Chapter 7

**Natural Language Processing**

Here we will learn *how* to *clean* *text* to prepare them for *machine* *learning* *models*.

How to create a *Bag Of Words Model* and apply ML models onto this *Bag Of Words* model.

**7.1 NLP: Natural Language Processing**

* NLP: *Natural Language Processing* or *NLP* is an area of Computer Science and Artificial Intelligence concerned with interactions between computers and human through natural languages.
* NLP is used to apply *Machine* *Learning* models to *text* and *language*.
* Teach machines to understand what is said in *spoken* and *written* word is the focus of *Natural* *Language* *Processing*. Whenever you dictate something into your iPhone / Android device that then converted to text, that’s an NLP algorithm in action.
* You can also think of Apple's Siri or Amazon's Alexa. These are NLP related algorithms, which is processing your language and converting them into data. Making possible to run the algorithm for the desired purposes.
* NLP history: The history of *Natural-Language Processing* generally started in the 1950s, although work can be found from earlier periods. In 1950, Alan Turing published an article titled "Computing Machinery and Intelligence" which proposed what is now called the Turing Test as a criterion of intelligence.
* Up to the 1980s, most ***natural-language processing systems*** were based on *complex* sets of *hand-written rules*. Starting in the late 1980s, however, there was a *revolution in natural-language processing* with the introduction of Machine Learning Algorithms for Language Processing.
* But even more recent the *research* has *focused* on unsupervised and semi supervised learning algorithm (especially as of lately a huge push into Deep Learning techniques for NLP). *Research in this area is* ***highly******active*** *and it's an exciting time for* ***NLP*** *related* ***algorithms***.

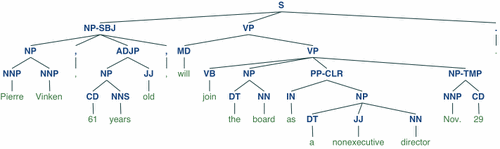
**7.2 Uses of NLP**

* Sentiment analysis: Identifying the mood or subjective opinions within large amounts of text, including average sentiment and opinion mining. We are going to be building a similar model in this section with restaurant reviews being positive or negative.
* Use it to predict the genre of the book: You can also use it to predict the genre of a book.
* Question Answering: use NLP for question and trade with Chatbot.
* Translator: Use NLP to build a machine translator or a speech recognition system
* Document Summarization

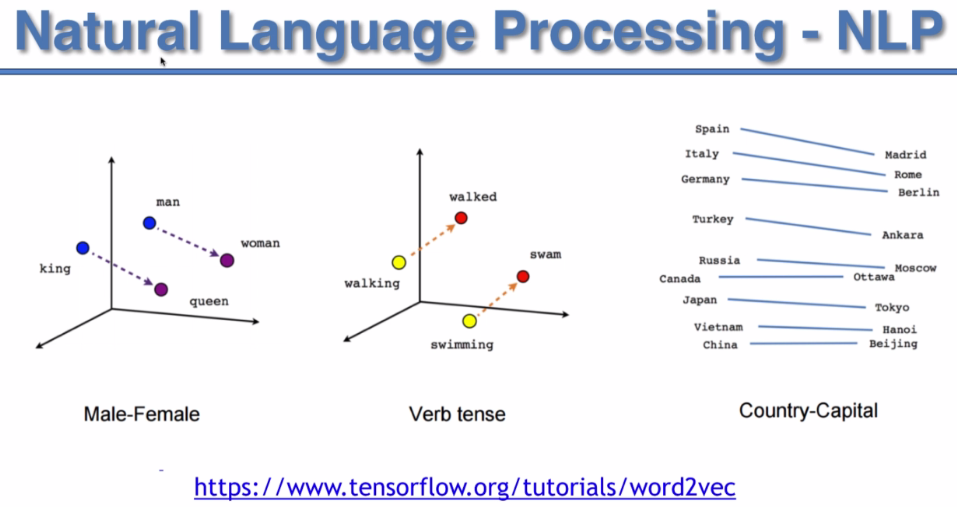
**7.3 NLP Libraries**

We also have a range of NLP related libraries that include the following examples:

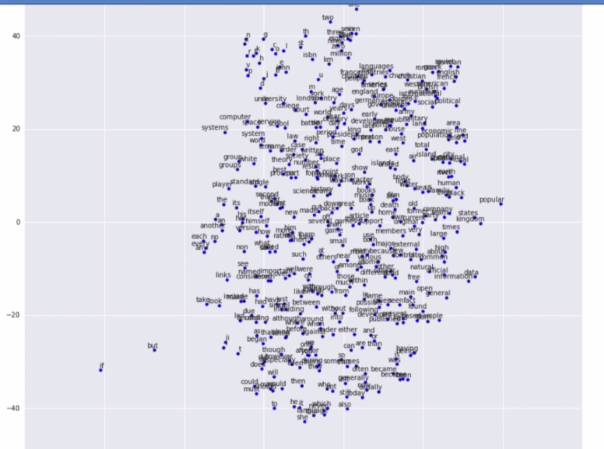
1. Natural Language Toolkit - NLTK
2. SpaCy
3. Stanford NLP
4. OpenNLP



* Example of NLTK: We need to take a look at what NLTK can do by breaking down a sentence. In this above example we can see the ***POS*** or ***part*** ***of*** ***speech*** ***tree***. It's breaking down the sentence. You can see for example we have a *determiner* or DT. you have the *cardinal number* CD and an example of an *adjective* for the JJ tag.
* You can run operations such as here to visualize *semantic* *information* about *words* and their relationships to one another.



* You can also visualize learned embeddings or have words that are similar cluster nearby each other.



* NLP - Bag of Words: For the purpose of the practical work in this section you'll be working with the bag of words.
* Bag of Words: A model used to ***preprocess*** ***the*** ***texts*** to classify before fitting the ***classification*** ***algorithms*** on the observations containing the text.
* It is Very popular *NLP model* - It is a model used to *preprocess* the *texts* to *classify* before fitting the *classification algorithms* on the observations containing the texts. It involves two things:

1. A *vocabulary* of *known* *words*.
2. A *measure* of the *presence* of *known* *words*.

* The things will learn in this section are give below:

1. ***Clean*** ***texts*** to prepare them for the ***Machine*** ***Learning*** ***models***,
2. Create a ***Bag*** of ***Words*** ***model***,
3. Apply ***Machine*** ***Learning*** models onto this ***Bag*** of ***Worlds*** model.

**7.5 CSV files vs TSV files**

* CSV: A comma-separated values (CSV) file is a *delimited* *text* *file* that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format. A CSV file typically stores ***tabular*** ***data*** (numbers and text) in ***plain*** ***text***, in which case each line will have the same number of fields.
* The CSV file format is not fully standardized. Separating fields with commas is the foundation, but commas in the ***data*** or ***embedded line breaks*** have to be handled specially. Some implementations disallow such content while others ***surround the field with quotation marks***, which yet again creates the need for escaping if quotation marks are present in the data.
* The term "CSV" also denotes several closely-related delimiter-separated formats that use other field delimiters such as ***semicolons***. These include ***tab-separated values*** and ***space-separated values***. A delimiter guaranteed not to be part of the data greatly simplifies ***parsing***.
* TSV: A *tab-separated values (TSV)* file is a simple text format for storing data in a *tabular* *structure*, e.g., a ***database*** ***table*** or ***spreadsheet*** data, and a way of ***exchanging*** ***information*** between ***databases***. Each record in the table is one line of the text file. Each field value of a record is separated from the next by a tab character. The TSV format is thus a variation of the comma-separated values format.
* TSV is a simple file format that is widely supported, so it is often used in data exchange to move tabular data between different computer programs that support the format. For example, a TSV file might be used to transfer information from a database program to a spreadsheet.

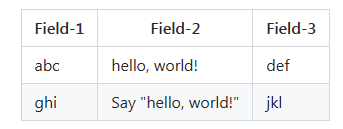
**\n** for newline,

**\t** for tab,

**\r** for carriage return,

**\\** for backslash

* Delimiter: A *delimiter* is a sequence of *one or more characters* for specifying the *boundary* *between* separate, independent regions in *plain* *text*, mathematical *expressions* or other *data* *streams*. An example of a delimiter is the ***comma*** character, which acts as a field delimiter in a sequence of ***comma-separated values***. Another example of a delimiter is the ***time*** ***gap*** used to separate ***letters*** and ***words*** in the ***transmission*** of ***Morse*** ***code***.
* Example: The most common CSV escape format uses quotes to delimit fields containing delimiters. Quotes must also be escaped, this is done by using a pair of quotes to represent a single quote. Consider the data in this table:



* In *Field-2*, the first value contains a *comma*, the second value contains *both quotes and a* *comma*. Here is the CSV representation, using escapes to represent commas and quotes in the data.

Field-1,Field-2,Field-3

abc,"hello, world!",def

ghi,"Say ""hello, world!""",jkl

* In the above example, only fields with delimiters are quoted. It is also common to quote all fields whether or not they contain delimiters. The following CSV file is equivalent:

"Field-1","Field-2","Field-3"

"abc","hello, world!","def"

"ghi","Say ""hello, world!""","jkl"

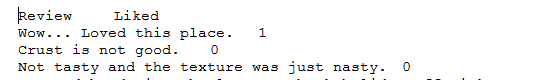
* Here's the same data in TSV. It is *much simpler* as *no* *escapes* are *involved*:

Field-1 Field-2 Field-3

abc hello, world! def

ghi Say "hello, world!" jkl

* TSV or CSV in NLP: Now the question is do we need a data-set where the columns are separated by a ***Comma*** or by a ***Tab***. We used csv for previous ML models but in NLP we are going to use TSV. The reason is:
* Because it *doesn't create any problem* with ***commas***, ***quotes*** and hence we don’t need any ***escapes***.



* For example: Our dataset is in tsv format like above. ***0*** or ***1*** means ***negative*** or ***positive***. Reviews like following in CSV can cause problem:

Wow, this place is amazing!!!, 1

* It is one sentence, but due to "***,***" spate ***Wow*** and ***this place is amazing !!!***  it makes them to be in different fields. Then ***1*** goes to the ***next*** ***field***. Hence algorithm doesn't work.
* Also we can use double-quote, but Reviews can also contain Double-quotes susc as:

"Good lord !!" , The place is Horrible., 1

* It's way better to take ***tabs*** here because you know when people ***write reviews*** they ***do not put tabs in the review***. That would be very rare.
* Because by pressing the ***Tab button*** when you're writing your review you would ***not be able to continue to write*** it. So we will ***never find a tab in the review*** and that's why we will *never have this problem of getting these anomalies due to* *duplicate* *delimiters* in one specific review.
* To avoid those problems we use ***tsv*** files. It makes easier to import dataset. So in NLP it is recommend to prepare your text datasets using tab-separator (TSV).

**7.4 NLP in python**

Natural Language Processing (NLP) is a branch of machine learning where we do some predictive analysis on text mostly. NLP is about ***analyzing text***. These texts can be ***books reviews***, some ***HTML*** web pages that you extract from ***web-scrapping***, all sorts of Texts.

* Problem Description: Here we will analyze some ***written reviews of restaurants***. So we will make some ML models that will ***predict*** if the review is ***positive*** or ***negative*** (according to 1 or 0 in our tsv file).
* The algorithm we will making this part is a general model, it will be very well applicable to other kinds of text such as:
* To predict the Genre of a book whether it is ***thriller***, ***comedy*** or ***romance***.
* Use it on ***HTML*** web pages to do whatever kind of analysis you want to do on those pages.
* You can also play it on ***newspapers*** you know to predict in which ***category*** an ***article*** belongs to.

we need to add some parameters because of course we are importing a tsv file whereas this ***read\_csv*** function by ***Pandas*** is expecting a ***csv***.

dataset = pd**.read\_csv**("Restaurant\_Reviews.tsv", delimiter = "\t", quoting = 3)

delimiter: This parameter ***delimiter = "\t"*** specify that it's *not* the *comma ","* but the tab "\t" delimiter .

Pandas will know that our two columns are separated by a tab.

quoting: But we will not stop here because when looking at the data sets we saw that we might have some other problems related to the ***double quotes "***. That's why we used ***quoting = 3***, to make sure that we won't have any problem with these double quotes.

We need to make sure that we have our 1000 reviews.

Also make sure that we don't have any ***review*** in the ***like*** ***column*** or a ***1 or 0*** in the ***review*** ***column***.

#*NLP: Natural Language Precessing*

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

#*-------------- Data preprocessing --------------*

#*importing dataset*

        #*since we are using "tsv" ins tead of "csv" we need to specify some parameters.*

        #*Because "Pandas" expecting some "csv" files*

dataset = pd**.read\_csv**("Restaurant\_Reviews.tsv", delimiter = "\t", quoting = 3)

****

**7.4.1 Clean the Data**

Next step is to clean the different reviews. This is a ***compulsory*** step in ***natural language processing (NLP)*** which consists of cleaning the text to make it ready for our future machine learning algorithms.

* We have to do this because in the end we will create a *bag of words model* or bag of words representation and this will consist of getting only the relevant words and the different reviews here.
* That means that we'll get rid of all the unnecessary words. EG: The, on, am etc. and these are not relevant words because these are not the words that will help the machine learning algorithm to predict if the review is positive or negative.
* Punctuation: We will also get rid of the ***punctuation*** like ***"..."***.
* numbers : We will also get rid of ***numbers*** unless numbers can have a significant impact.
* Stemming: And we will also do what's called ***stemming*** which consist of taking the *roots* of some *different* *versions* of a *same* *word*.
* For example we have this "Loved" word here past tense of the verb love and so ***apply stemming on this word*** will consist of only taking love here because whether we have ***loved*** or ***love*** that gives the ***same hints*** on whether the review is ***positive*** or ***negative***.
* We need to apply ***stemming*** in order not to have too many words in the end. And to regroup the same versions of a same word like ***love*** and ***love*** or even ***loving*** into a same word ***love***.
* Capitals: we will only have the reviews in lower text.
* Tokenization: Once all the reviews are cleaned, we'll proceed to the last step of creating our Bag Of Words Model which is the tokenization process.
* It splits all the different reviews into different words (only the relevant words because we did text pre-processing).
* And then we'll take all the words of the different reviews and we will ***create one column for each word*** (we'll have a lot of columns).
* And then for each review, each column will contain the number of times the associated word appears in the review. So we'll have a ***lot of 0's*** because for each review a ***lot of words don't appear in the review*** and we'll have a ***couple of 1's*** for the words that appear once/twice in the review.
* We will get here ***sparse matrix*** because we'll get a lot of zeros in the sparse matrix because for all the reviews ***most*** of the ***words*** will ***not*** be in the ***reviews*** so we will have to do something about it. That will be the ***last step*** of the creation of our ***bag of words***.
* Cleaning the text: At the beginning, we apply the *different* *steps* of cleaning process on the *first* *review*, and then we use *for –loop* for the *rest* of the *reviews*. So let's do this let's clean this first review and then we'll clean the rest.
* Regular Expression: Python has a built-in package called ***re***, which can be used to work with *Regular* *Expressions*. A *RegEx*, or ***Regular*** ***Expression***, is a sequence of characters that forms a ***search*** ***pattern***. ***RegEx*** can be used to check if a string contains the ***specified search pattern***. We are going to use it for cleaning text.

#*Cleaning the text*

**import** re

**print**(f"{dataset['Review'][0]}")

rev\_W = re**.sub**("[^a-zA-Z]", " ", dataset["Review"][0]) rev\_W = rev\_W**.lower**()  #*converting to Lower case*

1. In ***re.sub()***, the 1st parameter means ***remove all characters excluding*** ***a-z*** and ***A-Z***. "***^***" means ***Excluding***.
2. The 2nd parameter " " means we want to use a space as a separator.
3. dataset["Review"][0] is the sentence where re.sub() will be applied, it is the first review of our Data-set.



1. Then we convert the all words into lower case.



* Removing non-Significant words: That is we are going to remove all the words like: ***the***, ***that***, ***and***, ***in***, all the articles all the propositions. For example in ***"Wow loved this place"***: ***Loved*** is the important word here, this indicates positive. But the word "***this***" is not important because it may occur in negative review such as: "***this place sucks!!***" .
* Our goal is to avoid too much ***sparsity*** in the ***sparse*** ***matrix***.
* To do this we will need a new library which is the very famous ***NLTK*** library
* module nltk: The ***Natural Language Toolkit (NLTK)*** is an open source Python library for Natural Language Processing. A free online book is available. (If you use the library for academic research, please cite the book.)
* Steven Bird, Ewan Klein, and Edward Loper (2009). Natural Language Processing with Python. O'Reilly Media Inc. [http://nltk.org/book](file:///C:\Users\RTTI\AppData\Local\Programs\Microsoft%20VS%20Code\resources\app\out\vs\code\electron-browser\workbench\workbench.html?config=%7B%22_%22%3A%5B%22L%3A%5C%5C1_Development_2.0%5C%5CML_phase_3_ML_Intro%5C%5Cml_p3_ch7_NLP%5C%5Cprctc_nlp.py%22%5D%2C%22appRoot%22%3A%22c%3A%5C%5CUsers%5C%5CRTTI%5C%5CAppData%5C%5CLocal%5C%5CPrograms%5C%5CMicrosoft%20VS%20Code%5C%5Cresources%5C%5Capp%22%2C%22machineId%22%3A%228fe68423-e2d8-4e01-a7c5-5980feda7e4f%22%2C%22nodeCachedDataDir%22%3A%22C%3A%5C%5CUsers%5C%5CRTTI%5C%5CAppData%5C%5CRoaming%5C%5CCode%5C%5CCachedData%5C%5C6ab598523be7a800d7f3eb4d92d7ab9a66069390%22%2C%22mainPid%22%3A4340%2C%22execPath%22%3A%22C%3A%5C%5CUsers%5C%5CRTTI%5C%5CAppData%5C%5CLocal%5C%5CPrograms%5C%5CMicrosoft%20VS%20Code%5C%5CCode.exe%22%2C%22userEnv%22%3A%7B%22VSCODE_IPC_HOOK%22%3A%22%5C%5C%5C%5C.%5C%5Cpipe%5C%5C6ef7947227db608710a196042b78a050-1.39.2-main-sock%22%2C%22VSCODE_NLS_CONFIG%22%3A%22%7B%5C%22locale%5C%22%3A%5C%22en-us%5C%22%2C%5C%22availableLanguages%5C%22%3A%7B%7D%2C%5C%22_languagePackSupport%5C%22%3Atrue%7D%22%2C%22VSCODE_LOGS%22%3A%22C%3A%5C%5CUsers%5C%5CRTTI%5C%5CAppData%5C%5CRoaming%5C%5CCode%5C%5Clogs%5C%5C20220428T144746%22%7D%2C%22isInitialStartup%22%3Atrue%2C%22filesToOpenOrCreate%22%3A%5B%7B%22fileUri%22%3A%7B%22%24mid%22%3A1%2C%22path%22%3A%22%2FL%3A%2F1_Development_2.0%2FML_phase_3_ML_Intro%2Fml_p3_ch7_NLP%2Fprctc_nlp.py%22%2C%22scheme%22%3A%22file%22%7D%2C%22exists%22%3Atrue%7D%5D%2C%22filesToDiff%22%3A%5B%5D%2C%22backupPath%22%3A%22c%3A%5C%5CUsers%5C%5CRTTI%5C%5CAppData%5C%5CRoaming%5C%5CCode%5C%5CBackups%5C%5C1651135669494%22%2C%22windowId%22%3A1%2C%22logLevel%22%3A2%2C%22zoomLevel%22%3A0%2C%22maximized%22%3Atrue%2C%22frameless%22%3Atrue%2C%22perfEntries%22%3A%5B%22main%3Astarted%22%2C1651135664572%2C%22main%3AappReady%22%2C1651135665668%2C%22willLoadMainBundle%22%2C1651135665774%2C%22didLoadMainBundle%22%2C1651135666888%2C%22main%3AloadWindow%22%2C1651135668920%5D%2C%22partsSplashPath%22%3A%22C%3A%5C%5CUsers%5C%5CRTTI%5C%5CAppData%5C%5CRoaming%5C%5CCode%5C%5Crapid_render.json%22%7D)

**import** nltk

nltk**.download**("stopwords")

* Now we'll make a for-loop for each review ***" rev\_W "*** our first review: "Wow loved this place", to go through all the different words of this ***rev\_W***, then for each of the different words we see if the word is present in this stop words list and if that's the case we remove the word from the review.

#*Cleaning the text*

**import** re

**import** nltk

#*nltk.download("stopwords")*

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

**print**(f"{dataset['Review'][0]}")

rev\_W = re**.sub**("[^a-zA-Z]", " ", dataset["Review"][0])

rev\_W = rev\_W**.lower**()   #*converting to Lower case*

rev\_W = rev\_W**.split**()   #*converting "string" to "list"*

#*reView = [wrd for wrd in rev\_W if wrd not in stopwords.words("english")]*

reView = [wrd **for** wrd **in** rev\_W **if** wrd **not** **in** **set**(stopwords**.words**("english"))]

#*import nltk*

#*from nltk.corpus import stopwords*

#*print(stopwords.words('english'))*

#*print(set(stopwords.words('english')))*

* ***nltk.download("stopwords")*** not needed if the package already up-to-date.
* We use *list-comprehension*, and ***set*** is used to make a faster algorithm. ***set*** works faster than ***list***:

reView = [wrd **for** wrd **in** rev\_W **if** wrd **not** **in** **set**(stopwords**.words**("english"))]

In case you have some *bigger* *text (eg: Articles or Book review)* than this *simple* *review* here (because right now you know our review is composed of only 4 words) it is highly recommended to add this set() function here, makes your algorithm work much faster.

* Corpus: This word means a collection of written texts, especially the entire works of a particular author or a body of writing on a particular subject
* Stopwords: 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 when retrieving them as the result of a search query.

**import** nltk

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

**print**(stopwords**.words**('english'))

* Stemming: For example: keeping only the root of the word i.e. ***love*** because whether we have *love*, *loved* or *loving* or *will* *love* in a review, gives the same hints whether the review is ***positive*** or ***negative*** so we don't need this word conjugated in the ***past*** or in the ***future*** or in the ***present***.
* Because if we keep all the different versions of the same word, then in the end in our ***sparse*** ***matrix*** we'll have tons of words and therefore huge ***sparsity*** and therefore ***our algorithm will have trouble to run*** because we will simply have *too many columns* because in the end you know in the *sparse* *matrix* each *word* will have its *own* *column*. So that's why we are doing some simplification here.
* We import the ***PorterStemmer***, then create an object of it. And we apply it to our words in to our *list-comprehension.*
* Notice we changed ***wrd***  into ***prt\_stmr.stem(wrd)***in our list comprehension.

**from** nltk**.**stem**.**porter **import** PorterStemmer

and

prt\_stmr = **PorterStemmer**()

reView = [prt\_stmr**.stem**(wrd) **for** wrd **in** rev\_W **if** wrd **not** **in** **set**(stopwords**.words**("english"))]

* And then we join the resulting ***list*** of words into a ***string***

reVw = " "**.join**(reView)

|  |  |
| --- | --- |
| Before | After |
|  |  |

#*Cleaning the text*

**import** re

**import** nltk

#*nltk.download("stopwords")*

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

**from** nltk**.**stem**.**porter **import** PorterStemmer

**print**(f"{dataset['Review'][0]}")

rev\_W = re**.sub**("[^a-zA-Z]", " ", dataset["Review"][0])

rev\_W = rev\_W**.lower**()   #*converting to Lower case*

rev\_W = rev\_W**.split**()   #*converting "string" to "list"*

#*reView = [wrd for wrd in rev\_W if wrd not in stopwords.words("english")]*

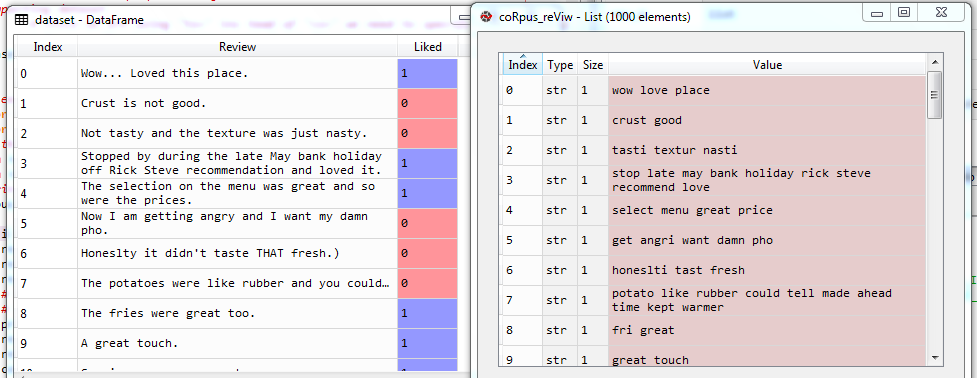
#*Removing stopwords. And stemming : Finding the root of different versions of a same word*

prt\_stmr = **PorterStemmer**()

reView = [prt\_stmr**.stem**(wrd) **for** wrd **in** rev\_W **if** wrd **not** **in** **set**(stopwords**.words**("english"))]

reVw = " "**.join**(reView)

* Loop through al 1000 reviews: We'll create a new list. And then inside our for-loop we are going to append it through the iterations.



#*Cleaning the text*

**import** re

**import** nltk

#*nltk.download("stopwords")*

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

**from** nltk**.**stem**.**porter **import** PorterStemmer

#*print(f"{dataset['Review'][0]}")*

coRpus\_reViw = []

**for** i **in** **range**(0, 1000):

    rev\_W = re**.sub**("[^a-zA-Z]", " ", dataset["Review"][i])

    rev\_W = rev\_W**.lower**()   #*converting to Lower case*

    rev\_W = rev\_W**.split**()   #*converting "string" to "list"*

    #*reView = [wrd for wrd in rev\_W if wrd not in stopwords.words("english")]*

    #*Removing stopwords. And stemming : Finding the root of different versions of a same word*

    prt\_stmr = **PorterStemmer**()

    reView = [prt\_stmr**.stem**(wrd) **for** wrd **in** rev\_W **if** wrd **not** **in** **set**(stopwords**.words**("english"))]

    reVw = " "**.join**(reView)

    coRpus\_reViw**.append**(reVw)

* All *stopwords* are *removed* and all words are *stemmed*.
* We will simplify those reviews even more. Because we will *actually* *keep* only the *words* that appear a *minimum* *number* of *times*. We'll remove some *irrelevant* *words* by filtering all the *words* that *appear* *rarely* and we'll do that while creating the *Bag* *of* *words* model.

**7.4.2 Clean the Data**

What is the *bag of words model* and why we need to create it? By understanding why we need to create the *bag of word model* you will understand even more why we *had to clean* the *text/the 1000 reviews*.

* Basically the first big step of natural language processing (NLP) not only *Cleaning* *The* *Texts* but also creating a *Corpus*.
* Now from this corpus we will create our bag of words model.
* The bag words model basically really simple. We're going to take all the different words of the 1000 reviews here but *without* taking *twice* or *three* times (*duplicates* or *triplicates*) i.e. the unique words of these 1000 reviews. Then we create a *table* where *one* *column* for each *word*. So there will be lots of columns.
* Then we will put all these columns in a table where the rows are nothing else than the 1000 reviews.
* So basically what we'll get is a *table* containing *1000 rows* where the *rows* correspond to the *reviews* and a lot of columns where the *columns* correspond to each of the *different* *words* in the *corpus* of the *reviews*.
* Each cell of this table will correspond to ***one specific review and one specific word*** of this corpus. And in this *cell* we're going to have a *number* which represents the *number of times* the *word* corresponding to the column *appears* in the *review*.
* So if a word appear in a review, we place a 1, and for all rest of the words we place 0's in the rows. Hence in each row(or review) of the table we mostly have ***0***'s and few ***1***'s.
* Sparse Matrix: And this table is actually a matrix, and it is called ***sparse*** ***matrix***. In numerical analysis and scientific computing, a ***sparse*** ***matrix*** or ***sparse*** ***array*** is a matrix in which most of the elements are ***zero***.
* And the fact that we have a lot of ***0***'s is called ***sparsity*** and that's a very common notion in ***machine*** ***learning***. We work a lot with ***sparse*** ***matrix*** and we're trying to ***reduce*** ***sparsity*** as much as possible when we work with ML models.
* Creating this Sparse Matrix is actually the bag of words model itself. The bag of words model is basically to ***simplify*** all the ***reviews***, ***clean*** all the ***reviews*** to ***simplify the words*** and try to ***minimize the number of words***.
* Tokenization: And it's also about creating the sparse matrix through the process of ***tokenization***. Tokenization is the process of taking all the different words of the review and creating one column for each of these words (which we just discussed).
* Our goal: In the end our goal is to predict if a review is positive or negative. Our ML model needs to be trained on all these reviews, and find the correlations between the ***hints*** (the words) that tell if the review is ***positive*** or ***negative*** and it's ***true*** result whether it is ***positive*** or ***negative***.
* Now remember we did this kind of stuff in classification models. So, here we also *classify* a review is *positive* or *negative* according to the given data (to train the model).
* Now *reducing the word*s and *sparsity* actually *reducing* the *"independent variable"* of our classification problem.
* The dependent variable is a categorical variable a binary outcome: ***1*** is a review positive or ***0*** if the reviews negative. So we are doing nothing else than classification.
* So, as soon as we managed to create this bag of words, then we simply need to copy or classification teplates and create our ML model.
* At the end we will have our *matrix of features* or *matrix of independent variables* which will be the different word appearing in all the reviews here that will be the columns of our matrix and we'll have our *dependent variable vector* which will be the *result* whether the review is *positive* **(1)** or *negative* **(0)**.
* So that's why we need to create this bag of words model and now we understand very well why we had to clean all the text all the reviews. Because since we created one column for each word that is one independent variable for each word.
* Hence we *clean* the *reviews* and *simplify* them as much as *possible* to *reduce* the total number of *words* in the *corpus* and therefore the total number of *independent* *variables*.
* Creating the bag of words: we will create this Bag Of Word Model through the process of tokenization. To create our *Bag Of Word* we need to use CountVectorizer of sklearn.feature\_extraction.text. It will simply create the ***sparse matrix***. Since it's a ***class*** we will create an ***object*** of this class

**from** sklearn**.**feature\_extraction**.**text **import** CountVectorizer

cntVctzr = **CountVectorizer**()

* Now we apply ***fit\_transform()*** to our ***coRpus\_reViw.*** And also we convert the result to an array (matrix). We call it ***X*** since it will be our ***feature matrix***.

X = cntVctzr**.fit\_transform**(coRpus\_reViw)**.toarray**()

* At this stage we don't need any parameters for ***CountVectorizer()*** but we should have a look at all the parameters because you'll see that some of them are very useful:
* stop\_words: It is what we did before in the ***cleaning*** part. But we can remove those ***stopwords*** by using this parameter directly.
* lowercase: Again it is what we did already, but can directly apply here.
* token\_pattern: basically it is same as we did in *Regular* *Expressions*. i.e. removing all characters other than ***a-z*** and ***A-Z***.

Basically what I'm showing you here is that what we did before in the text-cleaning manually, we can do directly in this ***CountVectorizer()*** by playing with *different parameters*.

* But using these parameters are not the best way to do it. There are two reasons:
* The *first* *reason* is that we get to see how to *clean* the *text* step by step for *learning* *purposes*.
* *Second* *reason* is by cleaning the reviews *manually* that gives you *more* *options*. Sometimes you will need to do further cleaning to clean whatever text you're working with. For example:
* If you are doing ***NLP*** for ***web scrapping***, in that case the text are going to be some ***HTML*** ***pages*** and in those ***HTML*** ***pages*** you'll get some ***HTML*** ***code***. So you would need to add ***another*** ***option*** to clean these ***HTML*** ***texts***. Basically doing this *manually* gives us more control and *more* *options*.
* Improving the Bag of Word: In our sparse matrix we can see, it has ***1000 lines*** and ***1565 words***. But we can ***reduce*** the ***sparsity*** by using ***another*** ***parameter*** called max\_features. It will keep the most frequent words in your reviews so you know that will remove the ***non-relevant words*** that appear only ***once*** or ***twice*** of use. Lets set this ***max\_features= 1500***
* Now our sparse matrix of features contains *1500* *columns* instead of *1565*. And not only that reduces the sparsity but also that gives us most relevant words to train our algorithm which therefore has more chance to *make better correlations* between the *presence of the words* in the reviews and the *outcome of the reviews* whether they are *positive* or *negative*.
* Now about sparsity, we can also REDUCE *sparsity* using *dimensionality* *reduction* techniques which we will see later in this course in Chapter 9: Dimensionality Reduction.
* Then we create the dependent variable (positive or negative) from our given dataset.

#*creating the bag of words model*

**from** sklearn**.**feature\_extraction**.**text **import** CountVectorizer

#*cntVctzr = CountVectorizer()*

cntVctzr = **CountVectorizer**(max\_features= 1500)

X = cntVctzr**.fit\_transform**(coRpus\_reViw)**.toarray**()

y = dataset**.**iloc[:, 1]**.**values

* Building the model: We now build the model using any classification algorithm.
* Now we have X and y, exactly what we had in Chapter 3: Classification that is we have a ***matrix*** of independent variable **X** here that contains 1500 independent variables (which are words actually). Each line here corresponds to one specific review and ***1***: word is in the review, ***0***: word is not in the review.
* So basically that gives us a classification model, we will train a ML model that will try to understand the ***correlations*** between the ***presence of the words*** in the reviews and the outcome (is ***0*** if it's a ***negative*** review and ***1*** if it's a ***positive*** review).
* And we will use ***Dimensionality*** ***Reduction*** to ***reduce sparsity*** of this ***Sparse Matrix***. But we will do this in Chapter 9.
* Choosing the Classification Models: We just need to do some copy-paste of our Machine Learning Classification Models that we built in ***chapter 3***. In ***chapter 3***, we have

1. Logistic Regression,
2. KNN
3. SVM,
4. Kernel SVM,
5. Naïve Bayes,
6. Decision tree classification and
7. Random forest classification.

* So which one would be the best for natural language processing? Well we have two options her:

1. Since we already have our *matrix of features* and our *dependent variable vector* and since *everything* is *well-prepared* for our *ML* *classification* *models*. So we can apply all of them and compare the *Confusion* *Matrix*. By looking at the *accuracy* the number of *false positive* and *false* *negatives* and look at all their *performance* *criteria* to decide what would be the best model.
2. Secondly, from experience, most Data-Scientists commonly use *Naïve* *Bayes*, *Decision tree* classification and *Random forest* classification for Natural Language Processing.

* For this section we choose the *Naïve* *Bayes*.

#*NLP: Natural Language Precessing*

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

#*-------------- Data preprocessing --------------*

#*importing dataset*

        #*since we are using "tsv" ins tead of "csv" we need to specify some parameters.*

        #*Because "Pandas" expecting some "csv" files*

dataset = pd**.read\_csv**("Restaurant\_Reviews.tsv", delimiter = "\t", quoting = 3)

#*Cleaning the text*

**import** re

**import** nltk

#*nltk.download("stopwords")*

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

**from** nltk**.**stem**.**porter **import** PorterStemmer

#*print(f"{dataset['Review'][0]}")*

coRpus\_reViw = []

**for** i **in** **range**(0, 1000):

    rev\_W = re**.sub**("[^a-zA-Z]", " ", dataset["Review"][i])

    rev\_W = rev\_W**.lower**()   #*converting to Lower case*

    rev\_W = rev\_W**.split**()   #*converting "string" to "list"*

    #*reView = [wrd for wrd in rev\_W if wrd not in stopwords.words("english")]*

    #*Removing stopwords. And stemming : Finding the root of different versions of a same word*

    prt\_stmr = **PorterStemmer**()

    reView = [prt\_stmr**.stem**(wrd) **for** wrd **in** rev\_W **if** wrd **not** **in** **set**(stopwords**.words**("english"))]

    reVw = " "**.join**(reView)

    coRpus\_reViw**.append**(reVw)

#*creating the bag of words model*

**from** sklearn**.**feature\_extraction**.**text **import** CountVectorizer

#*cntVctzr = CountVectorizer()*

cntVctzr = **CountVectorizer**(max\_features= 1500)

X = cntVctzr**.fit\_transform**(coRpus\_reViw)**.toarray**()

y = dataset**.**iloc[:, 1]**.**values

#*using ---------------- Naïve Bayes ---------------*

#*Data Split*

**from** sklearn**.**model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size= 0.20, random\_state = 0)

#*# Feature-Scaling*

#*from sklearn.preprocessing import StandardScaler*

#*st\_x= StandardScaler()*

#*X\_train= st\_x.fit\_transform(X\_train)*

#*X\_test= st\_x.transform(X\_test)*

#*Fit train set to Naïve Bayes classifier: No parmeter is needed*

**from** sklearn**.**naive\_bayes **import** GaussianNB

clsFier = **GaussianNB**()

clsFier**.fit**(X\_train, y\_train) #*fit the dataset*

#*Predict*

y\_prd = clsFier**.predict**(X\_test)

#*Making the confusion matrix use the function "confusion\_matrix"*

**from** sklearn**.**metrics **import** confusion\_matrix

cm = **confusion\_matrix**(y\_true= y\_test, y\_pred= y\_prd)

#*parameters of cm: y\_true: Real values, y\_pred: Predicted value*

accuracy = (cm[0][0] + cm[1][1])/X\_test**.**shape[0]

#*How can I find the length of a row (or column) of this matrix? Equivalently, how can I know the number of rows or columns?*

#*shape is a property of both numpy ndarray's and matrices.*

    #*A.shape*

#*will return a tuple (m, n), where m is the number of rows, and n is the number of columns.*

#*import nltk*

#*from nltk.corpus import stopwords*

#*print(stopwords.words('english'))*

#*print(set(stopwords.words('english')))*

#*python prctc\_nlp.py*

|  |  |
| --- | --- |
| * We take the ***test size*** 20%. * We also ***don’t need*** any ***feature scaling*** because we are only dealing with ***0***'s and ***1***'s. * From the confusion matrix we that out of 200 test points, ***146***(true negative 55 + true positive 91) are ***correct*** predictions and ***54*** (false positive 42 + false negative 12) are ***incorrect*** predictions and the accuracy is 73%. * From ***97*** negative review we predicted ***55*** correct * From ***103*** positive review we predicted ***91*** correct |  |

posve = [k for k in y\_test if k==1]

total\_posve = len(posve)