Structural Ambiguities

Grammatical Ambiguities

Lexical Ambiguities

Different levels of language Analysis

1. Morphological Analysis
2. Lexical Analysis
3. Syntactic Analysis
4. Semantics Analysis
5. Pragmatic Analysis
6. Discourse Knowledge

Context free grammers -CFG’s

Remove punctuations

Toeknization

Remove stop words

Lemmatize / Stem.

NLTK is platform used for building programs for text analysis.

N- Gram Language Modelling

* N-Grams
* Evaluating Language Models
  + Machine Translation
  + Sentence Completion
  + Spell Correction
  + Speech Recognition
* Generalization of Zeros
* Smoothing
* The interpolation of BackOff
* Demo of N-Gram

[A Comprehensive Guide to Build your own Language Model in Python! | by Mohd Sanad Zaki Rizvi | Analytics Vidhya | Medium](https://medium.com/analytics-vidhya/a-comprehensive-guide-to-build-your-own-language-model-in-python-5141b3917d6d)

Joint Probability

Conditional Probabiltiy

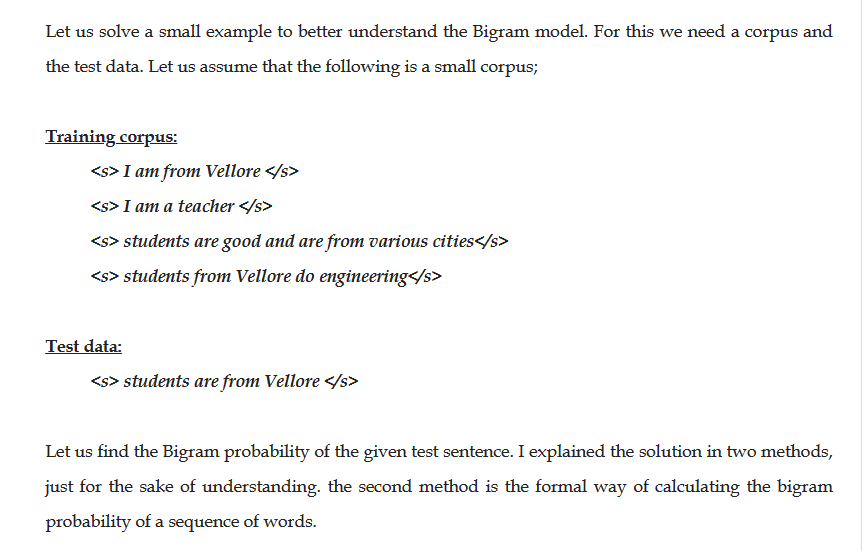
Markov assumption

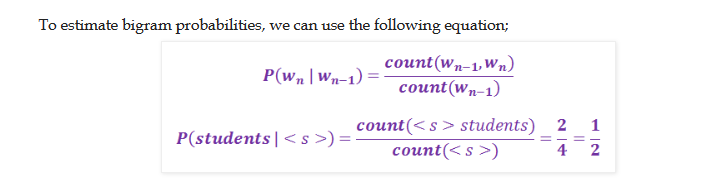
N-Gram

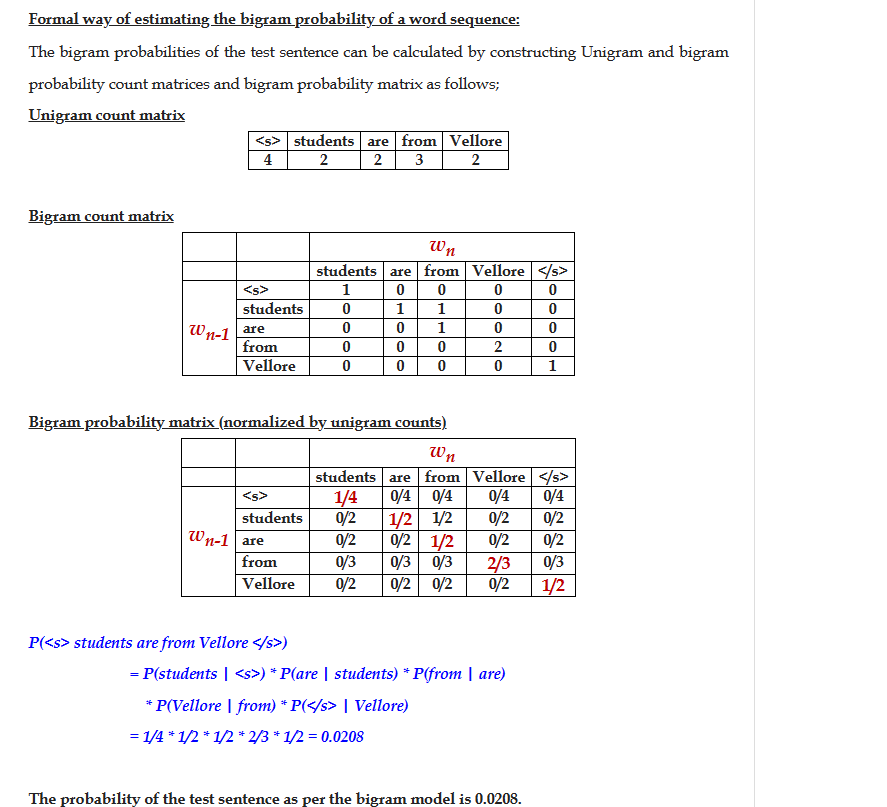
Unigram

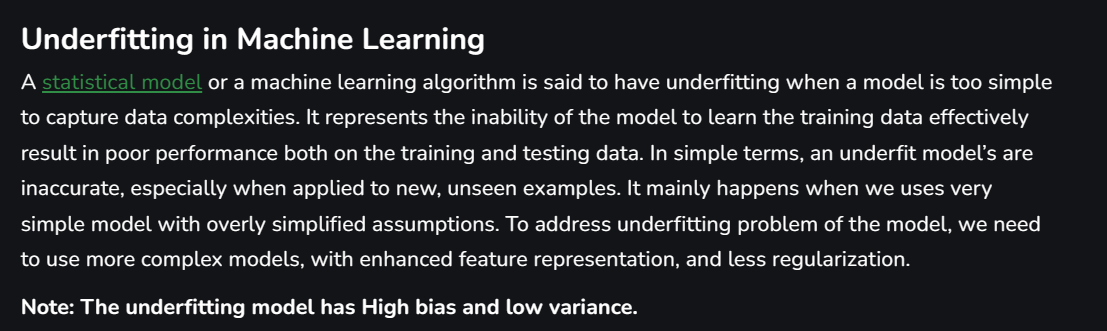
Bigram

Trigram









**Reasons for Underfitting**

1. The model is too simple, So it may be not capable to represent the complexities in the data.
2. The input features which is used to train the model is not the adequate representations of underlying factors influencing the target variable.
3. The size of the training dataset used is not enough.
4. Excessive regularization are used to prevent the overfitting, which constraint the model to capture the data well.
5. Features are not scaled.

**Techniques to Reduce Underfitting**

1. Increase model complexity.
2. Increase the number of features, performing [feature engineering](https://www.geeksforgeeks.org/what-is-feature-engineering/).
3. Remove noise from the data.
4. Increase the number of [epochs](https://www.geeksforgeeks.org/epoch-in-machine-learning/) or increase the duration of training to get better results.

[Computer Science and Engineering - Tutorials, Notes, MCQs, Questions and Answers: Explain add-1 (Laplace) smoothing with an example (exploredatabase.com)](https://www.exploredatabase.com/2020/10/explain-add-1-laplace-smoothing-with-example.html)

### Explain add-1 (Laplace) smoothing with an example

# **Natural language processing keywords, what is add-1 smoothing, what is Laplace smoothing, explain add-1 smoothing with an example, unigram and bi-gram with add-1 laplace smoothing**

## Add-1 (Laplace) smoothing

We have used Maximum Likelihood Estimation (MLE) for training the parameters of an N-gram model. The problem with MLE is that it assigns zero probability to unknown (unseen) words. This is because, MLE uses a training corpus. If the word in the test set is not available in the training set, then the count of that particular word is zero and it leads to zero probability.

To eliminate this zero probability, we can do smoothing. **Smoothing is about taking some probability mass from the events seen in training and assigns it to unseen events**. ***Add-1 smoothing*** (also called as ***Laplace smoothing***) is a simple smoothing technique that Add 1 to the count of all n-grams in the training set before normalizing into probabilities.

Recall that the unigram and bi-gram probabilities for a word w are calculated as follows;

P(w) = C(w)/N

P(wn|wn-1) = C(wn-1 wn)/C(wn-1)

Where, P(w) is the unigram probability, P(wn-1 wn) is the bigram probability, C(w) is the count of occurrence of w in the training set, C(wn-1 wn) is the count of bigram (wn-1 wn) in the training set, N is the total number of word tokens in the training set.

**Add-1 smoothing for unigrams**

PLaplace(w) = (C(w)+1)/N+|V|

Here, N is the total number of tokens in the training set and |V| is the size of the vocabulary represents the unique set of words in the training set.

As we have added 1 to the numerator, we have to normalize that by adding the count of unique words with the denominator in order to normalize.

**Add-1 smoothing for bigrams**

PLaplace(wn|wn-1) = (C(wn-1 wn)+1)/C(wn-1)+|V|

Interpolation and BackOff

# N-Gram Language Modelling with NLTK

**Read**

Courses

Video

Jobs

Language modeling is the way of determining the probability of any sequence of words. Language modeling is used in a wide variety of applications such as Speech Recognition, Spam filtering, etc. In fact, language modeling is the key aim behind the implementation of many state-of-the-art Natural Language Processing models.

**Methods of Language Modelings:**

Two types of Language Modelings:

* **Statistical Language Modelings**: Statistical Language Modeling, or Language Modeling, is the development of probabilistic models that are able to predict the next word in the sequence given the words that precede. Examples such as N-gram language modeling.
* **Neural Language Modelings**: Neural network methods are achieving better results than classical methods both on standalone language models and when models are incorporated into larger models on challenging tasks like speech recognition and machine translation. A way of performing a neural language model is through word embeddings.

# Intrinsic Evaluation:

What is Intrinsic Evaluation?

Intrinsic evaluation assesses the quality of an NLP model based on specific tasks or benchmarks directly related to the model’s performance. These tasks can include language modeling, part-of-speech tagging, sentiment analysis, and machine translation, among others.

Pros of Intrinsic Evaluation:

1. Task-Specific: Intrinsic evaluations are task-focused, providing insights into how well the model performs on a particular NLP task.
2. Quick Feedback: Results are obtained relatively quickly, allowing for rapid model iterations and improvements.
3. Benchmarking: Intrinsic evaluations often involve widely accepted benchmarks, making it easier to compare models and track progress.
4. Focused Metrics: Metrics such as accuracy, precision, recall, and F1-score provide detailed insights into model capabilities.
5. Controlled Environment: Researchers can control and manipulate evaluation conditions to gather precise data.

Cons of Intrinsic Evaluation:

1. Limited Scope: Intrinsic evaluations may not reflect the model’s performance in real-world applications, as they isolate specific tasks.
2. Not Always Predictive: Success in intrinsic tasks does not guarantee success in broader applications.
3. Task Dependency: The choice of evaluation task heavily influences the assessment, limiting generalizability.
4. Isolation from Real-World Use: Intrinsic evaluation may not reflect how the model performs in real-world applications where multiple tasks are involved.
5. Artificial Tasks: Some intrinsic tasks might be designed solely for evaluation purposes and lack practical significance.

# Extrinsic Evaluation:

What is Extrinsic Evaluation?

Extrinsic evaluation assesses the performance of an NLP model within the context of a real-world application or task. It measures how well the model contributes to achieving the overall goal, such as improving customer service chatbots, search engine performance, or language translation in healthcare.

Pros of Extrinsic Evaluation:

1. Holistic Assessment: It considers the model’s performance in a broader context, accounting for its interaction with other components or systems.
2. Generalization: Extrinsic evaluations provide insights into how well a model performs across diverse scenarios.
3. Real-World Relevance: Extrinsic evaluation reflects how the NLP model impacts real-world applications, providing a more accurate assessment of its practical value.
4. End-User Perspective: It aligns with the end-users’ perspective by focusing on the application’s overall success rather than individual tasks.
5. Complex Scenarios: Extrinsic evaluation considers the model’s performance in complex, multi-task environments.

Cons of Extrinsic Evaluation:

1. Complexity: Designing and conducting extrinsic evaluations can be more resource-intensive and time-consuming than intrinsic evaluations.
2. Subjectivity: Extrinsic evaluations may involve human judgment, introducing subjectivity in assessing the model’s performance.
3. Difficulty in Isolation: Isolating the model’s contribution from other factors in a real-world application can be challenging.
4. Dependent on the Application: The effectiveness of extrinsic evaluation heavily depends on the quality and complexity of the application.

# Use Case Example:

Scenario: Imagine you are developing a chatbot for a customer support service in an e-commerce company. The primary goal is to enhance user satisfaction and resolve customer queries efficiently.

Intrinsic Evaluation: In this case, intrinsic evaluation might involve assessing the chatbot’s language understanding capabilities, response time, and sentiment analysis accuracy. These metrics provide insights into how well the chatbot performs individual NLP tasks.

Extrinsic Evaluation: Extrinsic evaluation would assess the chatbot’s overall impact on customer satisfaction, response time reduction, and query resolution rate. This evaluation method considers the chatbot’s real-world performance in the context of improving customer support.

When to Use Which Approach:

* Intrinsic Evaluation: Use intrinsic evaluation when you want to fine-tune and assess the performance of individual NLP components or when benchmarking against specific tasks. It helps identify areas for improvement within the model.
* Extrinsic Evaluation: Choose extrinsic evaluation when you need to measure the model’s effectiveness in real-world applications or when assessing its contribution to achieving broader goals. This approach provides insights into how well the model performs in practical scenarios.

In conclusion, both intrinsic and extrinsic evaluation methods are essential in NLP, and the choice between them depends on your evaluation objectives and the context in which your model will be applied. A balanced approach that combines both methods can provide a comprehensive understanding of your NLP model’s capabilities and limitations.

In general, [perplexity](https://en.wikipedia.org/wiki/Perplexity) is a measurement of how well a probability model predicts a sample. In the context of Natural Language Processing, perplexity is one way to evaluate language models.

A [language model](https://en.wikipedia.org/wiki/Language_model) is a probability distribution over sentences: it’s both able to generate plausible human-written sentences (if it’s a good language model) and to evaluate the goodness of already written sentences. Presented with a well-written document, a good language model should be able to give it a higher probability than a badly written document, i.e. it should not be “perplexed” when presented with a well-written document.

Semantics

# Understanding Semantic Analysis – NLP

**Read**

Courses

Practice

Jobs

## Introduction to Semantic Analysis

Semantic Analysis is a subfield of Natural Language Processing (NLP) that attempts to understand the meaning of Natural Language. Understanding Natural Language might seem a straightforward process to us as humans. However, due to the vast complexity and subjectivity involved in human language, interpreting it is quite a complicated task for machines. Semantic Analysis of Natural Language captures the meaning of the given text while taking into account context, logical structuring of sentences and grammar roles.

## Parts of Semantic Analysis

Semantic Analysis of Natural Language can be classified into two broad parts:

**1. Lexical Semantic Analysis:**Lexical Semantic Analysis involves understanding the meaning of each word of the text individually. It basically refers to fetching the dictionary meaning that a word in the text is deputed to carry.

**2. Compositional Semantics Analysis:** Although knowing the meaning of each word of the text is essential, it is not sufficient to completely understand the meaning of the text.

For example, consider the following two sentences:

* **Sentence 1:** Students love GeeksforGeeks.
* **Sentence 2:** GeeksforGeeks loves Students.

Although both these sentences 1 and 2 use the same set of root words {student, love, geeksforgeeks}, they convey entirely different meanings.

Hence, under Compositional Semantics Analysis, we try to understand how combinations of individual words form the meaning of the text.

## Tasks involved in Semantic Analysis

In order to understand the meaning of a sentence, the following are the major processes involved in Semantic Analysis:

1. Word Sense Disambiguation
2. Relationship Extraction

### Word Sense Disambiguation:

In Natural Language, the meaning of a word may vary as per its usage in sentences and the context of the text. Word Sense Disambiguation involves interpreting the meaning of a word based upon the context of its occurrence in a text.

For example, the word ‘Bark’ may mean ‘the sound made by a dog’ or ‘the outermost layer of a tree.’

Likewise, the word ‘rock’ may mean ‘*a stone*‘ or ‘*a genre of music*‘ – hence, the accurate meaning of the word is highly dependent upon its context and usage in the text.

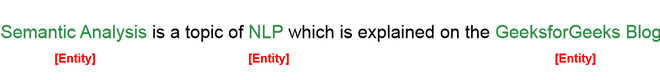
Thus, the ability of a machine to overcome the ambiguity involved in identifying the meaning of a word based on its usage and context is called Word Sense Disambiguation.

### Relationship Extraction:

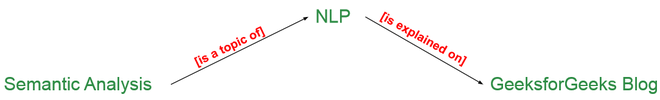
Another important task involved in Semantic Analysis is Relationship Extracting. It involves firstly identifying various entities present in the sentence and then extracting the relationships between those entities.

For example, consider the following sentence:

Semantic Analysis is a topic of NLP which is explained on the GeeksforGeeks blog. The entities involved in this text, along with their relationships, are shown below.



*Entities*



*Relationships*

## Elements of Semantic Analysis

Some of the critical elements of Semantic Analysis that must be scrutinized and taken into account while processing Natural Language are:

* **Hyponymy:** Hyponymys refers to a term that is an instance of a generic term. They can be understood by taking class-object as an analogy. For example: ‘*Color*‘ is a hypernymy while ‘*grey*‘, ‘*blue*‘, ‘*red*‘, etc, are its hyponyms.
* **Homonymy:** Homonymy refers to two or more lexical terms with the same spellings but completely distinct in meaning. For example: ‘*Rose*‘ might mean ‘*the past form of rise*‘ or ‘*a flower*‘, – same spelling but different meanings; hence, ‘*rose*‘ is a homonymy.
* **Synonymy:**When two or more lexical terms that might be spelt distinctly have the same or similar meaning, they are called Synonymy. For example: *(Job, Occupation), (Large, Big), (Stop, Halt).*
* **Antonymy:**Antonymy refers to a pair of lexical terms that have contrasting meanings – they are symmetric to a semantic axis. For example: *(Day, Night), (Hot, Cold), (Large, Small).*
* **Polysemy:** Polysemy refers to lexical terms that have the same spelling but multiple closely related meanings. It differs from homonymy because the meanings of the terms need not be closely related in the case of homonymy. For example: ‘*man*‘ may mean ‘*the human species*‘ or ‘*a male human*‘ or ‘*an adult male human*‘ – since all these different meanings bear a close association, the lexical term ‘*man*‘ is a polysemy.
* **Meronomy:**Meronomy refers to a relationship wherein one lexical term is a  constituent of some larger entity. For example: ‘*Wheel*‘ is a meronym of ‘*Automobile*‘

## Meaning Representation

While, as humans, it is pretty simple for us to understand the meaning of textual information, it is not so in the case of machines. Thus, machines tend to represent the text in specific formats in order to interpret its meaning. This formal structure that is used to understand the meaning of a text is called meaning representation.

### Basic Units of Semantic System:

In order to accomplish Meaning Representation in Semantic Analysis, it is vital to understand the building units of such representations. The basic units of semantic systems are explained below:

1. **Entity:** An entity refers to a particular unit or individual in specific such as a person or a location.  For example GeeksforGeeks, Delhi, etc.
2. **Concept:**A Concept may be understood as a generalization of entities. It refers to a broad class of individual units. For example Learning Portals, City, Students.
3. **Relations:**Relations help establish relationships between various entities and concepts. For example: ‘GeeksforGeeks is a Learning Portal’, ‘Delhi is a City.’, etc.
4. **Predicate:** Predicates represent the verb structures of the sentences.

In Meaning Representation, we employ these basic units to represent textual information.

### Approaches to Meaning Representations:

Now that we are familiar with the basic understanding of Meaning Representations, here are some of the most popular approaches to meaning representation:

1. First-order predicate logic (FOPL)
2. Semantic Nets
3. Frames
4. Conceptual dependency (CD)
5. Rule-based architecture
6. Case Grammar
7. Conceptual Graphs

## Semantic Analysis Techniques

Based upon the end goal one is trying to accomplish, Semantic Analysis can be used in various ways. Two of the most common Semantic Analysis techniques are:

### ****Text Classification****

In-Text Classification, our aim is to label the text according to the insights we intend to gain from the textual data.

For example:

* In **Sentiment Analysis,** we try to label the text with the prominent emotion they convey. It is highly beneficial when analyzing customer reviews for improvement.
* In **Topic Classification**, we try to categories our text into some predefined categories. For example: Identifying whether a research paper is of Physics, Chemistry or Maths
* In **Intent Classification**, we try to determine the intent behind a text message. For example: Identifying whether an e-mail received at customer care service is a query, complaint or request.

### Text Extraction

In-Text Extraction, we aim at obtaining specific information from our text.

For Example,

* In **Keyword Extraction**, we try to obtain the essential words that define the entire document.
* In **Entity Extraction**, we try to obtain all the entities involved in a document.

## Significance of Semantics Analysis

Semantics Analysis is a crucial part of Natural Language Processing (NLP). In the ever-expanding era of textual information, it is important for organizations to draw insights from such data to fuel businesses. Semantic Analysis helps machines interpret the meaning of texts and extract useful information, thus providing invaluable data while reducing manual efforts.

Besides, Semantics Analysis is also widely employed to facilitate the processes of automated answering systems such as chatbots – that answer user queries without any human interventions.

Don't miss your chance to ride the wave of the data revolution! Every industry is scaling new heights by tapping into the power of data. Sharpen your skills, become a part of the hottest trend in the 21st century.  
Dive into the future of technology - explore the [Complete Machine Learning and Data Science Program](https://www.geeksforgeeks.org/courses/data-science-live?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses) by GeeksforGeeks and stay ahead of the curve.

Lexical Semantics:

Word Vector Representation

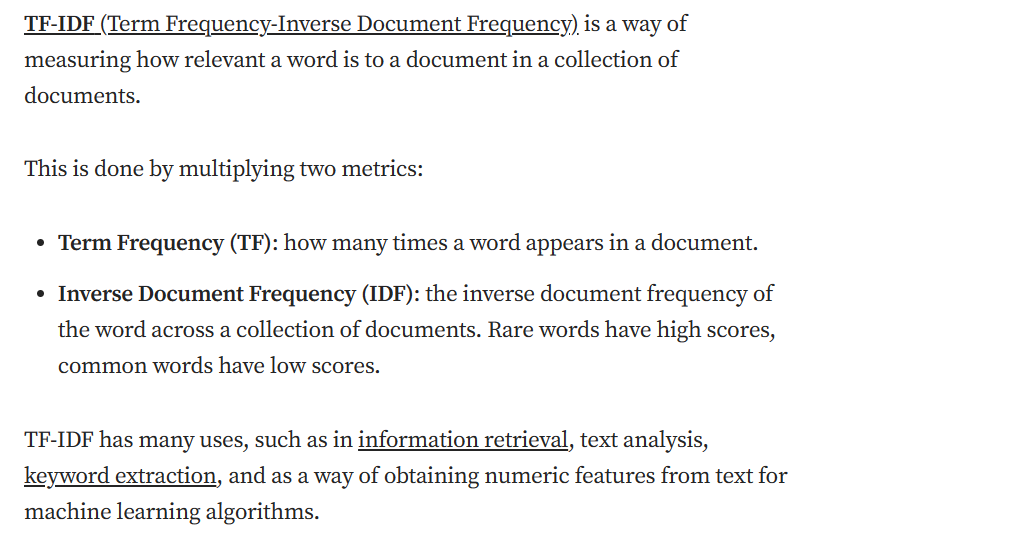
TF – Term frequency

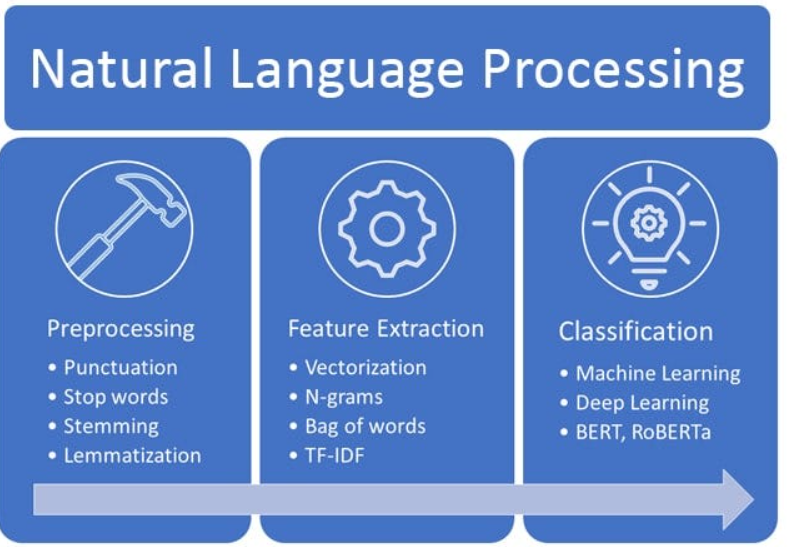
IDF – Inverse Document Frequency

# The use of TF-IDF in Machine Learning

TF-IDF is often used to transform text into a vector of numbers, otherwise known as [text vectorization](https://towardsdatascience.com/getting-started-with-text-vectorization-2f2efbec6685), where the numbers of the vectors are meant to somehow represent the content of the text.

TF-IDF gives us a way to associate each word in a document with a number that represents how relevant each word is in that document. Such numbers can be then used as features of machine learning models.





# What Does TF-IDF Mean?

TF-IDF stands for term frequency-inverse document frequency.

TF-IDF is typically used in the machine learning world and information retrieval.

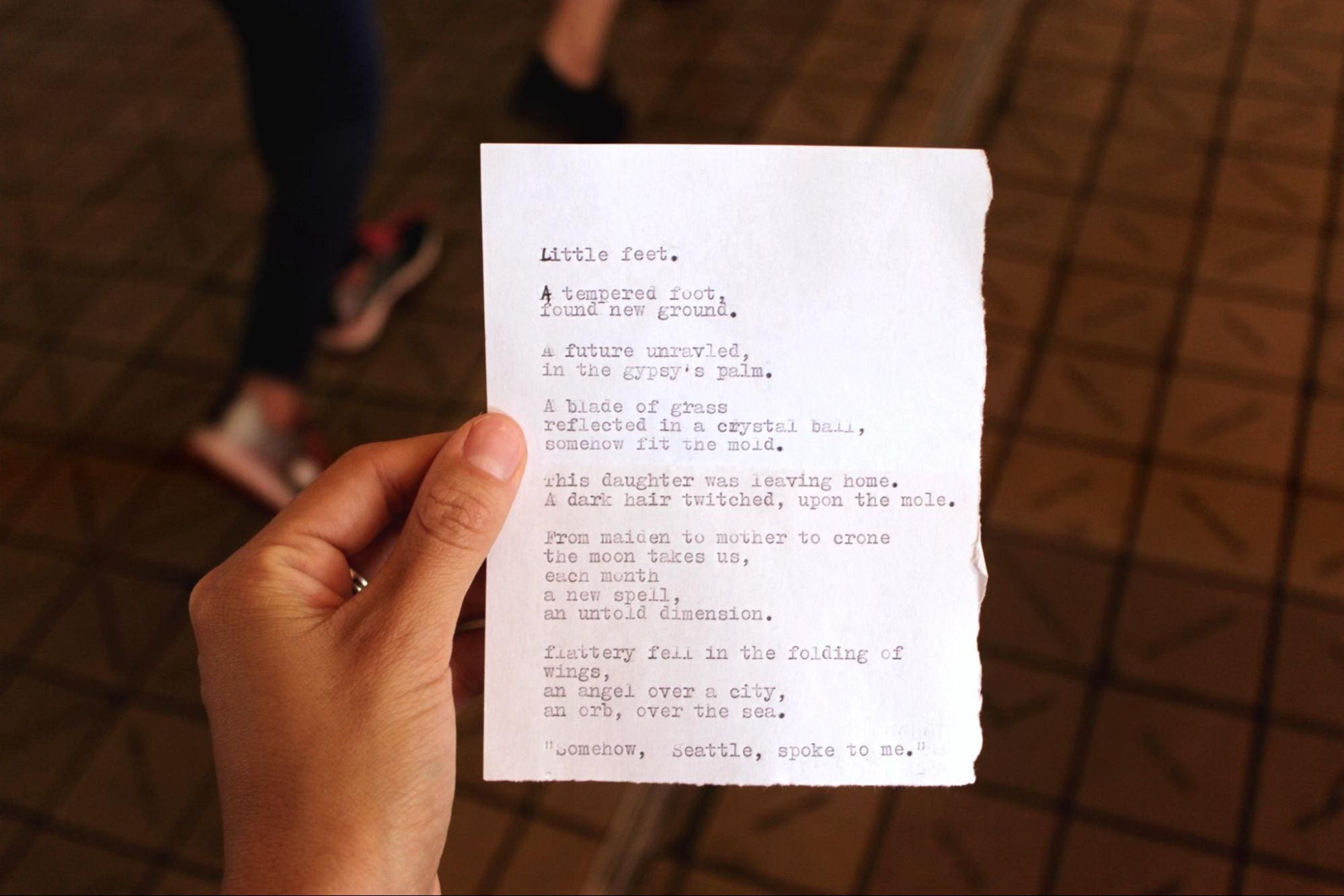
TF-IDF is a numerical statistic that measures the importance of string representations such as words, phrases and more in a corpus (document).

Let’s break the abbreviation up and go into further understanding.

# What is a Corpus?

When it comes to the art of language or Natural Language Processing in machine learning, a corpus is a collection of text or audio which has been organized into a dataset.

[TF-IDF Defined - KDnuggets](https://www.kdnuggets.com/2022/10/tfidf-defined.html)

  
[Sarah Mae](https://unsplash.com/@graystreet) via Unsplash

Let’s take this picture of this poem on a piece of paper for example. The corpus of this poem in a code format would look like this:

corpus = [

"Little feet.",

"A tempered foot,",

"found new ground.",

"A future unravled,",

"in the gypsy's palm.",

"A blade of grass",

"reflected in a crystal ball,",

"somehow fit the mold.",

"This daughter was leaving home.",

"A dark hair twitched, upon the mole.",

"From maiden to mother to crone",

"the moon takes us,",

"each month",

"a new spell.",

"an untold dimension.",

"Flattery fell in the folding of wings,",

"an angel over a city,",

"an orb, over the sea.",

"somehow, Seattle spoke to me."

]

# Mathematical Definition

This is the mathematical equation to define TF IDF:



* t stands for term
* d stands for document
* D stands for set of documents

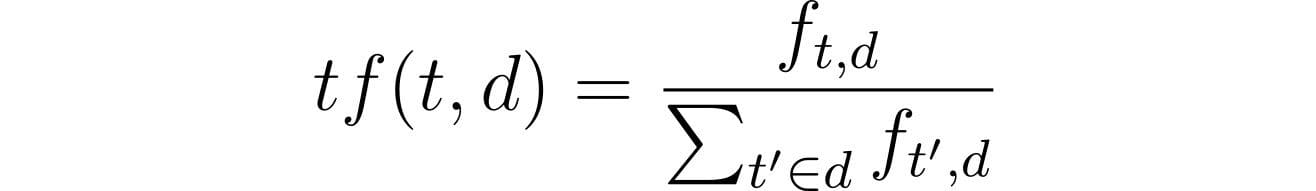
# Term Frequency (TF)

TF is **term frequency**. It measures exactly what it says - the frequency of a particular term. The number of times a particular term is available in a corpus can help us to measure the importance of that string.

You can measure the frequency in the following ways:

1. Raw count - You could do a raw count by counting manually how many times a word appears in the corpus.
2. Boolean frequency - a Boolean data type is when there are two possible values - true/false, yes/no, 0/1. You can use 1 if the term occurs or 0 if the term does not occur
3. Logarithmic scale - by using and displaying numerical data over a range of values.

## Mathematical equation for TF:



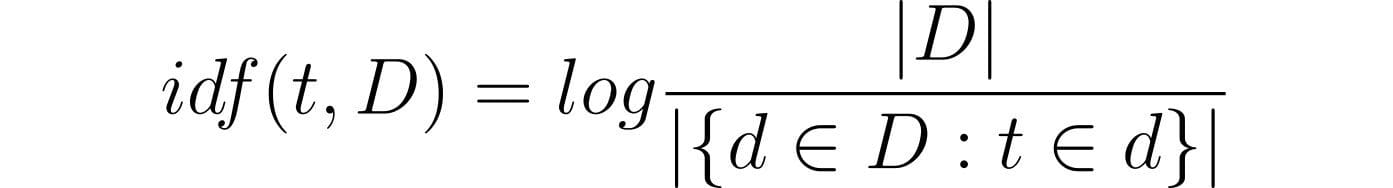
* t stands for term
* f stands for frequency
* d stands for document

# Inverse Document Frequency (IDF)

IDF is **inverse document frequency**. This goes further into looking at how common a word is found in a corpus - or how uncommon a word is found in a corpus.

IDF is important. Let’s take the English language for example, words such as “the”, “it”, “as”, “or” which appear frequently in many types of documents. Inverse document frequency essentially minimizes the weight of frequency terms such as those and puts terms which are not as frequent at the forefront to have a higher impact.

## Mathematical equation for IDF:



* t stands for term
* d stands for document
* D stands for set of documents

For IDF, you’re probably asking these questions:

### 1. Why do we take the inverse?

This is because we want to give the words that are uncommon a higher value in comparison to the words that are much more common.. If we did not take the inverses, common words such as “the” would have a higher value and we would never really find which terms in the corpus hold importance.

### 2. Why do we use logarithmic scale?

It is important to note that we are not focusing on the occurrence of a term in a corpus, it is the relevance and/or importance of that term in the corpus. Adding to the term frequency is essentially a sub-linear function, therefore using the logarithmic scale allows us to put these terms in the same scale or sub-linear function as the term frequency.

# The Importance of TF-IDF

Techniques in the Natural Language Processing world have been developing, and although TF IDF was first recognized in the 1970’s - it still holds relevance in 2022.

TF-IDF sounds simple in comparison to NLP techniques and tools that are being used today. But just because it’s simple does not mean that it does not hold value and does what it needs to do. TD-IDF can be used to better understand and interpret the outputs of algorithms that have been used on top of TF-IDF. There’s no harm in using more than one measure.

TF-IDF has also been known to solve major drawbacks from popular language processing techniques such as of Bag of words

Oh, and another reason: it’s quick, easy, and accessible.

# Conclusion

TF-IDF is a great starting point when it comes to language processing tasks, from building search engines to information retrieval. Although it is a simple measure, it still holds its intuitive approach to measuring the weight and relevance of words in a corpus.

# Word2Vec — Skip-Gram

[[](https://medium.com/@corymaklin?source=post_page-----904775613b4c--------------------------------)](https://medium.com/@corymaklin?source=post_page-----904775613b4c--------------------------------)

[Cory Maklin](https://medium.com/@corymaklin?source=post_page-----904775613b4c--------------------------------)

·

Follow

9 min read

·

Aug 11, 2022

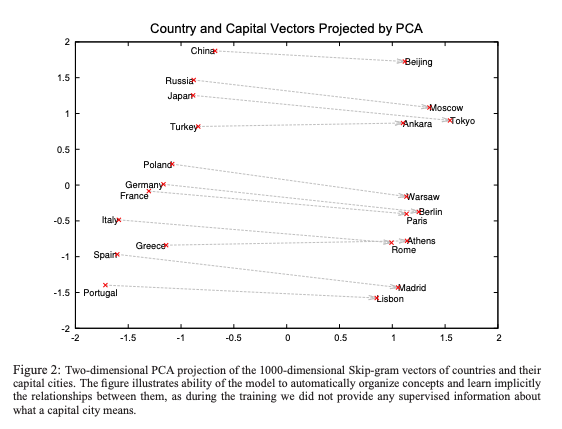
19

With a few exceptions, machine learning models do not accept raw text as input. The sequences of words must first be encoded in some fashion. We could represent each sentence as a Bag of Words (BOW). First, we find all the unique words in the text corpus. Then, we map every sentence to a vector whose length is equal to the length of the vocabulary (i.e. number of unique words) such that the values at the indices corresponding to words present in the sentence are set to 1, and the values at all other indices are left as 0.

There are two problems with this approach:

1. The vector is very sparse (i.e. most values are 0)
2. We lose information relating the context (i.e. order of the words in the sentence)

Alternatively, we could represent the text using word embeddings. A word embedding is a learned representation for text where related words will be closer to one another in feature space.



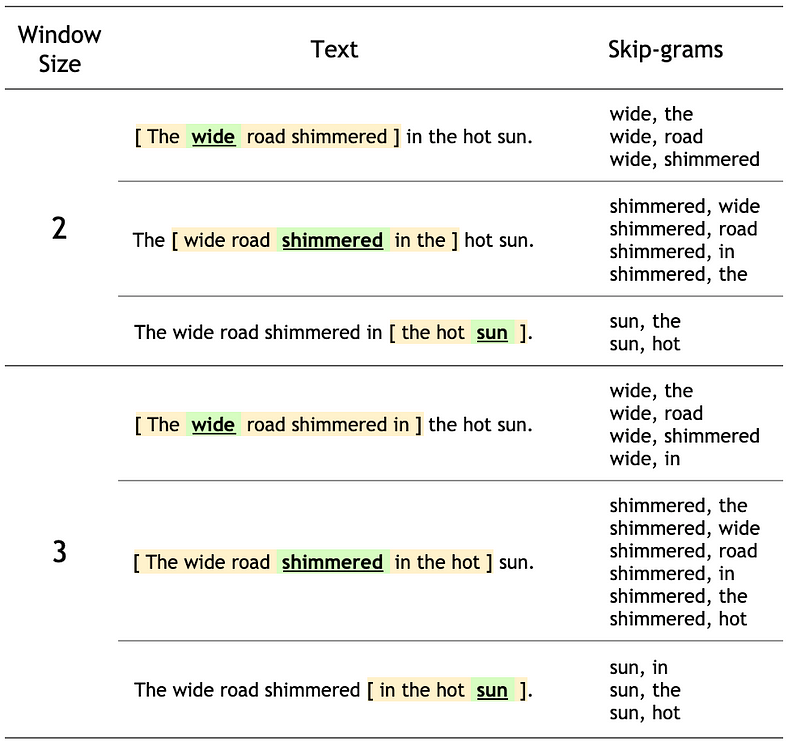
<https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf>

Word embeddings can be computed by training a machine learning model named [Word2Vec](https://arxiv.org/abs/1301.3781). There are two variants of Word2Vec — skip-gram and CBOW. The skip-gram variant takes a target word and tries to predict the surrounding context words, whereas the CBOW (continuous bag of words) variant takes a set of context words and tries to predict a target word. In this post, we will cover the skip-gram variant.

Suppose we had the following sentence:

The wide road shimmered in the hot sun.

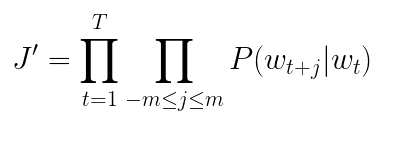
The window size determines the span of words on either side of a target\_word that can be considered a context word, as opposed to the number of context words.



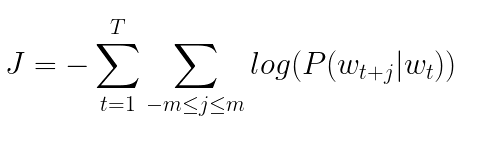
<https://www.tensorflow.org/tutorials/text/word2vec>

# Algorithm

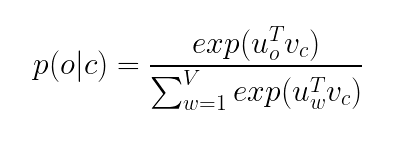
For each word t = 1 … T, we predict the surrounding words in a window of “radius” m. We train a machine learning model to maximize the probability of any context word given the current centre word.



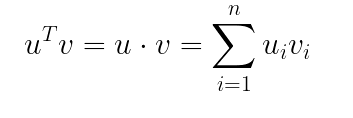
Like we do for other probabilistic models, we try to minimize the negative log likelihood.



where P(w\_{t+j}|w\_{t}) can be formulated as a Softmax function.



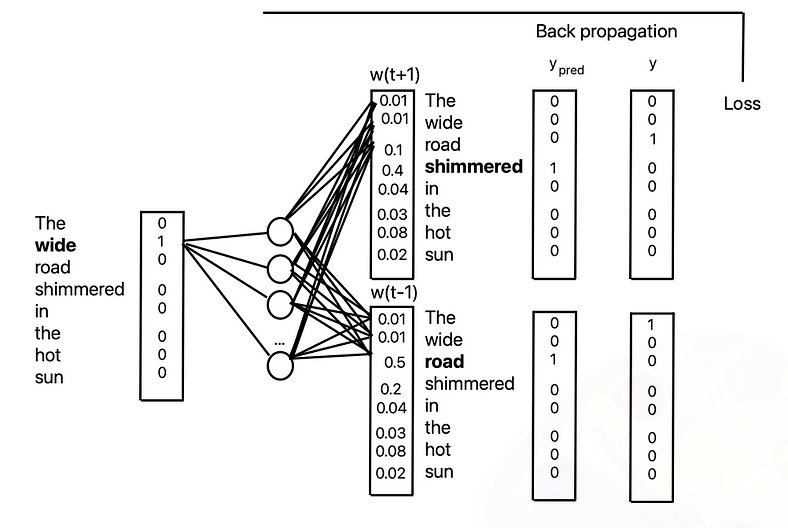
where



The latter reads as the probability of output word o given the centre word c. Recall that the denominator, in a Softmax function, is used to normalize the result to give a probability (i.e. number that ranges from 0 to 1).

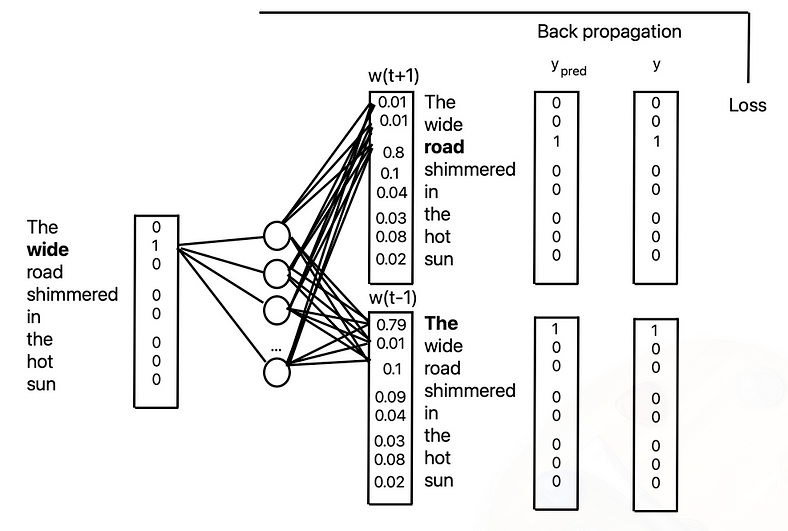
## Architecture

The skip-gram neural network is composed of a single hidden layer. The input is a Bag of Words (BOW) with a value of 1 at the position of the centre word. The output is the probability of finding a specific word at each position in the context window. In the following example, we assume we’re using a window size of 1. As we can see, initially, the model predicts that the word shimmered follows the word wide in a sentence and that the word road precedes the word wide in a sentence.



It’s important to note that we didn’t lowercase the words. Thus, we have The and the. However, in practice, you would.

Using back propagation, the model adjusts the weights until the error is minimized. As we can see, after a few iterations, it correctly predicts that the subsequent word after wide is road.



# Python

We will analyze individual sections of the code from Google’s detailed [word2vec tutorial](https://www.tensorflow.org/tutorials/text/word2vec).

Suppose we had the same sentence as before.

sentence = "The wide road shimmered in the hot sun"

We start off by splitting the sentence into individual tokens (i.e. words).

tokens = list(sentence.lower().split())  
print(len(tokens))8

Next, we map the words to numbers.

vocab, index = {}, 1 # start indexing from 1  
vocab['<pad>'] = 0 # add a padding token  
for token in tokens:  
 if token not in vocab:  
 vocab[token] = index  
 index += 1  
vocab\_size = len(vocab)inverse\_vocab = {index: token for token, index in vocab.items()}print(inverse\_vocab){0: '<pad>', 1: 'the', 2: 'wide', 3: 'road', 4: 'shimmered', 5: 'in', 6: 'hot', 7: 'sun'}

Then, we create a one dimensional vector.

example\_sequence = [vocab[word] for word in tokens]print(example\_sequence)[1, 2, 3, 4, 5, 1, 6, 7]

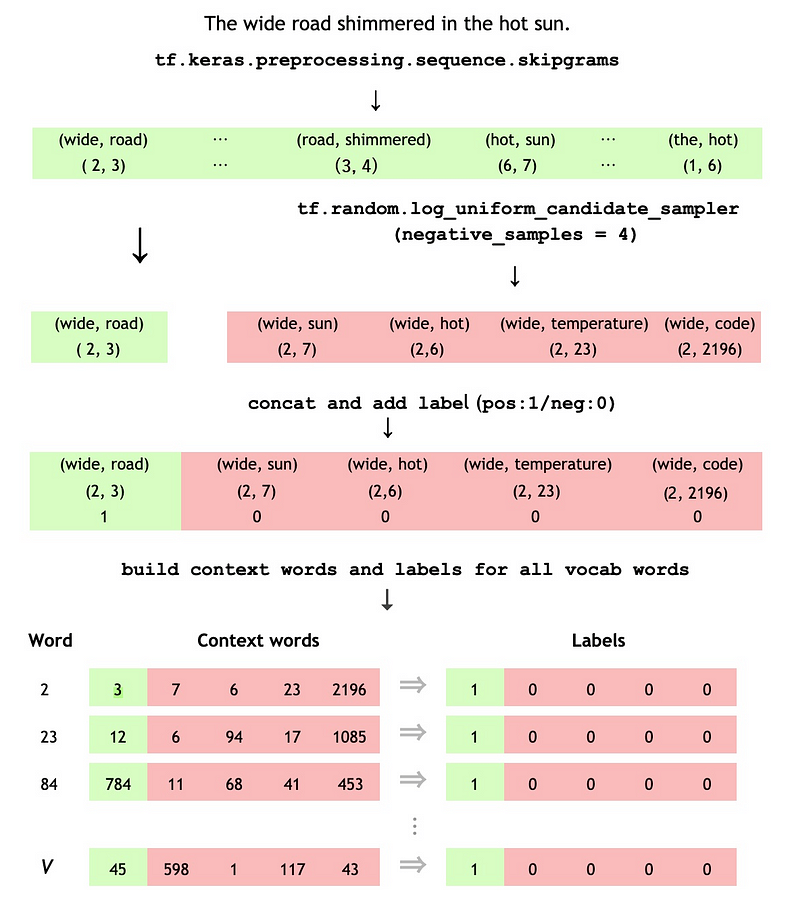
Using a window size of 2, we generate the list of all possible positive training samples given the example sentence.

window\_size = 2positive\_skip\_grams, \_ = tf.keras.preprocessing.sequence.skipgrams(  
 example\_sequence,  
 vocabulary\_size=vocab\_size,  
 window\_size=window\_size,  
 negative\_samples=0)for target, context in positive\_skip\_grams[:5]:  
 print(f"({target}, {context}): ({inverse\_vocab[target]}, {inverse\_vocab[context]})")(5, 4): (in, shimmered)  
(1, 6): (the, hot)  
(1, 5): (the, in)  
(1, 3): (the, road)  
(6, 5): (hot, in)

Using the first pair of target and context words, we generate num\_ns = 4 negative training samples. A sample is negative (i.e. assigned a label of 0) when the context word isn’t found inside of the context window.

# Get target and context words for one positive skip-gram.  
target\_word, context\_word = positive\_skip\_grams[0]  
  
# Set the number of negative samples per positive context.  
num\_ns = 4  
  
context\_class = tf.reshape(tf.constant(context\_word, dtype="int64"), (1, 1))  
negative\_sampling\_candidates, \_, \_ = tf.random.log\_uniform\_candidate\_sampler(  
 true\_classes=context\_class, # class that should be sampled as 'positive'  
 num\_true=1, # each positive skip-gram has 1 positive context class  
 num\_sampled=num\_ns, # number of negative context words to sample  
 unique=True, # all the negative samples should be unique  
 range\_max=vocab\_size, # pick index of the samples from [0, vocab\_size]  
 seed=SEED, # seed for reproducibility  
 name="negative\_sampling" # name of this operation  
)  
print(negative\_sampling\_candidates)  
print([inverse\_vocab[index.numpy()] for index in negative\_sampling\_candidates])# Add a dimension so you can use concatenation (in the next step).  
negative\_sampling\_candidates = tf.expand\_dims(negative\_sampling\_candidates, 1)  
  
# Concatenate a positive context word with negative sampled words.  
context = tf.concat([context\_class, negative\_sampling\_candidates], 0)  
  
# Label the first context word as `1` (positive) followed by `num\_ns` `0`s (negative).  
label = tf.constant([1] + [0]\*num\_ns, dtype="int64")  
  
# Reshape the target to shape `(1,)` and context and label to `(num\_ns+1,)`.  
target = tf.squeeze(target\_word)  
context = tf.squeeze(context)  
label = tf.squeeze(label)print(f"target\_index : {target}")  
print(f"target\_word : {inverse\_vocab[target\_word]}")  
print(f"context\_indices : {context}")  
print(f"context\_words : {[inverse\_vocab[c.numpy()] for c in context]}")  
print(f"label : {label}")target\_index : 6  
target\_word : hot  
context\_indices : [7 2 1 4 3]  
context\_words : ['sun', 'wide', 'the', 'shimmered', 'road']  
label : [1 0 0 0 0]

The following diagram summarizes the procedure of generating a training example:



<https://www.tensorflow.org/tutorials/text/word2vec>

We define a function that will dynamically generate training examples given a list of sentences, the window size, the number of negative samples, the vocabulary size and a random seed.

def generate\_training\_data(sequences, window\_size, num\_ns, vocab\_size, seed):  
 # Elements of each training example are appended to these lists.  
 targets, contexts, labels = [], [], []  
  
 # Build the sampling table for `vocab\_size` tokens.  
 sampling\_table = tf.keras.preprocessing.sequence.make\_sampling\_table(vocab\_size)  
  
 # Iterate over all sequences (sentences) in the dataset.  
 for sequence in tqdm.tqdm(sequences):  
  
 # Generate positive skip-gram pairs for a sequence (sentence).  
 positive\_skip\_grams, \_ = tf.keras.preprocessing.sequence.skipgrams(  
 sequence,  
 vocabulary\_size=vocab\_size,  
 sampling\_table=sampling\_table,  
 window\_size=window\_size,  
 negative\_samples=0)  
  
 # Iterate over each positive skip-gram pair to produce training examples  
 # with a positive context word and negative samples.  
 for target\_word, context\_word in positive\_skip\_grams:  
 context\_class = tf.expand\_dims(  
 tf.constant([context\_word], dtype="int64"), 1)  
 negative\_sampling\_candidates, \_, \_ = tf.random.log\_uniform\_candidate\_sampler(  
 true\_classes=context\_class,  
 num\_true=1,  
 num\_sampled=num\_ns,  
 unique=True,  
 range\_max=vocab\_size,  
 seed=seed,  
 name="negative\_sampling")  
  
 # Build context and label vectors (for one target word)  
 negative\_sampling\_candidates = tf.expand\_dims(  
 negative\_sampling\_candidates, 1)  
  
 context = tf.concat([context\_class, negative\_sampling\_candidates], 0)  
 label = tf.constant([1] + [0]\*num\_ns, dtype="int64")  
  
 # Append each element from the training example to global lists.  
 targets.append(target\_word)  
 contexts.append(context)  
 labels.append(label)  
  
 return targets, contexts, labels

Now, we can proceed to generate training examples from a larger list of sentences. That is, a text file of Shakespeare’s writing.

path\_to\_file = tf.keras.utils.get\_file('shakespeare.txt', 'https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt')text\_ds = tf.data.TextLineDataset(path\_to\_file).filter(lambda x: tf.cast(tf.strings.length(x), bool))def custom\_standardization(input\_data):  
 lowercase = tf.strings.lower(input\_data)  
 return tf.strings.regex\_replace(lowercase, '[%s]' % re.escape(string.punctuation), '')vocab\_size = 4096  
sequence\_length = 10vectorize\_layer = layers.TextVectorization(  
standardize=custom\_standardization,  
max\_tokens=vocab\_size,  
output\_mode='int',  
output\_sequence\_length=sequence\_length)vectorize\_layer.adapt(text\_ds.batch(1024))inverse\_vocab = vectorize\_layer.get\_vocabulary()text\_vector\_ds = text\_ds.batch(1024).prefetch(AUTOTUNE).map(vectorize\_layer).unbatch()sequences = list(text\_vector\_ds.as\_numpy\_iterator())for seq in sequences[:5]:  
 print(f"{seq} => {[inverse\_vocab[i] for i in seq]}")[ 89 270 0 0 0 0 0 0 0 0] => ['first', 'citizen', '', '', '', '', '', '', '', '']  
[138 36 982 144 673 125 16 106 0 0] => ['before', 'we', 'proceed', 'any', 'further', 'hear', 'me', 'speak', '', '']  
[34 0 0 0 0 0 0 0 0 0] => ['all', '', '', '', '', '', '', '', '', '']  
[106 106 0 0 0 0 0 0 0 0] => ['speak', 'speak', '', '', '', '', '', '', '', '']  
[ 89 270 0 0 0 0 0 0 0 0] => ['first', 'citizen', '', '', '', '', '', '', '', '']

We configure the dataset that will be used to train the model.

targets, contexts, labels = generate\_training\_data(sequences=sequences, window\_size=2, num\_ns=4, vocab\_size=vocab\_size, seed=SEED)targets = np.array(targets)  
contexts = np.array(contexts)[:,:,0]  
labels = np.array(labels)BATCH\_SIZE = 1024  
BUFFER\_SIZE = 10000dataset = tf.data.Dataset.from\_tensor\_slices(((targets, contexts), labels))  
dataset = dataset.shuffle(BUFFER\_SIZE).batch(BATCH\_SIZE, drop\_remainder=True)  
dataset = dataset.cache().prefetch(buffer\_size=AUTOTUNE)

We define a class for the Word2Vec model.

class Word2Vec(tf.keras.Model):  
 def \_\_init\_\_(self, vocab\_size, embedding\_dim):  
 super(Word2Vec, self).\_\_init\_\_()  
 self.target\_embedding = layers.Embedding(vocab\_size,  
 embedding\_dim,  
 input\_length=1,  
 name="w2v\_embedding")  
 self.context\_embedding = layers.Embedding(vocab\_size,  
 embedding\_dim,  
 input\_length=num\_ns+1)  
  
 def call(self, pair):  
 target, context = pair  
 # target: (batch, dummy?) # The dummy axis doesn't exist in TF2.7+  
 # context: (batch, context)  
 if len(target.shape) == 2:  
 target = tf.squeeze(target, axis=1)  
 # target: (batch,)  
 word\_emb = self.target\_embedding(target)  
 # word\_emb: (batch, embed)  
 context\_emb = self.context\_embedding(context)  
 # context\_emb: (batch, context, embed)  
 dots = tf.einsum('be,bce->bc', word\_emb, context\_emb)  
 # dots: (batch, context)  
 return dots

We will represent every word in the vocabulary using 128 dimensions. We instantiate an instance of our Word2Vec class. We compile the model using categorical crossentropy for our loss function.

embedding\_dim = 128word2vec = Word2Vec(vocab\_size, embedding\_dim)word2vec.compile(optimizer='adam', loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=True), metrics=['accuracy'])

Finally, we train the model.

word2vec.fit(dataset, epochs=20)

We obtain the embeddings (i.e. weights) between the input layer and the hidden layer.

weights = word2vec.get\_layer('w2v\_embedding').get\_weights()[0]

As we can see, we represent each of the 4096 words in the vocabulary using an embedding vector of length 128.

weights.shape(4096, 128)

We can examine the embedding vector corresponding to the first word in the vocabulary.

weights[0]array([ 0.02083263, 0.00343355, 0.03133059, 0.04064811, 0.02139286, 0.01668987, -0.01700681, 0.03104338, 0.00513292, 0.01149722, 0.00156037, 0.04110433, -0.02908002, -0.02072917, -0.04493903, -0.03360658, 0.02354895, 0.02986685, 0.01450031, -0.00434611, 0.02604233, 0.00688297, -0.00568321, 0.02448267, -0.04282743, 0.01752845, 0.02333864, -0.03737045, -0.03860588, 0.03164918, -0.03887875, 0.03344462, -0.04599243, -0.00912831, -0.03298129, -0.02165511, 0.00222781, -0.01334076, 0.03560077, -0.01657902, -0.04948949, 0.00923187, 0.03645227, 0.00624547, -0.00375736, 0.03080207, -0.03460135, 0.00123183, 0.0317348 , -0.03172968, -0.01598473, 0.03343581, 0.03939797, 0.01271281, 0.01737561, -0.04787338, 0.03081578, 0.02194339, 0.00668417, 0.0198779 , -0.03545182, 0.03608498, 0.03983852, 0.01381046, 0.02620314, -0.01378284, 0.04695277, -0.0301432 , -0.01917797, 0.03523597, 0.03922388, 0.02773141, 0.00329931, 0.02588192, 0.03493189, -0.02089679, 0.04374716, -0.03882134, -0.02024856, 0.04483554, -0.03621026, -0.04145117, -0.03030737, -0.02996567, -0.00220994, 0.0392569 , 0.03163559, -0.02619413, 0.04448912, -0.01938783, 0.02185104, 0.01294803, -0.01223926, -0.02752018, 0.02359452, 0.01469387, 0.01765844, -0.00813044, -0.04376047, -0.01028157, 0.00078993, -0.01525372, -0.0381612 , -0.00429031, 0.01438124, 0.03173996, 0.02320362, -0.03639726, -0.01158337, 0.04985858, 0.03488507, 0.0025389 , 0.03290978, 0.02607682, 0.04781124, 0.00342916, -0.03108559, 0.0361053 , 0.02612146, -0.00554097, -0.03796817, 0.03855484, -0.03623279, 0.0217861 , -0.01969334, -0.03057173, 0.03088465, -0.02974273], dtype=float32)

# Conclusion

Word embeddings are a widely used representation of words that captures the similarity between a given word and the other words in the corpus. In contrast to Bag of Words (BOW), word embeddings have the added advantage that they are dense. Word embeddings can be obtained by training a Word2Vec model and looking at the weights. There are two variations of the Word2Vec model: skip-gram and CBOW.

# Gradient Descent

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. In machine learning, we use gradient descent to update the [parameters](https://ml-cheatsheet.readthedocs.io/en/latest/glossary.html#glossary-parameters) of our model. Parameters refer to coefficients in [Linear Regression](https://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html) and [weights](https://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html#nn-weights) in neural networks.

Newral Networks in NLP

**What is POS tagging a preprocessing step for?**

A. Part-of-Speech (POS) tagging is a preprocessing step in natural language processing (NLP) that involves assigning a grammatical category or part-of-speech label (such as noun, verb, adjective, etc.) to each word in a sentence. It serves several purposes as a preprocessing step:  
1. **Syntactic Analysis**: POS tagging helps in understanding the grammatical structure of a sentence. It provides information about the roles of words in forming phrases and sentences, aiding in syntactic analysis.  
2. **Feature Extraction**: POS tags can be useful as features for various NLP tasks, such as text classification, named entity recognition, and machine translation. Different POS tags often convey different semantic or contextual information about words.  
3. **Disambiguation**: Many words in natural language have multiple possible interpretations (polysemy). POS tagging helps disambiguate word senses based on their grammatical context.  
4. **Language Modeling**: POS tagging is often used as a building block for language models, providing information about the relationships between words and their likely syntactic roles.  
**5**. **Rule-Based Processing**: POS tags can be used in rule-based processing to identify patterns and grammatical structures in text.  
6. **Lemmatization and Stemming**: POS information is valuable for lemmatization and stemming, where words are reduced to their base forms.  
7. **Parsing and Grammar Checking**: POS tagging aids in syntactic parsing and grammar checking by providing information about how words function within sentences.  
In summary, POS tagging is a fundamental preprocessing step that helps enhance the accuracy and effectiveness of various NLP tasks by providing insights into the grammatical and syntactic structure of textual data.

**Q2. What is an example of POS tagging?**

A. In the sentence “She quickly reads a book,” POS tagging assigns tags like “PRON” (pronoun) to “She,” “ADV” (adverb) to “quickly,” “VERB” to “reads,” “DET” (determiner) to “a,” and “NOUN” to “book.” This tagging clarifies the roles and grammatical functions of words, aiding syntactic and semantic analysis in natural language processing tasks.

**Q3.What is the use of POS tagging in sentiment analysis?**

POS tagging in sentiment analysis helps understand the emotional tone of words in a sentence. It labels words with their parts of speech, aiding in identifying nuances that influence sentiment.

**Q4.What is POS tagging text classification?**

POS tagging in text classification involves assigning parts of speech (like nouns, verbs, etc.) to words in a given text. This tagging adds linguistic context, enhancing the accuracy of algorithms in understanding and categorizing text content.

## **What are Hidden Markov Models?**

**A Hidden Markov Model (HMM) is a probabilistic model that consists of a sequence of hidden states, each of which generates an observation**. The hidden states are usually not directly observable, and the goal of HMM is to estimate the sequence of hidden states based on a sequence of observations. An HMM is defined by the following components:

* A set of N hidden states, S = {s1, s2, ..., sN}.
* A set of M observations, O = {o1, o2, ..., oM}.
* An initial state probability distribution, ? = {?1, ?2, ..., ?N}, which specifies the probability of starting in each hidden state.
* A transition probability matrix, A = [aij], defines the probability of moving from one hidden state to another.
* An emission probability matrix, B = [bjk], defines the probability of emitting an observation from a given hidden state.

The basic idea behind an HMM is that the hidden states generate the observations, and the observed data is used to estimate the hidden state sequence. This is often referred to as the **forward-backwards algorithm.**

* Filtering
* Prediction
* Smoothing
* Most Likely Explanation