studocu

NLP-3 - nlp notes all chapters

Natural Language Processing (Savitribai Phule Pune University)

Scan to open on Studocu

Studocu is not sponsored or endorsed by any college or university
Downloaded by Kumud (knowkumud@gmail.com)

Unit III -Language Modelling

Probabilistic language modeling, Markov models, Generative models of language, Log-Liner Models, Graph-based Models

N-gram models: Simple n-gram models, Estimation parameters and smoothing, Evaluating language models, Word Embeddings/ Vector Semantics: Bag-of-words, TFIDF, word2vec, doc2vec, Contextualized representations (BERT)

Topic Modelling: Latent Dirichlet Allocation (LDA), Latent Semantic Analysis, Non-Negative Matrix Factorization

#Exemplar/Case Studies Study of language modelling for Indian languages

What is Probabilistic language modelling

Probabilistic language modeling refers to the use of statistical methods and probability theory to model and generate natural language text. It involves estimating the likelihood of word sequences or sentences based on the patterns observed in a given training dataset.

In a probabilistic language model, the goal is to assign a probability to a sequence of words. This probability reflects the likelihood of that particular sequence occurring in the language. The model is trained on a large corpus of text, where it learns the statistical patterns and relationships between words.

One common approach to probabilistic language modeling is n-gram models. An n-gram is a contiguous sequence of n words in a sentence. For example, a trigram model considers three consecutive words. The model estimates the probability of a word given its preceding n-1 words, which is known as the conditional probability.

Probabilistic language models can be used for various natural language processing tasks, including speech recognition, machine translation, text generation, and text classification. They enable the generation of coherent and contextually appropriate sentences based on the patterns learned from the training data.

Advanced techniques, such as recurrent neural networks (RNNs) and transformer models, have further improved probabilistic language modeling. These models capture long-term dependencies and complex patterns in language by incorporating contextual information from a broader context, resulting in more accurate and fluent language generation.

Let's consider a simple example of probabilistic language modeling using a unigram model. In this model, we estimate the probability of each word independently, without considering the context.

Suppose we have a training corpus with the following sentences:

1. "I like to eat pizza."
2. "She enjoys reading books."
3. "He plays soccer in the park."

This document is available free of charge on studocu

Downloaded by Kumud (knowkumud@gmail.com)

Based on this data, we can calculate the probabilities of individual words. Let's calculate the probabilities of a few words in this corpus:

1. P("I") = 1/15 (since "I" appears once out of 15 total words)
2. P("pizza") = 1/15
3. P("eat") = 1/15
4. P("books") = 1/15
5. P("soccer") = 1/15
6. P("park") = 1/15

Now, let's say we want to generate a new sentence using this model. We start with a random word, and then for each subsequent word, we randomly select a word based on its probability in the corpus. For example, we start with "I," and using the probabilities calculated above, we randomly select the next word:

"I" -> "eat"

Then, we use the same process to select the subsequent words:

"I eat pizza."

Certainly! Let's consider an example of a bigram language model, where the probability of a word is estimated based on the preceding word.

Suppose we have a small corpus of text consisting of the following sentences:

1. I love cats.

2. Cats are cute.

3. Dogs are loyal.

To build a bigram language model, we need to calculate the probabilities of each word given the preceding word. Let's assume we want to estimate the probability of the word "cats" given the preceding word "love." We can calculate it as follows:

P(cats | love) = Count(love cats) / Count(love)

From the given corpus, we see that "love cats" occurs once, and "love" occurs once. Therefore:

Downloaded by Kumud (knowkumud@gmail.com)

P(cats | love) = 1 / 1 = 1

Similarly, we can estimate other bigram probabilities:

P(are | cats) = Count(cats are) / Count(cats)

P(cute | are) = Count(are cute) / Count(are)

P(are | dogs) = Count(dogs are) / Count(dogs)

Suppose we want to generate the next word after the word "cats." We can calculate the probabilities of the possible next words and choose the most likely one. Let's assume we have:

P(are | cats) = 1/1 = 1

P(loyal | cats) = 0/1 = 0

Based on these probabilities, the bigram model would predict that "cats are" is a more likely continuation compared to "cats loyal."

The bigram model allows us to estimate the probabilities of words based on the preceding word, capturing the statistical patterns and dependencies in the language. While bigram models have limitations in capturing long-range dependencies, they can still provide useful insights into word probabilities and aid in various NLP tasks such as text generation and next-word prediction.

By using probabilistic language modeling, we can generate sentences that have a similar distribution of words as observed in the training data. However, it's important to note that this simple unigram model does not capture any contextual dependencies or grammatical structure. Advanced models like n-gram models, RNNs, or transformer models are used to capture more complex language patterns and generate more coherent and contextually appropriate text.

Explain Markov models

Markov models, also known as Markov chains or Markov processes, are probabilistic models that capture the concept of "memorylessness" in sequential data. They are widely used in various fields, including language modeling, speech recognition, finance, and weather prediction.

In a Markov model, the probability of an event occurring depends only on its immediate preceding state. It assumes that the current state in a sequence is conditionally independent of all the preceding states given the immediately preceding state. This property is known as the Markov property or Markov assumption.

Mathematically, a Markov model can be represented by a set of states and transition probabilities. The states represent different configurations or observations in a system, and the transition

This document is available free of charge on studocu

Downloaded by Kumud (knowkumud@gmail.com)

probabilities indicate the likelihood of moving from one state to another. Each state has associated probabilities of transitioning to other states.

The key components of a Markov model are:

1. State Space: It represents the set of possible states in the system. For example, in a weather prediction model, the state space could include sunny, cloudy, and rainy.
2. Transition Matrix: It is a square matrix that defines the transition probabilities between states. The (i, j) entry of the matrix represents the probability of transitioning from state i to state j. The sum of transition probabilities from any state must be equal to 1.
3. Initial State Distribution: It specifies the initial probabilities of starting in each state. It represents the probability distribution over the states at the beginning of the sequence.

Markov models can be classified based on the number of preceding states considered for the transition probabilities. Some common types include:

1. First-order Markov Model: Also known as a unigram model, it only considers the immediate preceding state to estimate the transition probabilities.
2. Higher-order Markov Model: It considers a fixed number of preceding states (n-1) to estimate the transition probabilities. For example, a trigram model considers the two preceding states.

Markov models have various applications. In language modeling, they can be used to generate coherent text by modeling the conditional probabilities of words given the preceding words. In speech recognition, they can model the transitions between phonemes or words. They are also used in predicting stock prices, analyzing biological sequences, and simulating complex systems.

Note that while Markov models are useful for capturing short-term dependencies, they may not capture long-term dependencies or complex language structures. More advanced models like hidden Markov models (HMMs) and recurrent neural networks (RNNs) have been developed to address these limitations and capture more sophisticated patterns in sequential data.

Explain Hidden Markov Model with diagram

A Hidden Markov Model (HMM) is a statistical model used to describe the probabilistic relationships between a sequence of observable events and an underlying sequence of hidden states. It is commonly used in various applications, including speech recognition, natural language processing, and bioinformatics.

Here's a step-by-step explanation of a Hidden Markov Model with a diagram:

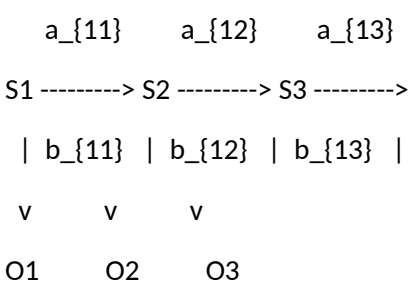
1. States: An HMM consists of a set of hidden states, denoted by S = {S1, S2, ..., Sn}. Each state represents an underlying condition or category that generates the observed sequence.
2. Transitions: There are transition probabilities between states, indicating the probability of moving from one state to another. These probabilities are represented by the transition matrix, A = {a_{ij}}, where a_{ij} represents the probability of transitioning from state Si to state Sj.
3. Observations: For each hidden state, there is an associated probability distribution over observable events or symbols. These symbols can be words, phonemes, or any other

Downloaded by Kumud (knowkumud@gmail.com)

observable units. The probability distribution for each state is represented by the emission matrix, B = {b_{ij}}, where b_{ij} represents the probability of observing symbol j given the hidden state Si.

4. Initial State Distribution: The initial state distribution, π = {π_i}, represents the probabilities of starting the sequence in each hidden state. π_i represents the probability of starting in state Si.

To visualize the structure of an HMM, we can use a directed graph representation:



In the diagram:

- The nodes represent the hidden states (S1, S2, S3, ...).
- The arrows represent the transition probabilities between the states (a_{ij}).
- The edges from the states to the observable symbols (O1, O2, O3) represent the emission probabilities (b_{ij}).

The initial state distribution π is not depicted in the diagram, but it represents the starting probabilities for each hidden state.

This diagram visually represents the structure of a simple Hidden Markov Model with three hidden states and three observable symbols. In practice, HMMs can have more complex structures with additional states and symbols.

The power of HMMs lies in their ability to model sequences of observable events while accounting for the underlying hidden states that generate those events.

Types of Probabilistic language modeling

There are several types of probabilistic language modeling techniques that have been developed to capture the statistical patterns and generate natural language text. Here are a few notable types:

1. N-gram Models: N-gram models are based on the n-gram concept, which represents a contiguous sequence of n words in a sentence. These models estimate the probability of a word given its preceding n-1 words, known as the conditional probability. N-gram models are simple and widely used, with unigram, bigram, and trigram models being the most common. However, they have limitations in capturing long-range dependencies and contextual information.

This document is available free of charge on studocu

Downloaded by Kumud (knowkumud@gmail.com)

2. Hidden Markov Models (HMMs): HMMs are probabilistic models that combine observed states (words) and hidden states to model sequences. In language modeling, HMMs use observed words to estimate the hidden states, which represent the underlying linguistic structure. HMMs are particularly useful in tasks such as speech recognition and part-of-speech tagging.

3. Recurrent Neural Networks (RNNs): RNNs are neural network architectures that excel at modeling sequential data. They can capture long-term dependencies by maintaining an internal memory, allowing information from previous words to influence the prediction of the current word. The use of recurrent connections enables RNNs to process sequences of arbitrary length. Variants of RNNs, such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), address the vanishing gradient problem and enhance the ability to retain long-term information.

4. Transformer Models: Transformer models, introduced by the "Attention is All You Need" paper, have revolutionized probabilistic language modeling. They rely on self-attention mechanisms to capture relationships between words in a sequence, removing the need for recurrent connections. Transformers are highly parallelizable and achieve state-of-the-art performance in various language tasks, including language modeling, machine translation, and text generation. The popular models such as GPT (Generative Pre-trained Transformer) and BERT (Bidirectional Encoder Representations from Transformers) are based on the transformer architecture.

These are just a few examples of probabilistic language modeling techniques. Other approaches, such as Maximum Entropy Models, Conditional Random Fields (CRFs), and Gaussian Mixture Models (GMMs), have also been applied in specific contexts. Each technique has its own strengths and weaknesses, and the choice of model depends on the specific task and requirements at hand.

Explain Generative models of language

Generative models of language in NLP refer to models that aim to understand and generate natural language text by capturing the underlying distribution of language. These models focus on learning the probability distribution of a sequence of words or characters in a language and utilize that knowledge to generate new text.

Here are a few popular generative models used in NLP:

1. n-gram models: n-gram models estimate the probability of a word based on its context, considering the n-1 preceding words. These models assume that the probability of a word depends only on the previous n-1 words and do not capture long-range dependencies.

2. Hidden Markov Models (HMMs): HMMs model the probability distribution of sequences of observed words based on a set of hidden states. They assume that each observed word is generated from a corresponding hidden state and utilize transition probabilities between states and emission probabilities for observed words.

3. Latent Dirichlet Allocation (LDA): LDA is a generative probabilistic model used for topic modeling. It assumes that documents are generated from a mixture of topics and words are generated from these topics. LDA aims to discover the latent topics and their distribution in a corpus.

Downloaded by Kumud (knowkumud@gmail.com)

4. Variational Autoencoders (VAEs): VAEs are generative models that learn a low-dimensional representation, called the latent space, of input text data. They consist of an encoder network that maps the input text to the latent space and a decoder network that reconstructs the input text from the latent space.

5. Generative Adversarial Networks (GANs): GANs are a class of generative models that involve a generator network and a discriminator network. The generator network learns to generate realistic text samples, while the discriminator network learns to distinguish between real and generated text. The two networks are trained in an adversarial manner, improving the quality of generated text over time.

Generative models of language play a crucial role in various NLP tasks, including text generation, machine translation, summarization, and dialogue systems. They enable the modeling and generation of coherent and contextually relevant text by learning the underlying patterns and structures of language. These models continue to be an active area of research in NLP, with ongoing advancements in deep learning and probabilistic modeling techniques.

Log-linear analysis

Log-linear analysis is a technique used in statistics to examine the relationship between more than two categorical variables. The technique is used for both hypothesis testing and model building. In both these uses, models are tested to find the most parsimonious (i.e., least complex) model that best accounts for the variance in the observed frequencies. (A Pearson's chi-square test could be used instead of log-linear analysis, but that technique only allows for two of the variables to be compared at a time

Fitting criterion

Log-linear analysis uses a likelihood ratio statistic that has an approximate chi-square distribution when the sample size is large:

Fitting criterion [edit]

Log-linear analysis uses a likelihood ratio statistic X^2 that has an approximate chi-square distribution when the sample size is large:[2]

$$\mathrm{X}^2 = 2\sum O_{ij} \ln \frac{O_{ij}}{E_{ij}},$$

where

$\ln$ = natural logarithm;
O_{ij} = observed frequency in cell$_{ij}$ (i = row and j = column);
E_{ij} = expected frequency in cell$_{ij}$.
X^2 = the deviance for the model.[3]

Assumptions

There are three assumptions in log-linear analysis:

1. The observations are independent and random;

2. Observed frequencies are normally distributed about expected frequencies over repeated samples. This is a good approximation if both (a) the expected frequencies are greater than or equal to 5 for 80% or more of the categories and (b) all expected frequencies are greater than 1. Violations to this assumption result in a large reduction in power. Suggested solutions to this violation are:

This document is available free of charge on studocu

Downloaded by Kumud (knowkumud@gmail.com)

delete a variable, combine levels of one variable (e.g., put males and females together), or collect more data.

3. The logarithm of the expected value of the response variable is a linear combination of the explanatory variables. This assumption is so fundamental that it is rarely mentioned, but like most linearity assumptions, it is rarely exact and often simply made to obtain a tractable model.

Additionally, data should always be categorical. Continuous data can first be converted to categorical data, with some loss of information. With both continuous and categorical data, it would be best to use logistic regression. (Any data that is analysed with log-linear analysis can also be analysed with logistic regression. The technique chosen depends on the research questions.)

Variables

In log-linear analysis there is no clear distinction between what variables are the independent or dependent variables. The variables are treated the same. However, often the theoretical background of the variables will lead the variables to be interpreted as either the independent or dependent variables.[1]

Models

The goal of log-linear analysis is to determine which model components are necessary to retain in order to best account for the data. Model components are the number of main effects and interactions in the model. For example, if we examine the relationship between three variables—variable A, variable B, and variable C—there are seven model components in the saturated model. The three main effects (A, B, C), the three two-way interactions (AB, AC, BC), and the one three-way interaction (ABC) gives the seven model components.

The log-linear models can be thought of to be on a continuum with the two extremes being the simplest model and the saturated model. The simplest model is the model where all the expected frequencies are equal. This is true when the variables are not related. The saturated model is the model that includes all the model components. This model will always explain the data the best, but it is the least parsimonious as everything is included. In this model, observed frequencies equal expected frequencies, therefore in the likelihood ratio chi-square statistic, the ratio and . This results in the likelihood ratio chi-square statistic being equal to 0, which is the best model fit.[2] Other possible models are the conditional equiprobability model and the mutual dependence model.[1]

Each log-linear model can be represented as a log-linear equation. For example, with the three variables (*A*, *B*, *C*) the saturated model has the following log-linear equation:[1]

$$\ln(F_{ijk}) = \lambda + \lambda_i^A + \lambda_j^B + \lambda_k^C + \lambda_{ij}^{AB} + \lambda_{ik}^{AC} + \lambda_{jk}^{BC} + \lambda_{ijk}^{ABC},$$

where

F_{ijk} = expected frequency in cell$_{ijk}$;
λ = the relative weight of each variable.

Hierarchical model[edit]

Log-linear analysis models can be hierarchical or nonhierarchical. Hierarchical models are the most common. These models contain all the lower order interactions and main effects of the interaction to be examined.[1]

Downloaded by Kumud (knowkumud@gmail.com)

Graphical model[edit]

A log-linear model is graphical if, whenever the model contains all two-factor terms generated by a higher-order interaction, the model also contains the higher-order interaction.[4] As a direct-consequence, graphical models are hierarchical. Moreover, being completely determined by its two-factor terms, a graphical model can be represented by an undirected graph, where the vertices represent the variables and the edges represent the two-factor terms included in the model.

Decomposable model[edit]

A log-linear model is decomposable if it is graphical and if the corresponding graph is chordal.

Model fit[edit]

The model fits well when the residuals (i.e., observed-expected) are close to 0, that is the closer the observed frequencies are to the expected frequencies the better the model fit. If the likelihood ratio chi-square statistic is non-significant, then the model fits well (i.e., calculated expected frequencies are close to observed frequencies). If the likelihood ratio chi-square statistic is significant, then the model does not fit well (i.e., calculated expected frequencies are not close to observed frequencies).

Backward elimination is used to determine which of the model components are necessary to retain in order to best account for the data. Log-linear analysis starts with the saturated model and the highest order interactions are removed until the model no longer accurately fits the data. Specifically, at each stage, after the removal of the highest ordered interaction, the likelihood ratio chi-square statistic is computed to measure how well the model is fitting the data. The highest ordered interactions are no longer removed when the likelihood ratio chi-square statistic becomes significant.[2]

Comparing models[edit]

When two models are nested, models can also be compared using a chi-square difference test. The chi-square difference test is computed by subtracting the likelihood ratio chi-square statistics for the two models being compared. This value is then compared to the chi-square critical value at their difference in degrees of freedom. If the chi-square difference is smaller than the chi-square critical value, the new model fits the data significantly better and is the preferred model. Else, if the chi-square difference is larger than the critical value, the less parsimonious model is preferred.

What is n-gram model

An n-gram model is a statistical language model commonly used in natural language processing (NLP) and computational linguistics. It models the probability distribution of sequences of n words (or characters) in a given text corpus. The n-gram model assumes that the probability of a word depends only on the previous (n-1) words in the sequence, and it estimates these probabilities based on the observed frequencies in the training data.

Here's how an n-gram model works:

1. Corpus and Training: The model is trained on a large corpus of text data, which can be sentences, paragraphs, or entire documents. The corpus is tokenized into individual words or characters.

This document is available free of charge on studocu

Downloaded by Kumud (knowkumud@gmail.com)

2. n-gram Generation: The corpus is then processed to generate n-grams, which are contiguous sequences of n words (or characters). For example, in a 3-gram model (also known as a trigram model), the sentence "I love OpenAI" would generate the following trigrams: "I love Open", "love OpenAI".

3. Counting Frequencies: The frequencies of each n-gram occurrence are counted in the training corpus. This involves counting the number of times each n-gram appears in the text.

4. Probability Estimation: Based on the n-gram frequencies, the model estimates the probabilities of the next word in the sequence given the (n-1) preceding words. These probabilities are typically calculated using relative frequencies, such as maximum likelihood estimation or add-one smoothing (Laplace smoothing).

5. Sequence Generation: The trained n-gram model can be used to generate new sequences of words. Starting with an initial n-gram, the model selects the next word based on the probabilities estimated from the training data. This process is repeated to generate a sequence of desired length.

The choice of n determines the context size considered by the model. Larger values of n capture more context but may suffer from data sparsity issues, especially when n is large and the training data is limited.

N-gram models are widely used in various NLP applications, including language modeling, text generation, spell checking, speech recognition, and machine translation. Despite their simplicity, n-gram models can capture some level of local syntactic and semantic patterns in the text and provide a foundation for more advanced language models. However, they have limitations in handling long-range dependencies and understanding the overall context of a sentence or document.

n-gram model with example

Sure! Let's illustrate the n-gram model with an example. Consider the following sentence:

"I like to play soccer with my friends."

We will use a 2-gram model (bigram model) to demonstrate the concept. Here's how it works:

1. Corpus and Training: Assume we have a corpus of text that includes this sentence along with other sentences. We tokenize the corpus into individual words:

"I", "like", "to", "play", "soccer", "with", "my", "friends."

2. n-gram Generation: Now, we generate the 2-grams (bigrams) by sliding a window of two words across the sentence:

"I like", "like to", "to play", "play soccer", "soccer with", "with my", "my friends."

3. Counting Frequencies: Next, we count the frequencies of each bigram in the corpus. Let's assume the bigram "like to" appears 10 times in the training corpus.

4. Probability Estimation: Based on the bigram frequencies, we estimate the probabilities of the next word given the previous word. In this case, we calculate the probability of "to" given "like" as:

P("to" | "like") = Count("like to") / Count("like")

Downloaded by Kumud (knowkumud@gmail.com)

Assume the count of "like to" is 10 and the count of "like" is 50. So, P("to" | "like") = 10/50 = 0.2.

5. Sequence Generation: Using the trained bigram model, we can generate new sequences of words. Starting with an initial word, we select the next word based on the probability distribution. Let's start with the seed word "I":

"I like" -> Select the next word based on the probability distribution of words that follow "I like". Assume the possible next words and their probabilities are as follows:

- "to": 0.5
- "play": 0.3
- "eat": 0.2

Based on these probabilities, we randomly select the next word. Let's say we choose "to".

"I like to" -> Select the next word based on the probability distribution of words that follow "like to". Assume the possible next words and their probabilities are as follows:

- "play": 0.7
- "eat": 0.2
- "dance": 0.1

Again, we randomly select the next word. Let's say we choose "play".

"I like to play" -> ...

We continue this process until we reach the desired length or a predefined end condition.

This demonstrates how an n-gram model, specifically the bigram model, can generate new sequences of words based on the probabilities learned from the training data.

Simple (Unsmoothed) N-gram in NLP

Overview

N-grams are **continuous sequences** of words or symbols or tokens in a document and are defined as the **neighboring sequences of items** in a document. They are used most importantly in tasks dealing with text data in NLP (Natural Language Processing).

N-gram models are widely used in statistical natural language processing, speech recognition, phonemes and sequences of phonemes, machine translation and predictive text input, and many others for which the modeling inputs are n-gram distributions.

Scope

- In this article, we will learn what n-grams are and the system of classification of n-grams.
- We will look at a few examples of n-grams and learn a step-by-step implementation of n-grams in Python.

This document is available free of charge on studocu

Downloaded by Kumud (knowkumud@gmail.com)

- We will then learn the concept of smoothing and see how to do smoothing on n-grams in Python.
- We will also learn how to use n-grams as features in machine learning models and evaluate the performance of the model with multiple experiments.
- We will then briefly look at the concept of smoothing and why we need to smooth the text corpus in certain instances.

What are n-grams?

N-grams are defined as the **contiguous sequence of n items** that can be extracted from a given sample of text or speech. The items can be letters, words, or base pairs, according to the application. The N-grams typically are collected from a **text or speech corpus** (Usually a corpus of long text dataset).

- N-grams can also be seen as a **set of co-occurring words** within a given window computed by basically moving the window some k words forward (k can be from 1 or more than 1).
- The co-occurring words are called "n-grams," and "n" is a number saying how long a string of words we have considered in the construction of n-grams.
- Unigrams are single words, bigrams are two words, trigrams are three words, 4-grams are four words, 5-grams are five words, etc.

Applications of N-grams: N-grams of texts are extensively used in the n-gram model in NLP, text mining, and natural language processing tasks.

- For example, when **developing language models** in natural language processing, n-grams are used to develop not just unigram models but also bigram and trigram models.
- Tech companies like Google and Microsoft have developed **web-scale n-gram models** that can be used in a variety of NLP-related tasks such as spelling correction, word breaking, and text summarization.
- One other major usage of n-grams is for developing features for supervised Machine Learning models such as SVMs, MaxEnt models, Naive Bayes, etc. The main idea is to use tokens such as bigrams (and trigrams and advanced n-grams) in the feature space instead of just unigrams.

Importance of order of words in text & NLP

- In general, the **order of the words** that are used in the natural language of the text is not random. If we consider the English language, we can use the words "the green apple" but not "apple green the" in a sentence.
- The **relationships** between words in the flow of the text are also very complex. N-grams are tooted as a relatively simple way of capturing some of these relationships between the words.
- We can capture quite a bit of information by just looking at which words tend to show up next to each other more often. By constructing the n-grams, we can construct the co-occurring words and use them in modeling relationships and other applications.
- **Example:** The general idea with n-grams is that we can look at each pair or triple or set of four or more words that occur next to each other in a sufficiently large corpus.
    - When we construct these pairs of words, we are more likely to see "the green" and "green apple" several times but less likely to see "apple green" and "green the".
    - This kind of context is useful to know and utilize in many use cases of NLP. We can figure out what someone is more likely to say to help decide between the possible outputs for an automatic speech recognition system.

Downloaded by Kumud (knowkumud@gmail.com)

How are n-grams Classified?

N-grams are classified into different types depending on the value that n takes. When $n=1$, it is said to be a unigram. When $n=2$, it is said to be a **bigram**. When $n=3$, it is said to be a **trigram**. When $n=4$, it is said to be a 4-gram, and so on.

- Different types of n-grams are suitable for different types of applications in the n-gram model in nlp, and we need to try different n-grams on the dataset to confidently conclude which one works the best among all for the text corpus analysis.
- It is also established with research and substantiated that trigrams and 4 grams work the best in the case of **spam filtering**.

An Example of n-grams

Let us look at the example sentence **Cowards die many times before their deaths; the valiant never taste of death but once** and generate the associated n-grams related to the sentence.

- **Unigrams**: These are simply the **unique words in the sentence**.
    - Cowards, die, many, times, before, their, deaths, the valiant, the valiant, never, taste, of, death, but, once.
- **Bigrams**: These are simply the **pairs of co-occurring words** in the sentence formed by sliding one word at a time in the forward direction to generate the next bigram.
    - cowards die, die many, many times, times before, before their, their deaths, deaths the, the valiant, valiant never, never taste, the taste of, of death, death but, but once
- **Trigrams**: These are the **3 pairs of co-occurring words** in the sentence formed by sliding two words at a time in the forward direction to generate the next trigram.
    - cowards die many, die many times, many times before, times before their, before their deaths, their deaths the, deaths the valiant, the valiant never, valiant never taste, never taste of, taste of death, of death but, death but once
- **4-grams**: Here we have the window such that we have **combinations of 4 words together**
    - cowards die many times, die many times before, many, times before their, times before their deaths, before their deaths the, their deaths the valiant, deaths the valiant never, the valiant taste, valiant never taste of, never taste of death, taste of death but, of death but once
- Simialary we can pick $n>4$ and generate 5-grams etc.

In n-gram model what is Estimation parameters and smoothing

In an n-gram model, estimation parameters refer to the process of estimating the probabilities associated with the n-grams based on the observed frequencies in the training data. These probabilities are crucial for the model to make predictions about the next word given the previous (n-1) words.

Smoothing, on the other hand, is a technique used to handle the issue of data sparsity and improve the robustness of the n-gram model. It addresses situations where certain n-grams are not present or have very low frequencies in the training data, resulting in zero probabilities or unreliable

This document is available free of charge on studocu

Downloaded by Kumud (knowkumud@gmail.com)

estimates. Smoothing methods assign non-zero probabilities to unseen or infrequent n-grams by redistributing the probabilities from more frequent n-grams.

The most commonly used smoothing technique in n-gram models is called add-one smoothing (or Laplace smoothing). It involves adding a constant value (usually 1) to the count of each n-gram, both in the numerator and denominator when calculating the probabilities. This ensures that no probability estimate becomes zero, and the probability mass is redistributed among all possible n-grams.

For example, let's consider a bigram model and suppose we encounter the bigram "like to" 10 times in the training data. If we use add-one smoothing, we would add 1 to both the numerator and the denominator of the probability estimation formula. This means the count of "like to" becomes 11, and the count of "like" (denominator) is increased by the total number of distinct bigrams in the training data.

Other smoothing techniques, such as Good-Turing smoothing, Jelinek-Mercer smoothing, and Kneser-Ney smoothing, are also used in n-gram models to address different aspects of the data sparsity problem and improve the quality of probability estimates.

Smoothing techniques help the n-gram model handle unseen or infrequent n-grams and provide more reliable probability estimates, leading to better performance in language modeling and other NLP tasks that rely on n-gram models.

Evaluation of an N-gram Model

The evaluation of an n-gram model involves assessing its performance and effectiveness in capturing the underlying language patterns and making accurate predictions. Here are some common evaluation measures used for n-gram models:

1. Perplexity: Perplexity is a widely used evaluation metric for language models, including n-gram models. It measures how well the model predicts a given test dataset. Lower perplexity indicates better performance. Perplexity is calculated using the probability distribution of the n-grams in the test data.
2. N-gram Coverage: N-gram coverage measures the percentage of n-grams in the test dataset that are covered by the n-gram model. Higher coverage indicates a better ability to capture the observed n-grams in the data.
3. Word Error Rate (WER): WER is often used in speech recognition tasks to evaluate the accuracy of the generated text compared to the reference transcript. It calculates the percentage of words that differ between the generated text and the reference transcript.
4. BLEU Score: BLEU (Bilingual Evaluation Understudy) is commonly used in machine translation tasks to evaluate the quality of the translated text. It measures the similarity between the generated translation and one or more reference translations. BLEU score ranges from 0 to 1, with higher scores indicating better translation quality.
5. Precision, Recall, and F1-score: These metrics are used for evaluating specific NLP tasks such as named entity recognition, part-of-speech tagging, or sentiment analysis. Precision measures the accuracy of the model's positive predictions, recall measures how well the

Downloaded by Kumud (knowkumud@gmail.com)

model finds all the positive instances, and the F1-score is the harmonic mean of precision and recall.

6. Human Evaluation: In some cases, human evaluation is conducted to assess the quality of the generated text or the performance of the model. Human judges may rate the fluency, coherence, and overall quality of the generated text to provide subjective feedback.

It's important to note that the evaluation measures may vary depending on the specific task and the application of the n-gram model. Additionally, n-gram models have limitations in capturing long-range dependencies and understanding the semantic context of the text. Therefore, more advanced language models, such as recurrent neural networks (RNNs) and transformer-based models, have gained popularity and often outperform n-gram models in various NLP tasks.

unknown words, out-of-vocabulary words

Unknown words, also referred to as out-of-vocabulary (OOV) words, are words that appear in the test or inference data but were not encountered during the training phase of an NLP model. Dealing with unknown words is a common challenge in NLP, and there are several approaches to address this issue:

1. Out-of-Vocabulary Handling: When an unknown word is encountered during inference, the model can assign a special token, such as "<UNK>", to represent the unknown word. This allows the model to at least acknowledge the presence of an unknown word in the output.

2. Morphological Analysis: Morphological analysis involves breaking down words into smaller meaningful units called morphemes. By analyzing the morphological structure of words, the model can potentially infer the meaning of unknown words based on known morphemes.

3. Subword Units: Instead of treating words as indivisible units, subword units can be used to represent words. Subword units can be character-based, such as character n-grams or byte-pair encoding (BPE), which break down words into smaller subword units. This approach helps the model handle unknown words by leveraging the knowledge of known subword units.

4. Language Resources: Utilizing external linguistic resources, such as dictionaries or word lists, can aid in handling unknown words. These resources can be used to check if an unknown word is a valid word or to provide additional information, such as part-of-speech tags, to guide the model's behavior.

5. Contextual Word Embeddings: Contextual word embeddings, such as those obtained from models like BERT or GPT, capture word meanings based on the context in which they appear. These embeddings can help the model handle unknown words by inferring their representations from the surrounding context.

6. Backoff Models: Backoff models are used in n-gram models to handle unknown words by gradually reducing the context window until a known n-gram is encountered. By progressively using shorter context windows, the model can fall back to lower-order n-grams that have been encountered during training.

7. Transfer Learning: Transfer learning approaches, such as fine-tuning pre-trained models, allow the model to leverage knowledge from a large-scale language model trained on vast amounts of data. These models have often encountered a wide range of words, including

This document is available free of charge on studocu

Downloaded by Kumud (knowkumud@gmail.com)

rare or unknown words, during their training and can generalize better to unknown words during inference.

Handling unknown words is an ongoing research area in NLP, and various techniques continue to evolve to address this challenge. The choice of approach depends on the specific task, available resources, and the characteristics of the data being processed.

Difference between open and closed class words

Open class words and closed class words are categories used in linguistics to classify words based on their characteristics and behavior within a language. Here's the difference between the two:

1. Open Class Words: Open class words, also known as content words or lexical words, are the main words that carry the content and convey most of the meaning in a sentence. These words can be added, modified, and expanded over time as language evolves. Open class words include:

- Nouns: Words that represent people, places, things, or concepts (e.g., "dog," "house," "love").
- Verbs: Words that express actions, processes, or states (e.g., "run," "eat," "think").
- Adjectives: Words that describe or modify nouns (e.g., "happy," "big," "red").
- Adverbs: Words that modify verbs, adjectives, or other adverbs, indicating manner, time, place, or degree (e.g., "quickly," "very," "here").

Open class words tend to have a relatively larger set of members and can easily accept new words or word forms. They are typically more content-rich and contribute to the substance and meaning of a sentence.

2. Closed Class Words: Closed class words, also called function words or grammatical words, are a set of words with a fixed number of members. They serve structural or grammatical functions in a sentence and have relatively stable forms. Closed class words include:

- Pronouns: Words used to replace nouns (e.g., "I," "you," "he").
- Prepositions: Words that express relationships between other words in terms of time, location, direction, etc. (e.g., "in," "on," "at").
- Conjunctions: Words that connect words, phrases, or clauses (e.g., "and," "but," "or").
- Determiners: Words that provide information about nouns or noun phrases (e.g., "the," "a," "this").
- Auxiliaries: Words used to help form verb tenses, moods, or voices (e.g., "be," "have," "will").

Closed class words have limited flexibility in terms of new additions or modifications. Their role is primarily to establish grammatical relationships and convey functional information within a sentence rather than adding new semantic content.

Downloaded by Kumud (knowkumud@gmail.com)

Understanding the distinction between open class words and closed class words can help analyze sentence structure, semantic relationships, and the grammatical patterns of a language.

Smoothing

Smoothing is a technique used in language modeling to address the issue of data sparsity and improve the performance of statistical models, such as n-gram models. Data sparsity occurs when certain n-grams have not been observed in the training data or have very low frequencies, resulting in unreliable probability estimates.

The goal of smoothing is to assign non-zero probabilities to unseen or infrequent n-grams, ensuring that the model can make reasonable predictions even for those cases. Smoothing redistributes the probability mass from more frequent n-grams to less frequent or unseen n-grams.

One commonly used smoothing technique is add-one smoothing (also known as Laplace smoothing). In add-one smoothing, a constant value (typically 1) is added to the count of each n-gram, both in the numerator and denominator when calculating the probabilities. This has the effect of increasing the counts of all n-grams, including the unseen ones.

The formula for calculating the smoothed probability of an n-gram is as follows:

P(w_n | w_1,...,w_{n-1}) = (Count(w_1,...,w_n) + 1) / (Count(w_1,...,w_{n-1}) + V)

Where:

- Count(w_1,...,w_n) is the count of the n-gram in the training data.
- Count(w_1,...,w_{n-1}) is the count of the (n-1)-gram (the context) in the training data.
- V is the vocabulary size, representing the total number of unique words in the training data.

The addition of 1 in the numerator ensures that no n-gram has a probability of 0. The addition of V in the denominator ensures that the probabilities sum up to 1.

Other smoothing techniques, such as Good-Turing smoothing, Jelinek-Mercer smoothing, and Kneser-Ney smoothing, provide more sophisticated ways of redistributing the probability mass and have been shown to produce better results in certain scenarios.

Overall, smoothing is a critical component in language modeling to handle unseen or infrequent n-grams and improve the robustness and generalization capability of the model.

Laplace Smoothing

Laplace smoothing, also known as add-one smoothing, is a simple technique used in language modeling to address the problem of zero probabilities for unseen or infrequent n-grams. It is one of the most basic and widely used smoothing methods in NLP.

In Laplace smoothing, a constant value (usually 1) is added to the count of each n-gram, both in the numerator and denominator, when calculating the probabilities. This has the effect of "smoothing out" the probability distribution and ensuring that no probability estimate becomes zero.

This document is available free of charge on studocu

Downloaded by Kumud (knowkumud@gmail.com)

The formula for calculating the smoothed probability of an n-gram using Laplace smoothing is as follows:

P(w_n | w_1,...,w_{n-1}) = (Count(w_1,...,w_n) + 1) / (Count(w_1,...,w_{n-1}) + V)

Where:

- Count(w_1,...,w_n) is the count of the n-gram in the training data.
- Count(w_1,...,w_{n-1}) is the count of the (n-1)-gram (the context) in the training data.
- V is the vocabulary size, representing the total number of unique words in the training data.

By adding 1 to both the numerator and denominator, Laplace smoothing guarantees that every n-gram has a non-zero probability. The addition of V in the denominator ensures that the probabilities sum up to 1, as it accounts for the extra counts added to all n-grams.

Laplace smoothing is a simple and intuitive method for handling data sparsity in language models, particularly in n-gram models. However, it assumes equal importance for all n-grams, regardless of their actual frequencies. More advanced smoothing techniques, such as Good-Turing smoothing, Jelinek-Mercer smoothing, or Kneser-Ney smoothing, take into account the frequency distribution of n-grams and provide more refined estimates.

Nevertheless, Laplace smoothing is often used as a baseline or starting point in NLP tasks, and it provides a straightforward way to assign non-zero probabilities to unseen or infrequent n-grams, improving the overall performance and robustness of the model.

Downloaded by Kumud (knowkumud@gmail.com)