**MACHINE LEARNING POWERED AUTO COMPLETION FEATURE**

**Dhruv Bajaj**

You all may have wondered how Google is able to predict the text that you are going to type or how your gmail gives suggestions for text or how our keyboards suggest alternate words or words that fit sentences or correct spellings of uncommon words like proper nouns. All of these things are done using state-of-the-art neural networks. All of them are powered by artificial intelligence and machine learning from the autocorrect to the text suggestion or alternate word suggestion.

Machine learning algorithms have been contributing to power such useful autocomplete features that have surely helped us to move faster in this fast growing world. All these tools have become part of our life. Let's dive in and look into how this technology works in real time and how we can make this technology understood. This article talks about the abstract idea of the network, its implementation and results.

**NEURAL NETWORKS**

Lets understand neural networks and how it's used in autocompletion of text.

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. A neural network contains layers of interconnected nodes. Each node is a perceptron (in other words, a single-layer neural network) and is similar to a multiple linear regression.

In simple words, This means the perceptron is used to classify data into two parts, hence it is also a binary classifier algorithm. The perceptron feeds the signal produced by a multiple linear regression into an activation function that may be nonlinear.

In autotext generation, Natural Language Processing (NLP) which is a very common neural networking technique, is used.

There are two main phases to natural language processing:

* Data preprocessing
* Algorithm development.

Data preprocessing involves preparing and "cleaning" text data for machines to be able to analyze it. Pre processing puts the data in a workable form and highlights features in the text that an algorithm can work with. There are several ways this can be done, including:

* **Tokenization** (to split words into units)

Tokenization- as the word may suggest- is the method of separating a piece of text into smaller units called token- either into words, characters, or subwords.

Code:-

Tokenize paragraph into sentences

**import** **nltk**

nltk.download('punkt')

**from** **nltk.tokenize** **import** sent\_tokenize

text = "Hello people. My name is Dhruv. How are you?"

sent\_tokenize(text)

**Tokenize sentence in words and grammatical aspect of words**

**import** **spacy**

**from** **spacy.lang.en.examples** **import** sentences

nlp = spacy.load("en\_core\_web\_sm") *# using preloaded data set*

doc = nlp(sentences[1])

print(doc.text)

**for** token **in** doc:

print(token.text, token.pos\_, token.dep\_)

Output :-

['Hello people.', 'My name is Dhruv.', 'How are you?']

Autonomous cars shift insurance liability toward manufacturers

Autonomous ADJ amod

cars NOUN nsubj

shift VERB ROOT

insurance NOUN compound

liability NOUN dobj

toward ADP prep

manufacturers NOUN pobj

* **Normalization** (reducing the randomness of text)

Reducing randomness, in terms of text normalization, means reducing words to their common contractions *(eg: we will- we’ll)* , and compressing them to a common, widely accepted base-form.

There are various steps in text pre-processing, some of which we’re going to use in the following text defining ‘Stock Market’:

*“The stock market refers to the collection of markets and exchanges where regular activities of buying, selling, and issuance of shares of publicly-held companies take place. Such financial activities are conducted through institutionalized formal exchanges or over-the-counter (OTC) marketplaces which operate under a defined set of regulations. There can be multiple stock trading venues in a country or a region which allow transactions in stocks and other forms of securities.”*

The steps are:

* Converting the string to one case(either lower or upper case),
* Removing all numbers (if not essential)
* Removing punctuations
* Removing white spaces
* Removing stop words

Code:-

**import** **re**

string = " The stock market refers to the collection of markets and exchanges where regular activities of buying, selling, and issuance of shares of publicly-held companies take place. Such financial activities are conducted through institutionalized formal exchanges or over-the-counter (OTC) marketplaces which operate under a defined set of regulations. There can be multiple stock trading venues in a country or a region which allow transactions in stocks and other forms of securities.."

lower\_string = string.lower()

*# removing numbers*

no\_number\_string = re.sub(r'\d+','',lower\_string)

*# removing all punctuations*

no\_punc\_string = re.sub(r'[^\w\s]','', no\_number\_string)

print(no\_punc\_string)

*# removing white spaces*

no\_wspace\_string = no\_punc\_string.strip()

print(no\_wspace\_string)

Output:-

the stock market refers to the collection of markets and exchanges where regular activities of buying selling and issuance of shares of publiclyheld companies take place such financial activities are conducted through institutionalized formal exchanges or overthecounter otc marketplaces which operate under a defined set of regulations there can be multiple stock trading venues in a country or a region which allow transactions in stocks and other forms of securities

* **Stopwords :-**

Stopwords are the most common words in any natural language, eg: “the”, “is”, “in”, “for”, “where”, “when”, “to”, “at” etc. For the purpose of analyzing text data and building NLP models, these stopwords might not add much value to the meaning of the document, hence we tend to remove them during preprocessing.

CODE :-

*# download stpwords*

**import** **nltk**

nltk.download('stopwords')

*# import nltk for stopwords*

**from** **nltk.corpus** **import** stopwords

stop\_words = set(stopwords.words('english'))

print(stop\_words)

*# assign string*

no\_wspace\_string=""" the stock market refers to the collection of markets and exchanges where regular activities of buying selling and issuance of shares of publiclyheld companies take place such financial activities are conducted through institutionalized formal exchanges or overthecounter otc marketplaces which operate under a defined set of regulations there can be multiple stock trading venues in a country or a region which allow transactions in stocks and other forms of securities

the stock market refers to the collection of markets and exchanges where regular activities of buying selling and issuance of shares of publiclyheld companies take place such financial activities are conducted through institutionalized formal exchanges or overthecounter otc marketplaces which operate under a defined set of regulations there can be multiple stock trading venues in a country or a region which allow transactions in stocks and other forms of securities"""

*# convert string to list of words*

lst\_string = [no\_wspace\_string][0].split()

print(lst\_string)

*# remove stopwords*

no\_stpwords\_string=""

**for** i **in** lst\_string:

**if** **not** i **in** stop\_words:

no\_stpwords\_string += i+' '

*# removing last space*

no\_stpwords\_string = no\_stpwords\_string[:-1]

print(no\_stpwords\_string)

Output:



Here we can observe that the text in question has been reduced to its canonical form using these steps.

The Algorithm development phase is facilitated by python libraries- the most commonly suggested being fast-autocomplete. This library was developed after it was concluded that Elasticsearch's (an open-source search and analytics engine) Autocomplete feature isn’t as fast and efficient as is required.

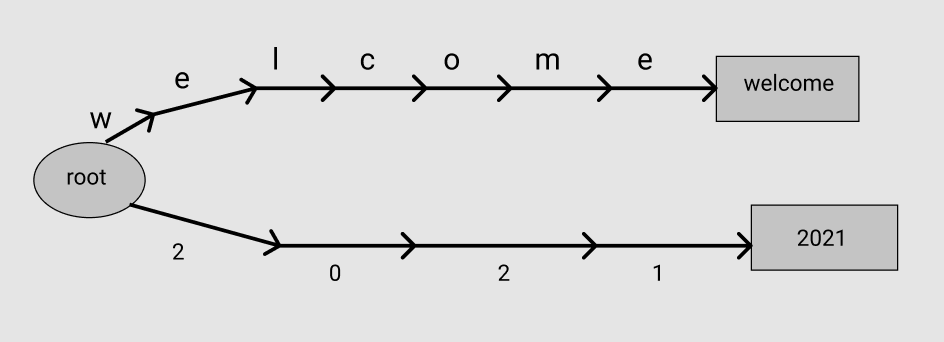
With Fast Autocomplete, an improvement of 3-4x in performance was observed and errors went down to zero!

Below are the various algorithms that this library works on:

**1. Trie-Tree (Radix Tree, Prefix Tree)**

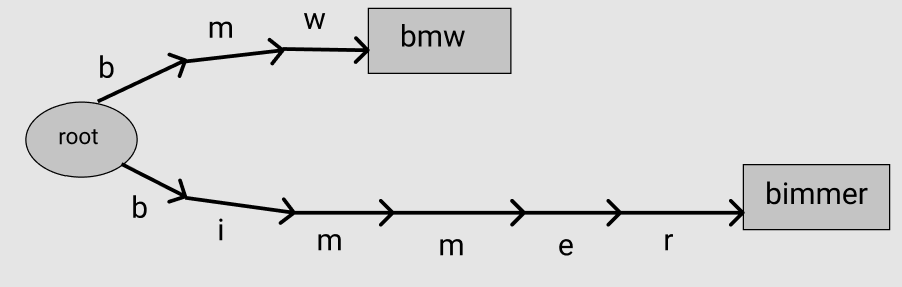
As the name might’ve suggested, a ‘Trie’ of a ‘Trie-Tree’ is a tree based data structure, but instead of storing numerical values, it stores sets of strings. It can search a word in the dictionary with the help of the word's prefix. Each node can have at maximum 26 children (from A-Z)

It is structured such that a node is a word or a flag that means “add up the edges”, and an edge is a letter of a word. As the user types each letter, the algorithm traverses through the tree, moving from node to node.



**2. Directed Acyclic Word Graph (DAWG)**

A DAWG is very similar to a trie-tree except each time we complete a branch in the tree, it is checked for duplicate nodes. If a duplicate is found, the branches are merged. This way it becomes an acyclic graph.



**3. Markov Chain**

Markov chains are a way to statistically model random processes. They essentially consist of various ‘states’ of a particular process or a system and give us the probability of transitioning to the next state from the current state in the system.

A “corpus” is a structured and well-defined dataset containing text. Markov chains help us in predicting words by taking a corpus of text and keeping a track of the number of times a pair of words succeed/follow each other. Imagine this as a system where states are words. Based on this it is able to determine the probability of which word follows which and hence incentivizes the autocomplete process.

**Fast-autocomplete library**

We looked at the algorithms and now know the concept on which the library works. Let's dive into the python library and work on real time applications!

**pip install fast-autocomplete**  
Download the library on your local computer or virtual environment

Let's take an example text data of my friends, their hobbies and the places where they live.

Code to create a csv of my friends and their details:

**import** **pandas** **as** **pd**

friend\_name=[]

friend\_hobbies=[]

friend\_location=[]

**for** i **in** range (10):

name=input()

hobbies= input()

location= input()

friend\_name.append(name)

friend\_hobbies.append(hobbies)

friend\_location.append(location)

data={

'friend':friend\_name,

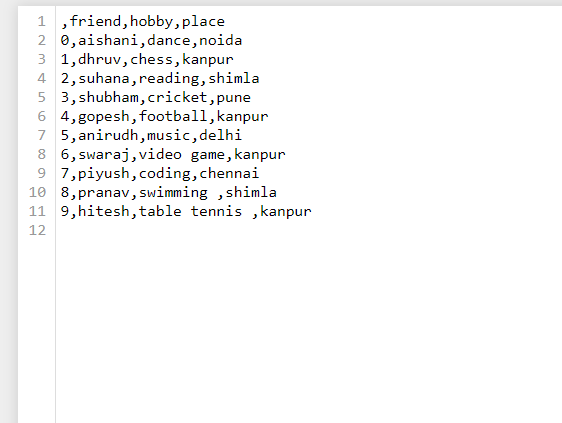
'hobby':friend\_hobbies,

'Place':friend\_location

}

df = pd.DataFrame(data)

df.to\_csv('friends.csv', mode='a', header=**False**)



We will convert this csv to a dictionary of words and their context.

**import** **csv**

**from** **fast\_autocomplete.misc** **import** read\_csv\_gen

**def** get\_words(path):

csv\_gen = read\_csv\_gen(path, csv\_func=csv.DictReader)

words = {}

**for** line **in** csv\_gen:

friend = line['friend']

hobby = line['hobby']

place = line['place']

**if** friend != place:

local\_words = [place, hobby,'**{}** **{}** **{}**'.format(friend, hobby, place)]

**while** local\_words:

word = local\_words.pop()

**if** word **not** **in** words:

words[word] = {}

**if** friend != hobby:

local\_words = [hobby,place, '**{}** **{}** **{}**'.format(friend, hobby ,place)]

**while** local\_words:

word = local\_words.pop()

**if** word **not** **in** words:

words[word] = {}

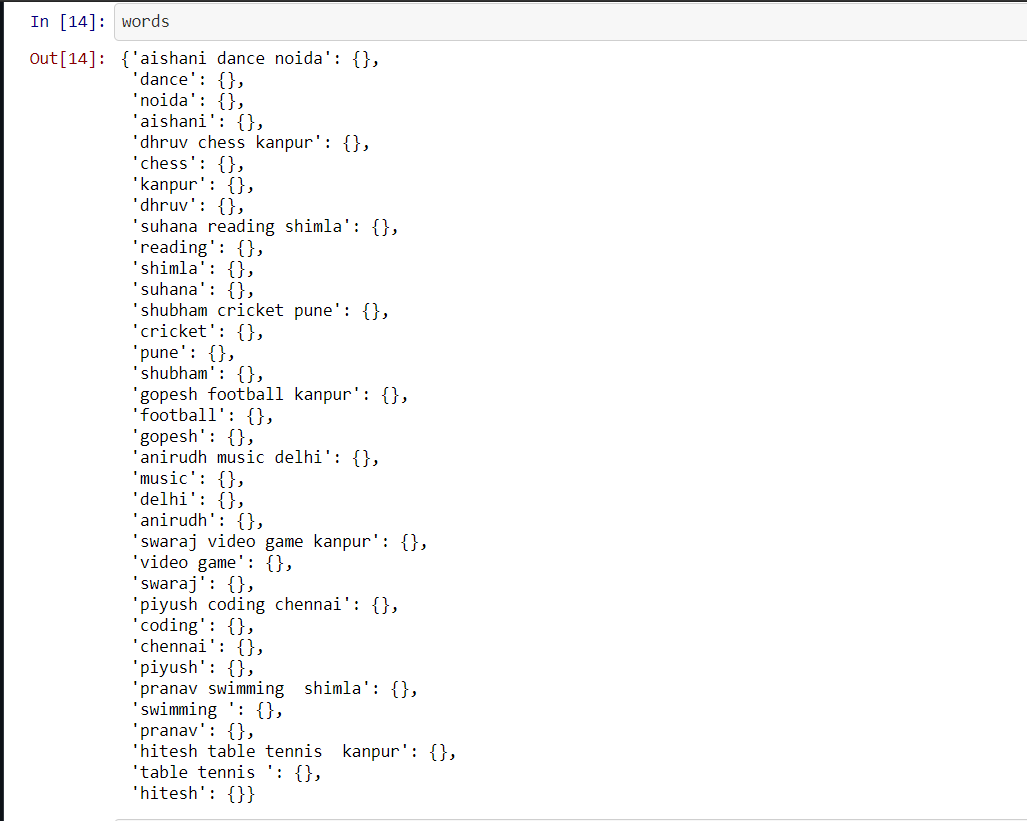
**if** friend **not** **in** words:

words[friend] = {}

**return** words

words = get\_words(r'C:\Users\Dhruv Bajaj\friends.csv')*# add the path file of csv created*

Output:-



Synonyms command used here is to add typing error or similar meaning word to dataset. Here, I've added my friends' nickname under their name .

synonyms = {

"aishani": ["aishu","aishi","ishi","ginni","giny"],

"dhruv": ["dhru", "duv","dhruvi",],

"suhana": ["su", "suzi","suz"],

"shubham": ["shubh"],

"gopesh": ["gopu"],

"swaraj": ["raj"],

"anirudh": ["ani"]

}

Lets run our dataset on the algorithm:

**from** **fast\_autocomplete** **import** AutoComplete

autocomplete = AutoComplete(words=words, synonyms=synonyms)

autocomplete.search(word='aishani', max\_cost=5, size=5)

Output:- [['aishani'], ['aishani dance noida']]



Now you can search!

* word: the word to return autocomplete results for
* max\_cost: Maximum Levenshtein edit distance to be considered when calculating results
* size: The max number of results to return

**Conclusion:-**

It is evident that Machine Learning related technologies have, and are constantly proving to make the simplest of tasks easier for mankind, and a reliable auto complete feature is one among many!

For more information on code refer Github link:-

<https://github.com/DhruvBajaj01/MSRF_Internship/blob/main/autocomplete%20library.ipynb>

**References :-**

* [**https://pypi.org/project/autocomplete/**](https://pypi.org/project/autocomplete/)
* [**https://pypi.org/project/fast-autocomplete/**](https://pypi.org/project/fast-autocomplete/)
* [**https://zepworks.com/posts/you-autocomplete-me/**](https://zepworks.com/posts/you-autocomplete-me/)
* [**https://spacy.io**](https://spacy.io)