1. **Explain One-Hot Encoding**

**One Hot Encoding in Machine Learning**

**Most real-life datasets we encounter during our data science project development have columns of mixed data type. These datasets consist of both**[**categorical**](https://www.geeksforgeeks.org/python-pandas-categorical/)**as well as numerical columns. However, various Machine Learning models do not work with categorical data and to fit this data into the machine learning model it needs to be converted into numerical data. For example, suppose a dataset has a *Gender*column with categorical elements like *Male and* *Female*. These labels have no specific order of preference and also since the data is string labels, machine learning models misinterpreted that there is some sort of hierarchy in them.**

**One approach to solve this problem can be label encoding where we will assign a numerical value to these labels for example *Male* and *Female* mapped to *0* and *1*. But this can add bias in our model as it will start giving higher preference to the *Female* parameter as 1>0 but ideally, both labels are equally important in the dataset. To deal with this issue we will use the One Hot Encoding technique.**

**One Hot Encoding**

**One hot encoding is a technique that we use to represent categorical variables as numerical values in a machine learning model.**

**The advantages of using one hot encoding include:**

1. **It allows the use of categorical variables in models that require numerical input.**
2. **It can improve model performance by providing more information to the model about the categorical variable.**
3. **It can help to avoid the problem of ordinality, which can occur when a categorical variable has a natural ordering (e.g. “small”, “medium”, “large”).**

**The disadvantages of using one hot encoding include:**

1. **It can lead to increased dimensionality, as a separate column is created for each category in the variable. This can make the model more complex and slow to train.**
2. **It can lead to sparse data, as most observations will have a value of 0 in most of the one-hot encoded columns.**
3. **It can lead to overfitting, especially if there are many categories in the variable and the sample size is relatively small.**
4. **One-hot-encoding is a powerful technique to treat categorical data, but it can lead to increased dimensionality, sparsity, and overfitting. It is important to use it cautiously and consider other methods such as ordinal encoding or binary encoding.**

**One Hot Encoding Examples**

**In One Hot Encoding, the categorical parameters will prepare separate columns for both Male and Female labels. So, wherever there is a Male, the value will be 1 in the Male column and 0 in the Female column, and vice-versa. Let’s understand with an example: Consider the data where fruits, their corresponding categorical values, and prices are given.**

| **Fruit** | **Categorical value of fruit** | **Price** |
| --- | --- | --- |
| **apple** | **1** | **5** |
| **mango** | **2** | **10** |
| **apple** | **1** | **15** |
| **orange** | **3** | **20** |

**The output after applying one-hot encoding on the data is given as follows,**

| **apple** | **mango** | **orange** | **price** |
| --- | --- | --- | --- |
| **1** | **0** | **0** | **5** |
| **0** | **1** | **0** | **10** |
| **1** | **0** | **0** | **15** |
| **0** | **0** | **1** | **20** |

1. **Explain Bag of Words**

**we are going to discuss a Natural Language Processing technique of text modeling known as Bag of Words model. Whenever we apply any algorithm in NLP, it works on numbers. We cannot directly feed our text into that algorithm. Hence, Bag of Words model is used to preprocess the text by converting it into a bag of words, which keeps a count of the total occurrences of most frequently used words.**

**This model can be visualized using a table, which contains the count of words corresponding to the word itself.**

**Applying the Bag of Words model:**

**Let us take this sample paragraph for our task :**

**Beans. I was trying to explain to somebody as we were flying in, that’s corn. That’s beans. And they were very impressed at my agricultural knowledge. Please give it up for Amaury once again for that outstanding introduction. I have a bunch of good friends here today, including somebody who I served with, who is one of the finest senators in the country, and we’re lucky to have him, your Senator, Dick Durbin is here. I also noticed, by the way, former Governor Edgar here, who I haven’t seen in a long time, and somehow he has not aged and I have. And it’s great to see you, Governor. I want to thank President Killeen and everybody at the U of I System for making it possible for me to be here today. And I am deeply honored at the Paul Douglas Award that is being given to me. He is somebody who set the path for so much outstanding public service here in Illinois. Now, I want to start by addressing the elephant in the room. I know people are still wondering why I didn’t speak at the commencement.**

1. **Explain Bag of N-Grams**

**N-Gram Language Modelling with NLTK**

**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.**

**N-gram**

**N-gram can be defined as the contiguous sequence of n items 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 (A long text dataset).**

**N-gram Language Model:**

**An N-gram language model predicts the probability of a given N-gram within any sequence of words in the language. A good N-gram model can predict the next word in the sentence i.e the value of p(w|h)**

**Example of N-gram such as unigram (“This”, “article”, “is”, “on”, “NLP”)  or bi-gram (‘This article’, ‘article is’, ‘is on’,’on NLP’).**

**Now, we will establish a relation on how to find the next word in the sentence using**

**. We need to calculate p(w|h), where is the candidate for the next word. For example in the above example, lets’ consider, we want to calculate what is the probability of the last word being “NLP” given the previous words:**

**After generalizing the above equation can be calculated as:**

**But how do we calculate it? The answer lies in the chain rule of probability:**

**Now generalize the above equation:**

**Simplifying the above formula using Markov assumptions:**

* **For unigram:**
* **For Bigram:**

1. **Explain TF-IDF**

**Understanding TF-IDF (Term Frequency-Inverse Document Frequency)**

**TF-IDF stands for Term Frequency Inverse Document Frequency of records. It can be defined as the calculation of how relevant a word in a series or corpus is to a text. The meaning increases proportionally to the number of times in the text a word appears but is compensated by the word frequency in the corpus (data-set).**

**Terminologies:**

* **Term Frequency: In document d, the frequency represents the number of instances of a given word t. Therefore, we can see that it becomes more relevant when a word appears in the text, which is rational. Since the ordering of terms is not significant, we can use a vector to describe the text in the bag of term models. For each specific term in the paper, there is an entry with the value being the term frequency.**

**The weight of a term that occurs in a document is simply proportional to the term frequency.**

**tf(t,d) = count of t in d / number of words in d**

* **Document Frequency: This tests the meaning of the text, which is very similar to TF, in the whole corpus collection. The only difference is that in document d, TF is the frequency counter for a term t, while df is the number of occurrences in the document set N of the term t. In other words, the number of papers in which the word is present is DF.**

**df(t) = occurrence of t in documents**

* **Inverse Document Frequency: Mainly, it tests how relevant the word is. The key aim of the search is to locate the appropriate records that fit the demand. Since tf considers all terms equally significant, it is therefore not only possible to use the term frequencies to measure the weight of the term in the paper. First, find the document frequency of a term t by counting the number of documents containing the term:**

**df(t) = N(t)**

**where**

**df(t) = Document frequency of a term t**

**N(t) = Number of documents containing the term t**

**Term frequency is the number of instances of a term in a single document only; although the frequency of the document is the number of separate documents in which the term appears, it depends on the entire corpus. Now let’s look at the definition of the frequency of the inverse paper. The IDF of the word is the number of documents in the corpus separated by the frequency of the text.**

**idf(t) = N/ df(t) = N/N(t)**

**The more common word is supposed to be considered less significant, but the element (most definite integers) seems too harsh. We then take the logarithm (with base 2) of the inverse frequency of the paper. So the if of the term t becomes:**

**idf(t) = log(N/ df(t))**

* **Computation: Tf-idf is one of the best metrics to determine how significant a term is to a text in a series or a corpus. tf-idf is a weighting system that assigns a weight to each word in a document based on its term frequency (tf) and the reciprocal document frequency (tf) (idf). The words with higher scores of weight are deemed to be more significant.**

**Usually, the tf-idf weight consists of two terms-**

1. **Normalized Term Frequency (tf)**
2. **Inverse Document Frequency (idf)**

**tf-idf(t, d) = tf(t, d) \* idf(t)**

**In python tf-idf values can be computed using *TfidfVectorizer()*method in *sklearn* module.**

**Syntax:**

***sklearn.feature\_extraction.text.TfidfVectorizer(input)***

***Parameters:***

* ***input: It refers to parameter document passed, it can be a filename, file or content itself.***

***Attributes:***

* ***vocabulary\_: It returns a dictionary of terms as keys and values as feature indices.***
* ***idf\_: It returns the inverse document frequency vector of the document passed as a parameter.***

***Returns:***

* ***fit\_transform(): It returns an array of terms along with tf-idf values.***
* ***get\_feature\_names(): It returns a list of feature names.***

1. **What is OOV problem?**
2. **What are word embeddings?**

**Word Embeddings are numeric representations of words in a lower-dimensional space, capturing semantic and syntactic information. They play a vital role in Natural Language Processing (NLP) tasks. This article explores traditional and neural approaches, such as TF-IDF, Word2Vec, and GloVe, offering insights into their advantages and disadvantages. Understanding the importance of pre-trained word embeddings, providing a comprehensive understanding of their applications in various NLP scenarios.**

**What is Word Embedding in NLP?**

[**Word Embedding**](https://www.geeksforgeeks.org/overview-of-word-embedding-using-embeddings-from-language-models-elmo/)**is an approach for representing words and documents. Word Embedding or Word Vector is a numeric vector input that represents a word in a lower-dimensional space. It allows words with similar meanings to have a similar representation.**

**Word Embeddings are a method of extracting features out of text so that we can input those features into a machine learning model to work with text data. They try to preserve syntactical and semantic information. The methods such as**[**Bag of Words (BOW)**](https://www.geeksforgeeks.org/bag-of-words-bow-model-in-nlp/)**, [CountVectorizer](https://www.geeksforgeeks.org/using-countvectorizer-to-extracting-features-from-text/) and TFIDF rely on the word count in a sentence but do not save any syntactical or semantic information. In these algorithms, the size of the vector is the number of elements in the vocabulary. We can get a sparse matrix if most of the elements are zero. Large input vectors will mean a huge number of weights which will result in high computation required for training. Word Embeddings give a solution to these problems.**

**Need for Word Embedding?**

* **To reduce dimensionality**
* **To use a word to predict the words around it.**
* **Inter-word semantics must be captured.**

1. **Explain Continuous bag of words (CBOW)**

**Continuous bag of words (CBOW) in NLP**

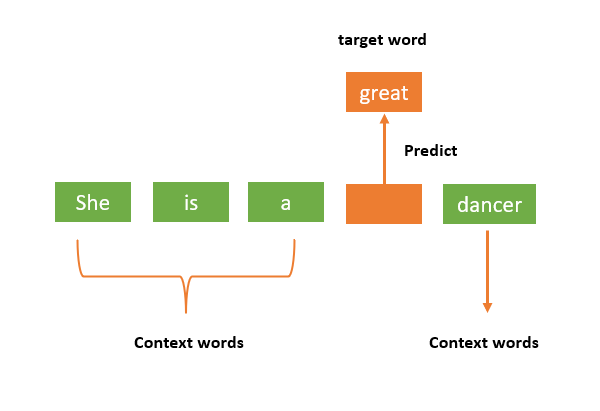
**In order to make the computer understand a written text, we can represent the words as numerical vectors. One way to do so is by Using Word embeddings, they are a way of representing words as numerical vectors. These vectors capture the meaning of the words and their relationships to other words in the language. Word embeddings can be generated using unsupervised learning algorithms such as Word2vec, [GloVe](https://www.geeksforgeeks.org/pre-trained-word-embedding-using-glove-in-nlp-models/), or [FastText](https://www.geeksforgeeks.org/fasttext-working-and-implementation/).**

**Word2vec is a neural network-based method for generating word embeddings, which are dense vector representations of words that capture their semantic meaning and relationships. There are two main approaches to implementing**[**Word2vec**](https://www.geeksforgeeks.org/python-word-embedding-using-word2vec/)**:**

* **Continuous bag-of-words (CBOW)**
* [**Skip-gram**](https://www.geeksforgeeks.org/implement-your-own-word2vecskip-gram-model-in-python/)

**What is a Continuous Bag of Words (CBOW)?**

**Continuous Bag of Words (CBOW) is a popular natural language processing technique used to generate word embeddings.**[**Word embeddings**](https://www.geeksforgeeks.org/word-embeddings-in-nlp/)**are important for many NLP tasks because they capture semantic and syntactic relationships between words in a language. CBOW is a neural network-based algorithm that predicts a target word given its surrounding context words. It is a type of “**[**unsupervised**](https://www.geeksforgeeks.org/supervised-unsupervised-learning/)**” learning, meaning that it can learn from unlabeled data, and it is often used to pre-train word embeddings that can be used for various NLP tasks such as**[**sentiment analysis**](https://www.geeksforgeeks.org/what-is-sentiment-analysis/)**,**[**text classification**](https://www.geeksforgeeks.org/text-mining-in-data-mining/)**, and**[**machine translation**](https://www.geeksforgeeks.org/machine-translation-of-languages-in-artificial-intelligence/)**.**

****

***Example of a CBOW Model***

**Is there any difference between  Bag-of-Words (BoW) model and the Continuous Bag-of-Words (CBOW)?**

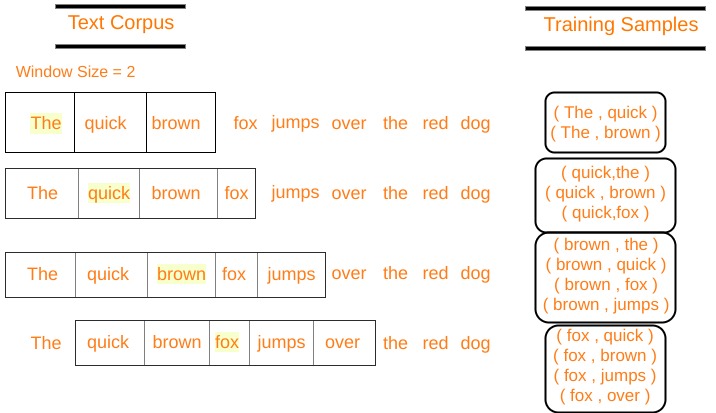
* **The Bag-of-Words model and the Continuous Bag-of-Words model are both techniques used in natural language processing to represent text in a computer-readable format, but they differ in how they capture context.**
* **The BoW model represents text as a collection of words and their frequency in a given document or corpus. It does not consider the order or context in which the words appear, and therefore, it may not capture the full meaning of the text. The BoW model is simple and easy to implement, but it has limitations in capturing the meaning of language.**
* **In contrast, the CBOW model is a neural network-based approach that captures the context of words. It learns to predict the target word based on the words that appear before and after it in a given context window. By considering the surrounding words, the CBOW model can better capture the meaning of a word in a given context.**

1. **ExplainSkipGram**

**Implement your own word2vec(skip-gram) model in Python**

**Last Updated : 04 Aug, 2022**

**Prerequisite:**[**Introduction to word2vec**](https://www.geeksforgeeks.org/python-word-embedding-using-word2vec/)[**Natural language processing**](https://www.geeksforgeeks.org/introduction-to-natural-language-processing/)**(NLP) is a subfield of computer science and**[**artificial intelligence**](https://www.geeksforgeeks.org/artificial-intelligence-an-introduction/)**concerned with the interactions between computers and human (natural) languages.   
In NLP techniques, we map the words and phrases (from vocabulary or corpus) to vectors of numbers to make the processing easier. These types of language modeling techniques are called word embeddings.   
In 2013, Google announced word2vec, a group of related models that are used to produce word embeddings.  
Let’s implement our own skip-gram model (in Python) by deriving the backpropagation equations of our neural network.  
In skip-gram architecture of word2vec, the input is the center word and the predictions are the context words. Consider an array of words W, if W(i) is the input (center word), then W(i-2), W(i-1), W(i+1), and W(i+2) are the context words if the *sliding window size* is 2.**

****

**Let's define some variables :**

**V Number of unique words in our corpus of text ( Vocabulary )**

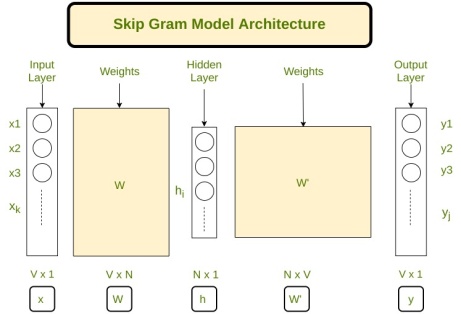
**x Input layer (**[**One hot encoding**](https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f) **of our input word ).**

**N Number of neurons in the hidden layer of neural network**

**W Weights between input layer and hidden layer**

**W' Weights between hidden layer and output layer**

**y A softmax output layer having probabilities of every word in our vocabulary**

****

***Skip gram architecture***

**Our neural network architecture is defined, now let’s do some math to derive the equations needed for gradient descent.**

**Forward Propagation:**

**Multiplying one hot encoding of the center word (denoted by x) with the first weight matrix W to get hidden layer matrix h (of size N x 1).   
  
*( Vx1 )        ( NxV )   ( Vx1 )*   
Now we multiply the hidden layer vector h with second weight matrix W’ to get a new matrix u  
  
*( Vx1 )        ( VxN )   ( Nx1 )*   
Note that we have to apply a softmax> to *layer u* to get our *output layer y*.  
Let uj be jth neuron of layer u   
Let wj be the jth word in our vocabulary where j is any index   
Let Vwj be the jth column of matrix W’(column corresponding to a word wj)  
  
*( 1×1 )        ( 1xN )   ( Nx1 )*   
y = softmax(u)   
yj = softmax(uj)   
yj denotes the probability that wj is a context word   
  
P(wj|wi) is the probability that wj is a context word, given wi is the input word.  
Thus, our goal is to maximize P( wj\* | wi ), where j\* represents the indices of context words  
Clearly, we want to maximize   
  
where j\*c are the vocabulary indexes of context words. Context words range from c = 1, 2, 3..C   
Let’s take a negative log-likelihood of this function to get our loss function, which we want to minimize   
  
Let *t* be the actual output vector from our training data, for a particular center word. It will have 1’s at the positions of context words and 0’s at all other places. tj\*c are the 1’s of the context words.   
We can multiply with   
  
Solving this equation we get our loss function as –**

**Back Propagation:**

**The parameters to be adjusted are in the matrices W and W’, hence we have to find the partial derivatives of our loss function with respect to W and W’ to apply the gradient descent algorithm.   
We have to find   
Now, Finding**

1. **Explain Glove Embeddings.**

**GloVe Embeddings Applications**

**GloVe embeddings are a popular option for representing words in text data and have found applications in various natural language processing (NLP) tasks. The following are some typical uses for GloVe embeddings:**

**Text Classification:**

* **GloVe embeddings can be utilised as features in**[**machine learning**](https://www.geeksforgeeks.org/machine-learning/)**models for sentiment analysis, topic classification, spam detection, and other applications.**

**Named Entity Recognition (NER):**

* **By capturing the semantic relationships between words and enhancing the model’s capacity to identify entities in text, GloVe embeddings can improve the performance of NER systems.**

**Machine Translation:**

* **GloVe embeddings can be used to represent words in the source and target languages in machine translation systems, which aim to translate text from one language to another, thereby enhancing the quality of the translation.**

**Question Answering Systems:**

* **To help models comprehend the context and relationships between words and produce more accurate answers, GloVe embeddings are used in question-answering tasks.**

**Document Similarity and Clustering:**

* **GloVe embeddings enable applications in information retrieval and document organization by measuring the semantic similarity between documents or grouping documents according to their content.**

**Word Analogy Tasks:**

* **In word analogy tasks, GloVe embeddings frequently yield good results. For instance, the generated vector for “king-man + woman” might resemble the “queen” vector, demonstrating the capacity to recognize semantic relationships.**

**Semantic Search:**

* **In semantic search applications, where retrieving documents or passages according to their semantic relevance to a user’s query is the aim, GloVe embeddings are helpful.**