Following are the steps required to create a text classification model in Python:

1. Importing Libraries
2. Importing The dataset
3. Text Preprocessing
4. Converting Text to Numbers
5. Training and Test Sets
6. Training Text Classification Model and Predicting Sentiment
7. Evaluating The Model
8. Saving and Loading the Model

2- Importing the dataset :

We will use the load\_files function from the sklearn\_datasets library to import the dataset into our application. The load\_files function automatically divides the dataset into data and target sets(Load text files with categories as subfolder names.). For instance, in our case, we will pass it the path to the "txtsentoken" directory. The *load\_files* will treat each folder inside the "txtsentoken" folder as one category and all the documents inside that folder will be assigned its corresponding category.

movie\_data = load\_files(r"D:\txt\_sentoken")

X, y = movie\_data.data, movie\_data.target

In the script above, the load\_files function loads the data from both "neg" and "pos" folders into the X variable, while the target categories are stored in y. Here X is a list of 2000 string type elements where each element corresponds to single user review. Similarly, y is a numpy array of size 2000. If you print y on the screen, you will see an array of 1s and 0s. This is because, for each category, the load\_files function adds a number to the target numpy array. We have two categories: "neg" and "pos", therefore 1s and 0s have been added to the target array.

#### 3- Text Preprocessing

Once the dataset has been imported, the next step is to preprocess the text. Text may contain numbers, special characters, and unwanted spaces

In the script above we use [Regex Expressions from Python re library](https://stackabuse.com/using-regex-for-text-manipulation-in-python/) to perform different preprocessing tasks. We start by removing all non-word characters such as special characters, numbers, etc.

Next, we remove all the single characters. For instance, when we remove the punctuation mark from "David's" and replace it with a space, we get "David" and a single character "s", which has no meaning. To remove such single characters we use \s+[a-zA-Z]\s+ regular expression which substitutes all the single characters having spaces on either side, with a single space.

Next, we use the \^[a-zA-Z]\s+ regular expression to replace a single character from the beginning of the document, with a single space. Replacing single characters with a single space may result in multiple spaces, which is not ideal.

We again use the regular expression \s+ to replace one or more spaces with a single space. When you have a dataset in bytes format, the alphabet letter "b" is appended before every string. The regex ^b\s+ removes "b" from the start of a string. The next step is to convert the data to lower case so that the words that are actually the same but have different cases can be treated equally.

The final preprocessing step is the [lemmatization](https://en.wikipedia.org/wiki/Tf%E2%80%93idf). In lemmatization, we reduce the word into dictionary root form. For instance "cats" is converted into "cat". Lemmatization is done in order to avoid creating features that are semantically similar but syntactically different. For instance, we don't want two different features named "cats" and "cat", which are semantically similar, therefore we perform lemmatization.

#### 4- Converting Text to Numbers

Different approaches exist to convert text into the corresponding numerical form. [The Bag of Words Model](https://en.wikipedia.org/wiki/Bag-of-words_model) and the [Word Embedding Model](https://en.wikipedia.org/wiki/Word_embedding) are two of the most commonly used approaches. In this article, we will use the bag of words model to convert our text to numbers.

\*\* Bag of Words \*\*

The script above uses CountVectorizer class from the sklearn.feature\_extraction.text library. There are some important parameters that are required to be passed to the constructor of the class. The first parameter is the max\_features parameter, which is set to 1500. This is because when you convert words to numbers using the bag of words approach, all the unique words in all the documents are converted into features. All the documents can contain tens of thousands of unique words. But the words that have a very low frequency of occurrence are unusually not a good parameter for classifying documents. Therefore we set the max\_features parameter to 1500, which means that we want to use 1500 most occurring words as features for training our classifier

The next parameter is min\_df and it has been set to 5. This corresponds to the minimum number of documents that should contain this feature. So we only include those words that occur in at least 5 documents. Similarly, for the max\_df, feature the value is set to 0.7; in which the fraction corresponds to a percentage. Here 0.7 means that we should include only those words that occur in a maximum of 70% of all the documents. Words that occur in almost every document are usually not suitable for classification because they do not provide any unique information about the document.

Finally, we remove the [stop words](https://en.wikipedia.org/wiki/Stop_words) from our text since, in the case of sentiment analysis, stop words may not contain any useful information. To remove the stop words we pass the stopwords object from the nltk.corpus library to the stop\_wordsparameter.

##

The fit\_transform function of the CountVectorizer class converts text documents into corresponding numeric features.

##### 5- Finding TFIDF

##### Finding TFIDF

The bag of words approach works fine for converting text to numbers. However, it has one drawback. It assigns a score to a word based on its occurrence in a particular document. It doesn't take into account the fact that the word might also be having a high frequency of occurrence in other documents as well. [TFIDF](https://en.wikipedia.org/wiki/Tf%E2%80%93idf) resolves this issue by multiplying the term frequency of a word by the inverse document frequency. The TF stands for "Term Frequency" while IDF stands for "Inverse Document Frequency".

The term frequency is calculated as:

Term frequency = (Number of Occurrences of a word)/(Total words in the document)

And the Inverse Document Frequency is calculated as:

IDF(word) = Log((Total number of documents)/(Number of documents containing the word))

The TFIDF value for a word in a particular document is higher if the frequency of occurrence of that word is higher in that specific document but lower in all the other documents.

------------------------------------------------------------

#### 6- Training and Testing Sets

Like any other supervised machine learning problem, we need to divide our data into training and testing sets. To do so, we will use the train\_test\_split utility from the sklearn.model\_selectionlibrary. Execute the following script: