UNIT-II

Rescaling

Working with Test Data

* Data types represented as strings :-

- Data types are the classifications or categorization of data items.
- It represents the kind value that tells what operation can be represented performed on a particular data.

1. Numeric
2. Sequence Type
3. Boolean
4. Set
5. Dictionary

Numeric :-

i) Numeric data type represent the data which has numeric value.

ii) Numeric value can be integer, floating number or even complex numbers.

iii) These values are defined as int, float and complex.

- Integers :- This value is represented by int class. It contains positive or negative.
- Float :- This value is represented by float class. It is a real number with floating point representation.
- Complex Numbers :- It is represented by complex class. (real part) + (imaginary part).

2) Sequence Type :- Ordered collection of similar or different data types.

- String
- List
- Tuple.

- String :- Strings are arrays of bytes representing unicode characters. A string is a collection of one or more character put in a single quote, double quote / triple quote.
- List :- Lists are just like the arrays, declared in other languages, which is ordered collection of data. It is very flexible.
- Tuple :- Tuple is also an ordered collection of python objects. Tuples cannot be modified after it is created. It is represented by tuple class.

3. Boolean :- Data type with one of the two built-in values True or False. Boolean are equal to True are truthy, and those equal to False are Falsy. It is denoted by class bool.

4. Set :- set is an unordered collection of data type that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it consists various elements.

5. Dictionary :- It is an unordered collection of data values, used to store data values like a map. Key-value is provided in the dictionary to make it more optimized. Dictionary is separated by a colon :, each key is separated by a 'comma'.

Representing Text data as Bag of words :-

- 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.
- Bag of words model is used to represent 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 :-

Step 1 :- First preprocess the data.

- Convert text to lower case.
- Remove all non-word characters.
- Remove all punctuations

Step 2 : Obtaining most fiequent words in our text.

- we declare a dictionary to hold our bag of words.
- Next we tokenize each sentence to words.
- Now for each word in sentence, we check if the word exists in our dictionary.
- If it does, then we increment its count by 1.
- If it doesn't, we add it to our dictionary and set its count as 1.

Step 3 :- Building the Bag of words model.

In this step we construct a vector, which would tell us whether a word in each sentence

is frequent word or not.

- If a word in a sentence is a frequent word, we set it as 1. else we set it as 0.

- A bag-of-words is a representation of text that describe the occurrence of words within a document.
- It involves two things : A vocabulary of known words.
- A measure of the presence of known words.
- To construct a bag-of-words model based on the word counts in the respective documents, the CountVectorizer class implemented in scikit-learn is used.
- The Bag-of-words model is an ~~address~~ orderless document representation - only the count of words matter.

Example :-

step 1 : Data collection

step 2 : Determine the vocabulary

step 3 : Counting

- Counts :- This is count every time the word appears in the document.
- Frequencies :- Calculate the frequency of the words in a document in constrast to the total words in the document

Drawbacks :-

If the new sentences contain new words, then our vocabulary size would increase and thereby, the length of the vectors would increase too.

4b) Advanced Tokenization:-

* Tokenization refers to breaking down the text into smaller units. It entails splitting paragraphs into sentences and sentences into words.

Word Tokenization :- It involves breaking down the text into words.

Ex:- ['I', 'study', 'Machine', 'Learning', '.'].

Sentence Tokenization :- It involves breaking down the text into individual sentences

['I study Machine Learning']

Ex:- From nltk import word_tokenize, sent_tokenize

Sent = " Machine learning subject".

print (word_tokenize (sent))

print (word_tokenize (sent))

output :- ['Machine', 'Learning', 'Subjeet']

Stemming :-

(i) It generates the base word from the inflected word by removing the affixes of the word.

(ii) It has a set of pre-defined rules that govern the dropping of these affixes.

(iii) It must be noted that Stemmers might not always result in Semantically meaningful base words.

(iv) Stemmers are faster and computationally less expensive than lemmatizers.

Ex:- from nltk.stem import PorterStemmer

```
porter = PorterStemmer()
print(porter.stem("play"))
print(porter.stem("playing"))
print(porter.stem("plays"))
print(porter.stem("played"))
```

output :-

```
play
play
play
play
```

Lemmatization :-

i) It involves grouping together the inflected forms of the same word.

ii) We can reach out the base form of any word which will be in meaningful in nature.

iii) The base from here is called Lemma.

Ex:- from nltk.stem import WordNetLemmatizer

```
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("plays", 'v'))
print(lemmatizer.lemmatize("played", 'v'))
print(lemmatizer.lemmatize("play", 'v'))
print(lemmatizer.lemmatize("playing", 'v'))
```

output :-

```
play
play
play
play
```

4a) Rescaling the Data with tf-idf :-

- Instead of dropping features that are seemed unimportant, another approach is to rescale features by how informative we expect them to be.
- one of the most common ways to do this is using the term frequency - inverse document frequency (tf-idf).
- The intuition of this method is to give high weight to any term that appears often in a particular document, but not in very many documents.
- It implements the tf-idf method in two classes :
  - i, TfidTransformer, which takes in sparse matrix output produced by countvectorizer and transforms it
  - ii, Tfid vectorizer which takes in the text data and does both the bag-of-words feature extraction and the tf-idf transformation.
- The tf-idf score for word w in document d as implemented in both the TfidTransformer and Tfid vectorizer classes is given by :

$$tfidf(w,d) = tf * \log\left(\frac{N+1}{N_w+1}\right)+1$$

Where N is the no. of documents in training set.
Nw is no. of documents in training set in word w.
tf = Term frequency

- Both classes also apply L_2 normalization after computing tf-idf representation.
- Rescale the representation of each document to have Euclidian length 1.

- Rescaling means the length of the document does not change the vectorized representation.
- Features with low tf-idf are those that either are very commonly used across documents.
- Many of the high tf-idf features actually identify certain shows or movies.

* Stop Words :-

The process of converting data to something a computer can understand is referred to as pre-processing. One of the major forms of preprocessing is to filter out useless data.

In natural language processing, useless words (data), are referred to as stop words.

A stop word is a commonly used word such as "the", "a", "an", "in" that a search engine has been programmed to ignore, both when indexing entries for searching and retrieving them as the result of a search query.

Sample text with stop words

i) A computer science portal for students.

ii) Can listening be exhausting?

iii) I like reading, so I read.

Without stop words.

i) Computer science, portal, students

ii) Listening, exhausting.

iii) like, reading, read.

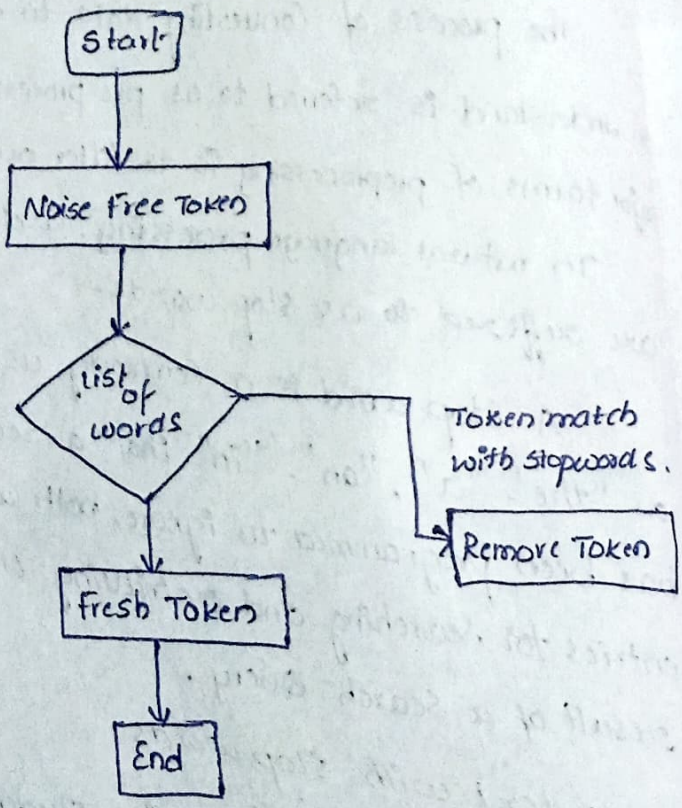
- To check stopwords

```
import nltk
from nltk.corpus import stopwords
print (stopwords.words('english'))
```

output :-

{'there', 'but', 'again', 'once', 'out', 'very', 'with', 'an', 'be' ... }

Steps :-



* The words which are generally filtered out before processing a natural language are called stopwords.
* These are actually the most common words in any language like articles, prepositions, pronouns, conjunctions, etc).
* It does not add much information to the text.

* Approaching a machine learning problem :-

- All the attributes are numeric and all attributes are of same scale and units.
- The problem in hand is a classification problem and thus gives us an option to explore many evaluation metrics.
- The dataset involved is small and clean and thus can be handled easily.

Step 1 :- Importing the required libraries.

```
import pandas as pd.
import numpy as np.
import matplotlib.pyplot as plt.
```

Step 2 :- Loading the data.

```
url = " https://www.github.com.
```

Step 3 :- Summarizing the Data.

a) Taking head of the data.

ex- data.head().

b) Finding dimension of data.

eg:- data.shape : (15,5)

c) Statistical summary all attributes

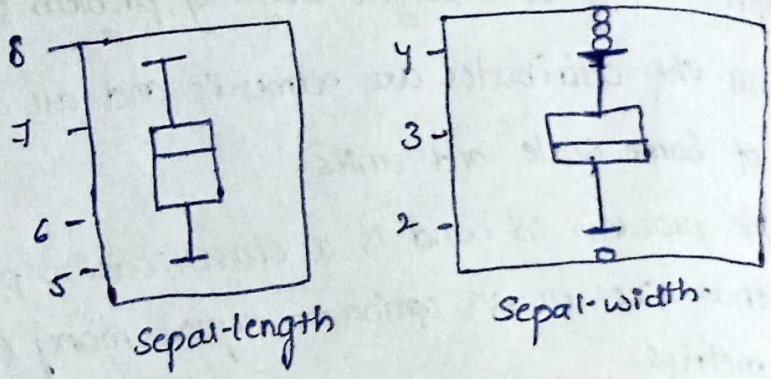
```
print (data.describe())
```

d) class distribution of the Data.

```
print ((data.groupby ('class')).size())
```

Step 4 :- Visualizing the Data.

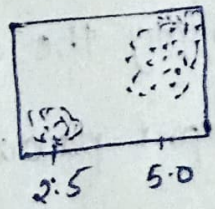
a) plotting univariate plots



b) plotting Multivariate plots.

scatter_matrix(data)

plt.show()



Steps :- Training and evaluating our models.

a) Splitting the training and testing data

b) Building and cross-validating the model.

c) visually comparing the results of different algorithms.

Step 6 :- Making predictions and evaluating the predictions.