# FastText

**Why FastText? - Features of FastText (basically fast - like really fast):**

1. We can train fastText on more than one billion words in less than ten minutes using a standard multicore CPU.
2. We can classify half a million sentences among 312K classes in less than a minute.

**FastText - working techniques**

- linear models with a rank constraint
- a fast loss approximation

**FastText - Model architecture**

Sentences -> bag of words(BoW) -> train a linear classifier(LR or SVM) -> Predict

**STEP1:** Character n-gram embeddings: FastText starts by representing each word as a bag of character n-grams, where n can range from 1 to some maximum value (e.g., 6). For example, the word "cat" would be represented as a bag of character 3-grams: {"cat", "at#", "t# "} (where "#" represents the end of a word). Each n-gram is mapped to a high-dimensional embedding vector using a lookup table.

**STEP2:** Word embeddings: The word embedding vector is obtained by summing up the character n-gram embeddings of the word. For example, the word embedding for "cat" would be the sum of the embeddings for "cat", "at#", and "t# ".

**STEP3:**Training: FastText trains a supervised classifier (e.g., logistic regression, SVM) on the word embeddings to classify text. During training, the algorithm uses a technique called negative sampling, where it randomly samples negative examples to balance the dataset and improve the accuracy of the classifier.

**Negative sampling:** During training, FastText uses a technique called negative sampling to improve the accuracy of the classifier. Negative sampling involves randomly sampling negative examples from the training data to balance the dataset and reduce the risk of overfitting. For each positive example (i.e., a correctly labeled example), FastText randomly selects a fixed number of negative examples (i.e., incorrectly labeled examples) and trains the classifier to distinguish between the positive and negative examples. By training on a mix of positive and negative examples, the classifier is able to learn more robust and generalizable patterns in the data.

**STEP4:**Prediction: To predict the label of a new text, FastText averages the word embeddings of the words in the text and feeds them into the trained classifier.

Overall, FastText is a powerful algorithm for text classification that can handle large datasets and achieve high accuracy with relatively small amounts of training data. It has become a popular choice for a wide range of natural language processing tasks, such as sentiment analysis, topic classification, and spam detection.

**Sources:**

First, we find vector representations such that text and its associated labels have similar vectors. In simple words, the vector corresponding to the text is closer to its corresponding label.

To find the probability score of a correct label given it's associated text we use the softmax function:

Here travel is the label and car is the text associated to it.
To maximize this probability of the correct label we can use the Gradient Descent algorithm.

This is quite computationally expensive because for every piece of text not only we have to get the score associated with its correct label but we need to get the score for every other label in the training set. This limits the use of these models on very large datasets.

FastText solves this problem by using a hierarchical classifier to train the model.

Hierarchical Classifier used by FastText:

In this method, it represents the labels in a binary tree. Every node in the binary tree represents a probability. A label is represented by the probability along the path to that given label. This means that the leaf nodes of the binary tree represent the labels.

FastText uses the Huffman algorithm to build these trees to make full use of the fact that classes can be imbalanced. Depth of the frequently occurring labels is smaller than the infrequent ones.

Using a binary tree speed up the time of search as instead of having to go through all the different elements you just search for the nodes. So now we won't have to compute the score for every single possible label, and we will only be calculating just the probability on each node in the path to the one correct label. Hence this method vastly reduces the time complexity of training the model.

Increasing the speed does not sacrifice the accuracy of the model.

When we have unlabeled dataset FastText uses the N-Gram Technique to train the model. Let us understand more in detail how this technique works-
Let us consider a word from our dataset, for example: "kingdom". Now it will take a look at the word "kingdom" and will break it into its n-gram components as-

kingdom = ['k','in','kin','king','kingd','kingdo','kingdom',...]
These are some n-gram components for the given words. There will be many more components for this word but only a few are stated here just to get an idea. The size of the n-gram components can be chosen as per your choice. The length of n-grams can be between the minimum and the maximum number of characters selected. You can do so by using the -minn and -maxn flags respectively.

Note: When your text is not words from a particular language then using n-grams won't make sense. for example: when the corpus contains ids it will not be storing words but numbers and special characters. In this case, you can turn off the -gram embeddings by selecting the -minn and -maxn parameters as 0.

When the model updates, fastText learns the weights for every n-gram along with the entire word token.

In this manner, each token/word will be expressed as the sum and an average of its n-gram components.

Word vectors generated through fastText hold extra information about their sub-words. As in the above example, we can see that one of the components for the word "kingdom" is the word "king". This information helps the model build semantic similarity between the two words.

It also allows for capturing the meaning of suffixes/prefixes for the given words in the corpus.
It allows for generating better word embeddings for different or rare words as well.
It can also generate word embeddings for out of vocabulary(OOV) words.
While using fastText even if you don’t remove the stopwords still the accuracy is not compromised. You can perform simple pre-processing steps on your corpus if you fell like.
As fastText has the feature of providing sub-word information, it can also be used on morphologically rich languages like Spanish, French, German, etc.