**CONTENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **Name of experiment** | **Page no.** | **Sign** |
| 1. | To learn about morphological features of a word by analyzing it. | 3 |  |
| 2. | To generate word forms from root and suffix information. | 5 |  |
| 3. | Understanding the morphology of a word by the use of Add-delete table. | 7 |  |
| 4. | To learn how to calculate bigrams from a given corpus and calculate probability of a sentence. | 8 |  |
| 5. | To learn how to apply add-one smoothing on sparse bigram table. | 10 |  |
| 6. | To know the importance of context and size of training corpus in learning parts of speech. | 12 |  |
| 7. | To understand the concept of chunking and get familiar with the basic chunk tagset. | 14 |  |
| 8. | To know the importance of selecting proper features for training a model and size of training corpus in learning how to do chunking. | 17 |  |
| 9. | To study Natural language processing library in python: NLTK/ any other. | 19 |  |
| 10. | To study Natural language processing CORPUS/Treebank (e.g. Wordnet/anyother). | 22 |  |
| 11. | Program to calculate TF/IDF, feature extraction and finding unique words. | 27 |  |
| 12. | Program for data collection , remove stop words ,data generation , tensor flow graph and word vector . | 29 |  |

**EXPERIMENT-1**

**AIM:** *To learn about morphological features of a word by analyzing it.*

**THEORY**

Analysis of a word into root and affix(es) is called as Morphological analysis of a word. It is mandatory to identify root of a word for any natural language processing task. A root word can have various forms. For example, the word 'play' in English has the following forms: 'play', 'plays', 'played' and 'playing'. Hindi shows more number of forms for the word 'खेल' (khela) which is equivalent to 'play'. The forms of 'खेल'(khela) are the following:

खेल(khela), खेला(khelaa), खेली(khelii), खेलूंगा(kheluungaa), खेलूंगी(kheluungii), खेलेगा(khelegaa), खेलेगी(khelegii), खेलते(khelate), खेलती(khelatii), खेलने(khelane), खेलकर(khelakar)

**Types of Morphology**

Morphology is of two types,

1. Inflectional morphology

Deals with word forms of a root, where there is no change in lexical category. For example, 'played' is an inflection of the root word 'play'. Here, both 'played' and 'play' are verbs.

1. Derivational morphology

Deals with word forms of a root, where there is a change in the lexical category. For example, the word form 'happiness' is a derivation of the word 'happy'. Here, 'happiness' is a derived noun form of the adjective 'happy'.

**PROCEDURE**

**STEP 1**: Select the language.

**OUTPUT**: Drop down for selecting words will appear.

**STEP 2**: Select the word.

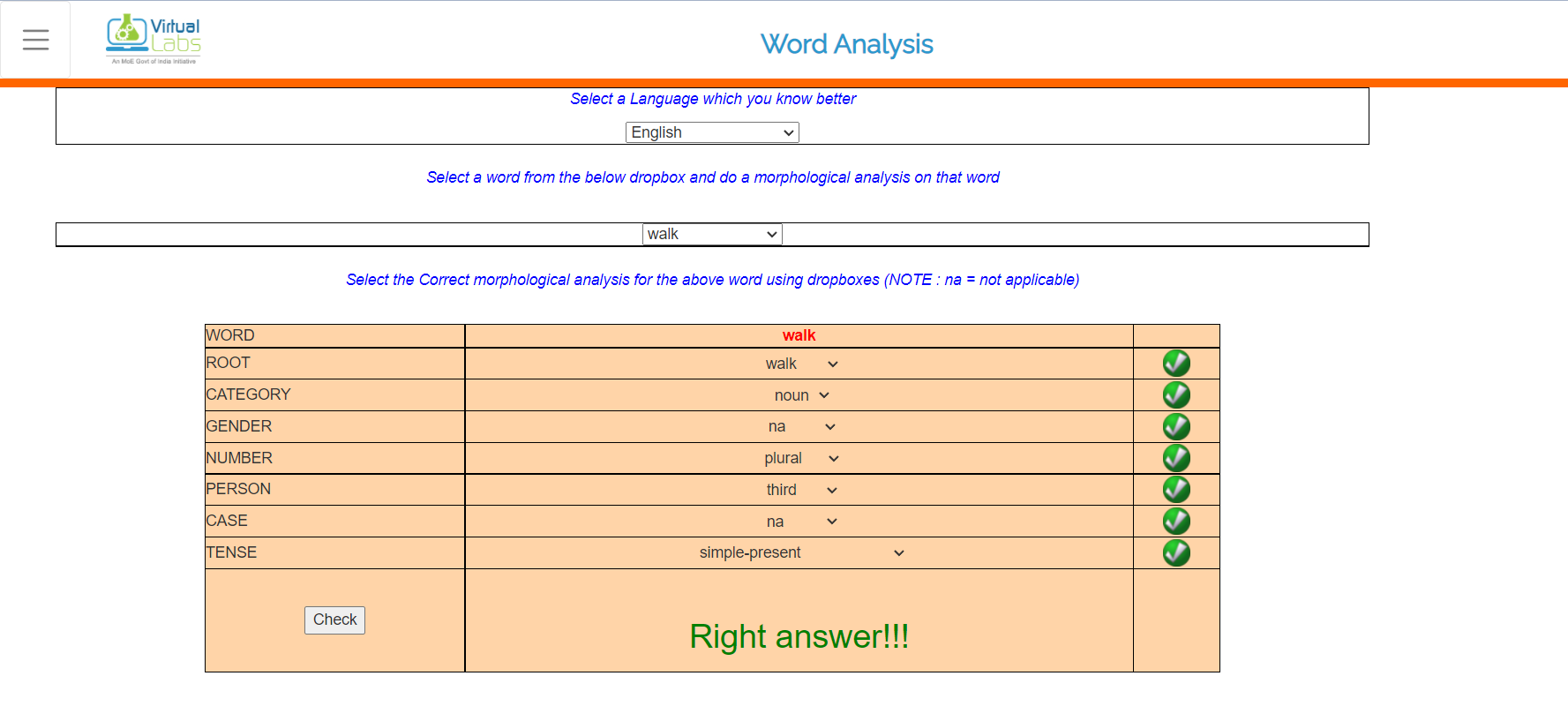
**OUTPUT**: Drop down for selecting features will appear.

**STEP 3**: Select the features.

**STEP 4**: Click "Check" button to check your answer.

**OUTPUT**: Right features are marked by tick and wrong features are marked by cross.

**SIMULATION**

****

**EXPERIMENT-2**

**AIM:** *To generate word forms from root and suffix information.*

**THEORY**

Given the root and suffix information, a word can be generated. For example,

| **Language** | **input:analysis** | **output:word** |
| --- | --- | --- |
| Hindi | rt=लड़का(ladakaa), cat=n, gen=m, num=sg, case=obl | लड़के(ladake) |
| Hindi | rt=लड़का(ladakaa), cat=n, gen=m, num=pl, case=dir | लड़के(ladake) |
| English | rt=boy, cat=n, num=pl | boys |
| English | rt=play, cat=v, num=sg, per=3, tense=pr | plays |

* Morphological analysis and generation: Inverse processes.
* Analysis may involve non-determinism, since more than one analysis is possible.
* Generation is a deterministic process. In case a language allows spelling variation, then till that extent, generation would also involve non-determinism.

**PROCEDURE**

**STEP 1**: Select the language.

**OUTPUT**: Drop downs for selecting root and other features will appear.

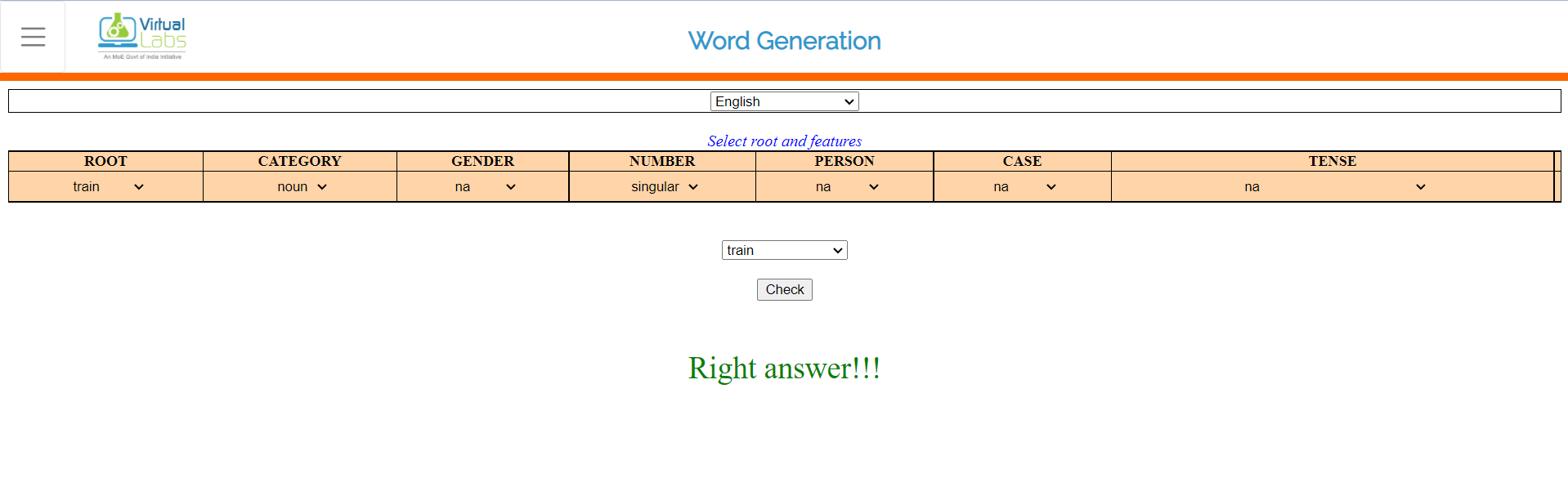
**STEP 2**: Select the root and other features.

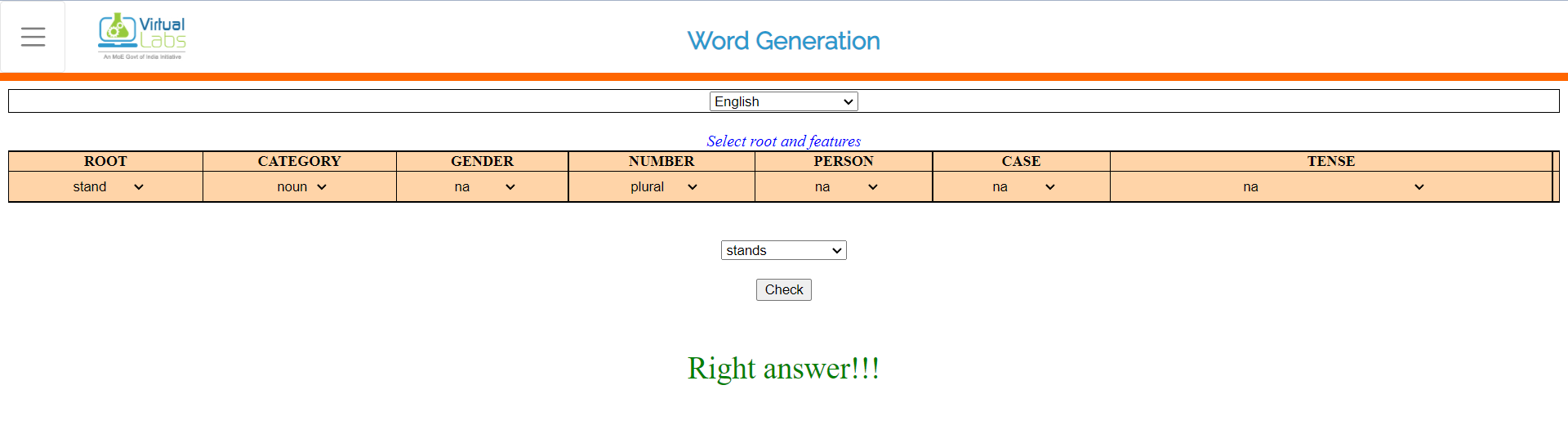
**STEP 3**: After selecting all the features, select the word corresponding above features selected.

**STEP 4**: Click the check button to see whether right word is selected or not

**OUTPUT**: Output tells whether the word selected is right or wrong

**SIMULATION**

****

****

**EXPERIMENT-3**

**AIM:** *Understanding the morphology of a word by the use of Add-delete table.*

**THEORY**

Morphology is the study of the way words are built up from smaller meaning bearing units i.e., morphemes. A morpheme is the smallest meaningful linguistic unit.

**Definition**

Morphemes are considered as smallest meaningful units of language. These morphemes can either be a root word(play) or affix(-ed). Combination of these morphemes is called morphological process. So, word "played" is made out of 2 morphemes "play" and "-ed". Thus finding all parts of a word(morphemes) and thus describing properties of a word is called "Morphological Analysis". For example, "played" has information verb "play" and "past tense", so given word is past tense form of verb "play".

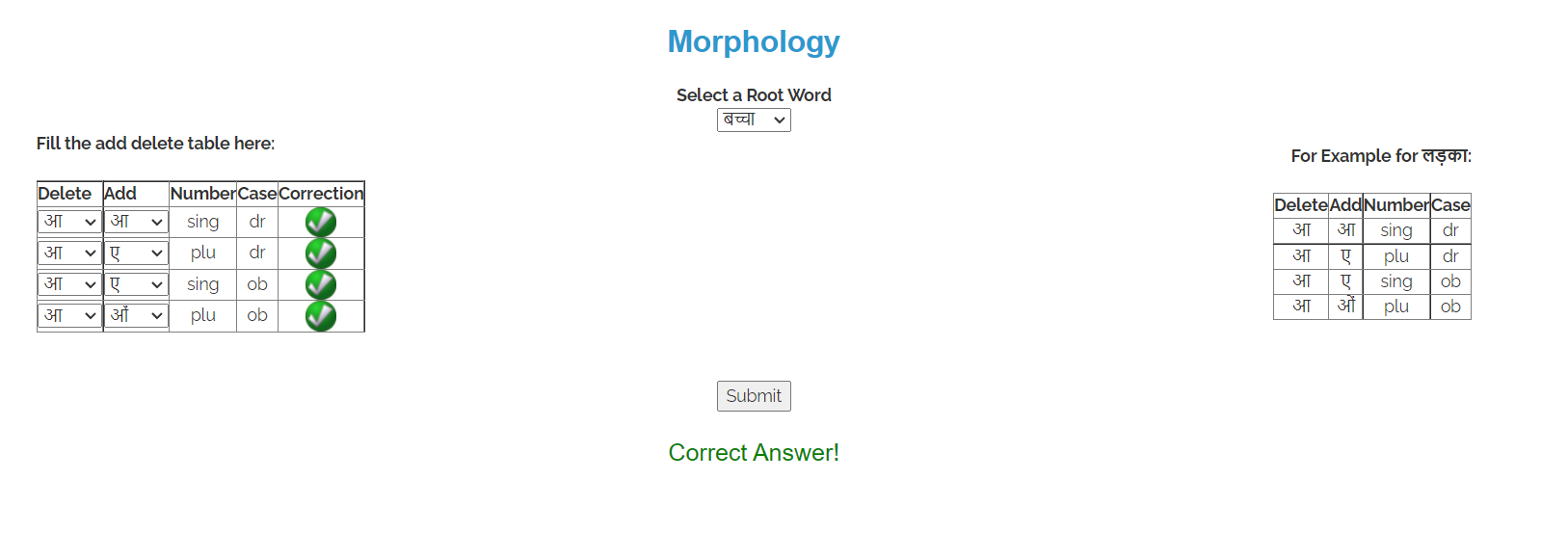
**PROCEDURE**

**STEP 1**: Select a word root.

**STEP 2**: Fill the add-delete table and submit.

**STEP 3**: If wrong, see the correct answer or repeat STEP1.

**SIMULATION**

****

**EXPERIMENT-4**

**AIM:** *To learn how to calculate bigrams from a given corpus and calculate probability of a sentence.*

**THEORY**

### Probability of a sentence

If we consider each word occurring in its correct location as an independent event,the probability of the sentences is : P(w(1), w(2)..., w(n-1), w(n))

Using chain rule: = **P(w(1))** \* **P(w(2) | w(1))** \* **P(w(3) | w(1)w(2))** ... **P(w(n) | w(1)w(2) ... w(n-1))**

### Bigrams

We can avoid this very long calculation by approximating that the probability of a given word depends only on the probability of its previous words. This assumption is called Markov assumption and such a model is called Markov model- bigrams. Bigrams can be generalized to the n-gram which looks at (n-1) words in the past. A bigram is a first-order Markov model.

Therefore , **P(w(1), w(2)..., w(n-1), w(n))** = **P(w(2)|w(1)) P(w(3)|w(2))** ... **P(w(n)|w(n-1))**

We use (eos) tag to mark the beginning and end of a sentence.

A bigram table for a given corpus can be generated and used as a lookup table for calculating probability of sentences.

**PROCEDURE**

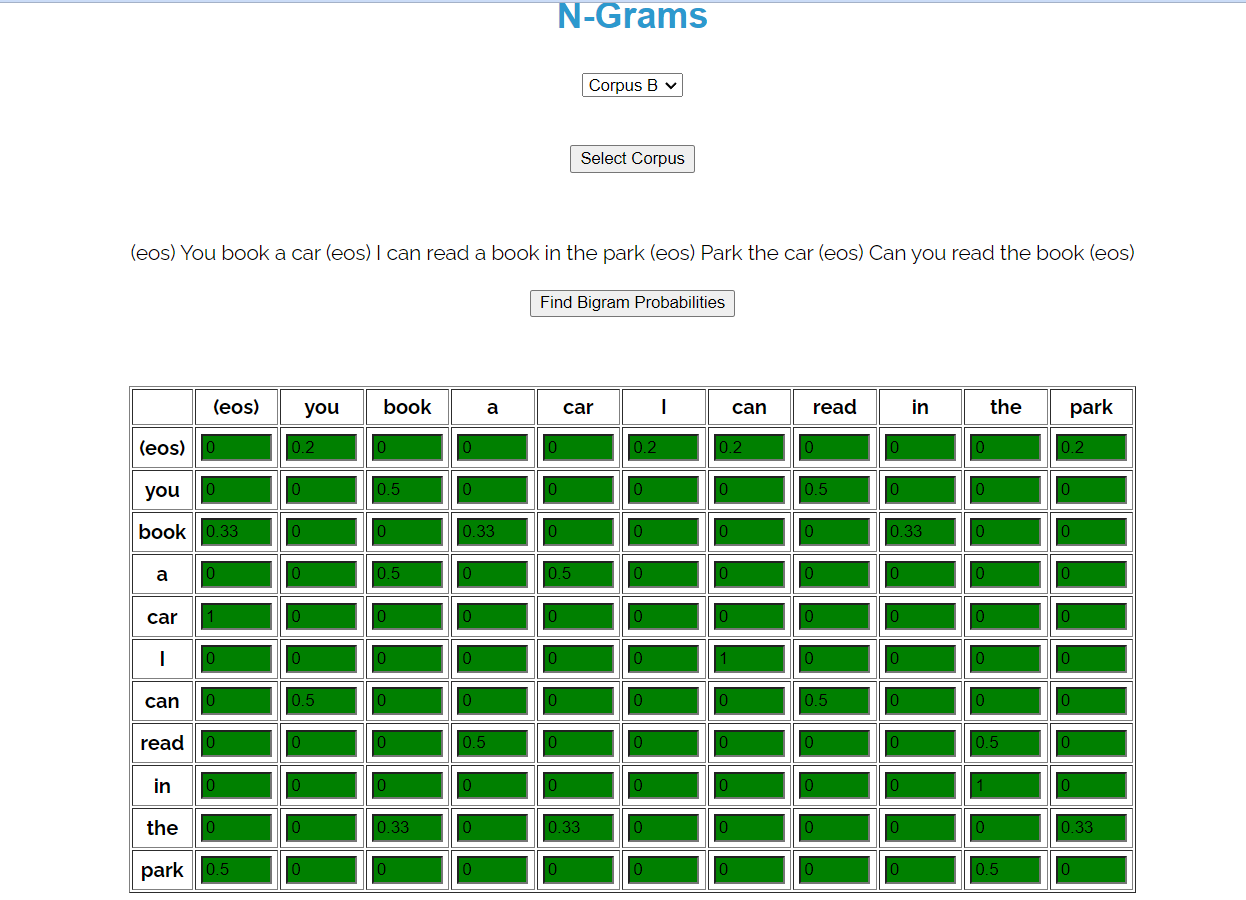
**STEP 1**: Select a corpus and click on Generate bigram table

**STEP 2**: Fill up the table that is generated and hit Submit

**STEP 3**: If incorrect (red), see the correct answer by clicking on show answer or repeat Step 2.

**STEP 4**: If correct (green), click on take a quiz and fill the correct answer

**SIMULATION**

****

**EXPERIMENT-5**

**AIM:** *To learn how to apply add-one smoothing on sparse bigram table.*

**THEORY**

The standard N-gram models are trained from some corpus. The finiteness of the training corpus leads to the absence of some perfectly acceptable N-grams. This results in sparse bigram matrices. This method tend to underestimate the probability of strings that do not occur in their training corpus.

There are some techniques that can be used for assigning a non-zero probabilty to these 'zero probability bigrams'. This task of reevaluating some of the zero-probability and low-probabilty N-grams, and assigning them non-zero values, is called smoothing. Some of the techniques are: Add-One Smoothing, Witten-Bell Discounting, Good-Turing Discounting.

### Add-One Smoothing

In Add-One smooting, we add one to all the bigram counts before normalizing them into probabilities. This is called add-one smoothing.

### Application on unigrams

The unsmoothed maximum likelihood estimate of the unigram probability can be computed by dividing the count of the word by the total number of word tokens N.

P(wx) = c(wx)/sumi{c(wi)} = c(wx)/N

Let there be an adjusted count c.ci = (c i+1 \* N/(N+V))  
where where V is the total number of word types in the language.  
Now, probabilities can be calculated by normalizing counts by N.  
pi\* = (c i+1)/(N+V)

### Application on bigrams

Normal bigram probabilities are computed by normalizing each row of counts by the unigram count:  
P(wn|wn-1) = C(wn-1wn)/C(wn-1)

For add-one smoothed bigram counts we need to augment the unigram count by the number of total word types in the vocabulary V:  
p\*(wn|wn-1) = ( C(wn-1wn)+1 )/( C(wn-1)+V )

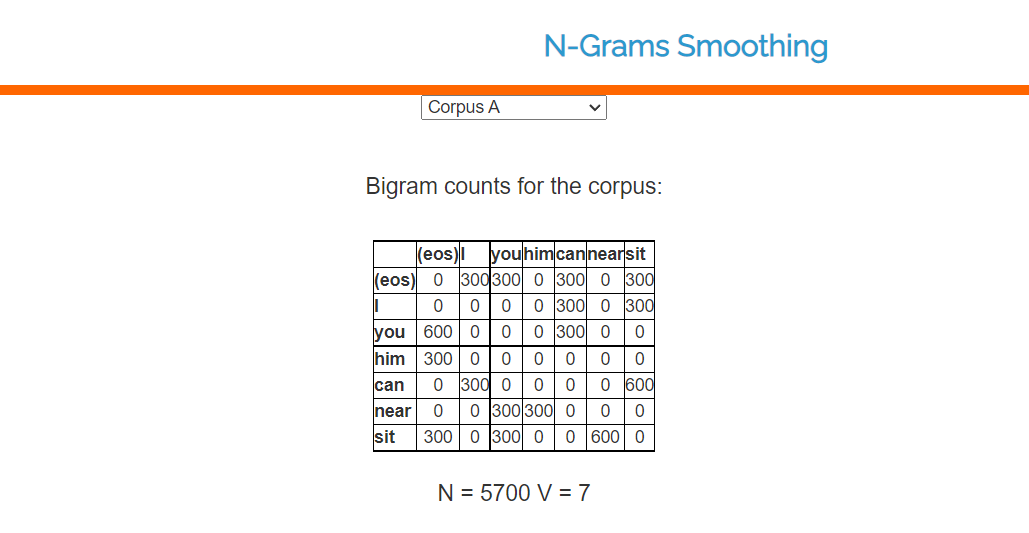
**PROCEDURE**

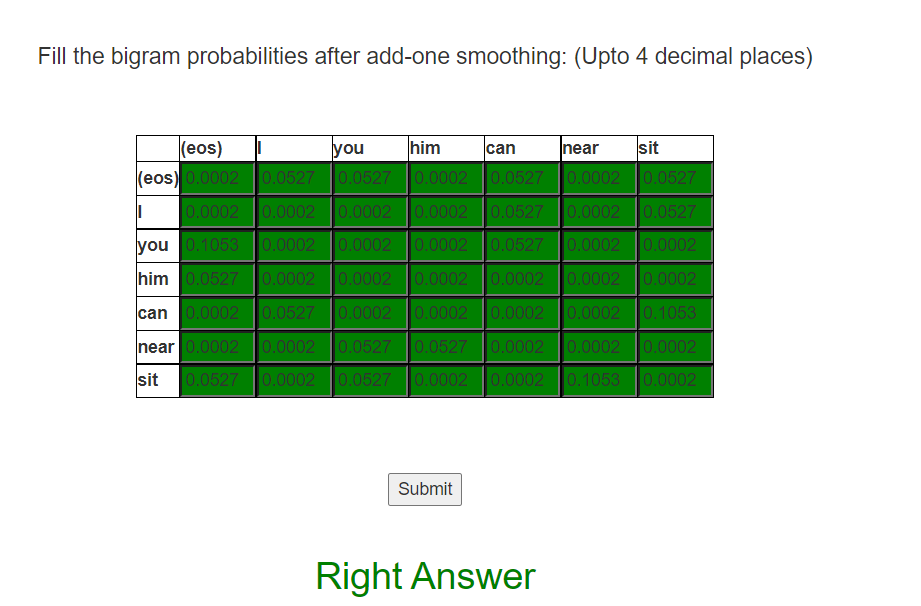
STEP 1: Select a corpus

STEP 2: Apply add one smoothing and calculate bigram probabilities using the given bigram counts,N and V. Fill the table and hit Submit

STEP 3: If incorrect (red), see the correct answer by clicking on show answer or repeat Step 2

**SIMULATION**

****

****

**EXPERIMENT-6**

**AIM:** *To know the importance of context and size of training corpus in learning parts of speech.*

**THEORY**

POS tagging or part-of-speech tagging is the procedure of assigning a grammatical category like noun, verb, adjective etc. to a word. In this process both the lexical information and the context play an important role as the same lexical form can behave differently in a different context.

For example the word "Park" can have two different lexical categories based on the context.

* The boy is playing in the park. ('Park' is Noun)
* Park the car. ('Park' is Verb)

**PROCEDURE**

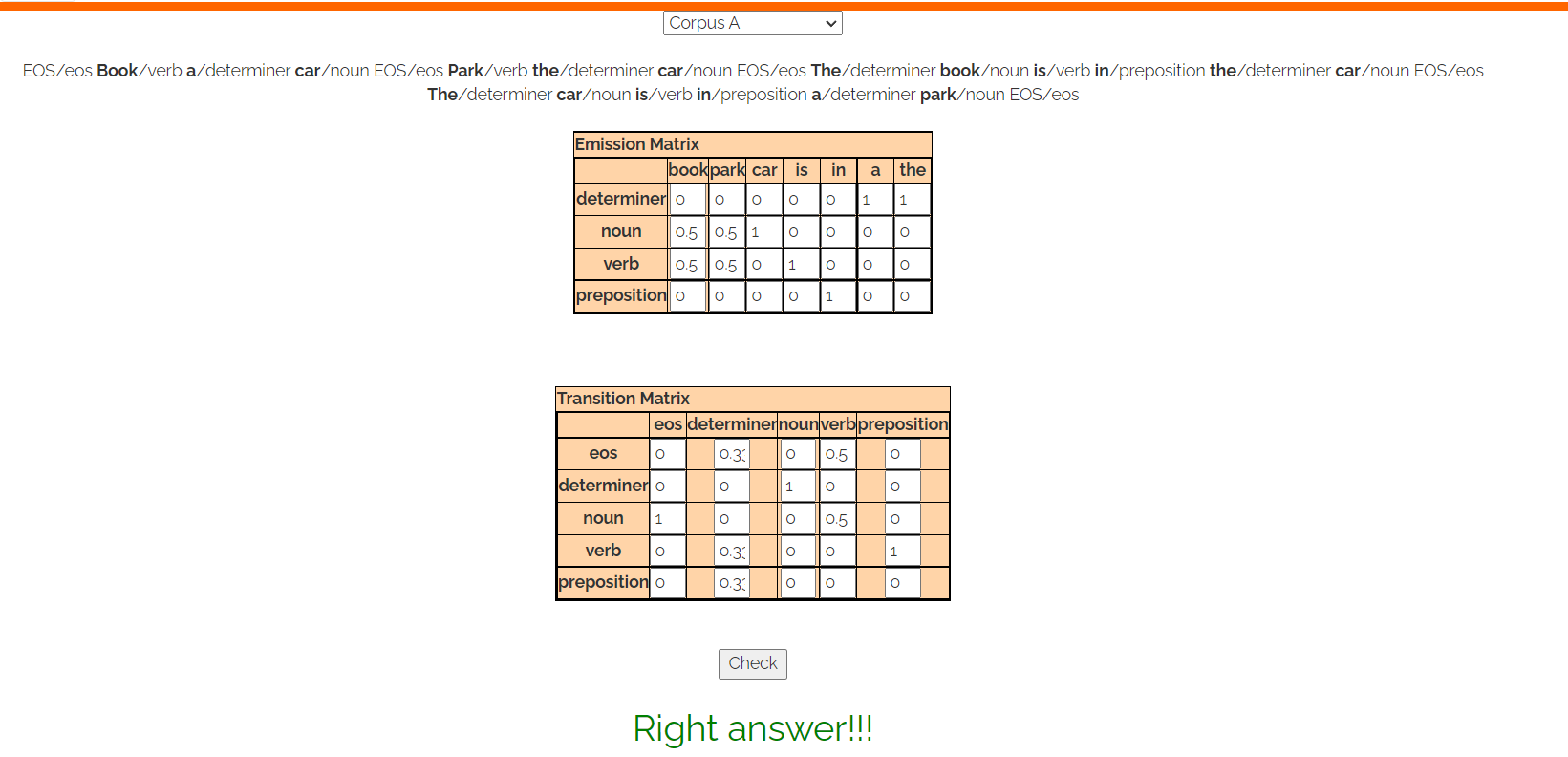
STEP1: Select the corpus.

STEP2: For the given corpus fill the emission and transition matrix. Answers are rounded to 2 decimal digits.

STEP3: Press Check to check your answer.

Wrong answers are indicated by the red cell.

**SIMULATION**

****

**EXPERIMENT-7**

**AIM:** *To understand the concept of chunking and get familiar with the basic chunk tagset.*

**THEORY**

Chunking of text invloves dividing a text into syntactically correlated words. For example, the sentence 'He ate an apple.' can be divided as follows:

https://nlp-iiith.vlabs.ac.in/exp/chunking/images/a.jpg

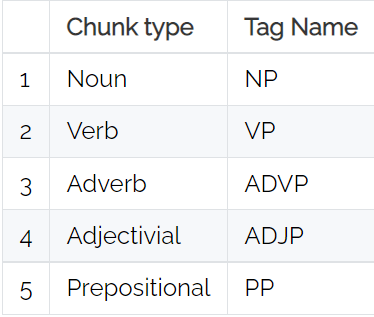
Each chunk has an open boundary and close boundary that delimit the word groups as a minimal non-recursive unit. This can be formally expressed by using IOB prefixes.

https://nlp-iiith.vlabs.ac.in/exp/chunking/images/a.jpg

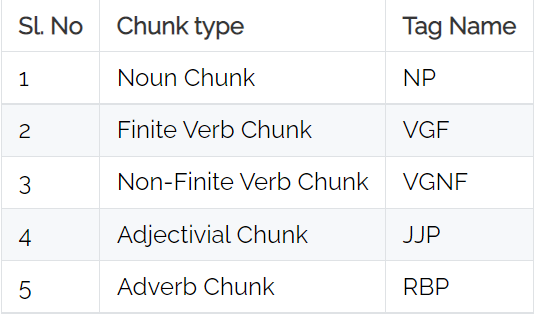
**Chunk Types**

The chunk types are based on the syntactic category part. Besides the head a chunk also contains modifiers (like determiners, adjectives, postpositions in NPs).

The basic types of chunks in English are:



The basic Chunk Tag Set for Indian Languages



**PROCEDURE**

**STEP1**: Select a language

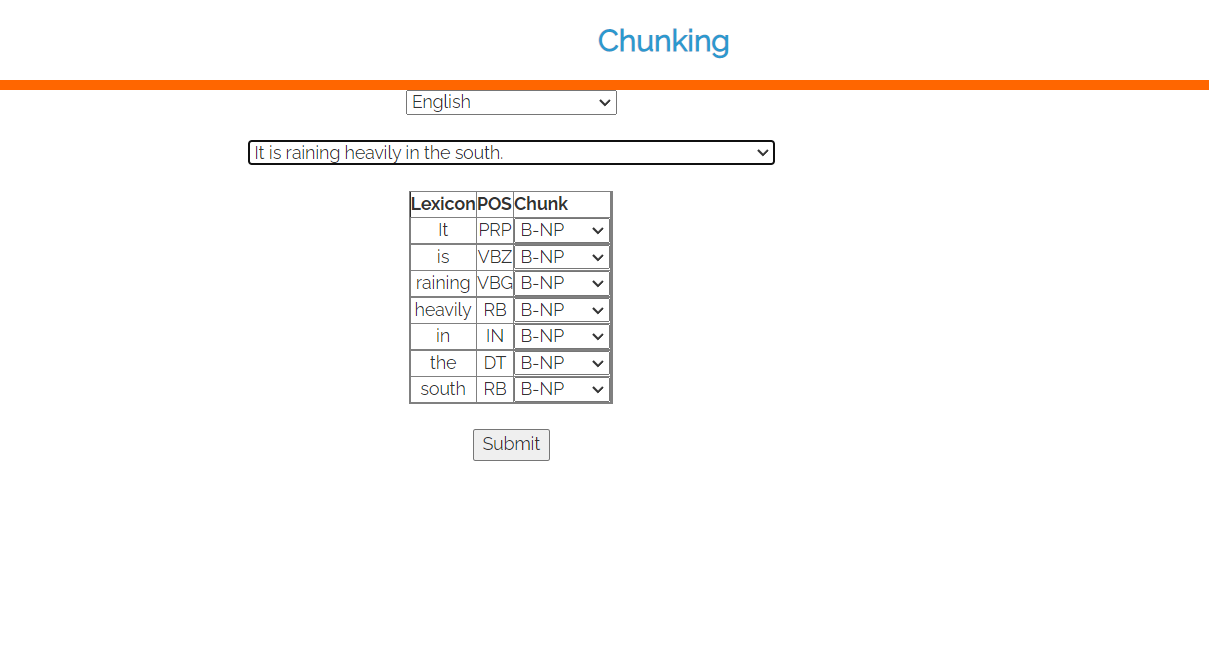
**STEP2**: Select a sentence

**STEP3**: Select the corresponding chunk-tag for each word in the sentence and click the Submit button.

**OUTPUT1**: The submitted answer will be checked.

Click on Get Answer button for the correct answer.

**SIMULATION**

****

**EXPERIMENT-8**

**AIM:** *To know the importance of selecting proper features for training a model and size of training corpus in learning how to do chunking.*

**THEORY**

**Chunking** is an analysis of a sentence which identifies the constituents (noun groups, verbs, verb groups, etc.) which are correlated. These are non-overlapping regions of text. Usually, each chunk contains a head, with the possible addition of some function words and modifiers either before or after depending on languages. These are non-recursive in nature i.e. a chunk cannot contain another chunk of the same category.

Some of the groups possible are:

1. Noun Group
2. Verb Group

For example, the sentence 'He reckons the current account deficit will narrow to only 1.8 billion in September.' can be divided as follows:

[NP He ] [VP reckons ] [NP the current account deficit ] [VP will narrow ] [PP to ] [NP only 1.8 billion ] [PP in ] [NP September ]

Each chunk has an open boundary and close boundary that delimit the word groups as a minimal non-recursive unit.

**PROCEDURE**

**STEP1**: Select the language.

**OUTPUT**: Drop down to select size of corpus, algorithm and features will appear.

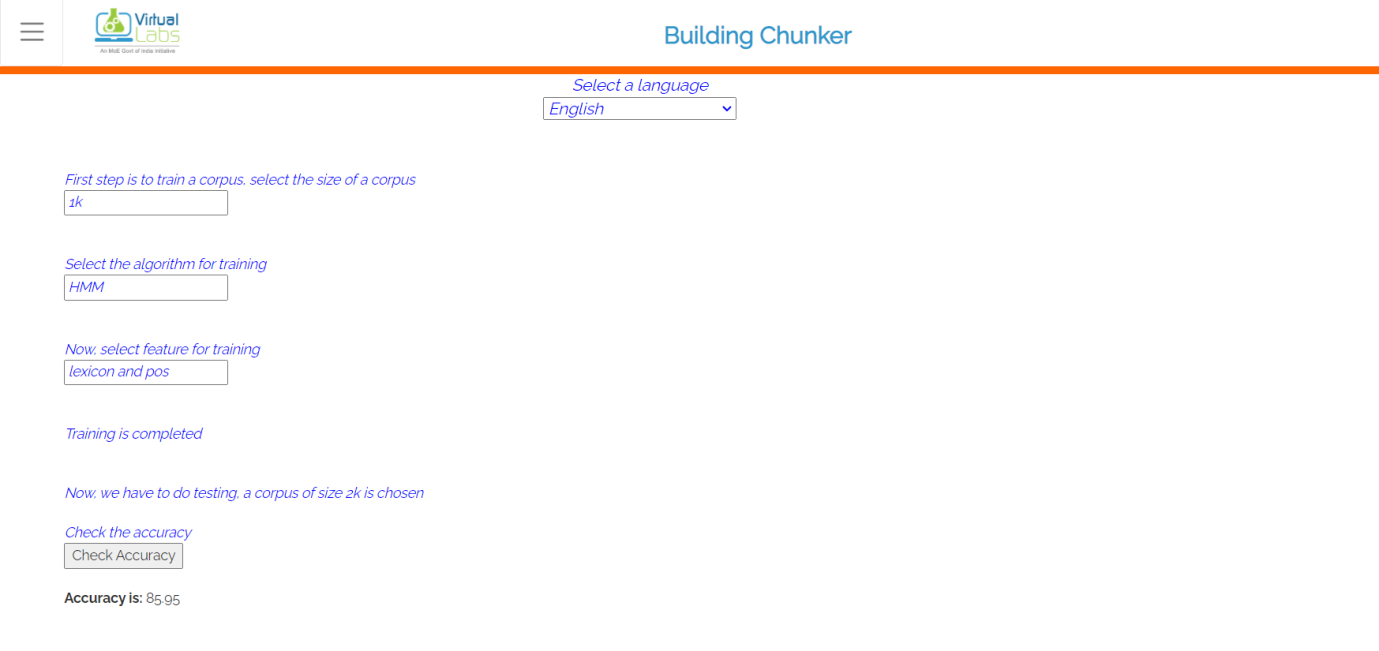
**STEP2**: Select corpus size.

**STEP3**: Select algorithm "CRF" or "HMM".

**STEP4**: Select feature "only lexicon", "only POS", "lexicon and POS".

**OUTPUT**: Corresponding accuracy wil be shown.

**SIMULATION**

****

**EXPERIMENT-9**

**AIM:** *To study Natural language processing library in python: NLTK/ any other.*

**THEORY**

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

**Things you can do with NLTK —**

### Tokenization:

The breaking down of text into smaller units is called tokens. tokens are a small part of that text. If we have a sentence, the idea is to separate each word and build a vocabulary such that we can represent all words uniquely in a list. Numbers, words, etc.. all fall under tokens.

from nltk.tokenize import sent\_tokenize, word\_tokenize

text = "Natural language processing is an exciting area."

print(sent\_tokenize(text))

# output: ['Natural language processing is an exciting area.', 'Huge budget have been allocated for this.']

### Lower case conversion:

We want our model to not get confused by seeing the same word with different cases like one starting with capital and one without and interpret both differently. So we convert all words into the lower case to avoid redundancy in the token list.

text = re.sub(r"[^a-zA-Z0-9]", " ", text.lower())

words = text.split()

print(words)

# output -> ['natural', 'language', 'processing', 'is', 'an', 'exciting', 'area', 'huge', 'budget', 'have', 'been', 'allocated', 'for', 'this']

### Stop Words removal:

When we use the features from a text to model, we will encounter a lot of noise. These are the stop words like the, he, her, etc… which don’t help us and, just be removed before processing for cleaner processing inside the model. With NLTK we can see all the stop words available in the English language.

### Stemming:

In our text we may find many words like playing, played, playfully, etc… which have a root word, play all of these convey the same meaning. So we can just extract the root word and remove the rest. Here the root word formed is called ‘stem’ and it is not necessarily that stem needs to exist and have a meaning. Just by committing the suffix and prefix, we generate the stems.

NLTK provides us with PorterStemmer LancasterStemmer and SnowballStemmer packages.

from nltk.stem.porter import PorterStemmer

# Reduce words to their stems

stemmed = [PorterStemmer().stem(w) for w in words]

print(stemmed)

# output -> ['natur', 'languag', 'process', 'excit', 'area', 'huge', 'budget', 'alloc']

### Lemmatization:

We want to extract the base form of the word here. The word extracted here is called Lemma and it is available in the dictionary. We have the WordNet corpus and the lemma generated will be available in this corpus. NLTK provides us with the WordNet Lemmatizer that makes use of the WordNet Database to lookup lemmas of words.

from nltk.stem.wordnet import WordNetLemmatizer

# Reduce words to their root form

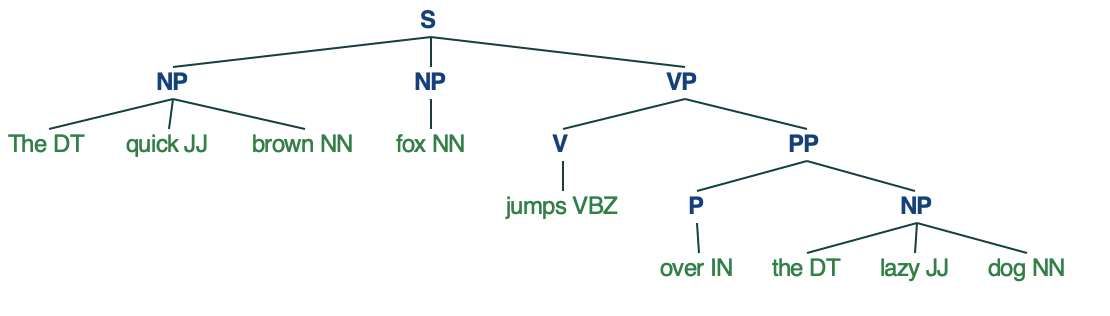
lemmed = [WordNetLemmatizer().lemmatize(w) for w in words]

print(lemmed)

#output -> ['natural', 'language', 'processing', 'exciting', 'area', 'huge', 'budget', 'allocated']

### Parse tree or Syntax Tree generation :

We can define grammar and then use NLTK RegexpParser to extract all parts of speech from the sentence and draw functions to visualize it.



**EXPERIMENT-10**

**AIM:** *To study Natural language processing CORPUS/Treebank (e.g. Wordnet/anyother).*

**THEORY**

A corpus is a large and structured set of machine-readable texts that have been produced in a natural communicative setting. Its plural is corpora. They can be derived in different ways like text that was originally electronic, transcripts of spoken language and optical character recognition, etc.

## Elements of Corpus Design

Language is infinite but a corpus has to be finite in size. For the corpus to be finite in size, we need to sample and proportionally include a wide range of text types to ensure a good corpus design.

Let us now learn about some important elements for corpus design −

### Corpus Representativeness

Representativeness is a defining feature of corpus design. The following definitions from two great researchers − Leech and Biber, will help us understand corpus representativeness −

* According to Leech (1991), “A corpus is thought to be representative of the language variety it is supposed to represent if the findings based on its contents can be generalized to the said language variety”.
* According to Biber (1993), “Representativeness refers to the extent to which a sample includes the full range of variability in a population”.

In this way, we can conclude that representativeness of a corpus are determined by the following two factors −

* Balance − The range of genre include in a corpus
* Sampling − How the chunks for each genre are selected.

### Corpus Balance

Another very important element of corpus design is corpus balance – the range of genre included in a corpus. We have already studied that representativeness of a general corpus depends upon how balanced the corpus is. A balanced corpus covers a wide range of text categories, which are supposed to be representatives of the language. We do not have any reliable scientific measure for balance but the best estimation and intuition works in this concern. In other words, we can say that the accepted balance is determined by its intended uses only.

### Sampling

Another important element of corpus design is sampling. Corpus representativeness and balance is very closely associated with sampling. That is why we can say that sampling is inescapable in corpus building.

* According to Biber(1993), “Some of the first considerations in constructing a corpus concern the overall design: for example, the kinds of texts included, the number of texts, the selection of particular texts, the selection of text samples from within texts, and the length of text samples. Each of these involves a sampling decision, either conscious or not.”

While obtaining a representative sample, we need to consider the following −

* Sampling unit − It refers to the unit which requires a sample. For example, for written text, a sampling unit may be a newspaper, journal or a book.
* Sampling frame − The list of al sampling units is called a sampling frame.
* Population − It may be referred as the assembly of all sampling units. It is defined in terms of language production, language reception or language as a product.

### Corpus Size

Another important element of corpus design is its size. How large the corpus should be? There is no specific answer to this question. The size of the corpus depends upon the purpose for which it is intended as well as on some practical considerations as follows −

* Kind of query anticipated from the user.
* The methodology used by the users to study the data.
* Availability of the source of data.

With the advancement in technology, the corpus size also increases. The following table of comparison will help you understand how the corpus size works −

|  |  |  |
| --- | --- | --- |
| Year | Name of the Corpus | Size (in words) |
| 1960s - 70s | Brown and LOB | 1 Million words |
| 1980s | The Birmingham corpora | 20 Million words |
| 1990s | The British National corpus | 100 Million words |
| Early 21st century | The Bank of English corpus | 650 Million words |

**Examples of corpus.**

## TreeBank Corpus

It may be defined as linguistically parsed text corpus that annotates syntactic or semantic sentence structure. Geoffrey Leech coined the term ‘treebank’, which represents that the most common way of representing the grammatical analysis is by means of a tree structure. Generally, Treebanks are created on the top of a corpus, which has already been annotated with part-of-speech tags.

## Types of TreeBank Corpus

Semantic and Syntactic Treebanks are the two most common types of Treebanks in linguistics. Let us now learn more about these types −

### Semantic Treebanks

These Treebanks use a formal representation of sentence’s semantic structure. They vary in the depth of their semantic representation. Robot Commands Treebank, Geoquery, Groningen Meaning Bank, RoboCup Corpus are some of the examples of Semantic Treebanks.

### Syntactic Treebanks

Opposite to the semantic Treebanks, inputs to the Syntactic Treebank systems are expressions of the formal language obtained from the conversion of parsed Treebank data. The outputs of such systems are predicate logic based meaning representation. Various syntactic Treebanks in different languages have been created so far. For example, Penn Arabic Treebank, Columbia Arabic Treebank are syntactic Treebanks created in Arabia language. Sininca syntactic Treebank created in Chinese language. Lucy, Susane and BLLIP WSJ syntactic corpus created in English language.

## Applications of TreeBank Corpus

### In Computational Linguistics:

If we talk about Computational Linguistic then the best use of TreeBanks is to engineer state-of-the-art natural language processing systems such as part-of-speech taggers, parsers, semantic analyzers and machine translation systems.

### In Corpus Linguistics:

In case of Corpus linguistics, the best use of Treebanks is to study syntactic phenomena.

### In Theoretical Linguistics and Psycholinguistics:

The best use of Treebanks in theoretical and psycholinguistics is interaction evidence.

## PropBank Corpus

PropBank more specifically called “Proposition Bank” is a corpus, which is annotated with verbal propositions and their arguments. The corpus is a verb-oriented resource; the annotations here are more closely related to the syntactic level. Martha Palmer et al., Department of Linguistic, University of Colorado Boulder developed it. We can use the term PropBank as a common noun referring to any corpus that has been annotated with propositions and their arguments.

In Natural Language Processing (NLP), the PropBank project has played a very significant role. It helps in semantic role labeling.

## VerbNet(VN)

VerbNet(VN) is the hierarchical domain-independent and largest lexical resource present in English that incorporates both semantic as well as syntactic information about its contents. VN is a broad-coverage verb lexicon having mappings to other lexical resources such as WordNet, Xtag and FrameNet. It is organized into verb classes extending Levin classes by refinement and addition of subclasses for achieving syntactic and semantic coherence among class members.

Each VerbNet (VN) class contains −

### A set of syntactic descriptions or syntactic frames:

For depicting the possible surface realizations of the argument structure for constructions such as transitive, intransitive, prepositional phrases, resultatives, and a large set of diathesis alternations.

### A set of semantic descriptions such as animate, human, organization:

For constraining, the types of thematic roles allowed by the arguments, and further restrictions may be imposed. This will help in indicating the syntactic nature of the constituent likely to be associated with the thematic role.

## WordNet

WordNet, created by Princeton is a lexical database for English language. It is the part of the NLTK corpus. In WordNet, nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms called **Synsets**. All the synsets are linked with the help of conceptual-semantic and lexical relations. Its structure makes it very useful for natural language processing (NLP).

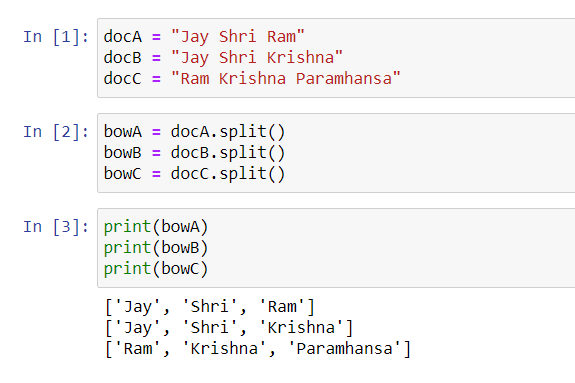
In information systems, WordNet is used for various purposes like word-sense disambiguation, information retrieval, automatic text classification and machine translation. One of the most important uses of WordNet is to find out the similarity among words. For this task, various algorithms have been implemented in various packages like Similarity in Perl, NLTK in Python and ADW in Java.

**EXPERIMENT-11**

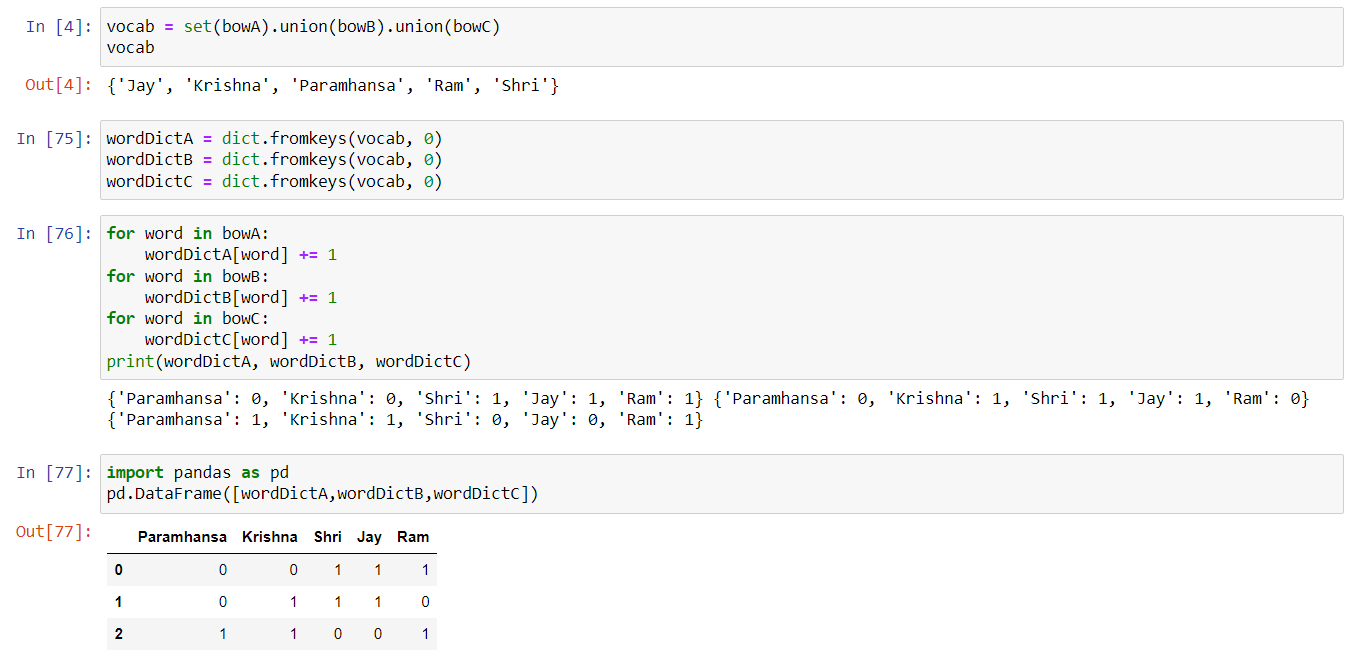
**AIM:** *Program to calculate TF/IDF , feature extraction and finding unique words*

**CODE**

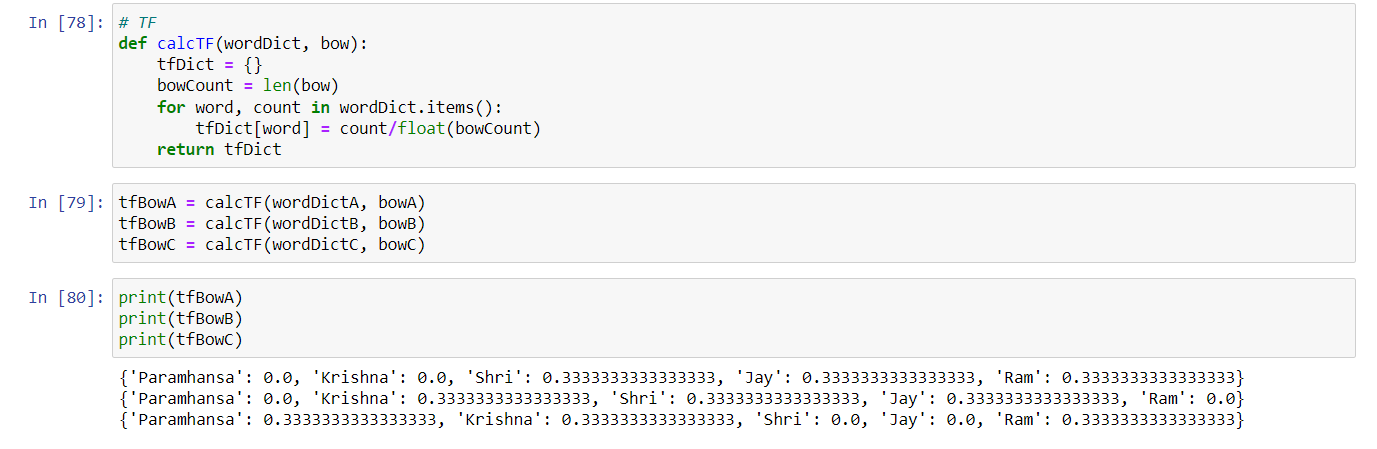
# Feature Extraction

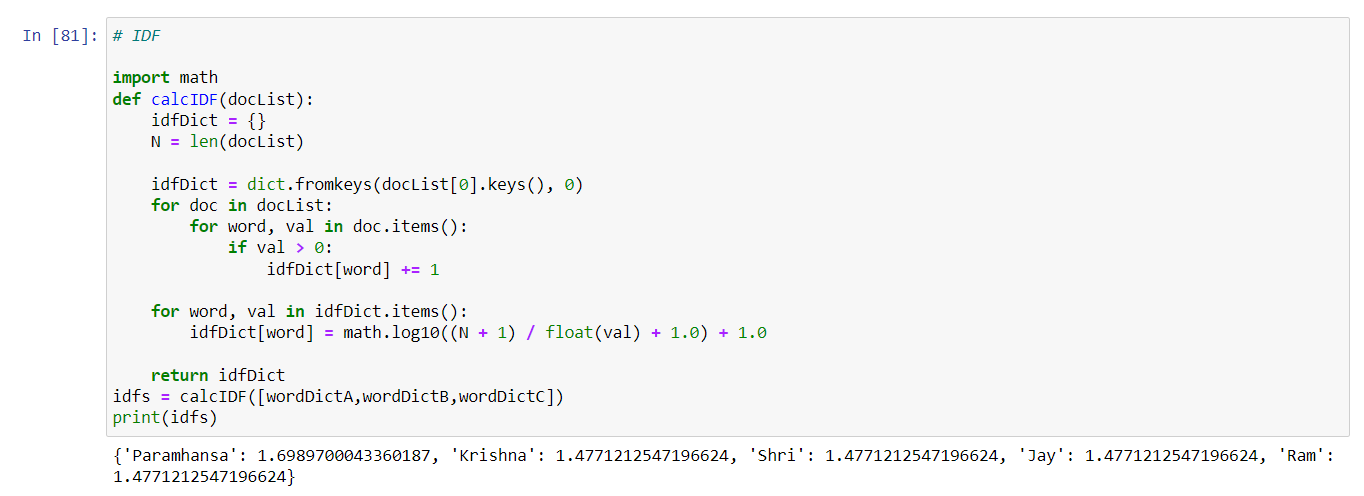


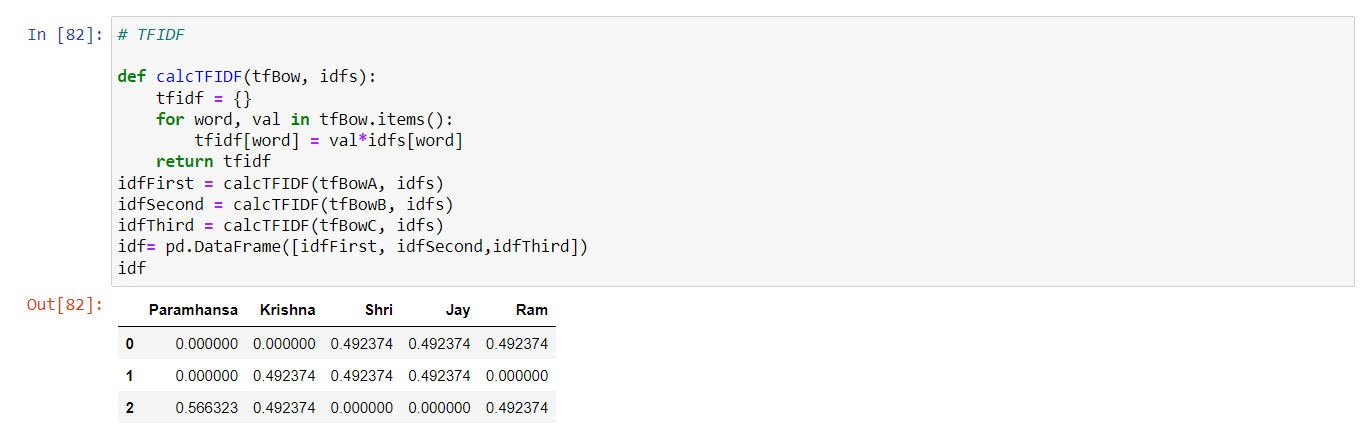
**Finding Unique Words**



# TF-IDF







**EXPERIMENT-12**

**AIM:** *Program for data collection , remove stop words ,data generation , tensor flow graph and word vector .*

**CODE**

# Data Collection

corpus = ['I like apple juice',

'I like orange juice',

'king is a strong man',

'queen is a wise woman',

'boy is a young man',

'girl is a young woman',

'prince is a young king',

'princess is a young queen',

'man is strong',

'woman is pretty',

'prince is a boy will be king',

'princess is a girl will be queen',

'Apple is good place for work']

Corpus

**OUTPUT**

['I like apple juice',

'I like orange juice',

'king is a strong man',

'queen is a wise woman',

'boy is a young man',

'girl is a young woman',

'prince is a young king',

'princess is a young queen',

'man is strong',

'woman is pretty',

'prince is a boy will be king',

'princess is a girl will be queen',

'Apple is good place for work']

# Remove stop words

def remove\_stop\_words(corpus):

stop\_words = ['is', 'a', 'will', 'be']

results = []

for text in corpus:

tmp = text.split(' ')

for stop\_word in stop\_words:

if stop\_word in tmp:

tmp.remove(stop\_word)

results.append(" ".join(tmp))

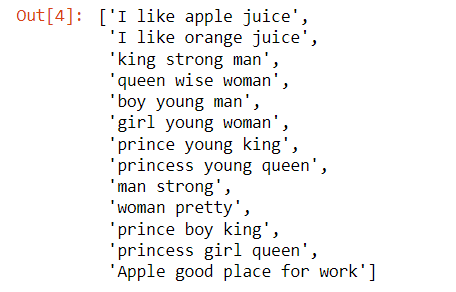
return results

#After removing all stop-words

corpus = remove\_stop\_words(corpus)

corpus

**OUTPUT**



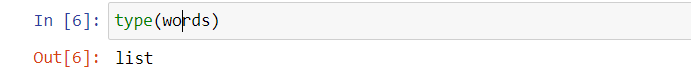
words = []

for text in corpus:

for word in text.split(' '):

words.append(word)

type(words)





# Data Generation

word2int = {}

#Here we assisgned number to each word store it into Dictionary

for i,word in enumerate(words):

word2int[word] = i

# Here we split corpus into sentences

sentences = []

for sentence in corpus:

sentences.append(sentence.split())

WINDOW\_SIZE = 2 # Dimension is 2 means "we consider 2 words from left and right to the centre word

data = []

for sentence in sentences:

for idx, word in enumerate(sentence):

for neighbor in sentence[max(idx - WINDOW\_SIZE, 0) : min(idx + WINDOW\_SIZE, len(sentence)) + 1] :

if neighbor != word:

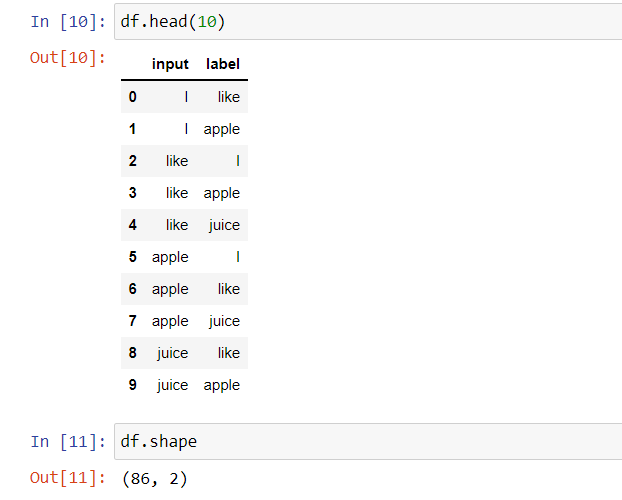
data.append([word, neighbor])

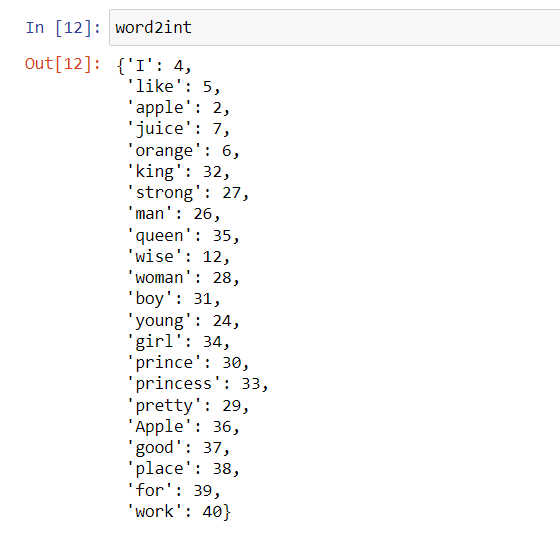
import pandas as pd

df = pd.DataFrame(data, columns = ['input', 'label'])

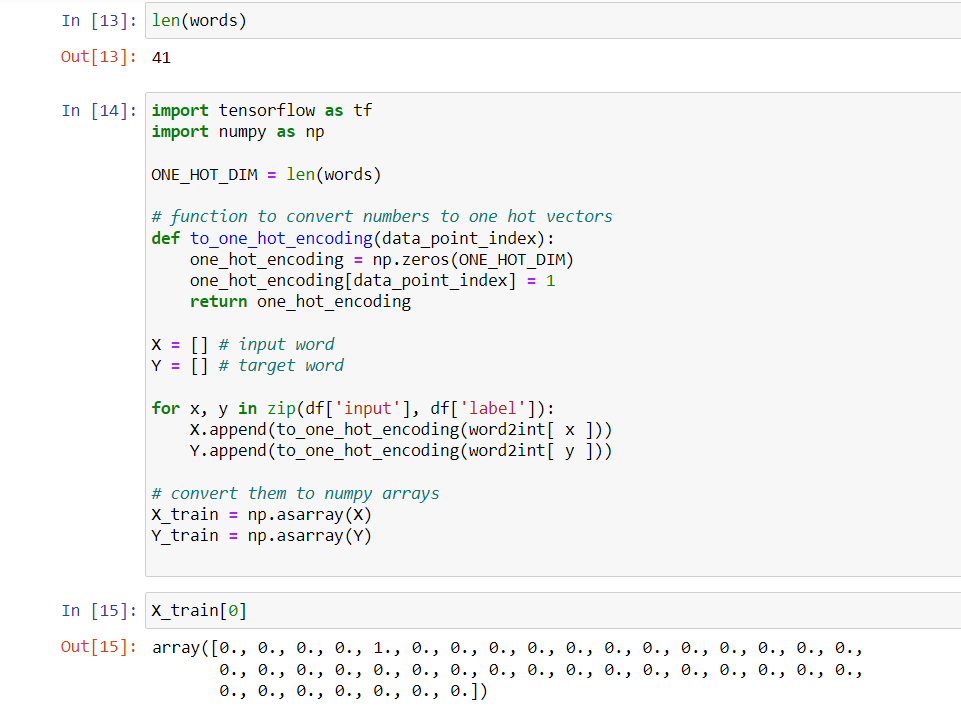
df



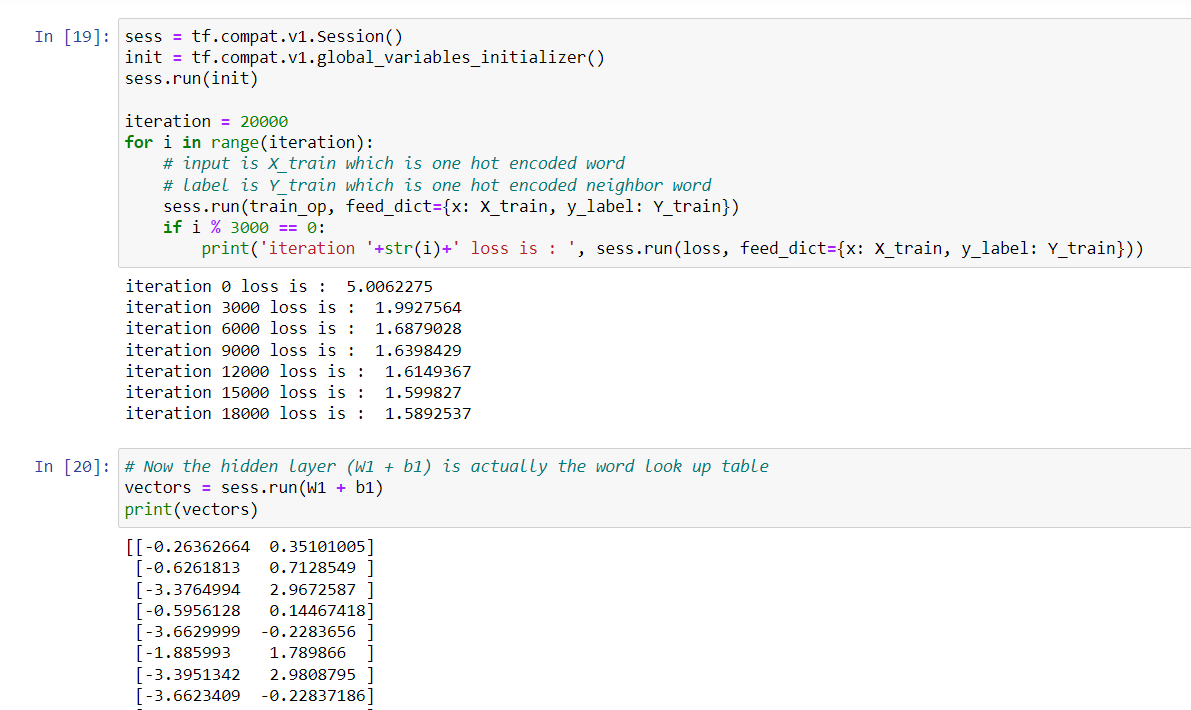


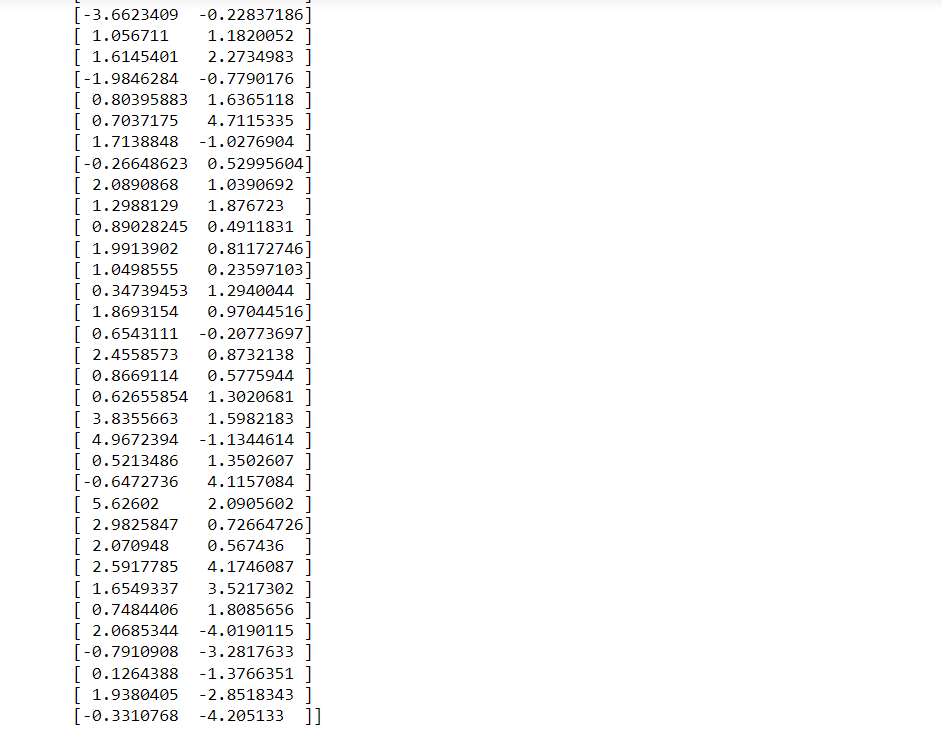
****

# Define Tensor flow Graph

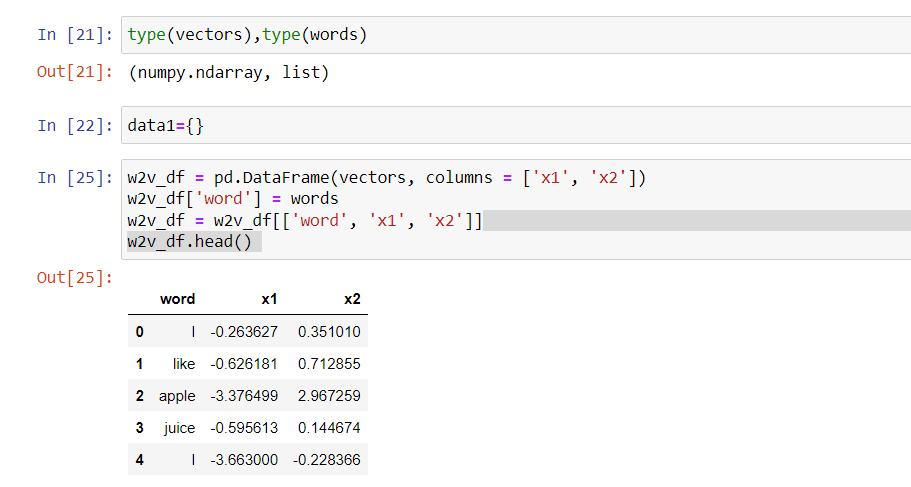


# Train

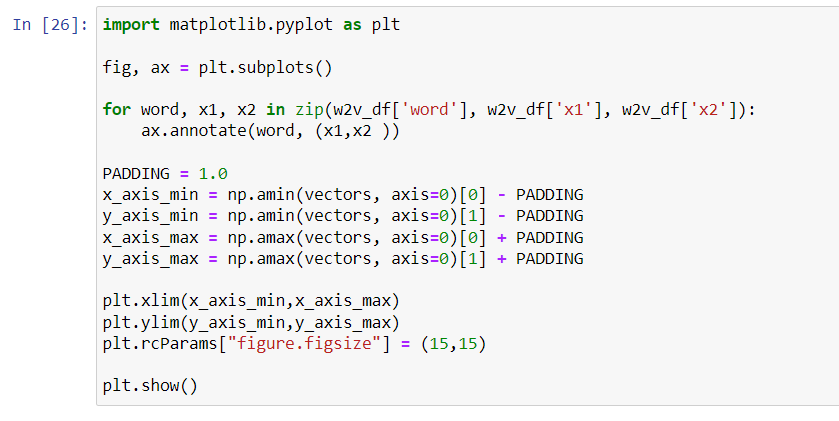




# Word vector in table



# Word vector in 2d chart



Output:

