	0	0	1	1
is	0	1	0	1	0	1
blue	0	0	0	1	1	0

Each row and column represents a unique word in the corpus, and the cells contain the number of times the corresponding words co-occur within a window size of +/- 1. For example, the cell at row "are" and column "is" has a value of 1, indicating that the words "are" and "is" co-occur once within a window size of +/- 1. Similarly, the cell at row "Sky" and column "blue" has a value of 1, indicating that the words "Sky" and "blue" co-occur once within a window size of +/- 1.

▼ Sentiment Analysis

A worked sentiment example with add-1 smoothing

Cat	Documents
Training	<ul style="list-style-type: none"> - just plain boring - entirely predictable and lacks energy - no surprises and very few laughs + very powerful + the most fun film of the summer
Test	<p>? predictable with no fun</p>

1. Prior from training:

$$\hat{P}(c_j) = \frac{N_{c_j}}{N_{total}}$$

$$P(-) = 3/5$$

$$P(+) = 2/5$$

2. Drop "with"

3. Likelihoods from training:

$$p(w_i|c) = \frac{\text{count}(w_i, c) + 1}{(\sum_{w \in V} \text{count}(w, c)) + |V|}$$

$$P(\text{"predictable"}|-) = \frac{1+1}{14+20}$$

$$P(\text{"predictable"}|+) = \frac{0+1}{9+20}$$

$$P(\text{"no"}|-) = \frac{1+1}{14+20}$$

$$P(\text{"no"}|+) = \frac{0+1}{9+20}$$

$$P(\text{"fun"}|-) = \frac{0+1}{14+20}$$

$$P(\text{"fun"}|+) = \frac{1+1}{9+20}$$

4. Scoring the test set:

$$P(-)P(S|-) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5}$$

$$P(+)P(S|+) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5}$$

Optimizing for sentiment analysis

For tasks like sentiment, word **occurrence** seems to be more important than word **frequency**.

- The occurrence of the word fantastic tells us a lot
- The fact that it occurs 5 times may not tell us much more.

Binary multinomial naive bayes, or binary NB

- Clip our word counts at 1
- Note: this is different than Bernoulli naive bayes; see the textbook at the end of the chapter.

Stanford

Binary Multinomial Naïve Bayes: Learning

- From training corpus, extract *Vocabulary*

Calculate $P(c_j)$ terms

- For each c_j in C do
 $docs_j \leftarrow$ all docs with class = c_j

$$P(c_j) \leftarrow \frac{|docs_j|}{\text{total # documents}}$$

Calculate $P(w_k | c_j)$ terms

- $Text_j \leftarrow$ single doc containing all $docs_j$
 - For each word w_k in *Vocabulary*
 $n_k \leftarrow$ # of occurrences of w_k in $Text_j$
- $$P(w_k | c_j) \leftarrow \frac{n_k + \alpha}{n + \alpha |Vocabulary|}$$

Stanford

Binary Multinomial Naive Bayes on a test document d

First remove all duplicate words from d

Then compute NB using the same equation:

$$c_{NB} = \operatorname{argmax}_{c_j \in C} P(c_j) \prod_{i \in positions} P(w_i | c_j)$$

Binary multinomial naive Bayes

Four original documents:

- it was pathetic the worst part was the boxing scenes
- no plot twists or great scenes
- + and satire and great plot twists
- + great scenes great film

After per-document binarization:

- it was pathetic the worst part boxing scenes
- no plot twists or great scenes
- + and satire great plot twists
- + great scenes film

Counts can still be 2! Binarization is within-doc!

	NB Counts		Binary Counts	
	+	-	+	-
and	2	0	1	0
boxing	0	1	0	1
film	1	0	1	0
great	3	1	2	1
it	0	1	0	1
no	0	1	0	1
or	0	1	0	1
part	0	1	0	1
pathetic	0	1	0	1
plot	1	1	1	1
satire	1	0	1	0
scenes	1	2	1	2
the	0	2	0	1
twists	1	1	1	1
was	0	2	0	1
worst	0	1	0	1

Sentiment Classification: Dealing with Negation

Das, Sanjiv and Mike Chen. 2001. Yahoo! for Amazon: Extracting market sentiment from stock message boards. In Proceedings of the Asia Pacific Finance Association Annual Conference (APFA).
Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? Sentiment Classification using Machine Learning Techniques. EMNLP-2002, 79–86.

Simple baseline method:

Add NOT_ to every word between negation and following punctuation:

didn't like this movie , but I



didn't NOT_like NOT_this NOT_movie but I

Sentiment Classification: Lexicons

Sometimes we don't have enough labeled training data

In that case, we can make use of pre-built word lists

Called lexicons

There are various publically available lexicons

Using Lexicons in Sentiment Classification

Add a feature that gets a count whenever a word from the lexicon occurs

- E.g., a feature called "this word occurs in the positive lexicon" or "this word occurs in the negative lexicon"

Now all positive words (*good, great, beautiful, wonderful*) or negative words count for that feature.

Using 1-2 features isn't as good as using all the words.

- But when training data is sparse or not representative of the test set, dense lexicon features can help

Naive Bayes in Other tasks: Spam Filtering

SpamAssassin Features:

- Mentions millions of (dollar) ((dollar) NN,NNN,NNN.NN)
- From: starts with many numbers
- Subject is all capitals
- HTML has a low ratio of text to image area
- "One hundred percent guaranteed"
- Claims you can be removed from the list

Summary: Naive Bayes is Not So Naive

Very Fast, low storage requirements

Work well with very small amounts of training data

Robust to Irrelevant Features

Irrelevant Features cancel each other without affecting results

Very good in domains with many equally important features

Decision Trees suffer from *fragmentation* in such cases – especially if little data

Optimal if the independence assumptions hold: If assumed independence is correct, then it is the Bayes Optimal Classifier for problem

A good dependable baseline for text classification

- But we will see other classifiers that give better accuracy

Slide from Chris Manning

▼ PMI

PMI helps us to find related words. In other words, it explains how likely the co-occurrence of two words than we would expect by chance. For example the word "Data Science" has a specific meaning when these two words "Data" and "Science" go together. Otherwise meaning of these two words are independent. Similarly "Great Britain" is meaningful since we know the word "Great" can be used with several other words but not so relevant in meaning like "Great UK, Great London, Great Dubai etc."

When words 'w1' and 'w2' are independent, their joint probability is equal to the product of their individual probabilities. Imagine when the formula of PMI as shown below returns 0, it means the numerator and denominator is same and then taking log of 1 produces 0. In simple words it means the words together has NO specific meaning or relevance. Question arises what are we trying to achieve here. We are focusing on the words which have high joint probability with the other word but having not so high probability of occurrence if words are considered separately. It implies that this word pair has a specific meaning.

Pointwise mutual information (PMI) in NLP

This tutorial describes interpretation and calculation of Pointwise mutual information (PMI) in NLP. It also includes how to run PMI in Python and R.

L <https://www.listendata.com/2022/06/pointwise-mutual-information-pmi.html>

$$\text{PMI}(w_1, w_2) = \log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

$$P(w) = \frac{\text{Freq}(w)}{\text{totalWordCount}}$$

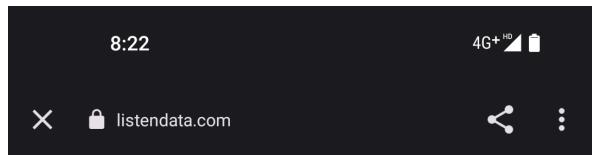
Smoothning

Introduction to Positive Point-wise mutual information (PPMI) -

Understand what is Positive Point-wise mutual information (PPMI) . Computing PPMI on a term-context matrix , Weighting PMI-K-Smoothing in PMI computation

L <https://towardsmachinelearning.org/positive-point-wise-mutual-information-ppmi/>





joint $p(w_1, w_2)$ and individual $p(w_1)$ and $p(w_2)$

are 0.001. PMI in that case would be -1.

Negative PMI means words are co-occurring

less than we expect by chance.

To handle negative PMI we replace negative

value with 0. In the formula below we are

capping negative values with 0.

$$PMI(w_1, w_2) = \max\left(\log_2 \frac{p(w_1, w_2)}{p(w_1)p(w_2)}, 0\right)$$

Remove Stopwords prior to PMI

In the above example we have not removed

stopwords so some of you might be

wondering if we need to remove stopwords

prior to PMI. It depends on the problem

statement but if your objective is to find the

related words, you should remove stopwords

prior to calculating PMI. In nutshell you

should also consider the following data



▼ HMM

HMM - https://www.youtube.com/watch?v=YErli89h0hs&list=PLR5USSocuZ5dA4-ivqzqNiDkICRMpudzS&index=8&ab_channel=SHUBHAMARORA

Forward Algorithm - https://www.youtube.com/watch?v=WDTc_SsZH70&list=PLR5USSocuZ5dA4-ivqzqNiDkICRMpudzS&index=9&ab_channel=SHUBHAMARORA

▼ Cosine similarity

Cosine for computing word similarity

$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

v_i is the count for word v in context i

w_i is the count for word w in context i .

$$\vec{a} \cdot \vec{b} = |\vec{a}| |\vec{b}| \cos \theta$$

$\rightarrow \rightarrow$

\rightarrow

\rightarrow

$$\frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|} = \cos \theta$$

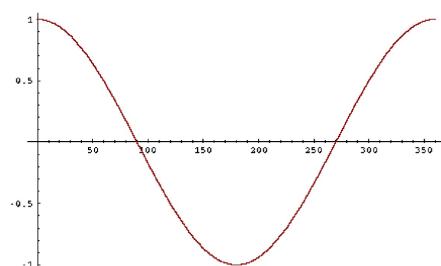
$\text{Cos}(v, w)$ is the cosine similarity of v and w

Cosine as a similarity metric

-1: vectors point in opposite directions

+1: vectors point in same directions

0: vectors are orthogonal



$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\| \|\vec{w}\|} = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\| \|\vec{w}\|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

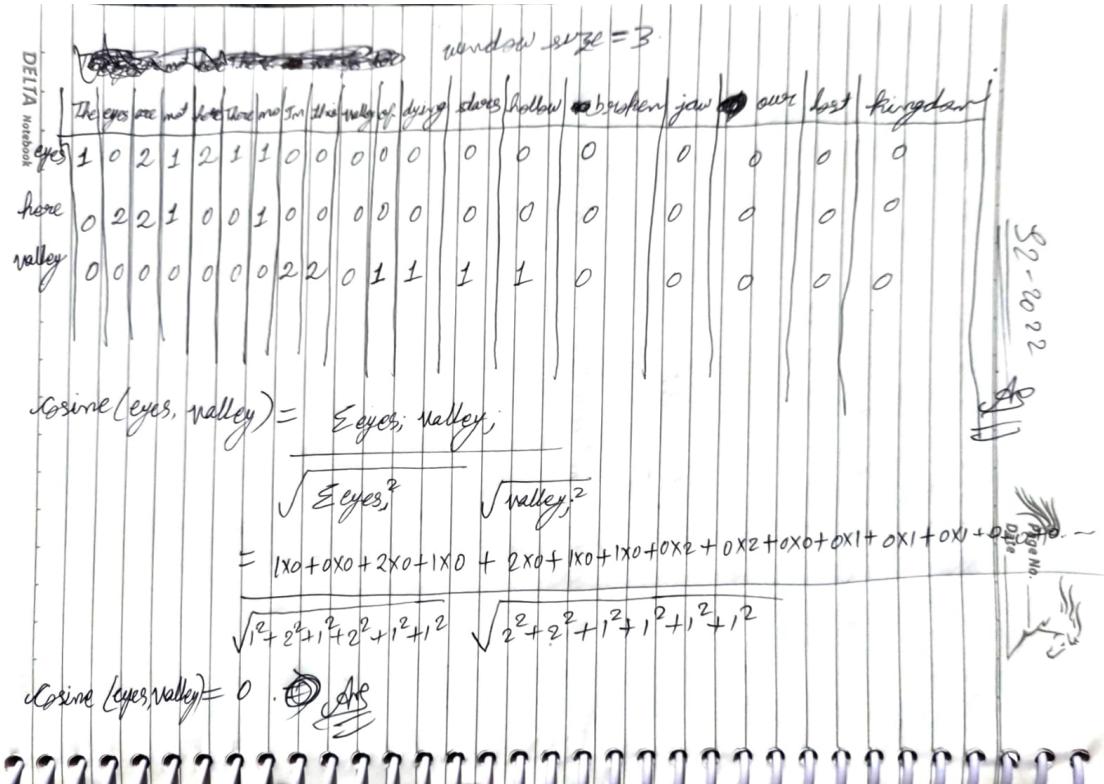
Which pair of words is more similar?

$$\text{cosine(apricot, information)} = \frac{1+0+0}{\sqrt{1+0+0} \sqrt{1+36+1}} = \frac{1}{\sqrt{38}} = .16$$

$$\text{cosine(digital, information)} = \frac{0+6+2}{\sqrt{0+1+4} \sqrt{1+36+1}} = \frac{8}{\sqrt{38} \sqrt{5}} = .58$$

$$\text{cosine(apricot, digital)} = \frac{0+0+0}{\sqrt{1+0+0} \sqrt{0+1+4}} = 0$$

	large	data	computer
apricot \leftarrow	1	0	0
digital	0	1	2
information \leftarrow	1	6	1



▼ Word2vec

Word2vec

- Instead of **counting** how often each word w occurs near "apricot"
- Train a classifier on a binary **prediction** task:
 - Is w likely to show up near "apricot"?
- We don't actually care about this task
 - But we'll take the learned classifier weights as the word embeddings

Word2Vec: Skip-Gram Task

Word2vec provides a variety of options.

Let's do

"skip-gram with negative sampling" (SGNS)

Approach: predict if candidate word c is a "neighbor"

- Treat the target word t and a true neighboring context word c as **positive examples**.
- Randomly sample other words in the lexicon to get negative examples
- Use logistic regression to train a classifier to distinguish those two cases
- Use the weights as the embeddings

Skip-Gram Training Data

Assume a +/- 2 word window, given training sentence:

- *I would like to go someplace nearby for lunch*
[target]

The training data:

- input/output pairs (centering on *go*)

Skip-Gram Training data

I would like to go someplace nearby for
lunch



Positive
{target, context}

- {go, like}
- {go, to}
- {go, someplace}
- {go, nearby}

Skip-Gram Training data

I would like to go someplace nearby for
lunch



Positive
{target, context}

- {go, like}
- {go, to}
- {go, someplace}
- {go, nearby}

That's fine for positive data. But for training a binary classifier we need negative examples.

Let's sample other words from the lexicon (that don't occur with the target word in this context).

Skip-Gram Training data

I would like to go someplace nearby for
lunch



Positive {+ target, context}	Negative {- target, context}
{go, like}	{go, aardvark}
{go, to}	{go, incubate}
{go, someplace}	{go, twelve}
{go, nearby}	{go, therefore}

Summary: How to learn word2vec (skip-gram) embeddings

- Start with V random 300-dimensional vectors as initial embeddings
- Use logistic regression:
 - Take a corpus and take pairs of words that co-occur as positive examples
 - Take pairs of words that don't co-occur as negative examples
 - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
 - Throw away the classifier code and keep the embeddings.

Or in other words

- Start with some initial embeddings (e.g., random)
- iteratively make the embeddings for a word
 - more like the embeddings of its neighbors
 - less like the embeddings of other words.

- Vectors based on term-frequency (counts of words in a corpus wrt to each document)
- Vectors based on Positive Pointwise Mutual Information (PPMI)

Characteristics:

- Sparse vectors (large no. of zero entries)
- Large size vectors (length of vector is equal to vocabulary size $|V|$)
- However, we can reduce the size using dimensionality reduction techniques such as PCA, SVD,etc

Word2Vec

- uses a technique called as skip gram
- negative sampling

Aim: Instead of counting or calculating some values the values, train a classifier.

Task: Prediction - Is a target word t is likely to show up near the context c .

However, we skip the prediction and the weights of the learned classifier will be used as embeddings.

For a classifier, we require a supervised or labelled training data.

The running text from the corpus can be considered as supervised training data.

Skip gram follows the steps as:

- 1) Treat the target word t and its neighbouring context c as positive sample.
- 2) Randomly use the other words in the vocab to obtain the negative samples.
- 3) Train the classifier such as logistic regression.
- 4) Use the regression weights as embeddings.

The Classifier:

Train classifier such that the tuple (t,c) , generates the probabilities as

$$P(+ | t,c) \\ P(- | t,c) = 1 - P(+ | t,c)$$

A target word is likely to appear near the context, if its embedding is similar to the context embedding.

$\text{Similarity}(t,c) == t \cdot c$ (dot product) // between - infinity and + infinity

We bound this similarity using the logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$



$$P(+|t,c) = \frac{1}{1+e^{-t \cdot c}}$$

$$t \cdot c = t_1 \cdot c_1 + t_2 \cdot c_2 + \dots + t_d \cdot c_d$$

t_1	t_2	\dots	t_d
-------	-------	---------	-------

$$P(-|t,c) = 1 - P(+|t,c) \\ = \frac{e^{-t \cdot c}}{1+e^{-t \cdot c}}$$

where t_1, t_2, \dots, t_d and c_1, c_2, \dots, c_d are randomly generated values.

positive examples +	
t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -			
t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

$$P(+|t,c_{1:k}) = \prod_{i=1}^k \frac{1}{1+e^{-t \cdot c_i}}$$

$$\log P(+|t,c_{1:k}) = \sum_{i=1}^k \log \frac{1}{1+e^{-t \cdot c_i}}$$

Since all the context words of target word are independent of each other we can multiply the probabilities.

$$L(\theta) = \sum_{(t,c) \in +} \log P(+|t, c) + \sum_{(t,c) \in -} \log P(-|t, c)$$

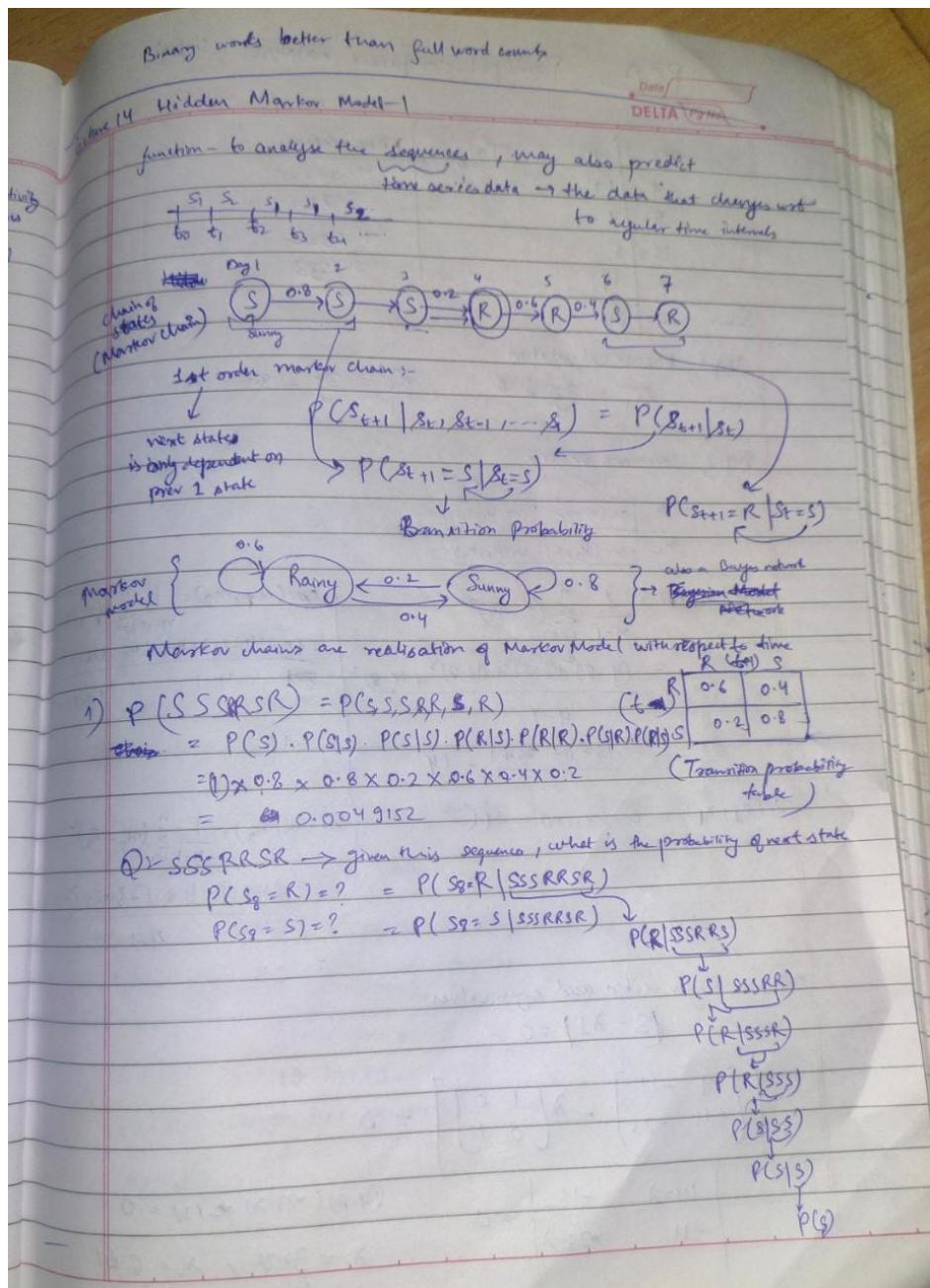
While training

Goal:

- 1) Maximize the loss of positive samples.
- 2) Minimize the loss of negative samples.

Optimizer like gradient descent may be used to optimize the weights or specifically, the randomly generated embeddings.

- <http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/>
- <https://medium.com/@zafaralibagh6/simple-tutorial-on-word-embedding-and-word2vec-43d477624b6d>
- <https://stackoverflow.com/questions/21979970/how-to-use-word2vec-to-calculate-the-similarity-distance-by-giving-2-words>
- <https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa>
- <https://www.kaggle.com/code/cdabakoglu/word-vectors-cosine-similarity>
- <https://jalammar.github.io/illustrated-word2vec/>
- <https://www.shanelynn.ie/get-busy-with-word-embeddings-introduction/>



2. Training: a. Calculate the prior probabilities for each class (category) of documents. This is done by counting the number of documents in each class and dividing by the total number of documents. b. For each word in the vocabulary, calculate its conditional probability given each class. This is done by counting the number of occurrences of the word in documents belonging to each class and dividing by the total number of words in documents belonging to that class.
3. Classification: a. Given a new document, tokenize it, remove stop words, and stem the words. b. For each class, calculate the product of the conditional probabilities of each word in the document given that class, and multiply it by the prior probability of that class. c. The class with the highest probability is assigned to the new document.
4. Evaluation: a. Calculate the accuracy, precision, recall, and F1 score of the classifier using a set of test documents.

Multinomial Naive Bayes algorithm is a probabilistic approach to text classification. It is based on Bayes' theorem, which states that the probability of a hypothesis (in this case, a class) given the observed evidence (in this case, a document) is proportional to the probability of the evidence given the hypothesis multiplied by the prior probability of the hypothesis. In the context of text classification, we can use this theorem to calculate the probability of a document belonging to a particular class given its words.

The Multinomial Naive Bayes algorithm assumes that the probability of each word in a document is independent of the other words, given the class label. This assumption is called the "naive" assumption and is not always true in practice, but it simplifies the computation and often works well in practice.

The algorithm works by first building a model from a set of labeled training documents. It then uses this model to predict the class of new, unlabeled documents. The model consists of a set of probabilities that describe how likely each word in the vocabulary is to appear in a document of each class.

To build the model, we first calculate the prior probabilities of each class, which are simply the proportion of training documents that belong to each class. We then count the number of occurrences of each word in each class, and use these counts to estimate the conditional probabilities of each word given each class. We can use Laplace smoothing to avoid zero probabilities, which can cause problems with the multiplication of probabilities.

To predict the class of a new document, we simply calculate the probability of the document given each class, using the conditional probabilities from the model. We then multiply these probabilities by the prior probabilities of each class, and normalize them to get the posterior probabilities. The class with the highest posterior probability is the predicted class for the document.

The Multinomial Naive Bayes algorithm has been shown to work well in practice for text classification tasks, particularly for tasks with limited training data and a large number of features (i.e., a large vocabulary). However, it is not always the best choice for every task, and there are other algorithms that may be more appropriate in certain situations.

- ▼ Q2) Same in Semester 2022

2. Consider the following training data:

(10 marks)

S.No.	Document	Class
1.	Natural Language Processing	A
2.	Language Model Learning	A
3.	Ngram Langauge Model	A
4.	Text Classification Model	A
5.	Text Processing Model	A
6.	Computer Vision	B
7.	Image Classification Model	B
8.	Image Segmentation	B
9.	Image Processing	B
10.	Object Recognition	B

And Test Data:

1.	Image Learning Model	?
2.	Text Learning Model	?

Predict the Class for test samples using Bag of Words Language and Multinomial Model Naïve Bayes Text Classification. Apply add-2 La Place smoothing and show the calculation of posterior, class conditional and prior probabilities.

▼ Sessional 1 2022

Same as Sessional 1 of 2021

▼ Sessional 1 2023

▼ Q1)

MAX TIME: 1hr

1. Given,

Training Corpus: <s> I am from Jamia Millia Islamia </s>
<s> I am a student </s>
<s> engineering students are good and are from various cities</s>
<s> students are from Jamia engineering</s>

Test sentence: <s> students are from Jamia 600

For the test sentence:

- Calculate the unigram count matrix
- Calculate the bigram count matrix
- Calculate the bigram probability matrix
- Calculate the probability using unigram and bigram language models

[Note: Ignore the pre-processing and use the text as it is.] σ, γ, 8 [5 marks]

2. Consider the following text:

Unigram ($\langle S \rangle$)	Students	are	from	Jamal	$\langle S \rangle$
Count	4	2	3	3	2
Probability	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{4}$
bigram count matrix.					
with word $\langle S \rangle$					
Probability	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{4}$
preceding word $\langle S \rangle$	0	0	$\frac{2}{3}$	0	0
Students	0	0	0	0	0
are	0	0	$\frac{2}{3}$	0	0
from	0	0	0	$\frac{2}{3}$	0
Jamal	0	0	0	0	0
$\langle S \rangle$	0	0	0	0	0

unigram count = 8	Sentence 1 - 8
+1	1 → 1
+1	2 → 6
+1	3 → 11
+1	4 → 7
32	Total

unigram probability = $P(S)$

$$= P(\langle S \rangle) \cdot P(\text{Students}) \cdot P(\text{are}) \cdot P(\text{from}) \cdot P(\text{Jamal})$$

$$P(S|S)$$

$$= \frac{4}{32} \times \frac{2}{32} \times \frac{3}{32} \times \frac{2}{32} \times \frac{1}{32} \times \frac{4}{32}$$

$$= 976 \times (32)^{-6} \approx 5.36 \times 10^{-7}$$

bigram probability = $P(S)$

$$= P(\langle S \rangle) \cdot P(\text{Students}|\langle S \rangle) \cdot P(\text{are}|\text{Students}) \cdot P(\text{from}|\text{are})$$

$$\text{don't consider } \langle S \rangle \text{ at end of sentence}$$

$$P(\text{Jamal}|\text{from})$$

$$= \frac{4}{32} \times \frac{1}{4} \times \frac{2}{2} \times \frac{2}{3} \times \frac{2}{3} \times \frac{1}{3} = \frac{32}{32 \times 72} = \frac{1}{72}$$

$$= 0.0138$$

▼ Q2) - Same in Sessional 1 of 2021 and 2022

▼ Sessional 2 2021

▼ Q1) What are word vectors? Illustrate with the help of a suitable diagram. Define the cosine similarity between two word vectors.

Same in Semester 2022

▼ Q2) A char-context matrix (also called character co-occurrence matrix) is a char x char matrix which captures the number of times a character appears in the context of another character. Choose your full name as the corpus and consider the context of window +-4, i.e. 4 characters left and 4 characters right of any character.

Generate the simple char-context matrix for the corpus.

Ans -

▼ Sessional 2 2022

▼ Q1)

Note: Attempt all questions. Assume suitable data, if necessary.			
S.No.	Questions	Marks	CO
1.	<p>Given the following Corpus</p> <p><i>The eyes are not here</i></p> <p><i>There are no eyes here</i></p> <p><i>In this valley of dying stars</i></p> <p><i>In this hollow valley</i></p> <p><i>This broken jaw of our lost kingdoms</i></p> <p>Construct word-context matrix for <i>eyes</i>, <i>here</i> and <i>valley</i> using a three-word window. Calculate the similarity between <i>eyes</i> and <i>valley</i>.</p>	7	4

Ans1: Here is the word-context matrix for "eyes", "here", and "valley" using a three-word window:

To calculate the similarity between "eyes" and "valley," we can use the cosine similarity measure. First, we need to represent the two words as vectors, which can be done by extracting their rows from the matrix above:

Next, we can calculate the cosine similarity between the two vectors:

$$= 0 / (\sqrt{2} * \sqrt{19})$$

- 0

Q2) Explain word2vec algorithm using skip-gram with negative sampling.

Word2Vec is a neural network-based algorithm for learning word embeddings, which are vector representations of words in a high-dimensional space. The Skip-gram model is a popular variant of Word2Vec that predicts the context words given a target word. Negative sampling is a technique used to simplify the learning task and speed up training.

Here's how the Skip-gram model with negative sampling works:

Given a corpus of text, create a vocabulary of unique words, and assign each word a unique ID.

For each word in the corpus, generate training examples of the form (target word, context word), where the target word is the center word and the context word is one of the words that appear within a fixed-size window around the target word.

For each training example (target word, context word), use the Skip-gram model to predict the probability of the context word given the target word. Specifically, the model uses a softmax function to compute the conditional

probability of each context word in the vocabulary, given the target

$P(\text{context word} \mid \text{target word}) = \exp(v_c \cdot v_t) / \sum \exp(v_c \cdot v_i)$
 where v_c and v_t are the vector representations of the context word and target word, respectively, and v_i are

The model is trained by maximizing the log-likelihood of the observed context words given the target words. This is equivalent to minimizing the negative log-likelihood, which is known as the cross-entropy loss. However,

computing
the softmax function over the entire vocabulary can be computationally expensive, especially for large

To speed up training, the Skip-gram model uses negative sampling, which involves randomly selecting a small number (typically between 5 and 20) of "negative" context words for each target word. Negative context words

randomly from the vocabulary, but with a probability that is proportional to their frequency in the corpus. The objective of the model is to maximize the probability of the observed context word and minimize the probability of the negative context words.

To do this, the model uses a modified version of the softmax function, called the negative sampling loss function, which is defined as follows:

$$J = -\log(\sigma(v_c \cdot v_t)) - \sum_{i=1}^k \log(\sigma(-v_i \cdot v_t))$$

where $\sigma(x) = 1 / (1 + \exp(-x))$, k is the number of negative context words, and v_i are the vector representations of the negative context words.

During training, the parameters of the model (i.e., the word vectors) are updated using stochastic gradient descent to minimize the negative sampling loss function.

In summary, the Skip-gram model with negative sampling is a variant of the Word2Vec algorithm that predicts the context words given a target word. It uses a modified softmax function to compute the conditional probability of the context words and a negative sampling loss function to speed up training by selecting a small number of negative context words. The model is trained using stochastic gradient descent to minimize the negative sampling loss function and learn high-quality word embeddings.

Word2Vec Tutorial Part 2 - Negative Sampling · Chris McCormick

 <http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/>

How to Use Negative Sampling With Word2Vec Model?

Word2Vec is a technique to get a good quality word embedding from a corpus. CBOW and Skip-gram are the two well known word2vec models. There are a few features that both models have in common.

 <https://analyticsindiamag.com/how-to-use-negative-sampling-with-word2vec-model/>



▼ Q3) Define Named Entity Recognition and give examples.

Named Entity Recognition (NER) is a natural language processing (NLP) task that involves identifying and categorizing named entities in unstructured text into predefined categories such as person names, organizations, locations, dates, and so on. The goal of NER is to extract meaningful information from text and enable machines to understand the context of words in a given text. and is useful in a wide range of applications, including information retrieval, question answering, text summarization, and machine translation.

Examples of named entities include:

Person: John Smith, Barack Obama, Mary Jane

Organization: Microsoft, Google, United Nations

Location: New York, Paris, Tokyo

Date and Time: January 1st, 2022, 2:00 PM

Monetary Value: \$10,000, 1.5 million euros

Percentage: 50%, 75.5%

Product: iPhone, Tesla Model S, Coca-Cola

For example, consider the following sentence: "John Smith works as a software engineer at Google in New York."

A named entity recognition system would identify the following named entities:

Person: John Smith

Organization: Google

Location: New York

NER can also identify relations between named entities, such as "John Smith works at Google." This information can be used for knowledge extraction and various applications in natural language processing.

Another example, consider the following sentence: "Barack Obama was the 44th President of the United States."

In this sentence, "Barack Obama" is a person name, "United States" is a location, and "44th" is a numerical expression that indicates a date. NER systems can identify these entities and categorize them into the appropriate categories.

Some other examples of named entities are:

- Organizations: Google, Microsoft, IBM, Facebook
- Locations: New York City, Paris, Tokyo, Mount Everest
- Dates: January 1, 2022, 4th of July, Christmas Day
- Times: 10:30 am, 3:00 pm, midnight
- Monetary values: \$5,000, €1.2 million, ¥100,000

NER is an important task in many applications such as information retrieval, sentiment analysis, and question answering. It is also used in text mining to extract structured information from unstructured text data, which can be used for business intelligence and decision-making purposes.

▼ Sessional 2 2023

▼ Q1)

1. Consider the following Markov Model:

```
graph LR; A((uniformly)) -- ".4" --> B((are)); A -- ".5" --> C((charming)); B -- ".1" --> A; B -- ".5" --> C; C -- ".2" --> B; C -- ".6" --> A;
```

(1) Generate all the possible first order Markov chains of three words.
(2) Find out the chain which has maximum probability.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/aaed96ba-f048-479f-ba64-6c904881907f/Adobe_Scan_14_May_2023.pdf

▼ Q2)

2. State all the components of a Hidden Markov Model.

Consider the following transition table

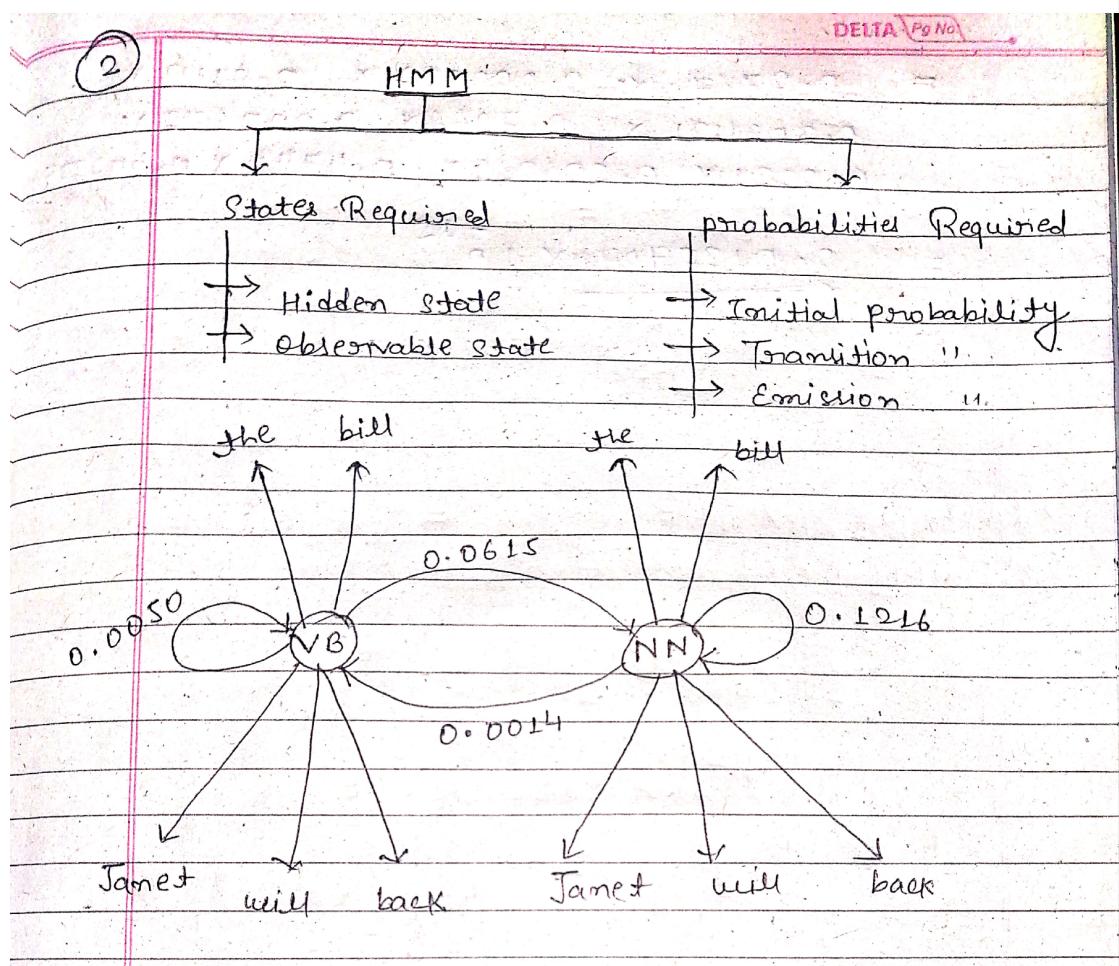
	NNP	MD	VB	JJ	NN	RB	DT
<S>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

and the following observation table

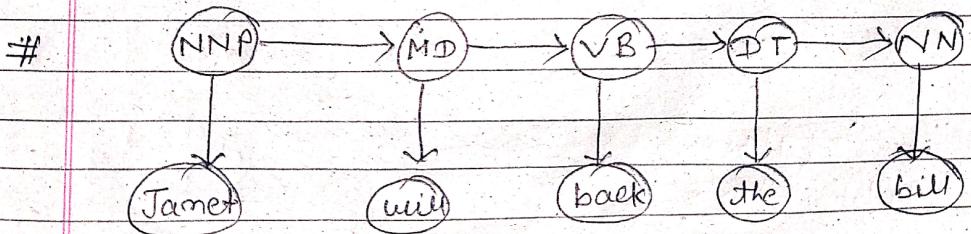
	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Using the tables, give the graphical representation of HMM for two states: VB and NN.

Calculate is the probability of the sequence: Janet/NNP will/MD back/VB the/DT bill/NN



Graphical Representation



$$\begin{aligned}
 &= p(\text{NNP}) \times p(\text{James} / \text{NNP}) \times p(\text{MD} / \text{NNP}) \times \\
 &\quad p(\text{will} / \text{MD}) \times p(\text{VB} / \text{MD}) \times p(\text{back} / \text{VB}) \times \\
 &\quad p(\text{DT} / \text{VB}) \times p(\text{the} / \text{DT}) \times p(\text{NN} / \text{DT}) \times p(\text{bill} / \text{NN})
 \end{aligned}$$

DETA Pg No.

$$\begin{aligned}
 &= 0.2767 \times 0.000032 \times 0.0110 \times \\
 &\quad 0.308432 \times 0.7968 \times 0.000672 \times \\
 &\quad 0.2231 \times 0.506099 \times 0.4744 \times 0.002397 \\
 &= \underline{\underline{2.01357071}} \times 10^{-15}
 \end{aligned}$$

▼ Q3) - Same as Sessional 1 2021 Q2

▼ Semester 2021

▼ Ans 1 (A) -

S.No.	Questions	Marks	CO
1(a)	Apply the Max-Match word segmentation algorithm on the following: "wecanonlyseashortdistanceahead" Assume you have the English all vocabulary set.	6	1

The Max-Match algorithm works by starting from the beginning of the text and attempting to match the longest possible word from a given dictionary. Once a word is found, it is added to the output, and the algorithm moves to the next unmatched portion of the text.

Q. 1(a) ~~2021~~ Endsen

- → Find longest prefix word present in dictionary.
- Recursively repeat for the left out suffix.
Repeat step 2

→ ~~we~~

~~we + canonlyseeas short distance ahead~~

↓

~~canon + lyseeas short distance ahead~~

↓

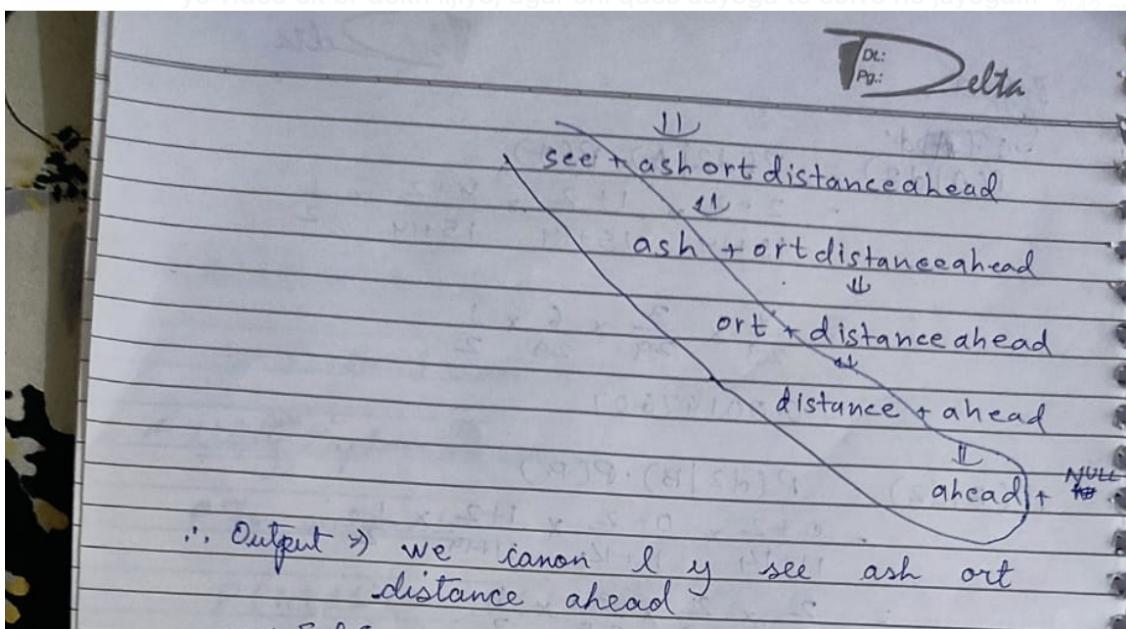
~~lyseeas short distance ahead~~

↓

~~lyseeas short distance ahead~~

↓

~~y + seeas short distance ahead~~



Ans - we can only see ash or t distance ahead

▼ Ans 1 (B) -

1(b)

Consider Count(w), Count(v,w) be unigram and bigram counts taken from a training corpus, where w is the single word, v,w is a bigram. Let N be the total number of words in the Corpus. What are the maximum likelihood estimates for unigram and bigram model?

6

1

The maximum likelihood estimates (MLE) for unigram and bigram models can be calculated as follows:

1. Unigram Model:

The probability of a word w in the unigram model is given by:

$$P(w) = \text{Count}(w) / N$$

where Count(w) is the number of occurrences of the word w in the corpus, and N is the total number of words in the corpus.

1. Bigram Model:

The probability of a bigram v,w in the bigram model is given by:

$$P(w | v) = \text{Count}(v,w) / \text{Count}(v)$$

where Count(v,w) is the number of occurrences of the bigram v,w in the corpus, and Count(v) is the number of times the word v appears as the first word in any bigram in the corpus.

It's worth noting that the denominator Count(v) is equivalent to the number of occurrences of the word v in the corpus minus one, since we can't count the last word in each document as the first word of a bigram.

If we want to smooth the probabilities to avoid zero probabilities for unseen bigrams, we can use techniques such as Laplace smoothing or add-k smoothing, where we add a constant k to the numerator and k times the vocabulary size to the denominator.

For example, Laplace smoothing for the bigram model would give:

$$P(w | v) = (\text{Count}(v,w) + \alpha) / (\text{Count}(v) + V*\alpha)$$

if alpha = 1, then

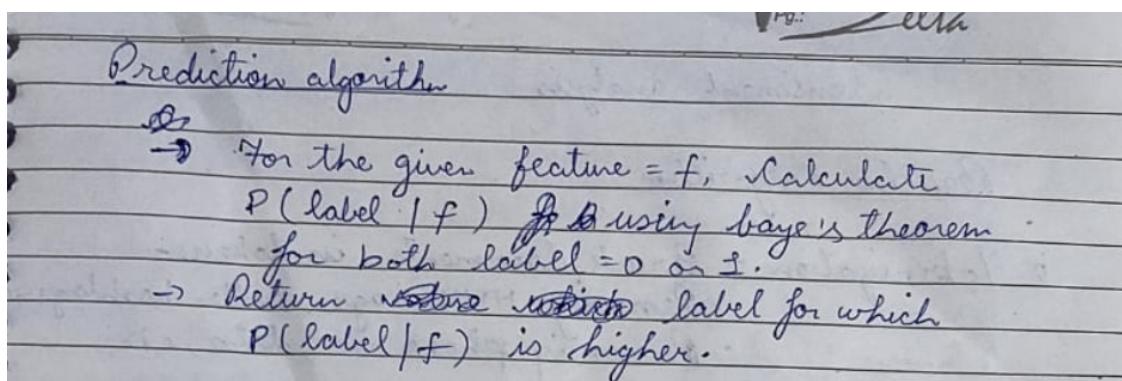
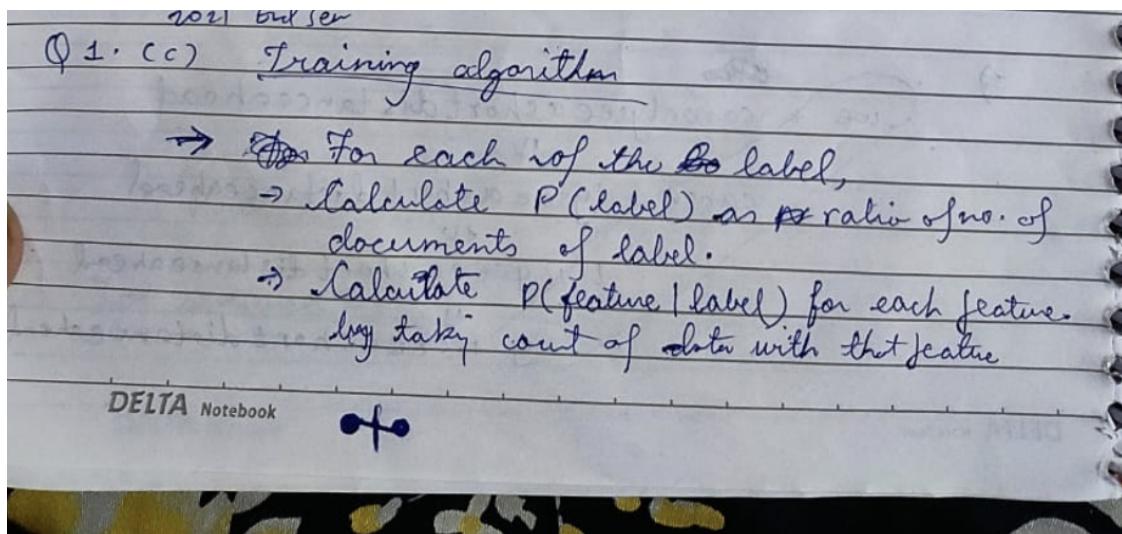
$$P(w | v) = (\text{Count}(v, w) + 1) / (\text{Count}(v) + V)$$

where V is the size of the vocabulary (i.e., the number of unique words) in the corpus.

Note that MLE estimates may not always be optimal, especially for rare events, and more sophisticated smoothing techniques and language models may be needed in practice.

▼ Ans 1 (C) -

 | |



Write the training and prediction algorithm for Bernoulli Naive Bayes Text Classification. Consider the following dataset:

Class Feature	0	1	Total no. of features
0	4	3	7
1	6	7	13
Total features per class	10	10	20

$P(a=0 | \text{feature}=0) = P(a=0, \text{feature}=0) / P(\text{feature}=0)$
 $= P(\text{feature}=0 | \text{class}=0) \cdot \frac{P(\text{class}=0)}{P(\text{feature}=0)}$
 $= \frac{4}{10} \times \frac{1/2}{7/20}$
 $P(a=1 | \text{feature}=0) = P(\text{feature}=0 | \text{class}=1) = \frac{4}{10} \times \frac{1}{2} \times \frac{20}{7} = \frac{4}{7}$
 $\times \frac{P(\text{class}=1)}{P(\text{feature}=0)}$
 $= \frac{3}{10} \times \frac{1/2}{7/20} = \frac{3}{10} \times \frac{1}{2} \times \frac{20}{7} = \frac{3}{7}$

$$\text{Similarly } P(b=0 \mid \text{feature}=1) = \frac{P(\text{feature}=1 \mid \text{class}=0) \times P(\text{class}=0)}{P(\text{feature}=1)}$$

$$= \frac{\frac{6}{10} \times \frac{1}{2} \times \frac{20}{13}}{\frac{6}{13}} = \frac{6}{13}$$

$$P(b=1 | \text{feature}=1) = P(\text{feature}=1 | \text{class}=1) \times \frac{P(\text{class}=1)}{P(\text{feature}=1)}$$

$$= \frac{7}{10} \times \frac{1}{2} \times \frac{20}{13} = \frac{7}{13}$$

$$\Rightarrow b = 1$$

- ▼ Ans 2 (A) - Same in 2022 Chinese one

Same in Sem 2022

- ▼ Ans 2 (B) - Not in syllabus

	Given a corpus $w_1, w_2, w_3, \dots, w_N$ such that each word is labelled with three POS tags: NOUN, VERB, ADJECTIVE, OTHER.		
2(b)	Construct the graphical sequence labelling model using HMM for POS tagging and define all the transition and emission probabilities.	6	2

Not in Syllabus

- ▼ Ans 2 (C) - Same in 2022 Precision Def

Same in Sem 2022

▼ Ans 3 (A) - Same in 2022

▼ Ans 3 (B) - Same in 2022

▼ Ans 3 (C) - Same in 2022

▼ Ans 4 (A) - Same in 2022

▼ Ans 4 (B) -

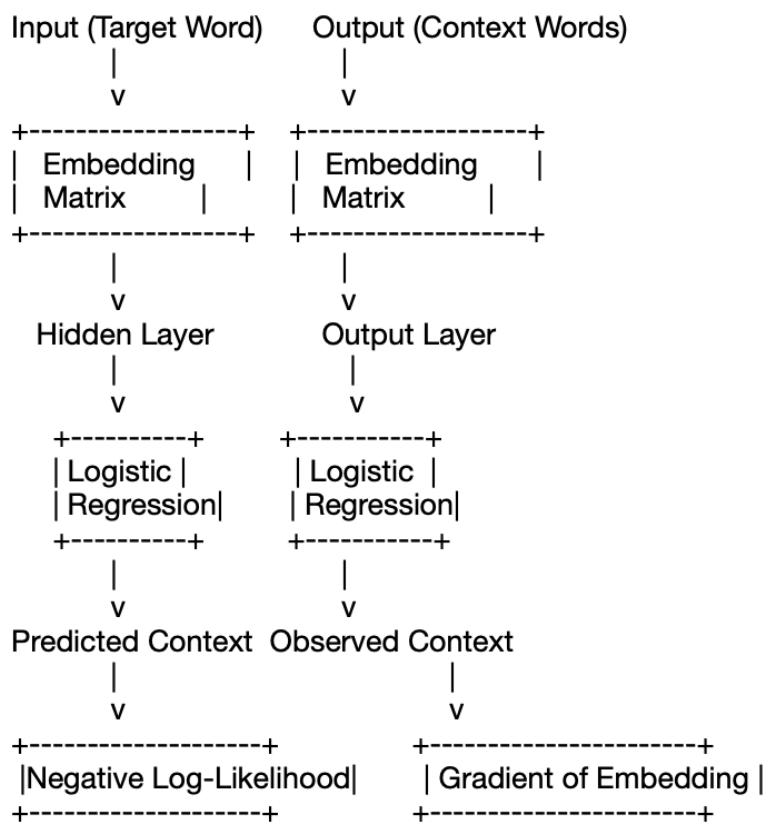
Describe the Logistic Regression based skip gram model with the help of suitable diagram.

The Logistic Regression based skip-gram model is a neural network language model that uses logistic regression to predict the context words given a target word. It is a modification of the original skip-gram model, which uses softmax regression for the same purpose.

The basic architecture of the Logistic Regression based skip-gram model consists of two layers: the input layer and the output layer. The input layer represents the target word, which is transformed into a vector representation using an embedding matrix. The output layer represents the context words, which are also transformed into vector representations using the same embedding matrix.

The goal of the model is to learn the probability distribution of the context words given the target word. To achieve this, the model uses logistic regression to estimate the conditional probability of each context word given the target word.

The training process involves minimizing the negative log-likelihood of the observed context words given the target word. This is done by adjusting the weights of the embedding matrix using backpropagation. The following diagram illustrates the architecture of the Logistic Regression based skip-gram model:



▼ Ans 4 (C) - Same in 2022

▼ Ans 5 (A) - Same in 2022

▼ Ans 5 (B) - Same in 2022

▼ Ans 5 (C) - Same in 2022

▼ Semester 2022

▼ Ans 1 (A) -

S.No.	Questions	Marks	CO										
1(a)	<p>What is tokenization? What is the advantage of using Byte Pair Encoding scheme for tokenization? Train the BPE algorithm using the following table.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>low</td> <td>5</td> </tr> <tr> <td>lowest</td> <td>2</td> </tr> <tr> <td>newer</td> <td>6</td> </tr> <tr> <td>wider</td> <td>3</td> </tr> <tr> <td>new</td> <td>2</td> </tr> </table> <p>Test the BPE algorithm using the word "lower" and show its tokenization. Show all the steps for training and testing clearly.</p>	low	5	lowest	2	newer	6	wider	3	new	2	6	1
low	5												
lowest	2												
newer	6												
wider	3												
new	2												

Q what is tokenization ? what is the advantage of using Byte pair encoding scheme for tokenization ? Train the BPE algorithm using the following table

low	5
lowest	2
newer	6
wider	3
new	2

Test the BPE algorithm using the word "lower" and show its tokenization . Show all the steps for training and testing clearly.

Token

Tokenization is the process of dividing a text into smaller units called tokens. These tokens can be further processed by NLP models. There are many different tokenization schemes, each with its own advantages and disadvantages.

Byte Pair Encoding (BPE) is a subword-based tokenisation scheme that is becoming increasingly popular in NLP. It works by iteratively merging the most frequent pair of characters or character sequences until a predefined vocabulary size is reached. BPE has an advantage that it can handle rare words that would not be represented in a traditional word-based tokenization scheme.

Training

The word frequencies given are

(low, 5), (lowest, 2), (newer, 6), (wider, 3), (new, 2)

lets split each word into characters.

"l", "o", "w", 5), ("l", "o", "w", "e", "s", "t", 2), ("n", "e", "w", "e", "r", 6), ("w", "i", "d", "e", "r", 3), ("n", "e", "w", 2)

we look at pairs. $\text{lo} \rightarrow 7$, $\text{ow} \rightarrow 7$, $\text{we} \rightarrow 8$, $\text{es} \rightarrow 2$, $\text{st} \rightarrow 2$, $\text{ne} \rightarrow 8$, $\text{et} \rightarrow 8$, $\text{id} \rightarrow 9$, $\text{wi} \rightarrow 3$, $\text{de} \rightarrow 3$.

\Rightarrow most frequent pair $\rightarrow \boxed{\text{er} \rightarrow 9}$

\Rightarrow Vocabulary: ["l", "o", "w", "e", "s", "t", "n", "r", "i", "d", "e", "x"]

Corpus: ("l", "o", "w", 5), ("l", "o", "w", "e", "s", "t", 2), ("n", "e", "w", "e", "r", 6), ("w", "i", "d", "e", "r", 3), ("n", "e", "w", 2)

again, mfp sequence = $\text{we} \rightarrow 8$, $\text{ne} \rightarrow 8$, $\text{et} \rightarrow 8$, $\text{lo} \rightarrow 7$, $\text{ow} \rightarrow 7$, $\text{we} \rightarrow 2$, $\text{es} \rightarrow 2$, $\text{st} \rightarrow 2$, $\text{ne} \rightarrow 8$, $\text{ew} \rightarrow 8$, $\text{wer} \rightarrow 6$, $\text{id} \rightarrow 3$, $\text{de} \rightarrow 3$, $\text{der} \rightarrow 3$.

\Rightarrow mfp $\rightarrow \boxed{\text{ne} \rightarrow 8}$

<p>vocabulary: [l, o, w, e, s, t, n, r, i, d, er, ne, new, low]</p> <p>corpus: ("low", 5), ("low", "e", "r", 2), ("new", "er", 6), ("wid", "er", 3) ("new", 2)</p> <p>mfp: $\begin{cases} \text{low} \rightarrow 7 \\ \text{ow} \rightarrow 7 \\ \text{we} \rightarrow 2 \\ \text{es} \rightarrow 2 \\ \text{st} \rightarrow 2 \\ \text{new} \rightarrow 8 \\ \text{newer} \rightarrow 6 \\ \text{wi} \rightarrow 3 \\ \text{id} \rightarrow 3 \\ \text{der} \rightarrow 3 \end{cases}$</p>
<p>vocab: [-]</p> <p>corpus: [-]</p> <p>mfp: $\begin{cases} \text{low} \rightarrow 7 \\ \text{ow} \rightarrow 7 \\ \text{we} \rightarrow 2 \\ \text{es} \rightarrow 2 \\ \text{st} \rightarrow 2 \\ \text{newer} \rightarrow 6 \\ \text{wi} \rightarrow 3 \\ \text{id} \rightarrow 3 \\ \text{der} \rightarrow 3 \end{cases}$</p>
<p>vocab: [-]</p> <p>corpus: [-]</p> <p>mfp: $\begin{cases} \text{low} \rightarrow 7 \\ \text{ow} \rightarrow 7 \\ \text{we} \rightarrow 2 \\ \text{es} \rightarrow 2 \\ \text{st} \rightarrow 2 \\ \text{newer} \rightarrow 6 \\ \text{wi} \rightarrow 3 \\ \text{id} \rightarrow 3 \\ \text{der} \rightarrow 3 \end{cases}$</p>
<p>vocab: [-]</p> <p>corpus: [-]</p> <p>mfp: $\begin{cases} \text{low} \rightarrow 7 \\ \text{ow} \rightarrow 7 \\ \text{we} \rightarrow 2 \\ \text{es} \rightarrow 2 \\ \text{st} \rightarrow 2 \\ \text{newer} \rightarrow 6 \\ \text{wi} \rightarrow 3 \\ \text{id} \rightarrow 3 \\ \text{der} \rightarrow 3 \end{cases}$</p>
<p>vocab: [-]</p> <p>corpus: [-]</p> <p>mfp: $\begin{cases} \text{low} \rightarrow 7 \\ \text{ow} \rightarrow 7 \\ \text{we} \rightarrow 2 \\ \text{es} \rightarrow 2 \\ \text{st} \rightarrow 2 \\ \text{newer} \rightarrow 6 \\ \text{wi} \rightarrow 3 \\ \text{id} \rightarrow 3 \\ \text{der} \rightarrow 3 \end{cases}$</p>
<p>vocab: [-]</p> <p>corpus: [-]</p> <p>mfp: $\begin{cases} \text{low} \rightarrow 7 \\ \text{ow} \rightarrow 7 \\ \text{we} \rightarrow 2 \\ \text{es} \rightarrow 2 \\ \text{st} \rightarrow 2 \\ \text{newer} \rightarrow 6 \\ \text{wi} \rightarrow 3 \\ \text{id} \rightarrow 3 \\ \text{der} \rightarrow 3 \end{cases}$</p>
<p>vocab: [-]</p> <p>corpus: [-]</p> <p>mfp: $\begin{cases} \text{low} \rightarrow 7 \\ \text{ow} \rightarrow 7 \\ \text{we} \rightarrow 2 \\ \text{es} \rightarrow 2 \\ \text{st} \rightarrow 2 \\ \text{newer} \rightarrow 6 \\ \text{wi} \rightarrow 3 \\ \text{id} \rightarrow 3 \\ \text{der} \rightarrow 3 \end{cases}$</p>
<p>vocab: [-]</p> <p>corpus: [-]</p> <p>mfp: $\begin{cases} \text{low} \rightarrow 7 \\ \text{ow} \rightarrow 7 \\ \text{we} \rightarrow 2 \\ \text{es} \rightarrow 2 \\ \text{st} \rightarrow 2 \\ \text{newer} \rightarrow 6 \\ \text{wi} \rightarrow 3 \\ \text{id} \rightarrow 3 \\ \text{der} \rightarrow 3 \end{cases}$</p>

<p>vocab: [-]</p> <p>corpus: [-]</p> <p>\Rightarrow Testing the final tokenisation of the word "lower"</p> <p>lower \Rightarrow $\begin{cases} l, o, w, e, s, t, n, r, i, d, \boxed{\text{er}}, \text{ne}, \text{new}, \text{lo}, \boxed{\text{low}}, \text{wi}, \text{wid}, \text{es}, \text{est} \end{cases}$ \Rightarrow low + er</p>
--

▼ Ans 1 (B) -

	Given the following bigram counts.									
1(b)	i	i	want	to	eat	Chinese	food	lunch	spend	
	i	5	827	0	9	0	0	0	2	
	want	2	0	608	1	6	6	5	1	
	to	2	0	4	686	2	0	6	211	
	eat	0	0	2	0	16	2	42	0	
	Chinese	1	0	0	0	0	82	1	0	
	food	15	0	15	0	1	4	0	0	
	lunch	2	0	0	0	0	1	0	0	
	spend	1	0	1	0	0	0	0	0	

6 1

	(1) Generate the Bigram probability matrix. (2) Generate the Bigram probability matrix with add-1 smoothing.		
--	---	--	--

Figure 3.2 shows the bigram probabilities after normalization (dividing each cell in Fig. 3.1 by the appropriate unigram for its row, taken from the following set of unigram probabilities):

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Unigrams: i 0.2533 want 0.0927 to 0.2417 eat 0.0746 chinese 0.0158 food 0.1093 lunch 0.0341 spend 0.0278

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

Here its given $V = 1446$ and the count matrix is also given, but in questions its not so calc it by adding rows and columns.

Also for laplace smoothing add 1 to all the places

With Laplace -

Now that we have the intuition for the unigram case, let's smooth our Berkeley Restaurant Project bigrams. Fig. 4.5 shows the add-one smoothed counts for the bigrams in Fig. 4.1.

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1		609	2	7	7	2
to	3	1		5	687	3	1	7
eat	1	1		3	1	17	3	1
chinese	2	1	1	1	1	1	83	2
food	16	1		16	1	2	5	1
lunch	3	1	1	1	1	2	1	1
spend	2	1		2	1	1	1	1

Figure 4.5 Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

Fig. 4.6 shows the add-one smoothed probabilities for the bigrams in Fig. 4.2. Recall that normal bigram probabilities are computed by normalizing each row of counts by the unigram count:

$$(4.22) \quad P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

For add-one smoothed bigram counts we need to augment the unigram count by the number of total word types in the vocabulary V :

$$(4.23) \quad P_{\text{Laplace}}^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

Thus each of the unigram counts given in the previous section will need to be augmented by $V = 1446$. The result is the smoothed bigram probabilities in Fig. 4.6.

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Figure 4.6 Add-one smoothed bigram probabilities for eight of the words (out of $V = 1446$) in the BeRP corpus of 9332 sentences.

1(c)	Define and give examples: (1) Segmentation (2) Lemmatization (3) Stemming	6	1

(1) Segmentation is the process of dividing a sentence or a document into smaller units, such as words or phrases, for further analysis. This is a common technique used in natural language processing and text analytics.

For example, consider the sentence: "I went to the store to buy some groceries."

Segmentation would break down this sentence into individual words: "I," "went," "to," "the," "store," "to," "buy," "some," "groceries."

(2) Lemmatization is the process of reducing a word to its base or root form, known as the lemma. This is done to simplify the analysis of text by grouping together words that have the same base form.

For example, consider the words: "am," "is," "are," "was," "were."

Lemmatization would reduce all of these words to their base form "be".

(3) Stemming is the process of reducing a word to its base form by removing the suffixes. This is a simpler approach than lemmatization because it does not consider the context of the word.

For example, consider the word "jumping."

Stemming would remove the "ing" suffix and reduce the word to its base form, "jump."

▼ Ans 2 (A) -

2(a)	<table border="1"> <thead> <tr> <th></th><th>Doc</th><th>Words</th><th>Class</th></tr> </thead> <tbody> <tr> <td rowspan="4">Training</td><td>1</td><td>Chinese, Beijing, Chinese</td><td>C</td></tr> <tr> <td>2</td><td>Chinese, Chinese, Shanghai</td><td>C</td></tr> <tr> <td>3</td><td>Chinese, Macao</td><td>C</td></tr> <tr> <td>4</td><td>Tokyo, Japan, Chinese</td><td>J</td></tr> <tr> <td>Test</td><td>5</td><td>Chinese, Chinese, Tokyo, Japan</td><td>?</td></tr> </tbody> </table> <p>Compute the most likely class for Doc 5. Assume a multinomial naive Bayes classifier and use add-α La Place smoothing for the likelihoods. Suitable value of α may be chosen.</p>		Doc	Words	Class	Training	1	Chinese, Beijing, Chinese	C	2	Chinese, Chinese, Shanghai	C	3	Chinese, Macao	C	4	Tokyo, Japan, Chinese	J	Test	5	Chinese, Chinese, Tokyo, Japan	?	6	2
	Doc	Words	Class																					
Training	1	Chinese, Beijing, Chinese	C																					
	2	Chinese, Chinese, Shanghai	C																					
	3	Chinese, Macao	C																					
	4	Tokyo, Japan, Chinese	J																					
Test	5	Chinese, Chinese, Tokyo, Japan	?																					

Naive Bayes

Relating to language model

Ex.

TARGET SITE

	Doc	Category
Training	1 Chinese Beijing Chinese	Next
	2 Chinese Chinese Shanghai	
	3 Chinese Macao	
Test	4 Tokio Japan Chinese	
	5 Chinese Chinese Tokyo Japanese	

Probability to come in the class

$\pi_c = \frac{p(c)}{p(c) + p(s)}$

Having a class

$p(c|ds) = \frac{\pi_c}{5} \times \left(\frac{2}{3}\right)^3 \times \left(\frac{1}{3}\right)^2 \times \left(\frac{1}{3}\right)$

≈ 0.0003

Probability to come in the class

$p(s|ds) = \frac{\pi_s}{5} \times \left(\frac{2}{3}\right)^2 \times \left(\frac{1}{3}\right)^3 \times \left(\frac{1}{3}\right)$

≈ 0.0001

∴ ds comes into class 'c'

Conditional Probabilistic

$p(c|ds) > p(s|ds)$

\rightarrow ds comes into class 'c'

$p(c) = \frac{3}{5}$ → Probability of class 'c'

$p(s) = \frac{2}{5}$ → Probability of class 's'

\rightarrow ds comes into class 'c'

▼ Ans 2 (B) -

	<p>Consider the following training data:</p> <table border="1"> <thead> <tr> <th>S.No.</th><th>Document</th><th>Class</th></tr> </thead> <tbody> <tr><td>1.</td><td>Natural Language Processing</td><td>A</td></tr> <tr><td>2.</td><td>Language Model Learning</td><td>A</td></tr> <tr><td>3.</td><td>Ngram Langauge Model</td><td>A</td></tr> <tr><td>4.</td><td>Text Classification Model</td><td>A</td></tr> <tr><td>5.</td><td>Text Processing Model</td><td>A</td></tr> <tr><td>6.</td><td>Computer Vision</td><td>B</td></tr> <tr><td>7.</td><td>Image Classification Model</td><td>B</td></tr> <tr><td>8.</td><td>Object Segmentation</td><td>B</td></tr> <tr><td>9.</td><td>Image Processing</td><td>B</td></tr> <tr><td>10.</td><td>Object Recognition</td><td>B</td></tr> </tbody> </table> <p>And Test Data:</p> <table border="1"> <tbody> <tr><td>1.</td><td>Object Recognition Model</td><td>?</td></tr> <tr><td>2.</td><td>Text Recognition Model</td><td>?</td></tr> </tbody> </table> <p>Predict the class for test samples using Multinomial Naïve Bayes and Bigram language model with La-Place smoothing.</p>	S.No.	Document	Class	1.	Natural Language Processing	A	2.	Language Model Learning	A	3.	Ngram Langauge Model	A	4.	Text Classification Model	A	5.	Text Processing Model	A	6.	Computer Vision	B	7.	Image Classification Model	B	8.	Object Segmentation	B	9.	Image Processing	B	10.	Object Recognition	B	1.	Object Recognition Model	?	2.	Text Recognition Model	?	6	2
S.No.	Document	Class																																								
1.	Natural Language Processing	A																																								
2.	Language Model Learning	A																																								
3.	Ngram Langauge Model	A																																								
4.	Text Classification Model	A																																								
5.	Text Processing Model	A																																								
6.	Computer Vision	B																																								
7.	Image Classification Model	B																																								
8.	Object Segmentation	B																																								
9.	Image Processing	B																																								
10.	Object Recognition	B																																								
1.	Object Recognition Model	?																																								
2.	Text Recognition Model	?																																								

S.No	Document			class
1	Natural	language	Processing	A
2	Language	Model	Learning	A
3	Ngiam	Language	Model	A
4	Text	Classification	Model	A
5	Text	Processing	Model	A
6	Computer	Vision		B
7	Image	classification	Model	B
8	Object	Segmentation		B
9	Image	Processing		B
10	object	Recognition		B

Sessinal ->
2022

Date:

Training

$P(A) = \frac{5}{10}$, $P(B) = \frac{5}{10}$

$= \frac{1}{2}$ $= \frac{1}{2}$

$|W| = \text{unique words}$
= 14

Now

Test data :-

1. object Recognition Model
 2. Text Recognition Model.

For

First Test data —

object Recognition Model

$$P(\text{object} | A) = \frac{0+1}{15+14} = \frac{1}{29}$$

$$P(\text{Recognition} | A) = \frac{0+1}{15+14} = \frac{1}{29}$$

$$P(\text{Model} | A) = \frac{4+1}{15+14} = \frac{5}{29}$$

$$P(\text{object} | B) = \frac{2+1}{11+14} = \frac{3}{25}$$

$$P(\text{Recognition} | B) = \frac{1+1}{11+14} = \frac{2}{25}$$

$$1.02 \times 10^{-4}$$

$$5.27 \times 10^{-4}$$

Now,

choosing a class : — $P(c) + P(w|c)$

$$P(A|d_1) = \frac{1}{2} \times \frac{1}{29} \times \frac{1}{29} \times \frac{5}{29}$$

$$= 0.0001$$

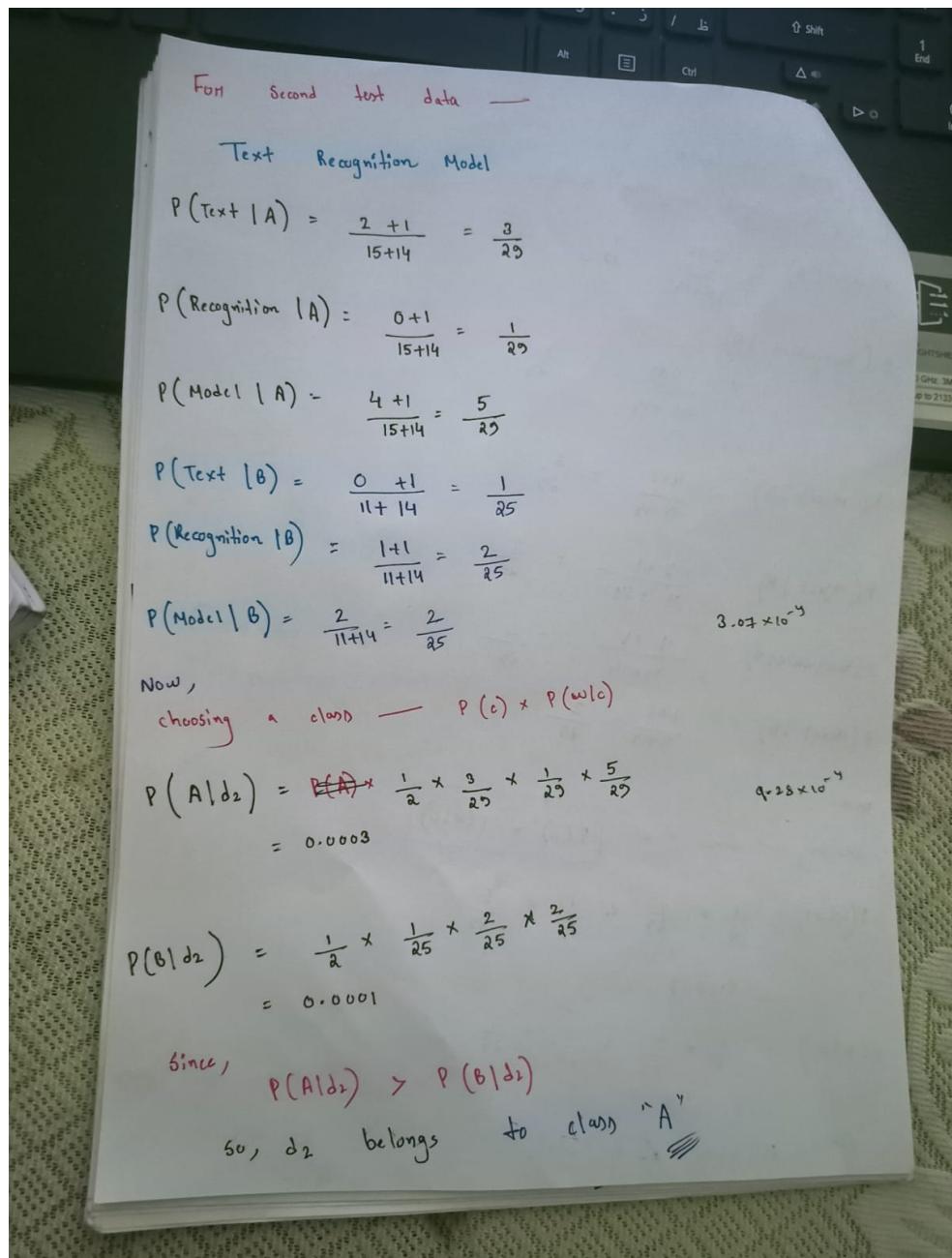
$$P(B|d_1) = \frac{1}{2} \times \frac{3}{25} \times \frac{2}{25} \times \frac{2}{25}$$

so, d_1 belongs to class

$$= 0.0003$$

since, $P(A|d_1) \leq P(B|d_1)$

"B"



▼ Ans 2 (C) -

2(c)

Define Precision, Recall and F-measure.

6

2

Precision, recall, and F-measure are three commonly used evaluation metrics in the field of information retrieval and machine learning.

(1) Precision: Precision is the fraction of retrieved instances that are relevant to the query. It measures how accurate the results are, in terms of true positives among all predicted positives.

Precision = True Positives / (True Positives + False Positives)

For example, in a spam detection system, precision would measure how many of the emails classified as spam are actually spam.

(2) Recall: Recall is the fraction of relevant instances that are retrieved by the system. It measures how complete the results are, in terms of true positives among all actual positives.

$$\text{Recall} = \text{True Positives} / (\text{True Positives} + \text{False Negatives})$$

For example, in a search engine, recall would measure how many of the relevant documents are returned in response to a query.

(3) F-measure: The F-measure is a weighted harmonic mean of precision and recall. It is used to evaluate the overall performance of a classification or retrieval system.

$$\text{F-measure} = 2 * ((\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall}))$$

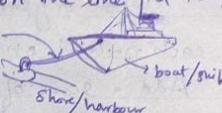
The F-measure balances the importance of precision and recall in the final evaluation, providing a single score that represents both aspects of the system's performance. It is commonly used as a single metric to compare different algorithms or systems.

▼ Ans 3 (A) -

3(a)	<p>Given the following Dictionary entry for line.</p> <p>line² a length of cord, rope, wire, or other material serving a particular purpose: <i>wring the clothes and hang them on the line</i> <i>a telephone line</i>.</p> <ul style="list-style-type: none">• one of a vessel's mooring ropes.• a telephone connection: <i>she had a crank on the line</i>.• a railroad track.• a branch or route of a railroad system: <i>the Philadelphia to Baltimore line</i>. <p>line³ a horizontal row of written or printed words.</p> <ul style="list-style-type: none">• a part of a poem forming one such row: <i>each stanza has eight lines</i>.• (lines) the words of an actor's part in a play or film.• a particularly noteworthy written or spoken sentence: <i>his speech ended with a line about the failure of justice</i>. <p>Which of these senses are related by homonymy, and which are related by polysemy? For any senses which are polysemous, give an argument as to how the senses are related.</p>	6	3

Q 3(a) 2nd sense → line²
 of line
 gloss → a length of cord, rope, wire, or other material serving a particular purpose: washing the clothes and hang them on the line / a telephone line.

examples
 1.1. one of a vessel's mooring ropes
 1.2. a telephone connection: she had a line.
 1.3. a railroad track
 1.4. a branch or route of a railroad system: the Philadelphia to Baltimore line



3rd sense of line → line³
 gloss - a horizontal row written or printed words.

examples
 2.1. a part of a poem forming one such row: each stanza has eight lines.
 2.2. (lines) the words of an actor's part in a play or film.
 2.3. a particularly noteworthy written or spoken sentence: his speech ended with a line about the failure of justice.

Homonymy: are ~~words~~ which have similar sound ~~or~~ spelling (form) but are unrelated i.e. completely different meaning.
 ↳ ~~→~~ (1.1), (1.2), (1.3), ~~2.1~~, (2.1, 2.2, 2.3)
 are homonyms
 ↳ a vessel's mooring rope is different from a telephone line which is different from a railroad track or route.
 ↳ well is different from a written sentence (part of poem, words of actor, spoken sentence)

Polysemy: ~~are~~ words which have related meaning.
 (1.3, 1.4) are polysemic as both point to railroad

(2.1, 2.2, 2.3) are polysemic as all three point to some written or spoken ~~sentence~~ sentences

The word "line" has multiple senses in the given dictionary entry, and some of them are related by homonymy while others are related by polysemy. The senses related by homonymy are:

line2: a length of cord, rope, wire, or other material serving a particular purpose.

line2: one of a vessel's mooring ropes.

line2: a railroad track.

These senses are related by homonymy because they have different meanings that are not related to each other and are not derived from the same source.

The senses related by polysemy are:

line2: a length of cord, rope, wire, or other material serving a particular purpose.

line2: a telephone line.

line3: a horizontal row of written or printed words.

line3: a part of a poem forming one such row.

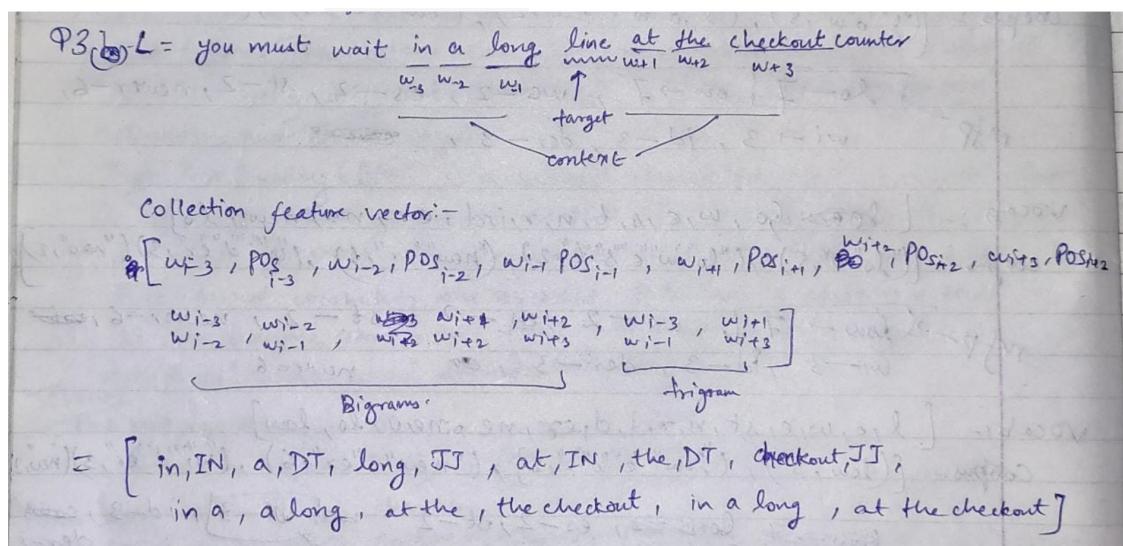
line3: (lines) the words of an actor's part in a play or film.

line3: a particularly noteworthy written or spoken sentence.

These senses are related by polysemy because they share a common meaning that is derived from the same source. The common meaning is related to the concept of "line" as a continuous or connected series or sequence of things. For example, a line of cord or rope is a continuous length of material, a telephone line is a connected series of wires or circuits, a line of text is a connected sequence of written or printed words, and a line in a play or film is a connected sequence of dialogue or action.

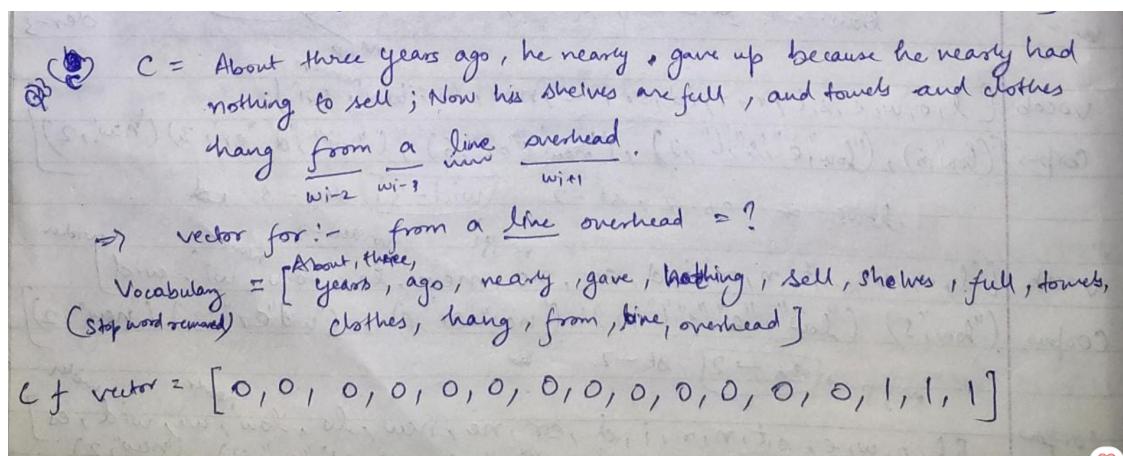
▼ Ans 3 (B) -

3(b)	<p>Assume the following sentence L in which the word <i>line</i> is in focus:</p> <p>L = you must wait in a long <i>line</i> at the checkout counter</p> <p>Give a collocation feature vector (including n-gram) for in the word <i>line</i> in L, given a window size of 3 words to the left and 3 words to the right.</p>	6	3
------	---	---	---



▼ Ans 3 (C) -

3(c)	<p>C = About three years ago, he nearly gave up because he nearly had nothing to sell; Now his shelves are full, and towels and clothes hang from a <u>line</u> overhead.</p> <p>For the word <u>line</u> in the above text L, generate the bag-of-words feature vector for window size = +2, assume C as the whole corpus.</p>	6	3
------	---	---	---



▼ Ans 4 (A) -

4(a)	For the following term document matrix:		6 4

For the following term document matrix:

	Document 1	Document 2	Document 3	Document 3
digital	1	0	7	13
computer	114	80	62	89
information	36	58	1	4

	data	20	15	2	3		
	Calculate the similarity between good and fool using (i) cosine similarity (ii) PPMI. Use add 2 smoothing if necessary.						

(i) Cosine Similarity:

To calculate the cosine similarity between "good" and "fool", we first need to represent these words as vectors using the term-document matrix.

Since neither "good" nor "fool" appears in the corpus, their vectors will be zero vectors. Therefore, the cosine similarity between "good" and "fool" is 0.

(ii) PPMI (Positive Pointwise Mutual Information):

PPMI measures the association between two words based on their co-occurrence in a corpus. It is calculated as:

$$\text{PPMI}(w_1, w_2) = \max(\log(P(w_1, w_2) / (P(w_1) * P(w_2))), 0)$$

where $P(w_1, w_2)$ is the probability of co-occurrence of w_1 and w_2 , and $P(w_1)$ and $P(w_2)$ are the probabilities of occurrence of w_1 and w_2 , respectively.

To calculate the PPMI between "good" and "fool", we first need to calculate the probabilities of their occurrence and co-occurrence in the corpus. Since neither "good" nor "fool" appears in the corpus, their probabilities of occurrence

are zero. Therefore, the PPMI between "good" and "fool" is also zero.

Note: If we use add 2 smoothing to the term-document matrix to avoid zero probabilities, we would get non-zero values for the probabilities of occurrence and co-occurrence of "good" and "fool". However, since neither "good" nor

"fool" co-occur with any of the words in the corpus, their PPMI would still be zero.

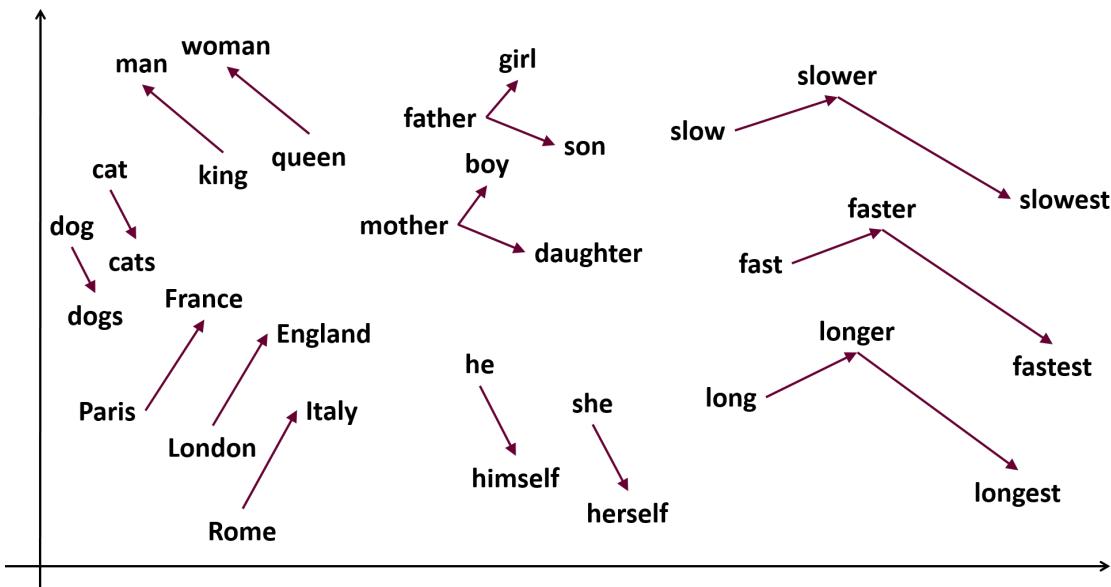
▼ Ans 4 (B) -

4(b)	What are word vectors? Illustrate with the help of a suitable diagram. Define the cosine similarity between two word vectors.	6	4
------	---	---	---

Word vectors, also known as word embeddings, are numerical representations of words in a high-dimensional space. They are created using machine learning algorithms that are trained on large collections of text to learn the relationships between words.

The basic idea behind word vectors is to represent each word as a vector of real numbers, where the values of the vector capture the meaning and context of the word in the text. Word vectors are typically high-dimensional, with hundreds or thousands of dimensions, and are learned in an unsupervised manner using techniques such as Word2Vec and GloVe.

Here is a simple diagram illustrating the concept of word vectors:



In this diagram, we see a 2D space where each word is represented as a vector. Each word is mapped to a point in this space, and similar words are placed close together. For example, "cat" and "dog" are similar words, so they are placed close together. "car" and "bus" are also similar words, but they are placed in a different direction.

The cosine similarity between two word vectors is a measure of their similarity or distance in the high-dimensional space. It is calculated by taking the cosine of the angle between the two vectors. The cosine similarity between two vectors ranges from -1 to 1, where -1 means the vectors are completely dissimilar, 0 means they are orthogonal or independent, and 1 means they are identical.

The cosine similarity between two word vectors x and y can be calculated as follows:

$$\text{cosine similarity}(x, y) = (x \cdot y) / (\|x\| * \|y\|)$$

where $x \cdot y$ is the dot product of x and y , and $\|x\|$ and $\|y\|$ are the magnitudes or lengths of the vectors x and y , respectively.

For example, consider the following word vectors in a 2D space:

$$\text{"cat"} = (1, 2)$$

$$\text{"dog"} = (2, 3)$$

$$\text{"car"} = (3, 1)$$

$$\text{"bus"} = (1, 3)$$

The cosine similarity between "cat" and "dog" can be calculated as follows:

$$\text{cosine similarity}(\text{"cat"}, \text{"dog"}) = ((1 * 2) + (2 * 3)) / (\sqrt(1^2 + 2^2) * \sqrt(2^2 + 3^2)) = 0.98$$

The high cosine similarity value indicates that "cat" and "dog" are similar words in the 2D space. Similarly, the cosine similarity between "cat" and "car" is -0.56, indicating that they are dissimilar words.

Word vectors have become an important tool in natural language processing (NLP) because they enable algorithms to capture the meaning and context of words in a more efficient and accurate way than traditional methods. Some benefits of using word vectors include:

1. Improved accuracy in NLP tasks: Word vectors can capture the semantic and syntactic relationships between words, which makes them useful in a wide range of NLP tasks such as machine translation, sentiment analysis, and named entity recognition.
2. Reduced dimensionality: Word vectors can represent words in a high-dimensional space, but they can also be used to reduce the dimensionality of text data. This is particularly useful when working with large datasets, where traditional methods can become computationally expensive.
3. Transfer learning: Word vectors can be trained on large datasets and then used as a starting point for other NLP tasks. This approach, known as transfer learning, can save time and computational resources when working with smaller datasets.

There are many techniques for generating word vectors, including Word2Vec, GloVe, and FastText. These techniques are based on neural networks and use different methods for learning word representations.

In practice, word vectors are often used as inputs to other machine learning algorithms, such as neural networks or support vector machines, in order to perform various NLP tasks. By using word vectors as inputs, these algorithms can capture the meaning and context of words in a more efficient and accurate way, which can lead to improved performance on NLP tasks.

▼ Ans 4 (C) -

4(c)	Construct the word co-occurrence matrix, window size+1, for the following text: Document: "Roses are red. Sky is blue."	6	4
------	--	---	---

4(c) Co-occurrence matrix, window size+1 Document : "Roses are red. Sky is blue".						
	Roses	are	red	Sky	is	blue
Roses	0	1	0	0	0	0
are	1	0	1	0	0	0
red	0	1	0	1	0	0
Sky	0	0	1	0	1	0
is	0	0	0	1	0	1
blue	0	0	0	0	1	0

▼ Ans 5 (A) -

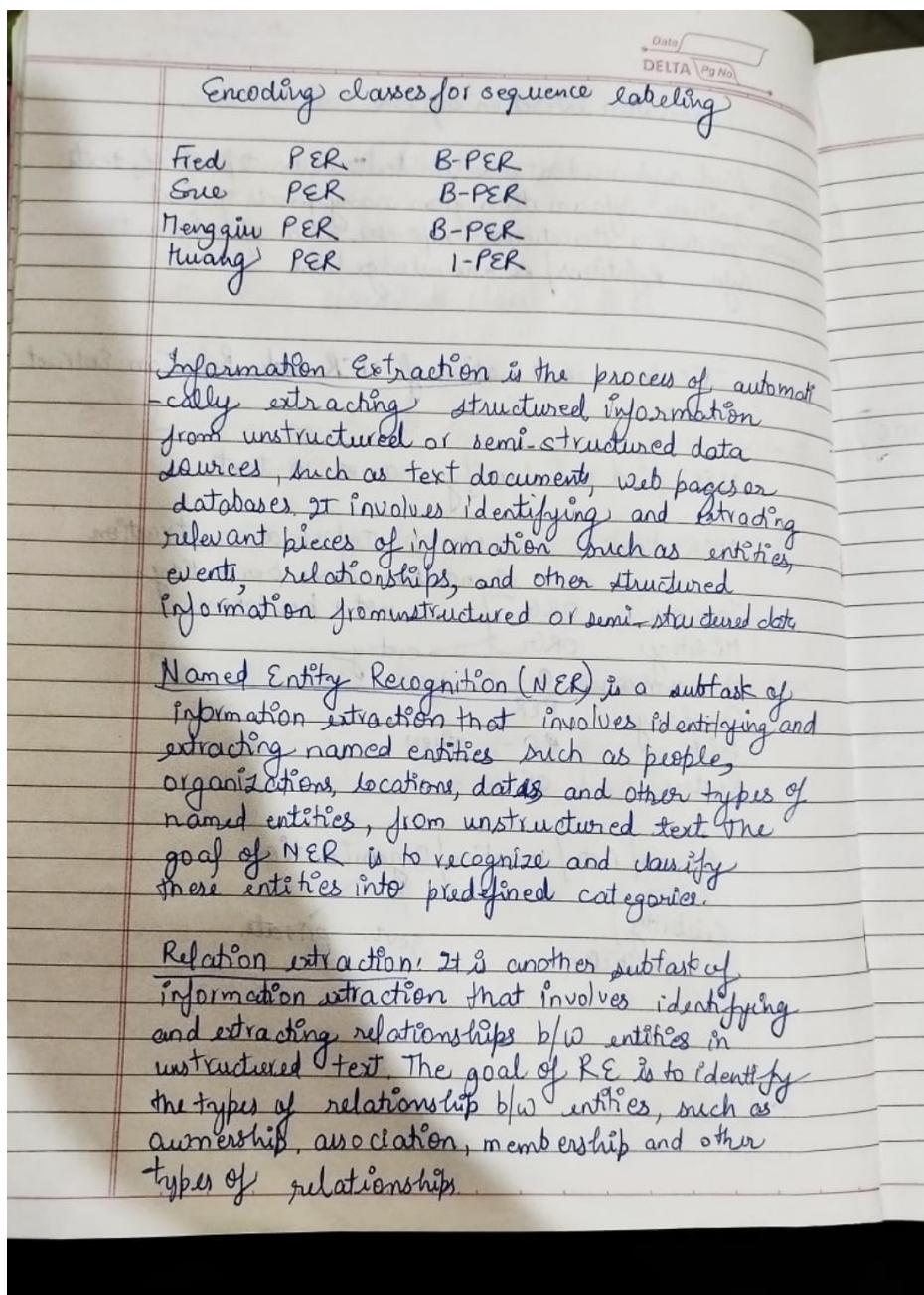
Q) Define Information Extraction, Named Entity Recognition and Relation Extraction. Why is Information Retrieval task not sufficient to perform information extraction tasks?

Information Extraction (IE) refers to the process of automatically extracting structured information from unstructured or semi-structured data sources, such as text documents or web pages. IE involves identifying and extracting specific types of information, such as named entities, relationships between entities, and events, from a given corpus of text.

Named Entity Recognition (NER) is a subtask of information extraction that involves identifying and classifying named entities (such as people, organizations, locations, and dates) in unstructured text.

Relation Extraction (RE) is another subtask of information extraction that involves identifying and extracting relationships between named entities in unstructured text. For example, RE may involve identifying the relationship between a person and a company mentioned in a news article.

Information Retrieval (IR) is the process of retrieving relevant information from a large collection of unstructured or semi-structured data sources, such as text documents or web pages. While IR can be used to search for documents that contain certain keywords or phrases, it is not sufficient to perform more complex information extraction tasks, such as named entity recognition and relation extraction. These tasks require deeper linguistic analysis and understanding of the structure and semantics of the text, which goes beyond the simple keyword matching used in IR.



Date _____
DELTA Pg No. _____

IR is a task that involves relevant information from a collection of documents based on a user's query while IR can help to find relevant documents containing certain keywords or phrases. It is not sufficient for performing information extraction tasks such as NER and RE. This is because IR does not involve extracting structured information from unstructured sources, which is primary goal of information extraction. IR relies on matching keywords and documents, while IE focuses on extracting structured information from unstructured resources.

▼ Ans 5 (B) -

Q) What are the different encoding schemes for Named Entity Recognition? Illustrate with examples.

There are different encoding schemes that can be used for Named Entity Recognition (NER) in natural language processing. Here are a few examples:

1. IOB (Inside-Outside-Beginning) Scheme:

The IOB encoding scheme is a widely used encoding scheme for NER. In this scheme, each token in a sentence is assigned one of three tags: I (Inside), O (Outside), or B (Beginning). The B tag is used to mark the first token of a named entity, the I tag is used for subsequent tokens of the entity, and the O tag is used for tokens that are not part of any entity. For example, consider the following sentence:

"Steve Jobs was the CEO of Apple."

Using the IOB scheme, the sentence would be tagged as follows:

```
"Steve B-PER
Jobs I-PER
was O
the O
CEO O
of O
Apple B-ORG
. O"
```

Here, "Steve Jobs" is a named entity of type "PER" (person), and "Apple" is a named entity of type "ORG" (organization).

2. IO (Inside-Outside) Scheme:

The IO encoding scheme is similar to the IOB scheme, but only two tags are used: I (Inside) and O (Outside). In this scheme, the first token of a named entity is assigned the I tag, and subsequent tokens are also assigned the I tag until the end of the entity is reached. For example, using the same sentence as above, the IO tagging would be:

```
"Steve I-PER
Jobs I-PER
was O
the O
```

CEO O
of O
Apple I-ORG
. O"

3. BILOU Scheme:

The BILOU encoding scheme is similar to IOB, but uses five tags: B (Beginning), I (Inside), L (Last), O (Outside), and U (Unit). The U tag is used for named entities that consist of a single token. For example, consider the sentence:

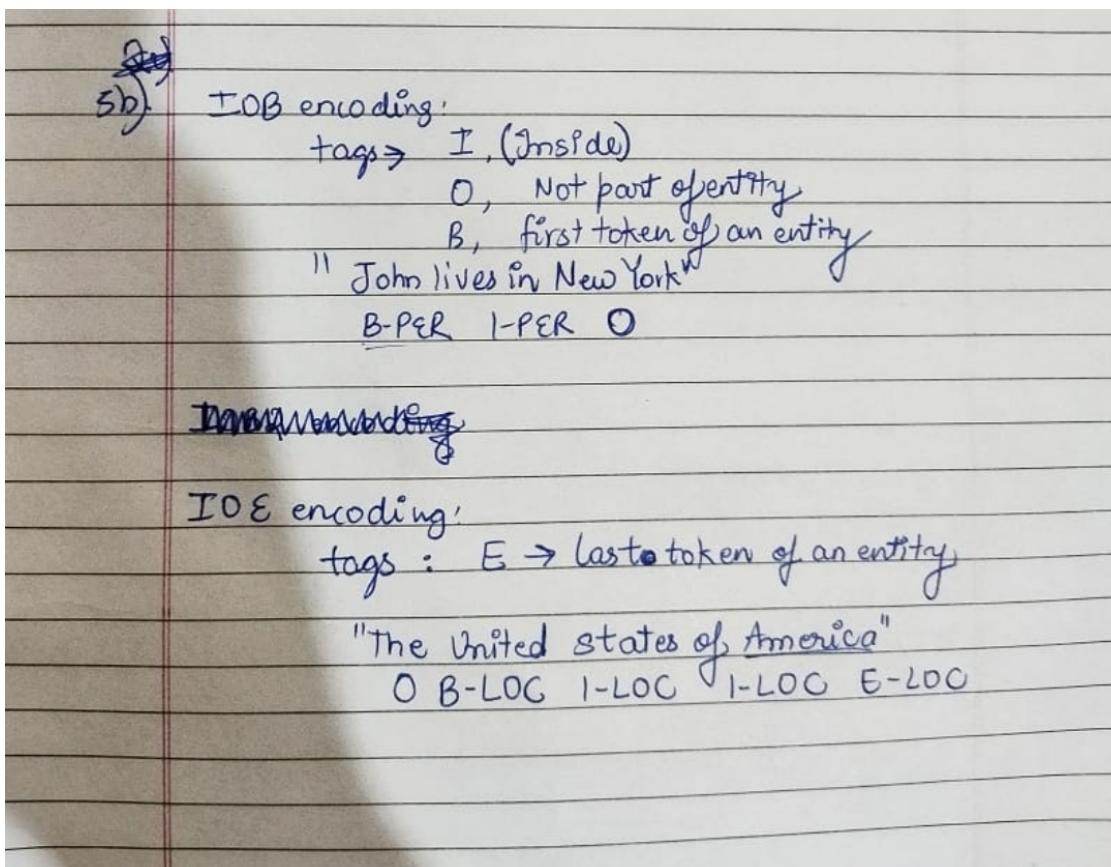
"George Washington was the first president of the United States."

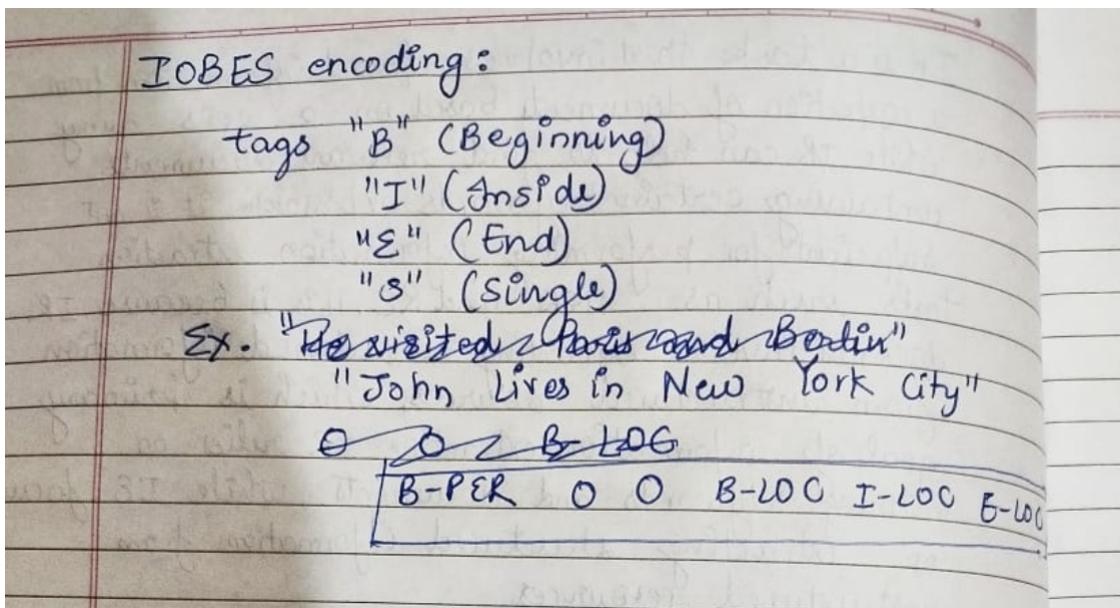
Using the BILOU scheme, the sentence would be tagged as follows:

```
"George B-PER  
Washington L-PER  
was O  
the O  
first O  
president O  
of O  
the B-LOC  
United I-LOC  
States L-LOC  
. O"
```

Here, "George Washington" is a named entity of type "PER", and "United States" is a named entity of type "LOC" (location).

These are just a few examples of the different encoding schemes used for Named Entity Recognition in NLP. The choice of encoding scheme may depend on the specific task and the requirements of the system being developed.





▼ Ans 5 (C) -

One ans -

A rule-based email extractor that can distinguish between sender and receiver email addresses:

1. Find all instances of the string "From:" in the email message.
2. For each instance of "From:", extract the email address that follows it.
3. Save the extracted email addresses in a list.
4. Find all instances of the string "To:" in the email message.
5. For each instance of "To:", extract the email address that follows it.
6. Save the extracted email addresses in a list.
7. Return the list of sender email addresses and the list of receiver email addresses.

This rule-based extractor can be used to extract email addresses from any email message. The extractor is based on the following rules:

- The sender email address is always preceded by the string "From:".
- The receiver email address is always preceded by the string "To:".
- The email address is always a string of characters that contains an "@" symbol.

The extractor is able to distinguish between sender and receiver email addresses because the "From:" and "To:" strings are always used to indicate the sender and receiver of an email message. The extractor is also able to handle email messages that contain multiple sender and receiver email addresses.

This rule-based extractor can be used for information extraction. For example, the extractor can be used to extract the email addresses of all the people who have sent you an email message. The extractor can also be used to extract the email addresses of all the people who you have sent an email message to.

Basically

1. Define Patterns for Sender and Receiver Email Addresses:
 - Sender Pattern: Look for phrases like "From:", "Sent by:", or "Sender:" followed by an email address.
 - Receiver Pattern: Look for phrases like "To:", "Cc:", "Bcc:", or "Recipient:" followed by one or multiple email addresses.
2. Implement the Rule-Based Email Extractor:

- Iterate through the lines or segments of the email.
- Check each line for the defined patterns.
- Extract the email address following the pattern for sender or receiver, based on the identified pattern.

Here's a possible approach for a rule-based email extractor that can distinguish between sender and receiver email addresses:

1. Identify the email headers: The email header is the portion of an email message that contains metadata such as the sender, recipient, subject, and date/time.
2. Extract the "From" field: The "From" field in the email header usually contains the email address of the sender. Extract this field and store it as the sender email address.
3. Extract the "To" field: The "To" field in the email header usually contains the email address of the primary recipient. However, an email message can have multiple recipients. Extract this field and store it as the receiver email address.
4. Check for other recipient fields: The email header may have other fields such as "Cc" (carbon copy) or "Bcc" (blind carbon copy) that contain additional recipient email addresses. Extract these fields and store the email addresses appropriately.
5. Handle edge cases: In some cases, the sender email address may not be explicitly mentioned in the email header, or the email may be sent to a group email address instead of an individual email address. In such cases, you can look for common patterns in the email address and make educated guesses about the sender and receiver email addresses.

youtube -

Byte Pair Encoding tokenization algorithm explained

Byte Pair Encoding is a powerful tokenization algorithm used in models like BERT for tokenization. Understanding it's working in this video

#artificialintelligence #datascience #machinelearning #nlp #tokenization

https://www.youtube.com/watch?v=sPHsY8tYtRM&ab_channel=DataScienceinyourpocket

Byte Pair Encoding tokenization algorithm

P.S. It can't merge *any* two words (we don't merge *are*)

Let's have a quick example and consider the below scenario:

dictionary	vocabulary
1 <i>love</i> -	→ <i>d, w, i, l, n, o, r, s, t, u</i>
2 <i>saw</i> -	
6 <i>never</i> -	
3 <i>idea</i> -	
2 <i>n is</i> -	

Byte Pair Encoding Tokenization in NLP

#tokenization #transformers #nlp

Tokenization is the process of representing text into smaller meaningful lexical units. Byte Pair Encoding (BPE) is a popular subword-based tokenization algorithm used by state-of-the-art NLP

https://www.youtube.com/watch?v=i0D5GbudU6c&ab_channel=TechViz-TheDataScienceGuy

Byte-pair Encoding

SUBWORD TOKENIZATION METHOD

Nlp - 1.7 - Computing Minimum Edit Distance

<https://www.coursera.org/>

https://www.youtube.com/watch?v=xWGav6hhM8A&ab_channel=AndréRibeiroMiranda

The Edit Distance Table

N	I	T	E	T	N	O	S	P	R	L	D	F	G	H	M	A	C	U	B	V	W	X	Y	Z
#	0	1	2	3	4	5	6	7	8	9	-	-	-	-	-	-	-	-	-	-	-	-	-	
#	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	-	

Hidden Markov Model | Hidden Markov Model in Pattern Recognition | HMM | Pattern Recognition | AKTU

Hello friends, in this video we are going to discuss about Hidden Markov Model i.e. HMM in Pattern Recognition.

Hope u like the video, So do SUBSCRIBE to the Channel and Press the BELL icon to get the latest video notifications.

https://www.youtube.com/watch?v=YErli89h0hs&list=PLR5USSocuZ5dA4-ivqzqNiDkICRMpudzS&index=9&ab_channel=SHUBHAMARORA

Hidden Marrov

Hidden + Complete Explai

Forward Algorithm Hidden Markov Model |Forward Algorithm in Pattern Recognition| Pattern Recognition

Hey Guys,, in this video we are going to discuss Forward Algorithm Hidden Markov Model.

So we will be discussing Forward Algorithm in Pattern Recognition.

And also forward backward algorithm hidden markov model with forward algorithm hidden markov model example.

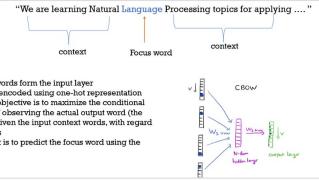
👉 https://www.youtube.com/watch?v=WDTC_SsZH70&list=PLR5USSocuZ5dA4-ivqzqNiDkICRMpudzS&index=10&ab_channel=SHUBHAMARORA



Exploring NLP—Kickstarting NLP (Step #3)

If you are new to my "Exploring NLP" series, please check my introductory article here.

👉 <https://medium.com/@deepthi.sudharsan/exploring-nlp-kickstarting-nlp-step-3-e6ea6963679d>



Understanding NLP Word Embeddings—Text Vectorization

Processing natural language text and extract useful information from the given word, a sentence using machine learning and deep learning...

👉 <https://towardsdatascience.com/understanding-nlp-word-embeddings-text-vectorization-1a23744f7223>

