**1. Explain One-Hot Encoding ?**

**Ans:**

**One Hot Encoding** is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.

One hot encoding is the most widespread approach, and it works very well unless your categorical variable takes on a large number of values (i.e. you generally won't it for variables taking more than 15 different values. It'd be a poor choice in some cases with fewer values, though that varies.)

One hot encoding creates new (binary) columns, indicating the presence of each possible value from the original data. Let's work through an example.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **COLOR** |  | **RED** | **YELLOW** | **GREEN** |
| RED |  | 1 | 0 | 0 |
| RED |  | 1 | 0 | 0 |
| YELLOW |  | 0 | 1 | 0 |
| GREEN |  | 0 | 0 | 1 |
| YELLOW |

The values in the original data are Red, Yellow and Green. We create a separate column for each possible value. Wherever the original value was Red, we put a 1 in the Red column.

**2. Explain Bag of Words ?**

**Ans:**

The **bag-of-words (BOW)** model is a representation that turns arbitrary text into fixed-length vectors by counting how many times each word appears. This process is often referred to as **vectorization**.

Suppose we wanted to vectorize the following:  
the cat sat  
the cat sat in the hat  
the cat with the hat

We’ll refer to each of these as a text document.

* We first define our vocabulary, which is the set of all words found in our document set. The only words that are found in the 3 documents above are: the, cat, sat, in, the, hat, and with.
* To vectorize our documents, all we have to do is count how many times each word appears:
* Now we have length-6 vectors for each document!
  + the cat sat: [1, 1, 1, 0, 0, 0]
  + the cat sat in the hat: [2, 1, 1, 1, 1, 0]
  + the cat with the hat: [2, 1, 0, 0, 1, 1]
* The Problem with BOW is, it does not preserve contextual information among the words. Notice that we lose contextual information, e.g. where in the document the word appeared, when we use BOW. It’s like a literal bag-of-words: it only tells you what words occur in the document, not where they occurred.

**3. Explain Bag of N-Grams ?**

**Ans:**

A **Bag-of-N-Grams model** is a way to represent a document, similar to a [bag-of-words][/terms/bag-of-words/] model.

A bag-of-n-grams model represents a text document as an unordered collection of its n-grams.

For example, let’s use the following phrase and divide it into bi-grams (n=2).

**James is the best person ever.**

Becomes

* **<start>James**
* **James is**
* **is the**
* **the best**
* **best person**
* **person ever.**
* **ever.<end>**

In a typical bag-of-n-grams model, these 6 bigrams would be a sample from a large number of bigrams observed in a corpus. And then James is the best person ever would be encoded in a representation showing which of the corpus’s bigrams were observed in the sentence.

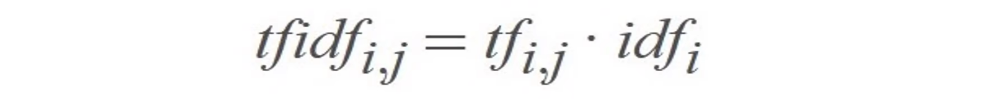
A bag-of-n-grams model has the simplicity of the bag-of-words model, but allows the preservation of more word locality information.

**4. Explain TF-IDF ?**

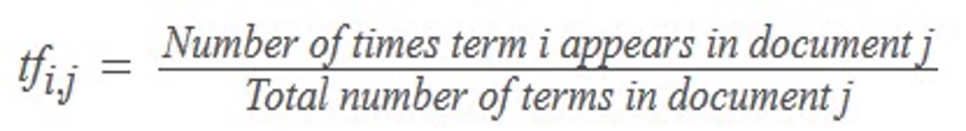
**Ans:**

TF-IDF stands for “Term Frequency – Inverse Document Frequency.” It reflects how important a word is to a document in a collection or corpus. This technique is often used in information retrieval and text mining as a weighing factor.

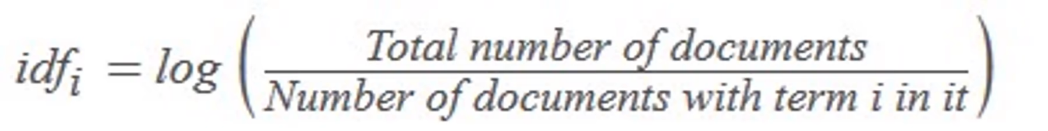
TF-IDF is composed of two terms:



**Term Frequency (TF):** The number of times a word appears in a document divided by the total number of words in that document.



**Inverse Document Frequency (IDF):** The logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.



So, essentially, the TF-IDF value increases as the word’s frequency in a document (TF) increases. However, this is offset by the number of times the word appears in the entire collection of documents or corpus (IDF).

**Applications of TF-IDF:**

Determining how relevant a word is to a document, or TD-IDF, is useful in many ways, for example:

**Information retrieval:**

TF-IDF was invented for document search and can be used to deliver results that are most relevant to what you’re searching for. Imagine you have a search engine and somebody looks for LeBron. The results will be displayed in order of relevance. That’s to say the most relevant sports articles will be ranked higher because TF-IDF gives the word LeBron a higher score.

It’s likely that every search engine you have ever encountered uses TF-IDF scores in its algorithm.

**Keyword Extraction:**

TF-IDF is also useful for extracting keywords from text. How? The highest scoring words of a document are the most relevant to that document, and therefore they can be considered keywords for that document. Pretty straightforward.

**5. What is OOV problem?**

**Ans:**

Out-of-vocabulary (OOV) are terms that are not part of the normal lexicon found in a natural language processing environment.

* We often use word2vec or glove to process documents to create word vector or word embedding.
* However, we may ignore some words that appear rarely in documents, which may cause OOV problem.
* Meanwhile, we may use some pre-trained word representation file, which may do not contain some words in our data set. It also can cause OOV problem.

**How to fix OOV problem?**  
There are three main ways that often be used in AI application.

* **Ignoring them**
  + Generally, words that are out of vocabulary often appear rarely, the will contribute less to our model. The performance of our model will drop scarcely, it means we can ignore them.
* **Replacing them using <UNK>**
  + We can replace all words that are out of vocabulary by using word <UNK>.
* **Initializing them by a uniform distribution with range [-0.01, 0.01]**
  + Out-Of-Vocabulary (OOV) words can be initialized from a uniform distribution with range [-0.01, 0.01]. We can use this uniform distribution to train our model.

**6. What are word embeddings?**

**Ans:**

It 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 meaning to have a similar representation. They can also approximate meaning. A word vector with 50 values can represent 50 unique features.

**Features:** Anything that relates words to one another. Eg: Age, Sports, Fitness, Employed etc. Each word vector has values corresponding to these features.

**Goal of Word Embeddings**

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

How are Word Embeddings used?

* They are used as input to machine learning models.  
  Take the words —-> Give their numeric representation —-> Use in training or inference
* To represent or visualize any underlying patterns of usage in the corpus that was used to train them.

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

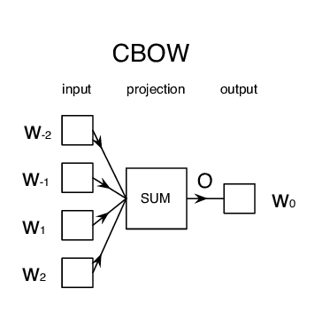
**Ans:**

The word2vec model has two different architectures to create the word embeddings. They are:

* Continuous bag of words(CBOW)
* Skip-gram model

**Continuous bag of words(CBOW):**

The CBOW model tries to understand the context of the words and takes this as input. It then tries to predict words that are contextually accurate. Let us consider an example for understanding this. Consider the sentence: It is a pleasant day and the word pleasant goes as input to the neural network. We are trying to predict the word day here. We will use the one-hot encoding for the input words and measure the error rates with the one-hot encoded target word. Doing this will help us predict the output based on the word with least error.



**The Model Architecture:**  The CBOW model architecture is as shown above. The model tries to predict the target word by trying to understand the context of the surrounding words. Consider the same sentence as above, ‘It is a pleasant day’. The model converts this sentence into word pairs in the form (contextword, targetword). The user will have to set the window size. If the window for the context word is 2 then the word pairs would look like this: ([it, a], is), ([is, pleasant], a),([a, day], pleasant). With these word pairs, the model tries to predict the target word considered the context words.

If we have 4 context words used for predicting one target word the input layer will be in the form of four 1XW input vectors. These input vectors will be passed to the hidden layer where it is multiplied by a WXN matrix. Finally, the 1XN output from the hidden layer enters the sum layer where an element-wise summation is performed on the vectors before a final activation is performed and the output is obtained.

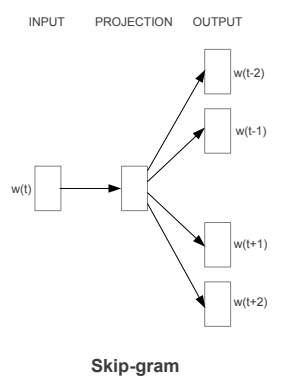
**8. Explain SkipGram**

**Ans:**

The word2vec model has two different architectures to create the word embeddings. They are:

* Continuous bag of words(CBOW)
* Skip-gram model

**The Skip-gram Model:** The Skip-gram model architecture usually tries to achieve the reverse of what the CBOW model does. It tries to predict the source context words (surrounding words) given a target word (the center word). Considering our simple sentence from earlier, “the quick brown fox jumps over the lazy dog”. If we used the CBOW model, we get pairs of (context\_window, target\_word) where if we consider a context window of size 2, we have examples like ([quick, fox], brown), ([the, brown], quick), ([the, dog], lazy) and so on. Now considering that the skip-gram model’s aim is to predict the context from the target word, the model typically inverts the contexts and targets, and tries to predict each context word from its target word. Hence the task becomes to predict the context [quick, fox] given target word ‘brown’ or [the, brown] given target word ‘quick’ and so on. Thus the model tries to predict the context\_window words based on the target\_word.



We need to model this Skip-gram architecture now as a deep learning classification model such that we take in the target word as our input and try to predict the context words. This becomes slightly complex since we have multiple words in our context. We simplify this further by breaking down each (target, context\_words) pair into (target, context) pairs such that each context consists of only one word. Hence our dataset from earlier gets transformed into pairs like (brown, quick), (brown, fox), (quick, the), (quick, brown) and so on. But how to supervise or train the model to know what is contextual and what is not?

For this, we feed our skip-gram model pairs of (X, Y) where X is our input and Y is our label. We do this by using [(target, context), 1] pairs as positive input samples where target is our word of interest and context is a context word occurring near the target word and the positive label 1 indicates this is a contextually relevant pair. We also feed in [(target, random), 0] pairs as negative input samples where target is again our word of interest but random is just a randomly selected word from our vocabulary which has no context or association with our target word. Hence the negative label 0indicates this is a contextually irrelevant pair. We do this so that the model can then learn which pairs of words are contextually relevant and which are not and generate similar embeddings for semantically similar words.

**9. Explain Glove Embeddings.**

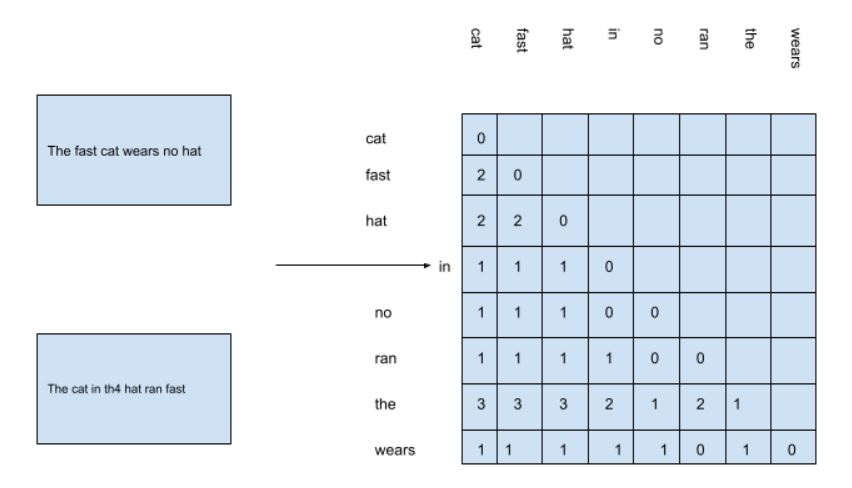
**Ans:**

GloVe word is a combination of two words- Global and Vectors. In-depth, the GloVe is a model used for the representation of the distributed words. This model represents words in the form of vectors using an unsupervised learning algorithm. This unsupervised learning algorithm maps the words into space where the semantic similarity between the words is observed by the distance between the words. These algorithms perform the Training of a corpus consisting of the aggregated global word-word co-occurrence statistics, and the result of the training usually represents the subspace of the words in which our interest lies. It is developed as an open-source project at Stanford and was launched in 2014.

**Training Procedure for GloVe Model:**

The glove model uses the matrix factorization technique for word embedding on the word-context matrix. It starts working by building a large matrix which consists of the words co-occurrence information, basically, The idea behind this matrix is to derive the relationship between the words from statistics. The co-occurrence matrix tells us the information about the occurrence of the words in different pairs.

Let’s have a look at the below image:



Here we have a matrix which is formed by considering the unique words from two sentences given on the left side of the image and in this we can see that the word “the” has accrued 3 times with cat fast and hat so the value in the matrix for the combination is 3 every time.