**NAME :** VISHWAS P

**REG NO.** : 231057008

NLP ASSIGNMENT

**1. Distinguish the following with relevant examples: (10 Marks)**

1. **Homonyms 2. Homographs 3. Homophones 4. Heteronyms 5. Heterographs**

**1. Homonyms**

These are words that have the same spelling and pronunciation but have different meanings. For example, "bat" can refer to a flying mammal or a piece of sports equipment used in baseball.

**2. Homographs**

These are words that have the same spelling but may have different pronunciations and meanings. For example, "lead" can be pronounced as "leed" and refer to a type of metal, or pronounced as "led" and refer to the act of guiding someone.

**3. Homophones**

These are words that have the same pronunciation but different meanings, origins, or spelling. For example, "two", "too", and "to" are homophones because they sound the same but have different meanings and spellings.

**4. Heteronyms**

These are words that have the same spelling but different pronunciations and meanings. For example, "lead" can be pronounced as "leed" and refer to a type of metal, or pronounced as "led" and refer to the act of guiding someone.

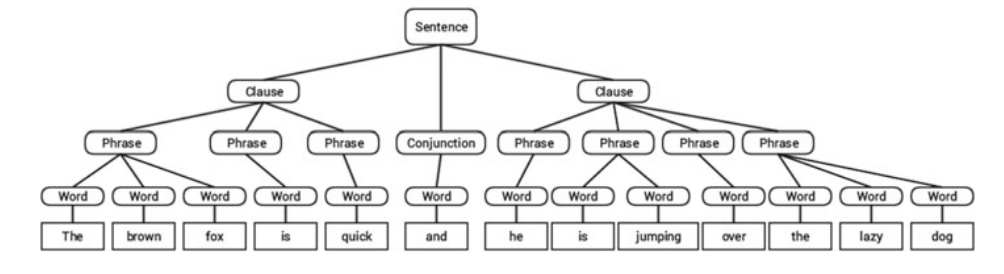
**5. Heterographs**

These are words that are spelled differently but sound the same. For example, "flour" and "flower" sound the same but have different spellings and meanings.

1. **A. Describe the hierarchical syntax of a structured sentence by annotating it with POS tags for a suitable sentence.**

In English, words usually combine together to form other constituent units . These constituents include words, phrases, clauses, and sentences. All these constituents exist together in any message and are related to each other in a hierarchical structure.

. Considering our previous hierarchy of sentence → clause → phrase → word, we can construct the hierarchical sentence tree in Figure

****

Usually, words can fall into one of the following major categories.

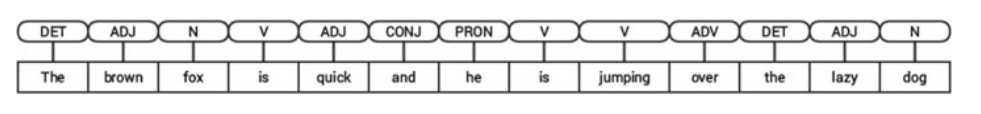
• N(oun) : This usually denotes words that depict some object or entity which may be living or nonliving. Some examples would be fox , dog , book , and so on. The POS tag symbol for nouns is N .

• V(erb) : Verbs are words that are used to describe certain actions, states, or occurrences. There are a wide variety of further subcategories, such as auxiliary, reflexive, and transitive verbs (and many more). Some typical examples of verbs would be running , jumping , read , and write . The POS tag symbol for verbs is V .

• Adj(ective) : Adjectives are words used to describe or qualify other words, typically nouns and noun phrases. The phrase beautiful flower has the noun (N) flower which is described or qualified using the adjective (ADJ) beautiful . The POS tag symbol for adjectives is ADJ .

• Adv(erb) : Adverbs usually act as modifiers for other words including nouns, adjectives, verbs, or other adverbs. The phrase very beautiful flower has the adverb (ADV) very , which modifies the adjective (ADJ) beautiful , indicating the degree to which the flower is beautiful. The POS tag symbol for adverbs is ADV .

Considering our previous example sentence ( The brown fox is quick and he is jumping over the lazy dog ) where we built the hierarchical syntax tree, if we were to annotate it using basic POS tags, it would look like below Figure



The tag DET stands for determiner , which is used to depict articles like a , an , the , and so on. The tag CONJ indicates conjunction , which is usually used to bind together clauses to form sentences. The PRON tag stands for pronoun , which represents words that are used to represent or take the place of a noun.

The tags N, V, ADJ and ADV are typical open classes and represent words belonging to an open vocabulary. Open classes are word classes that consist of an infinite set of words and commonly accept the addition of new words to the vocabulary which are invented by people. Words are usually added to open classes through processes like morphological derivation , invention based on usage, and creating compound lexemes . Some popular nouns added fairly recently include Internet and multimedia. Closed classes consist of a closed and finite set of words and do not accept new additions. Pronouns are a closed class.

**B. Write the categories and significance of different POS tags.**

Part-of-speech (POS) tags categorize words based on their syntactic roles and functions within a sentence. Here are some common POS tags along with their categories and significance:

Noun (NN):

Categories: Common nouns, proper nouns, abstract nouns, concrete nouns.

Significance: Nouns represent people, places, things, or ideas. They serve as subjects, objects, or complements in sentences.

Verb (VB):

Categories: Main verbs, auxiliary verbs.

Significance: Verbs express actions, states, or occurrences. They convey the main meaning of a sentence and indicate tense, aspect, and mood.

Adjective (JJ):

Categories: Descriptive adjectives, limiting adjectives.

Significance: Adjectives modify or describe nouns or pronouns by providing additional information about their qualities, characteristics, or attributes.

Adverb (RB):

Categories: Time adverbs, manner adverbs, place adverbs, degree adverbs.

Significance: Adverbs modify verbs, adjectives, or other adverbs by specifying how, when, where, or to what extent an action or state occurs.

Pronoun (PR):

Categories: Personal pronouns, possessive pronouns, demonstrative pronouns, relative pronouns.

Significance: Pronouns replace nouns to avoid repetition in a sentence. They refer to specific or general entities, individuals, or concepts.

Preposition (IN):

Categories: Simple prepositions, compound prepositions.

Significance: Prepositions establish relationships between nouns or pronouns and other words in a sentence, indicating location, time, direction, or possession.

Conjunction (CC):

Categories: Coordinating conjunctions, subordinating conjunctions.

Significance: Conjunctions join words, phrases, or clauses within a sentence to indicate relationships such as addition, contrast, cause and effect, or condition.

Interjection (UH):

Categories: Expressive interjections, attention-drawing interjections.

Significance: Interjections are exclamatory words or phrases that express emotions, feelings, or reactions. They are often used independently to convey surprise, joy, pain, or other sentiments.

1. **A) Compare the need for propositional Logic and First Order Logic**

Propositional Logic:

Simplicity: Propositional logic is simpler and more straightforward compared to first-order logic. It deals with propositions that can be either true or false, without considering the internal structure of sentences or the relationships between individual objects.

Suitability: Propositional logic is suitable for certain NLP tasks where the complexity of natural language expressions is low, and the focus is primarily on basic truth-functional relationships. For example, it can be used for simple information retrieval tasks or keyword-based document search.

First-Order Logic (FOL):

Expressiveness: First-order logic provides a more expressive framework for representing linguistic knowledge compared to propositional logic. It allows for the representation of relationships between objects, quantification over variables, and the expression of complex logical statements involving predicates and functions.

Applicability: First-order logic is indispensable for many advanced NLP tasks that require reasoning about structured linguistic information, such as semantic parsing, question answering, natural language inference, and knowledge representation. It enables the formalization of linguistic theories, the definition of ontologies, and the implementation of sophisticated reasoning systems.

**B) Write the FOL representation for the Natural Language Statement: "There is at least one student who failed the exam".**

**Write the Natural Language Statement for the FOL representation: ( ∃ x player(x) ∧ score(x, goal) ∧ ( ∀ y score(y, goal) → x=y))**

"There is at least one student who failed the exam" - ∃x student(x) ∧ fail(x, exam)

( ∃ x player(x) ∧ score(x, goal) ∧ ( ∀ y score(y, goal) → x=y)) - There exists a player who scored a goal, and for all players who scored a goal, there exists only one player who scored the goal.

1. **Explain the Disjunction, Grouping, and Precedence with respect to regular expression using relevant examples.**

Disjunction:

Definition: Disjunction, often denoted by the pipe symbol |, allows you to specify alternative patterns. It matches any one of the patterns separated by the pipe symbol.

Example: Consider the regular expression (cat|dog). This pattern matches either "cat" or "dog". So, it will match strings like "cat", "dog", but not "bat" or "mat".

Grouping:

Definition: Grouping in regular expressions allows you to apply quantifiers, modifiers, or alternations to a group of characters. You can create a group using parentheses ( ).

Example: Suppose you want to match "ab" followed by either "cd" or "ef". You can use grouping like this: ab(cd|ef). This pattern will match "abcd" or "abef", but not "ab", "abcf", or "abde".

Precedence:

Definition: Precedence in regular expressions determines the order in which operators are evaluated. It specifies which parts of the regular expression should be evaluated first.

Example: In the regular expression a|b\*, the Kleene star \* has higher precedence than the disjunction |. So, it will match "a", "b", "bb", "bbb", and so on. However, if you want to match either "a" or "bb", you need to use parentheses to group the disjunction: (a|b)\*.

Disjunction allows for specifying alternative patterns, grouping allows for applying operators to specific parts of the pattern, and precedence determines the order in which operators are evaluated. Understanding these concepts helps in creating more precise and effective regular expressions for pattern matching.

This idea that one operator may take precedence over another, requiring us to

sometimes use parentheses to specify what we mean, is formalized by the operator

precedence hierarchy for regular expressions. The following table gives the order of RE operator precedence, from highest precedence to lowest precedence.

Parenthesis ()

Counters \* + ? {}

Sequences and anchors the ˆmy end$

Disjunction

1. **Define the minimum edit distance between two strings.**

**Compute the edit distance (using insertion cost 1, deletion cost 1, substitution cost 1) of "explanation" to "description". Show your work (using the edit distance grid).**

Given two strings, the source string X of length n, and target string Y of length m, we’ll define D[i, j] as the edit distance between X[1..i] and Y[1.. j], i.e., the first i characters of X and the first j characters of Y. The edit distance between X and Y is thus D[n,m].

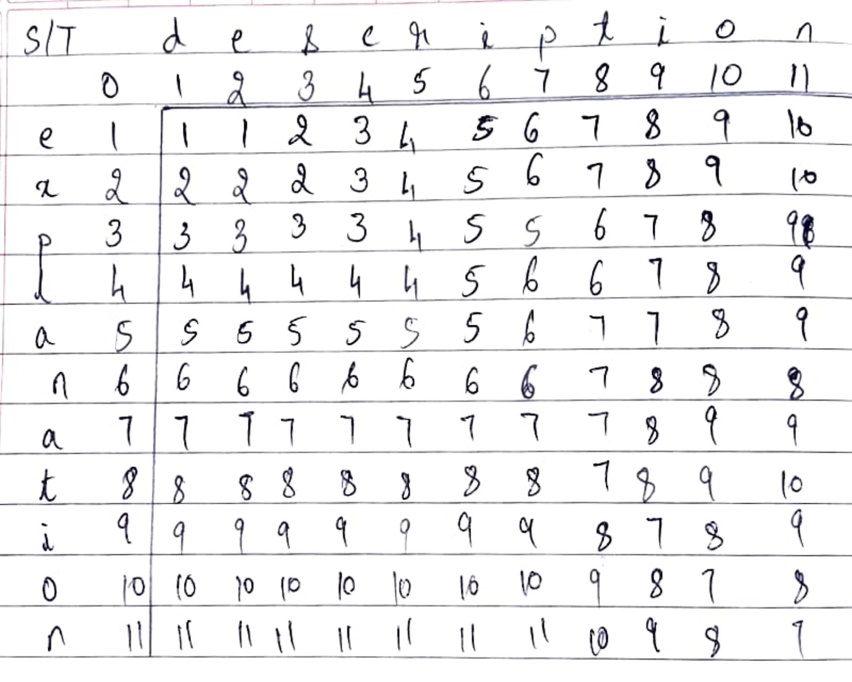
We’ll use dynamic programming to compute D[n,m] bottom up, combining solutions to subproblems. In the base case, with a source substring of length i but anempty target string, going from i characters to 0 requires i deletes. With a target substring of length j but an empty source going from 0 characters to j characters requires j inserts. Having computed D[i, j] for small i, j we then compute larger D[i, j] based on previously computed smaller values. The value of D[i, j] is computed by taking the minimum of the three possible paths through the matrix which arrive there:

D[i, j] = min

D[i−1, j] +del-cost(source[i])

D[i, j −1] +ins-cost(target[ j])

D[i−1, j −1] +sub-cost(source[i],target[ j])

****

Hence by backtrace using edit distance grid the edit distance = 7

1. **(A) Write regular expressions for the following languages.**

**1. the set of all alphabetic strings;**

**2. the set of all lower case alphabetic strings ending in a b;**

**3. the set of all strings from the alphabet a,b such that each a is immediately preceded by and immediately followed by a b;**

The set of all alphabetic strings:

Regular Expression: [a-zA-Z]+

Explanation: This regular expression matches one or more alphabetic characters (both uppercase and lowercase) in any order.

The set of all lower case alphabetic strings ending in a "b":

Regular Expression: [a-z]\*b

Explanation: This regular expression matches zero or more lowercase alphabetic characters followed by the letter "b" at the end of the string.

The set of all strings from the alphabet {a, b} such that each "a" is immediately preceded by and immediately followed by a "b":

Regular Expression: b(ab)\*b

Explanation: This regular expression matches a "b" followed by zero or more occurrences of the sequence "ab", and ends with a "b". This ensures that every "a" is immediately preceded and followed by a "b".

1. **Write regular expressions for the following languages. By “word”, we mean an alphabetic string separated from other words by whitespace, any relevant punctuation, line breaks, and so forth.**

**1. the set of all strings with two consecutive repeated words (e.g., “Humbert Humbert” and “the the” but not “the bug” or “the big bug”);**

**2. all strings that start at the beginning of the line with an integer and that end at the end of the line with a word;**

**3. all strings that have both the word grotto and the word raven in them (but not, e.g., words like grottos that merely contain the word grotto);**

**4. write a pattern that places the first word of an English sentence in a register. Deal with punctuation.**

Here are the regular expressions for the given languages:

The set of all strings with two consecutive repeated words:

Regular Expression: \b(\w+)\s+\1\b

Explanation: This regular expression matches any word (\w+) followed by one or more whitespace characters (\s+), and then the same word again (\1) surrounded by word boundaries (\b). This ensures that the word is repeated consecutively.

All strings that start at the beginning of the line with an integer and that end at the end of the line with a word:

Regular Expression: ^\d+\s.\*\b\w+$

Explanation: This regular expression matches strings that start (^) with one or more digits (\d+), followed by whitespace (\s), then any characters (.\*), and end (\b) with a word (\w+) at the end of the line ($).

All strings that have both the word "grotto" and the word "raven" in them:

Regular Expression: \bgrotto\b.\*\braven\b|\braven\b.\*\bgrotto\b

Explanation: This regular expression matches strings that contain both "grotto" and "raven" in any order. It ensures that both words are surrounded by word boundaries (\b).

Pattern that places the first word of an English sentence in a register:

Regular Expression: ^[^\w]\*(\w+)

Explanation: This regular expression matches the first word of an English sentence by ignoring leading non-word characters (^[^\w]\*) and capturing the first word ((\w+)). It handles punctuation by skipping over non-word characters at the beginning of the sentence.

1. **Provide an explanation of each term: tokenization, lemmatization, sentence segmentation, and edit distance, highlighting their significance in natural language processing?**

Tokenization is the process of breaking down a text into smaller units, such as words or phrases, known as tokens. This is a fundamental step in natural language processing (NLP) as it enables computers to understand and analyze the text by treating each token as a separate entity. Tokenization is essential for tasks such as text analysis, information retrieval, and machine translation.

Lemmatization is the process of determining the base or dictionary form of a word, known as its lemma, considering the context of the word in the sentence. It involves reducing inflected or derived words to their base form to normalize variations and facilitate analysis. Lemmatization is crucial in NLP for tasks like information retrieval, text mining, and sentiment analysis, where understanding the semantic meaning of words is essential.

Sentence segmentation, also known as sentence boundary detection, is the process of identifying and separating individual sentences within a text. It involves detecting punctuation marks, such as periods, question marks, and exclamation points, that indicate the end of one sentence and the beginning of another. Sentence segmentation is vital in NLP for tasks such as text summarization, machine translation, and sentiment analysis, where analyzing sentences separately is necessary.

Edit distance, also known as Levenshtein distance, is a metric used to measure the similarity between two strings based on the minimum number of operations (insertions, deletions, or substitutions) required to transform one string into the other. It quantifies the degree of difference or similarity between strings and is often used in tasks such as spelling correction, string matching, and approximate string matching in NLP. Edit distance is significant in NLP as it provides a way to compare and evaluate strings, enabling various text processing and analysis tasks.

1. **Write Basic components in FOL.**

The basic components in FOL are as follows:

• Objects : These are specific entities or terms with individual

unique identities like people, animals, and so on.

• Relations : These are also known as predicates and usually hold

among objects or sets of objects and express some form of

relationship or connection, like is\_man , is\_brother , is\_mortal .

Relations typically correspond to verbs.

• Functions : These are a subset of relations where there is always

only one output value or object for some given input. Examples

would be height , weight , age\_of .

• Properties : These are specific attributes of objects that help in

distinguishing them from other objects, like round, huge, and so on.

• Connectives : These are the logical connectives that are similar to

the ones in PL, which include not (¬), and (∧), or (∨), implies

(→), and iff (if and only if ↔).

• Quantifiers : These include two types of quantifiers: universal

(∀), which stands for “for all” or “all,” and existential (∃), which

stands for “there exists” or “exists.” They are used for quantifying

entities in a logical or mathematical expression.

• Constant symbols : These are used to represent concrete entities or

objects in the world. Examples would be John , King , Red , and 7 .

• Variable symbols : These are used to represent variables like x , y ,

and z .

• Function symbols : These are used to map functions to outcomes.

Examples would be, age\_of(John) = 25 or color\_of(Tree) =

Green.

• Predicate symbols : These map specific entities and a relation or

function between them to a truth value based on the outcome.

Examples would be color(sky, blue) = True.

1. **Explain the roles of phrases and clauses in sentence structure and semantic interpretation. Provide examples of each type of phrase and clause and discuss their significance in natural language processing.**

* Phrases and clauses serve as building blocks of sentence structure, organizing words into meaningful units.
* Phrases encompass various categories such as noun phrases (e.g., "the brown fox"), verb phrases (e.g., "is jumping over the lazy dog"), adjective phrases (e.g., "too quick"), adverb phrases (e.g., "pretty soon"), and prepositional phrases (e.g., "up the stairs").
* Clauses, containing a subject and a predicate, are classified into declarative (e.g., "The brown fox is quick"), imperative (e.g., "Please do not talk in class"), relative (e.g., "he wanted a soda"), interrogative (e.g., "Did you get my mail?"), and exclamative (e.g., "What an amazing race!") types.
* Phrases and clauses provide syntactic structure to sentences, facilitating comprehension and interpretation.
* Understanding phrases and clauses aids in syntactic parsing, a crucial step in natural language processing tasks like information extraction and sentiment analysis.
* Phrases and clauses contribute to semantic interpretation by conveying relationships between entities, actions, and attributes in text.
* In machine translation systems, recognizing and translating phrases and clauses accurately enhances the quality of translated output.
* Sentiment analysis algorithms utilize phrases and clauses to identify sentiment-bearing expressions and gauge the overall sentiment of text.
* Phrases and clauses play a role in text summarization by capturing key information and relationships within sentences.

Overall, mastering the analysis of phrases and clauses is essential for robust natural language processing systems, enabling accurate understanding and generation of human language.

1. **Explain the minimum edit distance algorithm and its applications in natural language processing.**

Explanation of the Minimum Edit Distance Algorithm:

The minimum edit distance algorithm is used to find the shortest path, or sequence of edits, required to transform one string into another.

It involves dynamic programming, where a table is used to store intermediate results and optimize the computation.

The algorithm defines the edit distance between two strings as the minimum number of edit operations (insertion, deletion, substitution) required to transform one string into the other.

Applications in Natural Language Processing (NLP):

Spelling Correction: The minimum edit distance algorithm is used in spelling correction systems to suggest the most likely correct word based on the similarity between the misspelled word and words in a dictionary.

Speech Recognition: It is used to compute the word error rate by aligning recognized words with the actual words spoken.

Machine Translation: Alignment between sentences in parallel corpora is computed using minimum edit distance to improve the accuracy of translation models.

Named Entity Recognition: It can be used to identify similar named entities by measuring the edit distance between them.

These applications demonstrate the importance of the minimum edit distance algorithm in various NLP tasks, where determining the similarity or dissimilarity between strings is crucial.

String Alignment:

The minimum edit distance algorithm can also be used to perform string alignment, where the algorithm computes the optimal alignment between two strings by identifying corresponding characters or words.

String alignment is essential in tasks such as DNA sequence alignment, where it helps identify similarities and differences between genetic sequences.

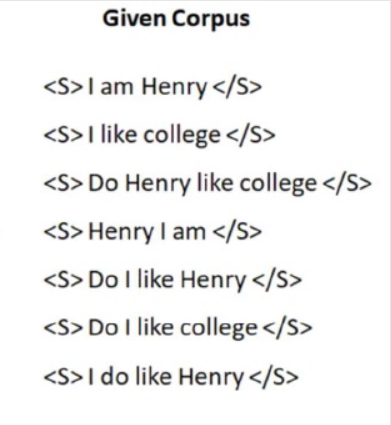
Error Detection and Correction:

In addition to spelling correction, the minimum edit distance algorithm is utilized in error detection and correction systems for identifying and rectifying typographical errors in text.

It plays a vital role in autocorrect features of word processing software and messaging applications, improving the accuracy of text input.

These additional points highlight further applications of the minimum edit distance algorithm in NLP and related fields, showcasing its versatility and importance in various computational tasks involving text processing and analysis. \

1. **What is n-gram? Illustrate the various types and the need of using n-grams in NLP. Estimate the Bi-gram probability? What is the most probable next word predicted by the model for the following word sequence? <S> I ?**



n-gram: An n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application. The n-grams typically are collected from a text or speech corpus.





Types of n-gram:

Unigram (1-gram): No history is used.

Bi-gram (2-gram): One word history.

Tri-gram (3-gram): Two words history.

Four-gram (4-gram): Three words history.

Five-gram (5-gram): Four words history is used. ‘

N-grams are a powerful tool for natural language processing (NLP) that can help with a variety of tasks. They can help capture the probability distribution of words in a language, which can be useful for: machine translation, speech recognition, auto-completion, text classification and clustering, and feature engineering.

|  |  |
| --- | --- |
| **Word** | **Frequency** |
| <S> | 7 |
| </S> | 7 |
| I | 6 |
| am | 2 |
| henry | 5 |
| like | 5 |
| college | 3 |
| do | 4 |

Next word prediction probability Wi-1 = I.

|  |  |
| --- | --- |
| **Next word** | **Probability of next word =** |
| P(</S> | I) | 0/6 |
| P(am | I) | 2/6 |
| P(Henry | I) | 0/6 |
| P(like | I) | 3/6 |
| P(college | I) | 0/6 |
| P(do | I) | 1/6 |

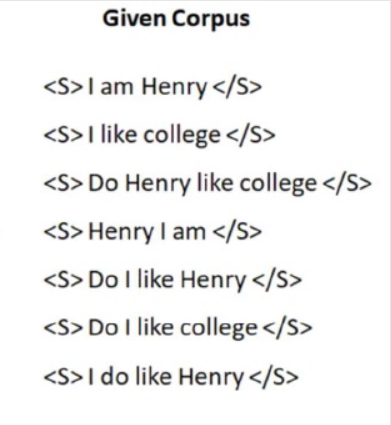
Probable next word is **like.**

1. **Identify the sentence that gets a higher probability with tri-gram model. (Use the same corpus as in Q1)**

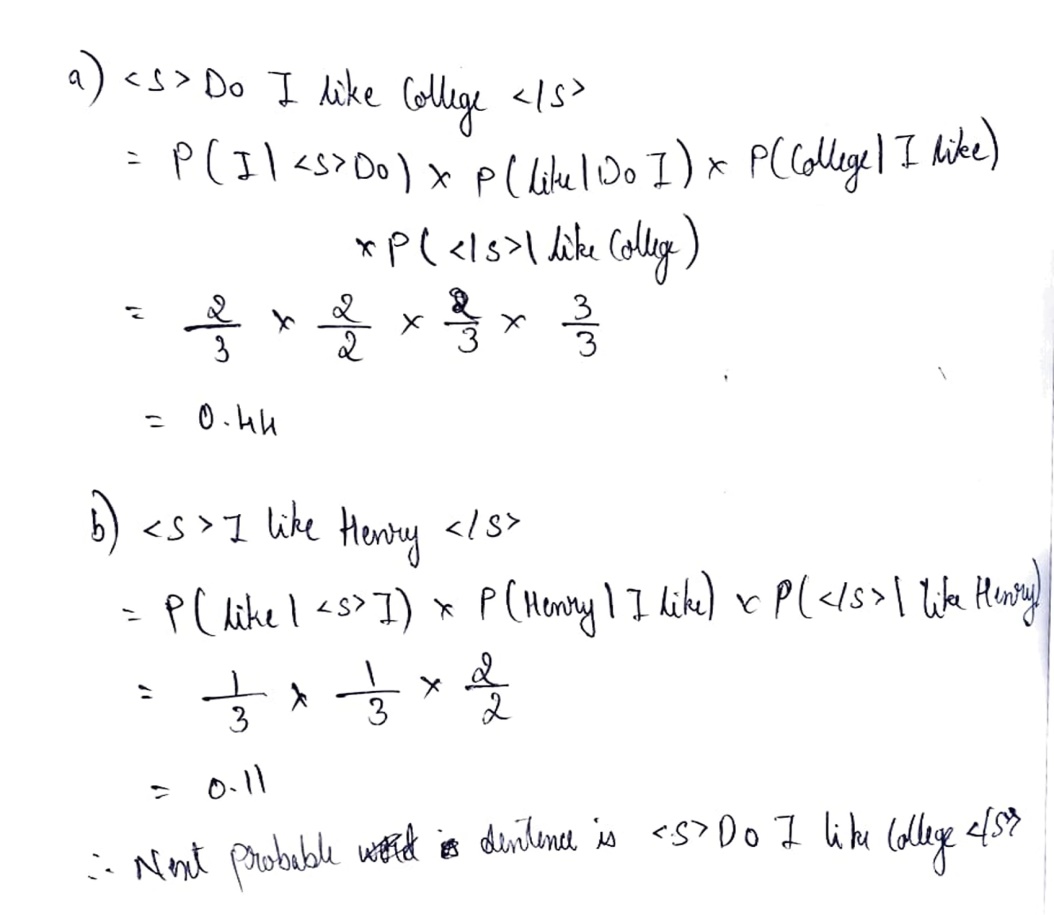
**a) < S> Do I like college </S>**

**b) < S> I like Henry </S>**

**Comment if there is a need for Laplace smoothing required here.**



|  |  |
| --- | --- |
| **Word** | **Frequency** |
| <S> | 7 |
| </S> | 7 |
| I | 6 |
| am | 2 |
| henry | 5 |
| like | 5 |
| college | 3 |
| do | 4 |

****

Laplace smoothing prevents the model from assigning zero probabilities to features not present in the training data, ensuring that the model can make predictions for previously unseen words. Since we don’t have any zero probabilities here, we don’t need to use Laplace Smoothing.

If Laplace Smoothing needs to be done, then we add 1 to the numerator of all the probabilities and add the total number of words in the vocabulary to the denominator.

1. **Classify the different relations the words can have with each other in the context of NLP with suitable examples.**

Relations that words can have with each other are:

1. **Synonyms:** Have the same meaning in some or all contexts.

**Examples:** Filbert/hazelnut**,** Couch/sofa**,** Big/large, Automobile/car, Vomit/throw up, Water/H₂O

Note that there are probably no examples of perfect synonym.

* Even if many aspects of meaning are identical
* Still may differ based on politeness, slang, register, genre, etc.

For example: Water / H20: "H2O" in a surfing guide?

Big / large: my big sister != my large sister

1. **Similarity:** Words with similar meaning. Not synonyms, but sharing some element of meaning.

**Example:** car-bicycle, Cow-horse

Words similarity can be measured as:



1. **Word Relatedness:** Words can be related in any way, perhaps via a semantic word or field.

**Example:** coffee, tea – similar

Coffee, cup – related not similar

1. **Semantic field:** Words that:

* cover a particular semantic domain
* bear structured relations with each other.

**Example:**

**Hospitals** - surgeon, scalpel, nurse, anaesthetic, hospital

**Restaurants** - waiter, menu, plate, food, menu, chef

**Houses** - door, roof, kitchen, family, bed

1. **Antonyms:** Senses that are opposite to only one feature of the meaning. Otherwise, they are similar.

**Example:** Dark/light, Hot/cold, Short/long, fast/slow, up/down, rise/fall, in/out

More formally antonyms can

* Define a binary opposition or be at opposite ends of a scale - long/short, fast/slow
* Be reverses - rise/fall, up/down

1. **Connotation (Sentiment):**

* Words have affective meanings
* Positive connotations (happy)
* Negative connotations (sad)
* Connotations can be subtle:
* Positive connotation: copy, replica, reproduction
* Negative connotation: fake, knockoff, forgery
* Evaluation (sentiment!)
* Positive evaluation (great, love)
* Negative evaluation (terrible, hate)

Words seem to vary along 3 affective dimensions:

* **Valence:** the pleasantness of the stimulus
* **Arousal:** the intensity of emotion provoked by the stimulus
* **Dominance:** the degree of control exerted by the stimulus



1. **Define connotation of a word. Interpret the different ideas of defining the meaning of a word.**

**Connotation (Sentiment):**

* Words have affective meanings
* Positive connotations (happy)
* Negative connotations (sad)
* Connotations can be subtle:
* Positive connotation: copy, replica, reproduction
* Negative connotation: fake, knockoff, forgery
* Evaluation (sentiment!)
* Positive evaluation (great, love)
* Negative evaluation (terrible, hate)

Words seem to vary along 3 affective dimensions:

* **Valence:** the pleasantness of the stimulus
* **Arousal:** the intensity of emotion provoked by the stimulus
* **Dominance:** the degree of control exerted by the stimulus



Different types of ideas defining the meaning of a word are:

1. **Synonyms:** Have the same meaning in some or all contexts.

Filbert/hazelnut**,** Couch/sofa**,** Big/large, Automobile/car, Vomit/throw up, Water/H₂O

1. **Semantic field:** Words that cover a particular semantic domainand bear structured relations with each other.

**Hospitals** - surgeon, scalpel, nurse, anaesthetic, hospital

**Restaurants** - waiter, menu, plate, food, menu, chef

1. **Word Relatedness:** Words can be related in any way, perhaps via a semantic word or field.

Coffee, tea – similar

Coffee, cup – related not similar

1. **Similarity:** Words with similar meaning. Not synonyms, but sharing some element of meaning.

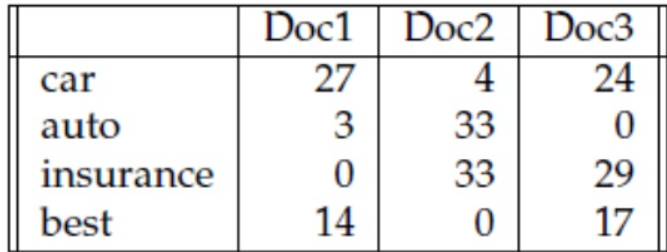
car-bicycle, Cow-horse

1. **Antonyms:** Senses that are opposite to only one feature of the meaning. Otherwise, they are similar.

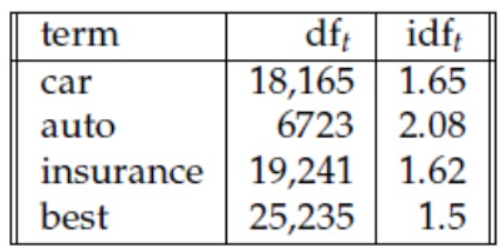
Dark/light, Hot/cold, Short/long, fast/slow, up/down, rise/fall, in/out

1. **What is tf-idf? Discuss the significance of tf-idf algorithm in NLP.**

**Consider the table of term frequencies for 3 documents denoted Doc1, Doc2, and Doc3 in the Figure below**

****

**Compute the tf-idf weights for the terms car, auto, insurance, best, for each document, using the idf values from Figure below**

****

The co-occurrence matrices above represent each cell by frequencies, either of words with documents or words with other words. But raw frequency is not the best measure of association between words. Raw frequency is very skewed and not very discriminative. If we want to know what kinds of contexts are shared by cherry and strawberry but not by digital and information, we’re not going to get good discrimination from words like the, it, or they, which occur frequently with all sorts of words and aren’t informative about any particular word.

It’s a bit of a paradox. Words that occur nearby frequently (maybe pie nearby cherry) are more important than words that only appear once or twice. Yet words that are too frequent—ubiquitous, like the or good— are unimportant.

The tf-idf weighting (the ‘-’ here is a hyphen, not a minus sign) is the product of two terms, each term capturing one of these two intuitions:

The first term frequency is the term frequency the frequency of the word t in the document d. We can just use the raw count as the term frequency:

tft, d  = count(t, d)

More commonly we squash the raw frequency a bit, by using the log10 of the frequency instead. The intuition is that a word appearing 100 times in a document doesn’t make that word 100 times more likely to be relevant to the meaning of the document. Because we can’t take the log of 0, we normally add 1 to the count:

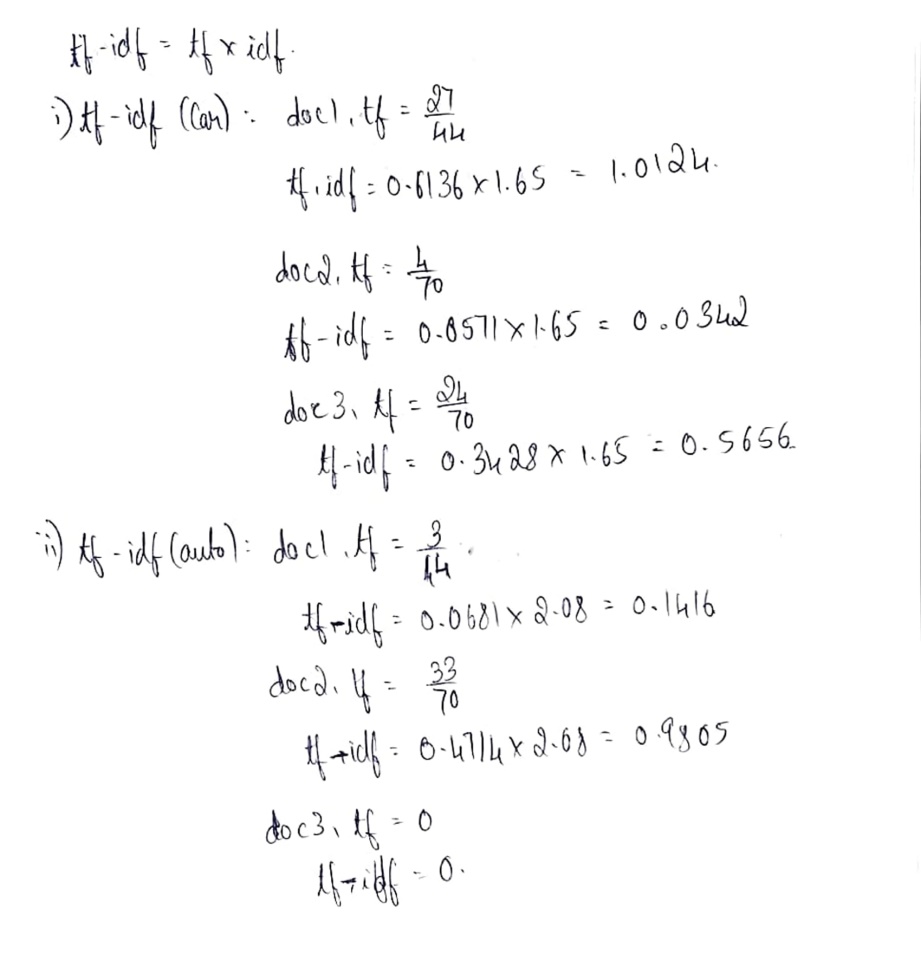
tft, d = log10(count(t, d)+1)

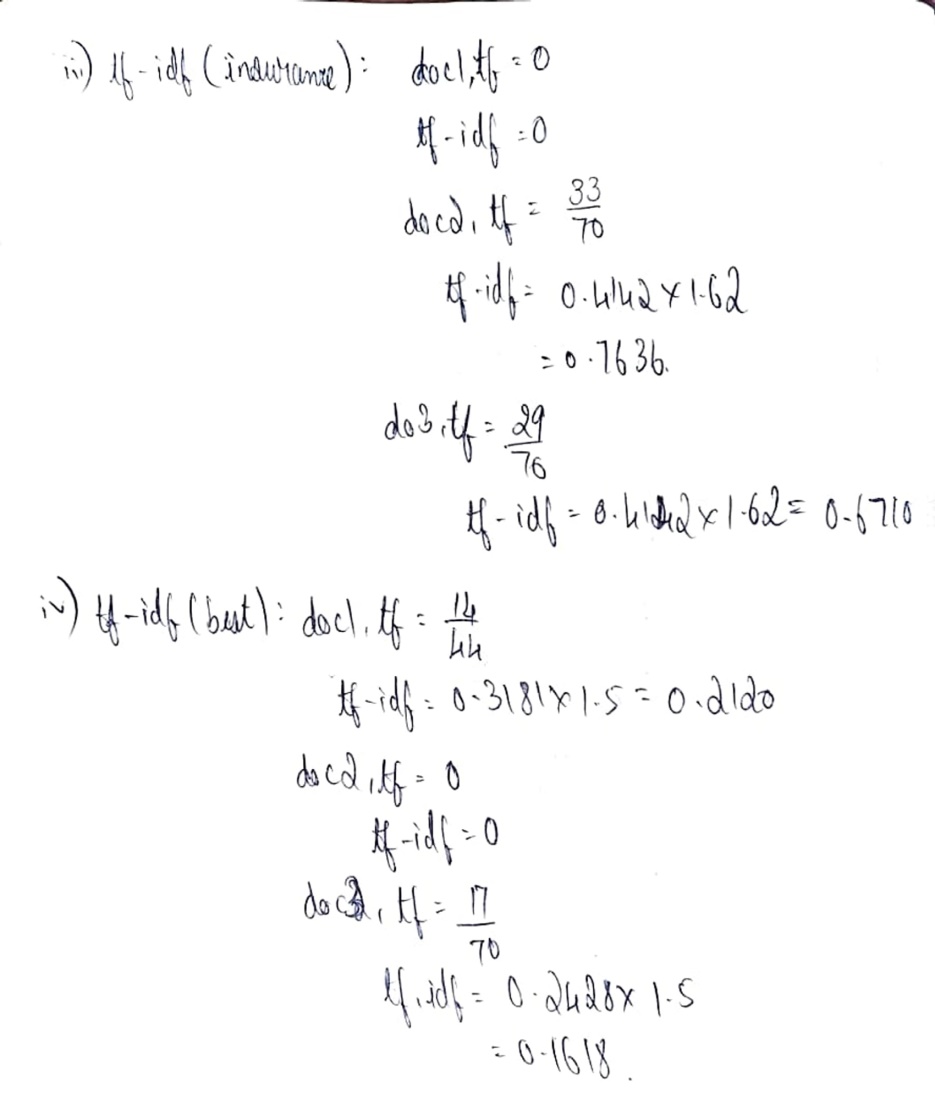
The second factor in tf-idf is used to give a higher weight to words that occur only in a few documents. Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection; terms that occur document frequently across the entire collection aren’t as helpful. The document frequency dft of a term t is the number of documents it occurs in. Document frequency is not the same as the collection frequency of a term, which is the total number of times the word appears in the whole collection in any document.

The idf is defined using the fraction N/df­t, where N is the total number of documents in the collection, and dft is the number of documents in which term t occurs. The fewer documents in which a term occurs, the higher this weight. The lowest weight of 1 is assigned to terms that occur in all the documents.

Because of the large number of documents in many collections, this measure too is usually squashed with a log function. The resulting definition for inverse document frequency (idf) is thus

The tf-idf weighted value wt;d for word t in document d thus combines term frequency tft,d with idf





1. **Explain Cosine Similarity. Suppose you have the following two 4-dimensional word vectors for two words w1 and w2 respectively:**

**w1 = (0.2, 0.1, 0.3, 0.4) and w2 = (0.3, 0, 0.2, 0.5)**

**What is the cosine similarity between w1 and w2? Are the words w1 and w2 similar or dissimilar?**

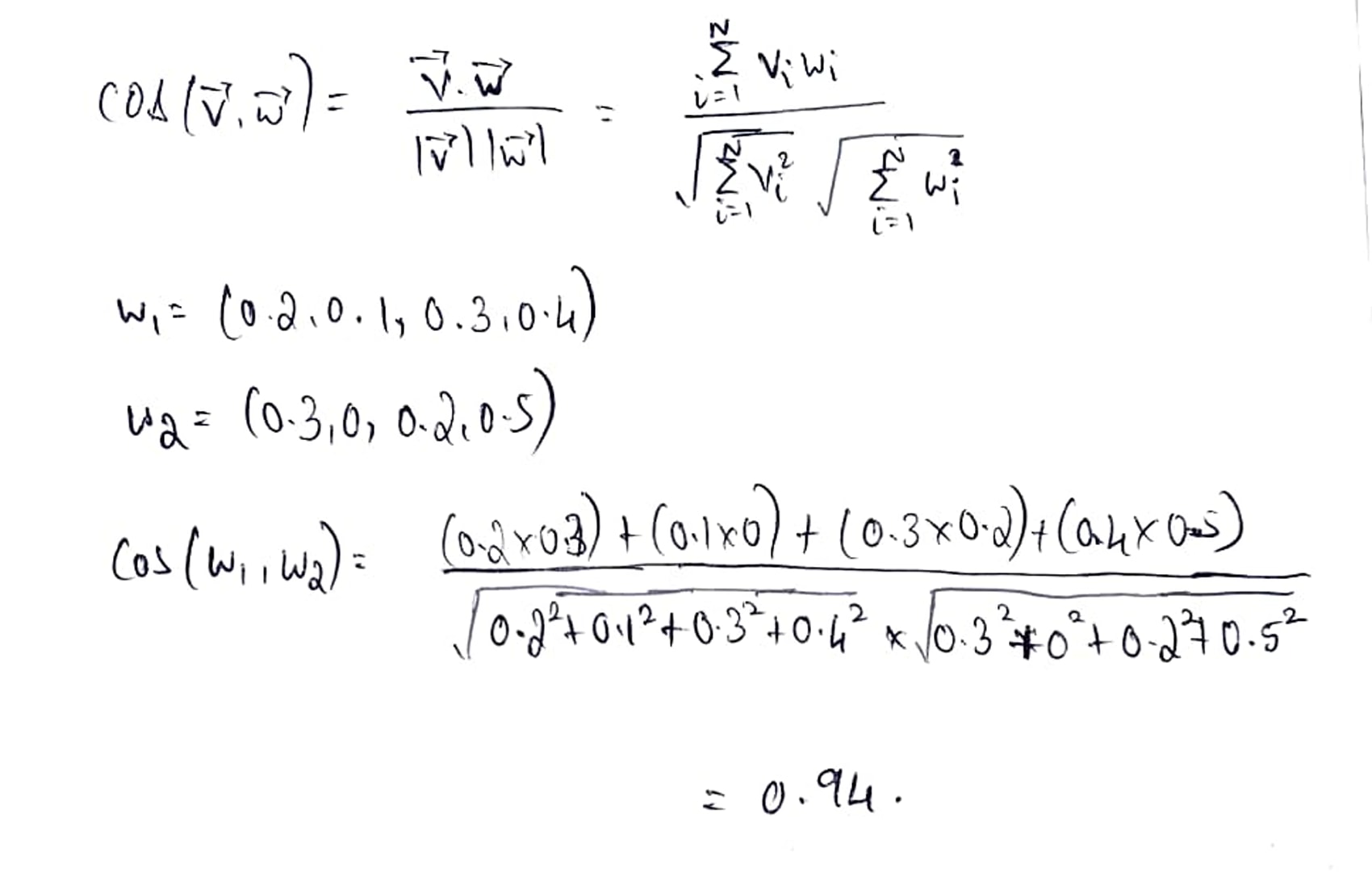
To measure similarity between two words, we need a metric that compares their representations.

By far, the most common similarity metric is the cosine of the angle between the two vectors.

The cosine, like most measures for vector similarity used in NLP, is based on

the dot product operator from Linear Algebra, also called the inner product, in which we multiply the vectors element wise and add up to get a single scalar value. The dot-product acts as a similarity metric because it tend to be high just when the two vectors have large values in the same dimension. Alternatively, vectors that have zeros in different dimension orthogonal vectors will have a dot product of zero representing their strong dissimilarity. The raw dot product, however, has a problem as a similarity metric. It favours long vectors. So we modify the dot product to normalize for the vector lengthby dividing the dot product by the length of each of the two vectors. And this normalized dot product turns out to be the same as the cosine of the angle between the two vectors. This is based on the geometric definition of the dot product as the product of Euclidean magnitudes of the two vectors and the coastline of the angle between them.

The cosine value ranges from one for vectors pointing in the same direction with an angle of zero between them to minus one for vectors pointing in opposite directions, But since raw frequency values are non-negative, the cosine for these vectors ranges from 0 to 1.

****

Yes, the 2 words are similar.

1. **We are given the following corpus:**

**<S> I am Sam </S>**

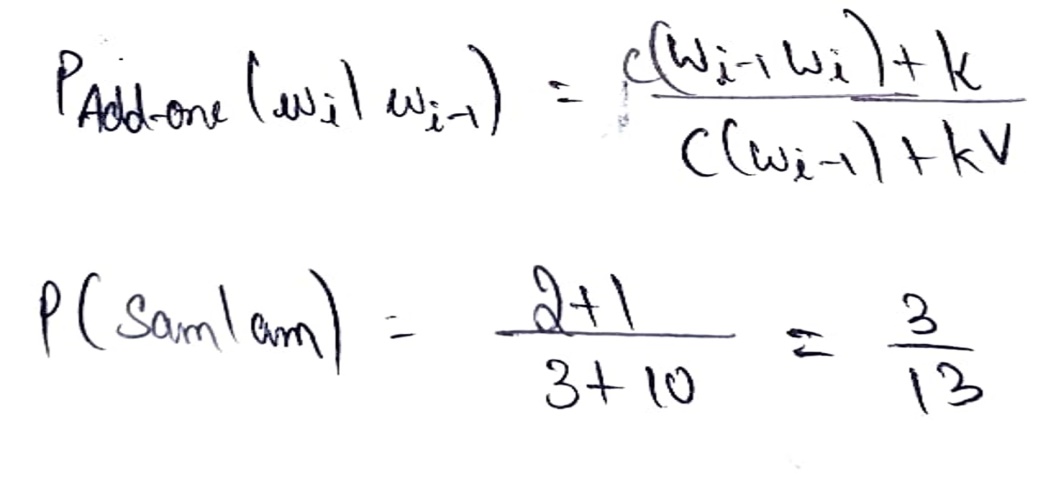
**<S> Sam I am </S>**

**<S> I am Sam </S>**

**<S> I do not like green eggs and Sam </S>**

**Using a bigram language model with add-one smoothing, what is P(Sam|am)? Include <S> and </S> in your counts just like any other token.**

|  |  |
| --- | --- |
| **Word** | **Frequency** |
| I | 4 |
| am | 3 |
| Sam | 4 |
| Do | 1 |
| Not | 1 |
| Like | 1 |
| Green | 1 |
| Eggs | 1 |

****

1. **What is perplexity in the context of evaluating language models, and how is it calculated? How can the concept of perplexity be applied to evaluating the performance of a language model in a task like digit recognition in English?**

The perplexity (sometimes called PPL for short) of a language model on a test set is the inverse probability of the test set, normalized by the number of words. For a test set

W = w1, w2, . . .wN:

We can use the chain rule to expand the probability of W:

The perplexity of a test setW depends on which language model we use. Here’s the perplexity of W with a unigram language model (just the geometric mean of the unigram probabilities):

The perplexity of W computed with a bigram language model is still a geometric mean, but now of the bigram probabilities:

Note that because of the inverse in Eq. 1, the higher the conditional probability of the word sequence, the lower the perplexity. Thus, minimizing perplexity is equivalent to maximizing the test set probability according to the language model. What we generally use for word sequence in Eq. 1 or Eq.2 is the entire sequence of words in some test set. Since this sequence will cross many sentence boundaries, we need to include the begin- and end-sentence markers <s> and </s> in the probability computation. We also need to include the end-of-sentence marker </s> (but not the beginning-of-sentence marker <s>) in the total count of word tokens N.

The task of recognizing the digits in English (zero, one, two,..., nine), given that (both in some training set and in some test set) each of the 10 digits occurs with equal probability P=1/10 . The perplexity of this mini-language is in fact 10. To see that, imagine a test string of digits of length N, and assume that in the training set all the digits occurred with equal probability.

By Eq. 1, the perplexity will be

1. **How does the perplexity of language models vary with respect to the level of information provided by n-grams, as illustrated by the perplexity values obtained for unigram, bigram, and trigram grammars on a 1.5 million word test set from the Wall Street Journal? What does a lower perplexity indicate about the predictive capabilities of a language model?**

****

As we see above, the more information the n-gram gives us about the word sequence, the higher the probability the n-gram will assign to the string. A trigram model is less surprised than a unigram model because it has a better idea of what words might come next, and so it assigns them a higher probability. And the higher the probability, the lower the perplexity (perplexity is related inversely to the likelihood of the test sequence according to the model). So a lower perplexity can tell us that a language model is a better predictor of the words in the test set. Note that in computing perplexities, the n-gram model P must be constructed without any knowledge of the test set or any prior knowledge of the vocabulary of the test set. Any kind of knowledge of the test set can cause the perplexity to be artificially low. The perplexity of two language models is only comparable if they use identical vocabularies. An (intrinsic) improvement in perplexity does not guarantee an (extrinsic) improvement in the performance of a language processing task like speech recognition.

1. **How does vector semantics contribute to representing word meaning in natural language processing (NLP)? How are word embedding derived, and how do they enable NLP applications to capture word similarity and semantic relationships?**

Vector semantics is the standard way to represent word meaning in NLP, helping semantics

us model many of the aspects of word meaning. The roots of the model lie in the 1950s when two big ideas converged: Osgood’s 1957 idea mentioned above to use a point in three-dimensional space to represent the connotation of a word, and the proposal by linguists like Joos (1950), Harris (1954), and Firth (1957) to define the meaning of a word by its distribution in language use, meaning its neighboring words or grammatical environments. Their idea was that two words that occur in very similar distributions (whose neighboring words are similar) have similar meanings.

For example, suppose you didn’t know the meaning of the word ongchoi (a recent borrowing from Cantonese) but you see it in the following contexts:

1. Ongchoi is delicious sauteed with garlic.
2. Ongchoi is superb over rice.
3. ...ongchoi leaves with salty sauces...

And suppose that you had seen many of these context words in other contexts:

1. ...spinach sauteed with garlic over rice...
2. ...chard stems and leaves are delicious...
3. ...collard greens and other salty leafy greens

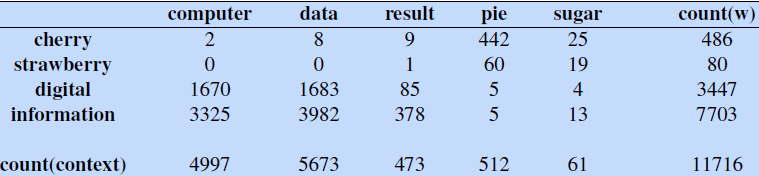
The fact that ongchoi occurs with words like rice and garlic and delicious and salty, as do words like spinach, chard, and collard greens might suggest that ongchoi is a leafy green similar to these other leafy greens.1 We can do the same thing computationally by just counting words in the context of ongchoi.

The idea of vector semantics is to represent a word as a point in a multidimensional semantic space that is derived from the distributions of embeddings word neighbors. Vectors for representing words are called embeddings (although the term is sometimes more strictly applied only to dense vectors like word2vec, rather than sparse tf-idf or PPMI vectors. The word “embedding” derives from its mathematical sense as a mapping from one space or structure to another, although the meaning has shifted.



Figure shows a visualization of embeddings learned for sentiment analysis, showing the location of selected words projected down from 60-dimensional space into a two dimensional space. Notice the distinct regions containing positive words, negative words, and neutral function words.

1. **Describe the intuition behind PPMI and how it addresses the limitations of PMI, particularly regarding the handling of negative values and bias toward infrequent events. Compute the Positive Pointwise Mutual Information (PPMI) value for the word pair "information" and "data" based on the provided co-occurrence counts in the table below.**



PPMI draws on the intuition that the best way to weigh the association between two words is to ask how much more the two words co-occur in our corpus than we would have a priori expected them to appear by chance. Pointwise mutual information is one of the most important con- pointwise mutual information cepts in NLP. It is a measure of how often two events x and y occur, compared with what we would expect if they were independent:

The pointwise mutual information between a target word w and a context word c is then defined as:

The numerator tells us how often we observed the two words together (assuming we compute probability by using the MLE). The denominator tells us how often we would expect the two words to co-occur assuming they each occurred independently; recall that the probability of two independent events both occurring is just the product of the probabilities of the two events. Thus, the ratio gives us an estimate of how much more the two words co-occur than we expect by chance. PMI is a useful tool whenever we need to find words that are strongly associated. PMI values range from negative to positive infinity. But negative PMI values (which imply things are co-occurring less often than we would expect by chance) tend to be unreliable unless our corpora are enormous. To distinguish whether two words whose individual probability is each 10􀀀6 occur together less often than chance, we would need to be certain that the probability of the two occurring together is significantly less than 10􀀀12, and this kind of granularity would require an enormous corpus. Furthermore it’s not clear whether it’s even possible to evaluate such scores of ‘unrelatedness’ with human judgments. For this reason it is more common PPMI to use Positive PMI (called PPMI) which replaces all negative PMI values with zero.

We can compute PPMI(information,data) as follows:

