Text Analytics

With the availability of comment or feedback feature for many of the things / products, users of those products are able to provide a valuable qualitative feedback for it. This qualitative feedback carries a lot of power in it which can help in making your product / service a grand success or destroy the product completely. That's why organizations can not ignore this feedback and has started taking it very seriously. The primary problem they face with such feedback is its completely unstructured - it can have text, numbers, images etc added in it which makes it very difficult when some models needs to be based on it. That's why this field of data analytics - text mining has started gaining a lot of importance in todays world!

Text mining is "the discovery by computer of new, previously unknown information, by automatically extracting information from different written resources." Written resources can be websites, books, emails, reviews, articles. Text mining, also referred to as text data mining, roughly equivalent to text analytics, is the process of deriving high-quality information from text. 'High quality' in text mining usually refers to some combination of relevance, novelty, and interest.

Text mining usually involves the process of

- structuring the input text (usually parsing, along with the addition of some derived linguistic features and the removal of others, and subsequent insertion into a database)
- deriving patterns within the structured data
- evaluation and interpretation of the output

Typical text mining tasks include

- text categorization
- text clustering
- concept/entity extraction
- sentiment analysis
- document summarization
- entity relation modeling (i.e., learning relations between named entities)

Text Mining with Stackoverflow data

Stack Overflow is an open community for anyone that codes. It help you get answers to your toughest coding questions, share knowledge with your coworkers in private, and find your next dream job. It is an question and answer site for professional and enthusiast programmers.The website serves as a platform for users to ask and answer questions, and, through membership and active participation, to vote questions and answers up or down and edit questions and answers. It is a privately held website, the flagship site of the Stack Exchange Network created in 2008 by Jeff Atwood and Joel Spolsky. It features questions and answers on a wide range of topics in computer programming. The name for the website was chosen by voting in April 2008 by readers of Coding Horror, Atwood's popular programming blog.

We are going to consider the text mining with reference to the Stackoverflow data. All of us must have used stackoverflow question - answer forum for raising the programming questions or providing the answers. One of the biggest challege lies there for organizing the questions in appropriate categories based on the progamming languages. Through this exercise, we are going to see how we can interpret the question and classify it based on programming language. The dataset looks very simple - it has only two fields in it

- an question / review text termed as "post"
- a tag for the programming language like Java, JS etc.

Lets explore this process of classifying the questions / reviews using text mining techniques.The data can be obtained from this link.

We will follow the process outlined below to complere the text classification task

- Importing the data
- Exploring the dataset
- Data preprocessing
- Naive-bayes model for post classification

1 Importing the data

1.1. Import required data processing libraries

Lets import the text data available in the csv format.

In [1]:

```
#import data management libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1.2. Import required data

In [23]:

```
text_data = pd.read_csv('stack-overflow-data.csv')
```

In [24]:

```
#lets have quick look at first and last five rows of data
text_data.head(5)
```

Out[24]:

	post	tags
0	.net framework 4 redistributable just wonderi...	.net
1	connection in .net i got <blockquote> ...	.net
2	in asp.net mvc4 using javascript when the but...	.net
3	how to upgrade from one version to latest vers...	.net
4	when to use abstract class and when to use an ...	.net

In [25]:

```
text_data.tail(5)
```

Out[25]:

	post	tags
9994	how to write this query using using clause. i ...	sql
9995	query issue in sql query for finding empty rec...	sql
9996	sql throws an error for trigger delete i have...	sql
9997	delete statement with a subquery i have to wr...	sql
9998	select from table where both id exist i need ...	sql

Seems there are 10000 records present in the dataset having two columns post and tags.

1.3. Quick look at data

Lets explore the data quickly.

In [26]:

```
#viewing random samples in the data
print('Sample 1 - only post:\n\n',text_data.iloc[0,0])
```

```
Sample 1 - only post:

 .net framework 4 redistributable  just wondering where we can get .
net framework 4 beta redistributable. we would like to include it in
our cd so we can distribute it to our clients and they need to insta
ll it from the cd and not from web as it is not necessary to have in
ternet for our application. any suggestions will be appreciated. tha
nks navin
```

In [27]:

```
print('Sample 2 - only post:\n\n',text_data.iloc[15,0])
```

```
Sample 2 - only post:

 add dynamic dotnet webcontrol into attribute value of static html
easy one to explain. is there any way i can do this:   <pre><code>&l
t;div id= header  style= &lt;asp:literal runat= server  id= litbackg
roundimage &gt;&lt;/asp:literal&gt;  &gt; </code></pre>   it looks v
alid  but visual studio will not recognise litbackgroundimage as a v
alid control in the code-behind. setting the div as runat= server  w
on t work either because the style property is read-only.    any sug
gestions gratefully received
```

In [28]:

```
print('Sample 2: - only tag\n\n',text_data.iloc[15,1])
```

```
Sample 2: - only tag

 .net
```

If we take a careful look at these samples, lot of things are appearing in it apart from gereral words like programming language symbols , tags, code etc. From this, we can conclude that the these posts data needs definitive cleaning.

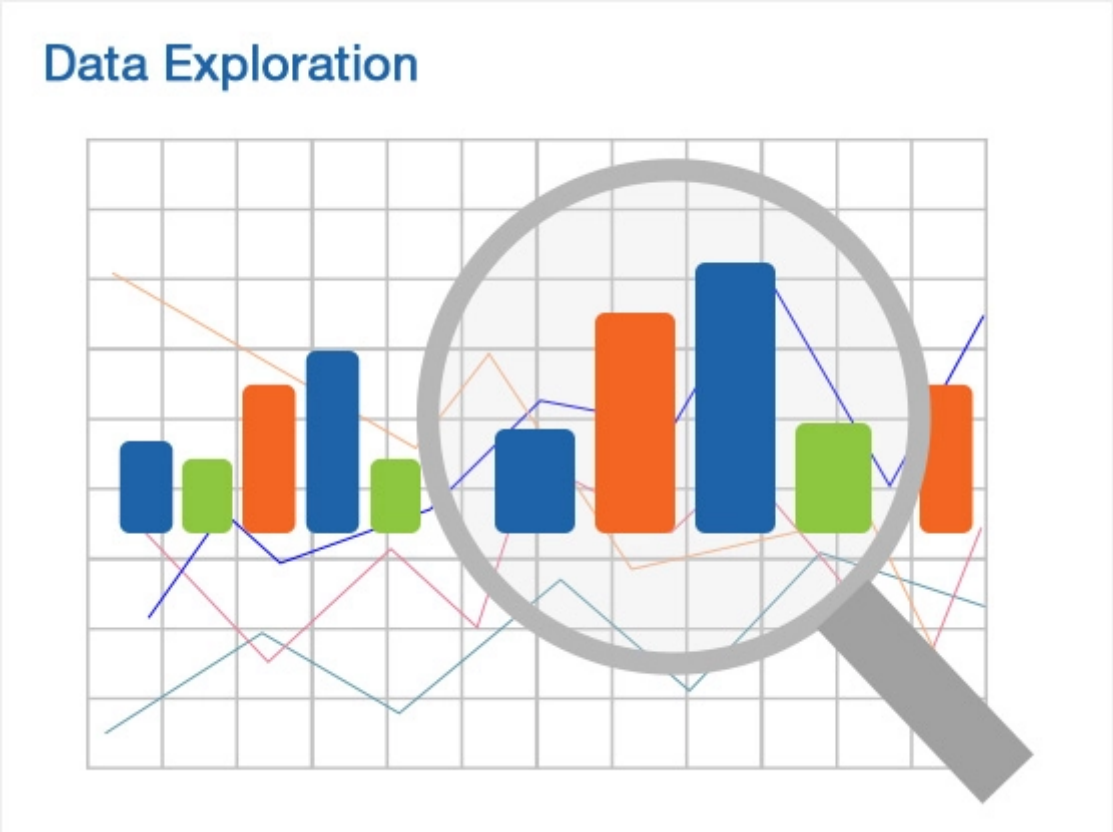
1.4. Quick look at structure of data

In [29]:

```
#size related queries
print("Shape of data:", text_data.shape)
print('Number of Rows:',text_data.shape[0])
print('Number of Columns:',text_data.shape[1])
print('Number of Classes/Tags:',len(text_data.tags.value_counts()))
print('Number of words in dataset:',text_data['post'].apply(lambda x: len(x.spli
t(' '))).sum())
```

```
Shape of data: (9999, 2)
Number of Rows: 9999
Number of Columns: 2
Number of Classes/Tags: 20
Number of words in dataset: 2604650
```

2 Exploring the dataset



Now lets spend some more time to understand our dataset in closer manner. For that purpose we need to play around with the columns present within the dataset.

2.1. Column descriptions

We know there are only two columns present in the dataset with 9999 posts in it but lets confirm it again.

In [30]:

```
text_data.shape
```

Out[30]:

```
(9999, 2)
```

In [31]:

```
text_data.columns
```

Out[31]:

```
Index(['post', 'tags'], dtype='object')
```

Lets print the metadata about the columns.

In [32]:

```
text_data.info()
```

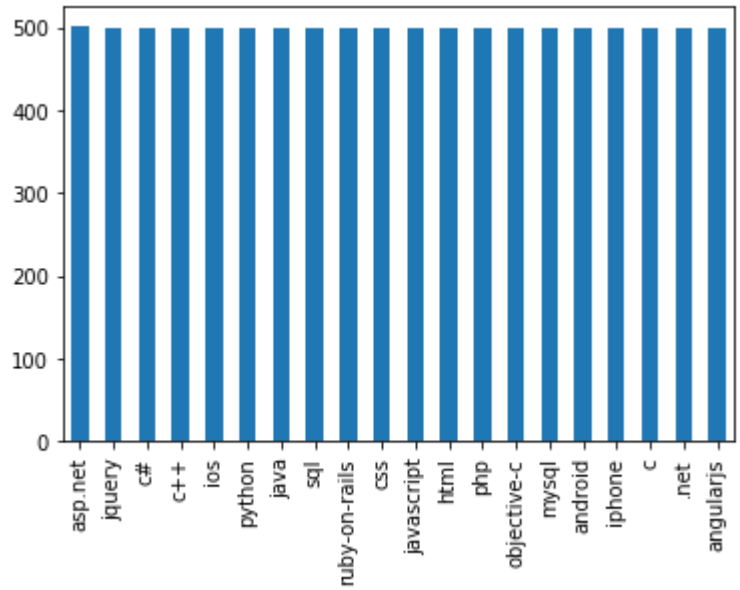
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9999 entries, 0 to 9998
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   post    9999 non-null   object
 1   tags    9999 non-null   object
dtypes: object(2)
memory usage: 156.4+ KB
```

2.2. Data distribution

Lets quickly see the distribution of posts by tags.

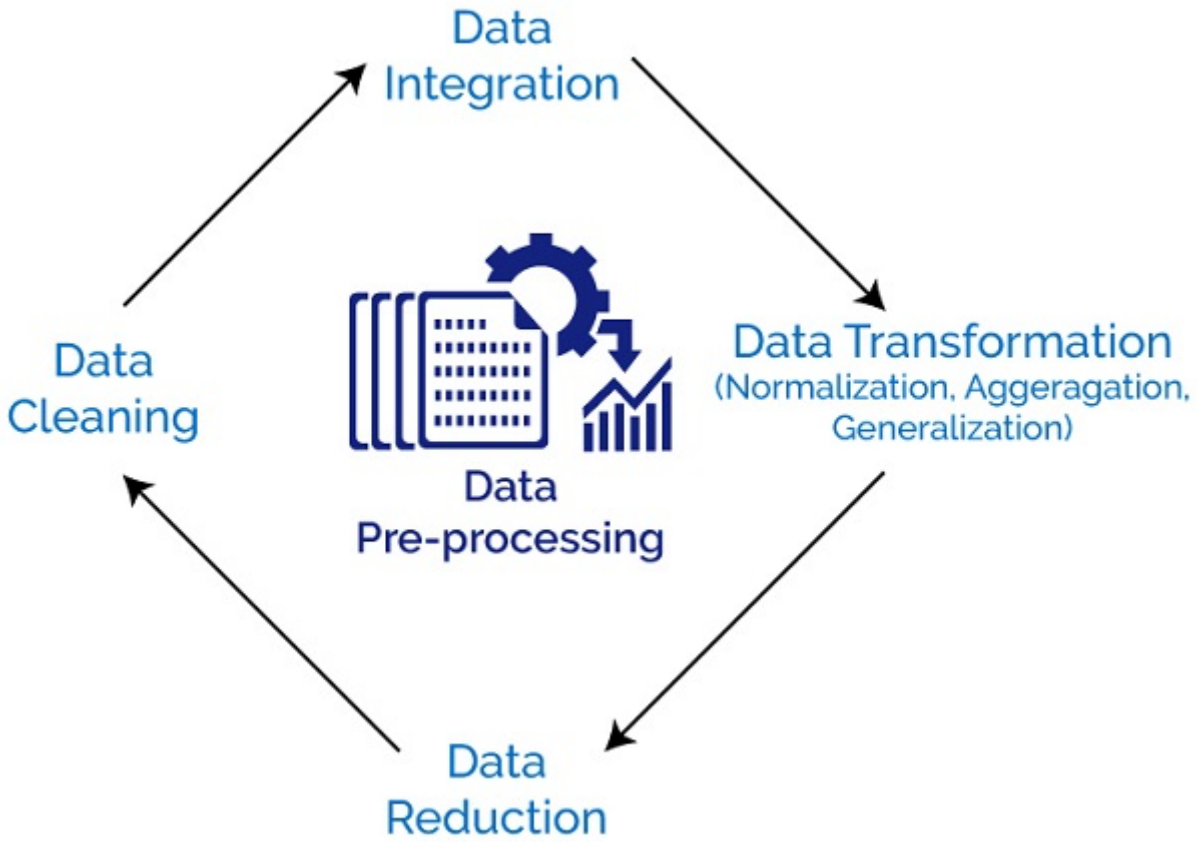
In [33]:

```
#Plot frequency chart for the classes/tags
text_data.tags.value_counts().plot(kind='bar')
plt.show()
```



This portrays that the dataset contains equal representation of all the classes/tags - i.e. 500 posts per programming language. This is good symbol that we don't need to worry about the class imbalance problem.

3 Data preprocessing



Independant variables (features) are not directly available in text data, hence we need to preprocess that data to extract out the features from it. Lets try to understand what are the steps involved in this process and available approached for tacking them.

3.1. Null check

In [34]:

```
text_data.isnull().sum()
```

Out[34]:

```
post    0
tags    0
dtype: int64
```

Fortunately we dont have any null values present in our data. But if it is then following code snippet can be used to get rid of it. The decision to remove all null values should be taken with lot of care as because of it we might be loosing the data present in other cells of the same record. Hence a caution is required before doing it!

In [35]:

```
#Remove rows/instances without any tag
text_data = text_data[pd.notnull(text_data['tags'])]
```

In [36]:

```
text_data.shape
```

Out[36]:

```
(9999, 2)
```

3.2. Symbol check

Its quite possible that the questions / posts have single characters, special characters, symbols , numbers present in it which may not be having a specific meaning but appearing in the text. Lets get rid of them. A regular expression or regex is a special text or string to define a search pattern. The "re" library provides operations to add, edit, rate and test regular expressions.

In [37]:

```
#pre-declared definitions
import re
BAD_SYMBOLS_TO_BE_REPLACED = re.compile('[^0-9a-z #+_]')
SPECIAL_CHARS_TO_BE_REPLACED =re.compile('[/(){}\[\]\|@,;]')
NUMBER_TO_BE_REPLACED = re.compile('[0-9]')
```

Lets write a function that helps us in doing this replacement.Beautiful Soup is a Python library for pulling data out of HTML and XML files. As our text data is having some html in it, we can use it for the processing here.

In [38]:

```
from bs4 import BeautifulSoup
```

In [39]:

```
def process_symbols(post_text):

    #decode html text
    post_text = BeautifulSoup(post_text,"lxml").text

    #converting all words to lowercase such that words like 'Word' and 'word' ar
e not treated differently
    post_text = post_text.lower()

    #Replacing symbols defined in REPLACE_BY_SPACE with ' '
    post_text = SPECIAL_CHARS_TO_BE_REPLACED.sub(' ',post_text)

    #Removing numbers altogether from text
    post_text = NUMBER_TO_BE_REPLACED.sub('',post_text)

    #Removing unnatural symbols
    post_text = BAD_SYMBOLS_TO_BE_REPLACED.sub('',post_text)

    return post_text
```

In [40]:

```
#Applying Preprocessing Function
text_data['post'] =  text_data['post'].apply(process_symbols)
```

In [41]:

```
#Checking Processed Samples
print('Sample 1 (Processed):\n',text_data.iloc[0,0])
print('\n\nSample 2 (Processed):\n',text_data.iloc[15,0])
```

```
Sample 1 (Processed):
 net framework  redistributable  just wondering where we can get net
framework  beta redistributable we would like to include it in our c
d so we can distribute it to our clients and they need to install it
from the cd and not from web as it is not necessary to have internet
for our application any suggestions will be appreciated thanks navin


Sample 2 (Processed):
 add dynamic dotnet webcontrol into attribute value of static html
easy one to explain is there any way i can do this   div id header
style aspliteral runat server  id litbackgroundimage  aspliteral
it looks valid  but visual studio will not recognise litbackgroundim
age as a valid control in the codebehind setting the div as runat se
rver  won t work either because the style property is readonly    an
y suggestions gratefully received
```

In [42]:

```
print('Present number of words:',text_data['post'].apply(lambda x: len(x.split('
'))).sum())
```

```
Present number of words: 2922606
```

The unique number of words present in the posts is too high to prepare any vector representation of the same. So lets explore some ways by which we can construct these vectors and then also understand how we can identify the features (words) which are meaningful for the classification of posts.

3.3. Understand Document models

Independent variables in the text data needs to be extracted out of the given dataset. Simple way is to just check whether the word is present in the post or not - termed as BoW. Here each post is considered as bag of words. Each record is called as document and the all documents are together called as corpus.

BoW (Bag of Words) Model

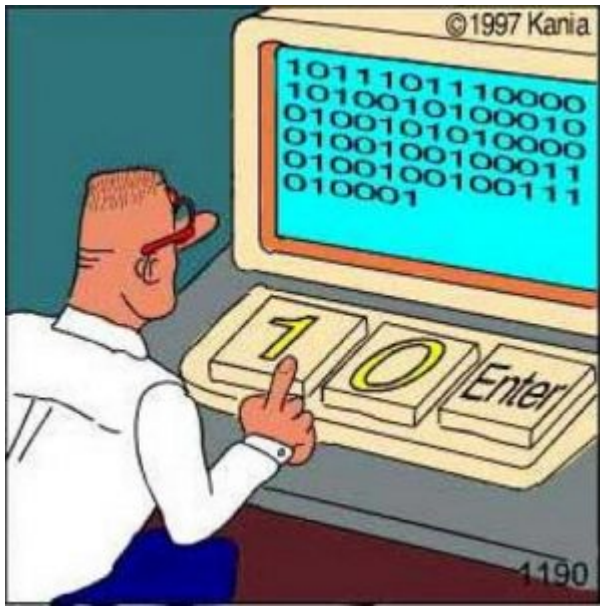
Steps involved

- Prepare the dictionary of all the words used in the corpus
- Convert each document to a vector that represents words present in the document

This BoW model can be created by three ways

- Count vector model
- Term Frequency (TF) Vector model
- Term Frequency - Inverse Document Frequency (TF-IDF) Vector model

Count vector model



Tf-IDf calculations.

Example documents —.

Doc 1 - I stay in Pune.

Doc 2. - I like Pune lot, I stay there.

Count vector. (Presence, Absense)

Documents. words →	I	stay	in	Pune	like	lot	there
↓							
D1	1	1	1	1	0	0	0
D2	1	1	0	1	1	1	1

Count vector (Countwise).

Docs. words →	I	stay	in	Pune	like	lot.	there
↓							
D1	1	1	1	1	0	0	0
D2	2	1	0	1	1	1	1

Term Frequency vector model

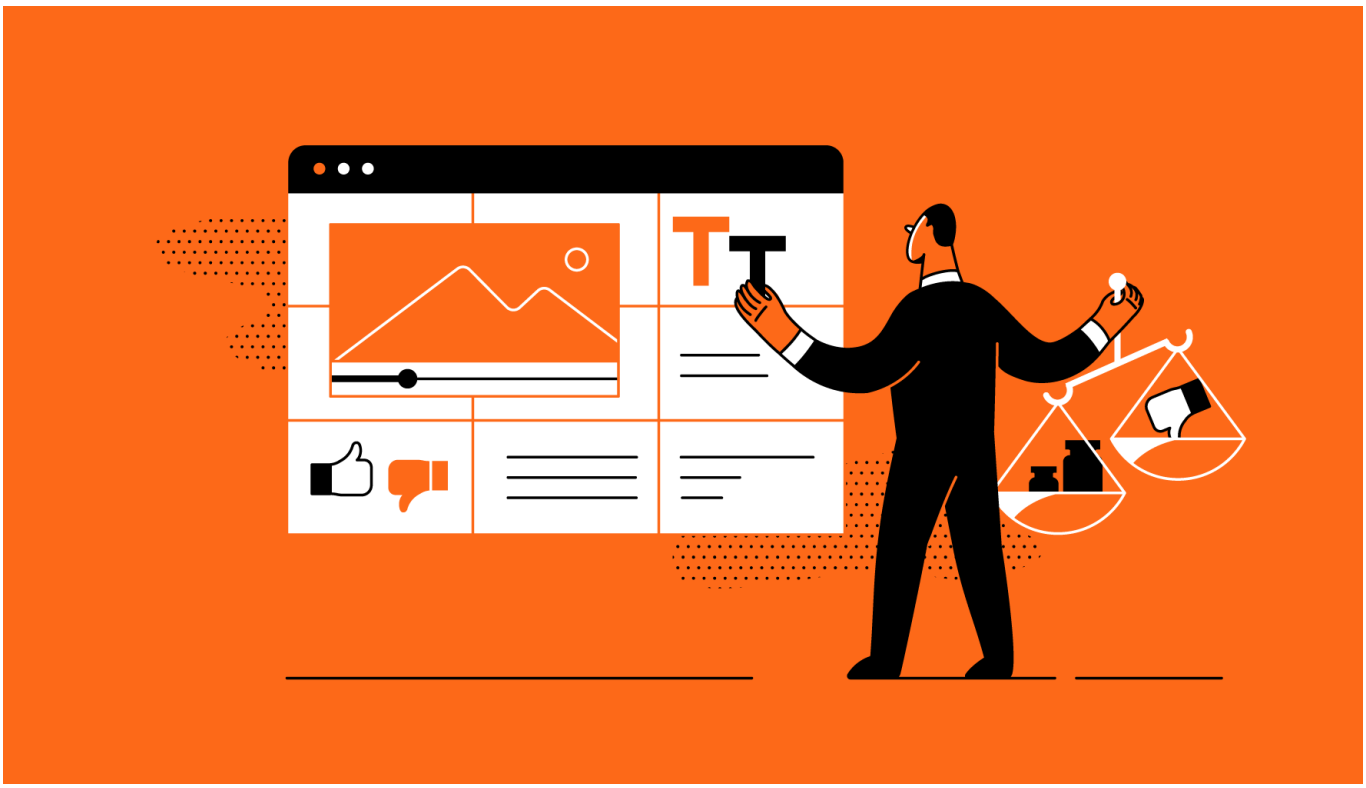
Tf Vector model.

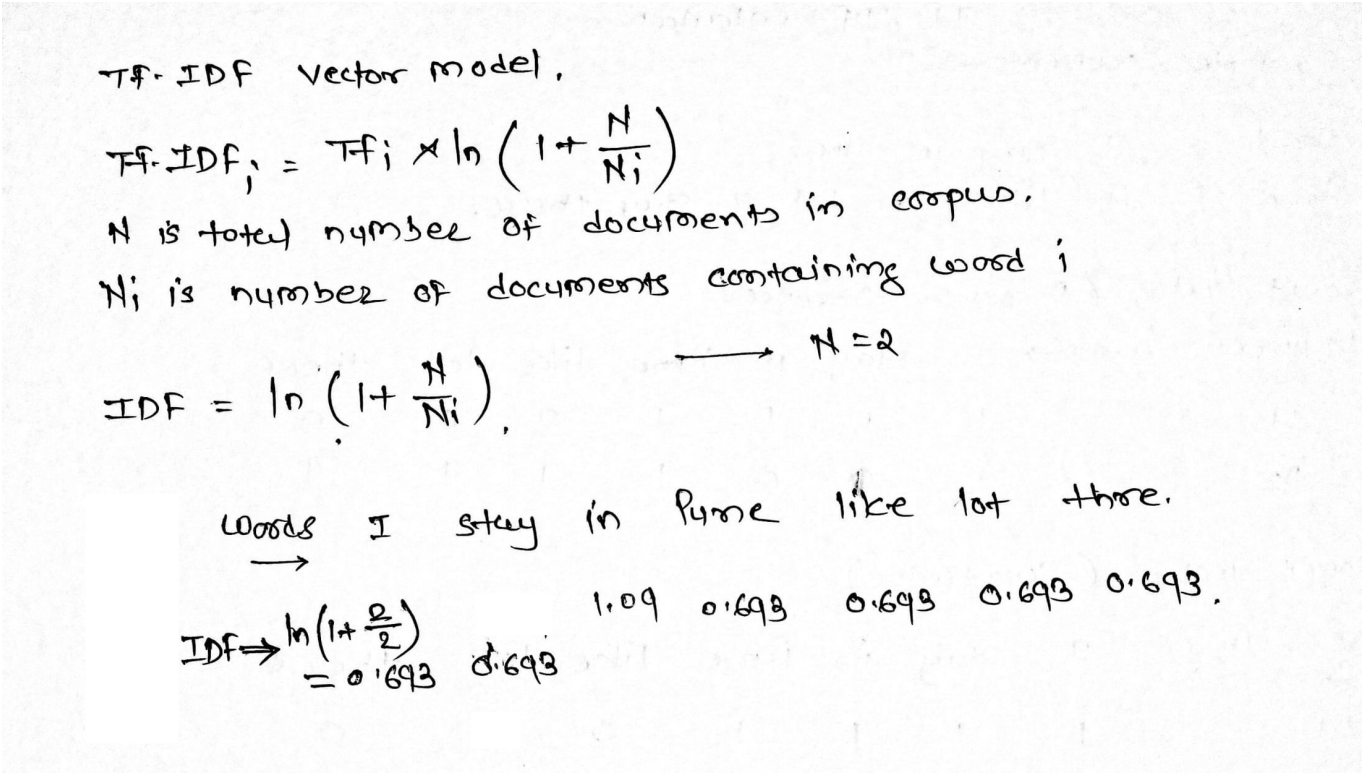
Term freq^n TF_i = $\frac{\#\text{ of occurrances of word } i \text{ in the document}}{\text{Total number of words in document}}$

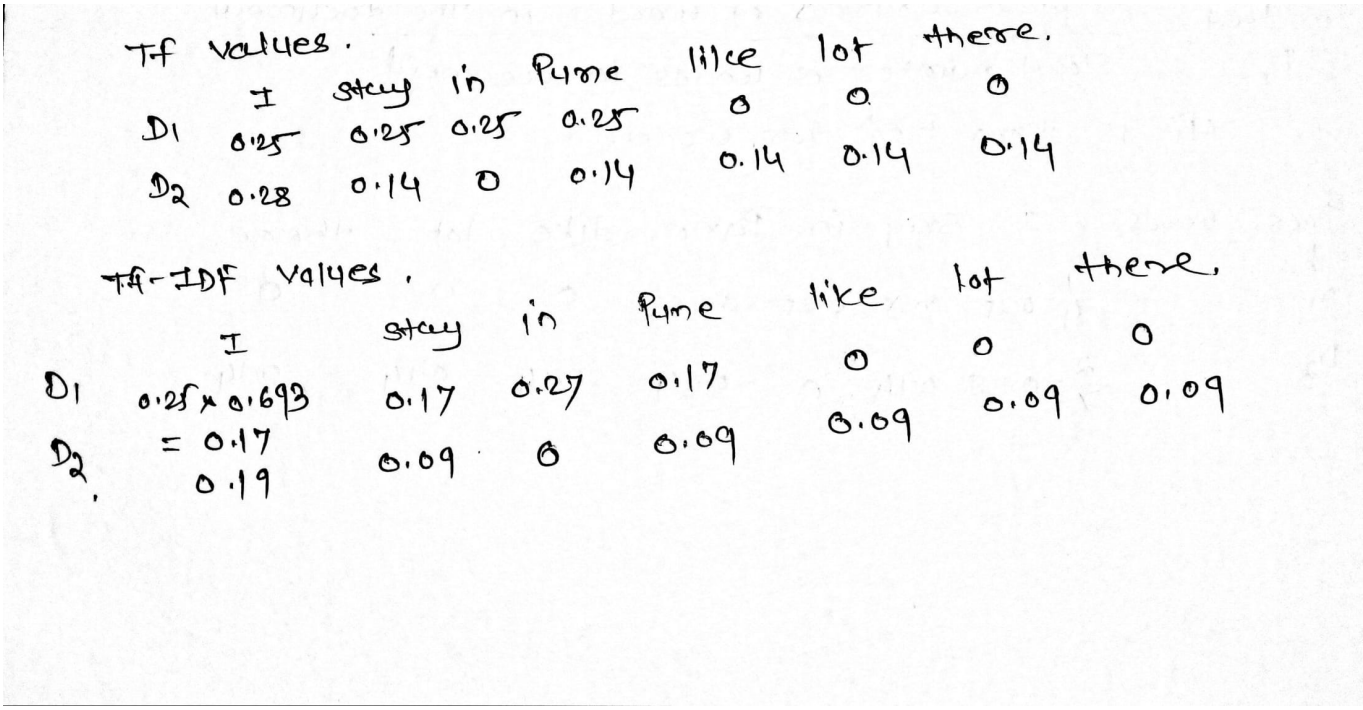
TF_i is term freq^n for word.

Docs.	words →	I	stay	in	Pune	like	lot	there.
↓								
D_1		$\frac{1}{4}=0.25$	0.25	0.25	0.25	0	0	0
D_2		$\frac{2}{7}=0.28$	0.14	0	0.14	0.14	0.14	0.14

Term Frequency - Inverse Term Frequenecy vector model







3.4. Preparing Count Vectors

Each document (post) needs to be converted into vectors which can be easily done with the help of sklearn library. Lets explore CountVectorizer to create count vectors. Here, the documents are represented as the number of times each word appears in the document.

3.4.1 Prepare word dictionary

In [43]:

```
# Import the required module
from sklearn.feature_extraction.text import CountVectorizer

# Initialize the CountVectorizer
count_vectorizer = CountVectorizer()

# Create the dictionary from the corpus
feature_vector = count_vectorizer.fit( text_data.post )

# Get the feature names
features = feature_vector.get_feature_names()

print( "Total number of features: ", len(features))
```

```
Total number of features:  75106
```

Still total number of features is too high. This is nothing but our dictionary of words. Lets have quik look at some of them through random sampling.

In [44]:

```
import random
random.sample(features, 5)
```

Out[44]:

```
['comid', 'jogging', 'collectionsort', 'sessions', 'token_count']
```

3.4.2 Transform dictionary to count vector

Lets convert each post into a count vector by using the transform method.

In [45]:

```
data_features = count_vectorizer.transform( text_data.post )
type(data_features)
print(data_features.shape)
```

```
(9999, 75106)
```

This is very big, sparse matrix that we have obtained. Each records is converted into a count vector which has 75106 dimensions present in it and the cell value corresponds to the number of times that word appears in document.

3.4.3 Showing Document vectors

In order to see the document vector, need to convert into dataframe.

In [47]:

```
# Converting the matrix to a dataframe
data_df = pd.DataFrame(data_features.todense())

# Setting the column names to the features i.e. words
data_df.columns = features
```

Execute the following code only if you dont hit the memory error in above cell.

First record

In [48]:

```
text_data[0:1]
```

Out[48]:

	post	tags
0	net framework redistributable just wondering...	.net

Transformed vector

In [50]:

```
data_df.iloc[0:1, 100:105]
```

Out[50]:

	__main__	__main__myclass	__main__xint	__mainc	__maincpp
0	0	0	0	0	0

3.4.4 Removing low frequency words

As the number of words appearing in the dictionary is too large, we can think of removing the words which are not that frequent so that we can have some reduction in the feature list size.

In [53]:

```
# summing up the occurances of features column wise
features_counts = np.sum( data_features.toarray(), axis = 0 )

feature_counts_df = pd.DataFrame( dict( features = features,
counts = features_counts ) )
feature_counts_df
```

Out[53]:

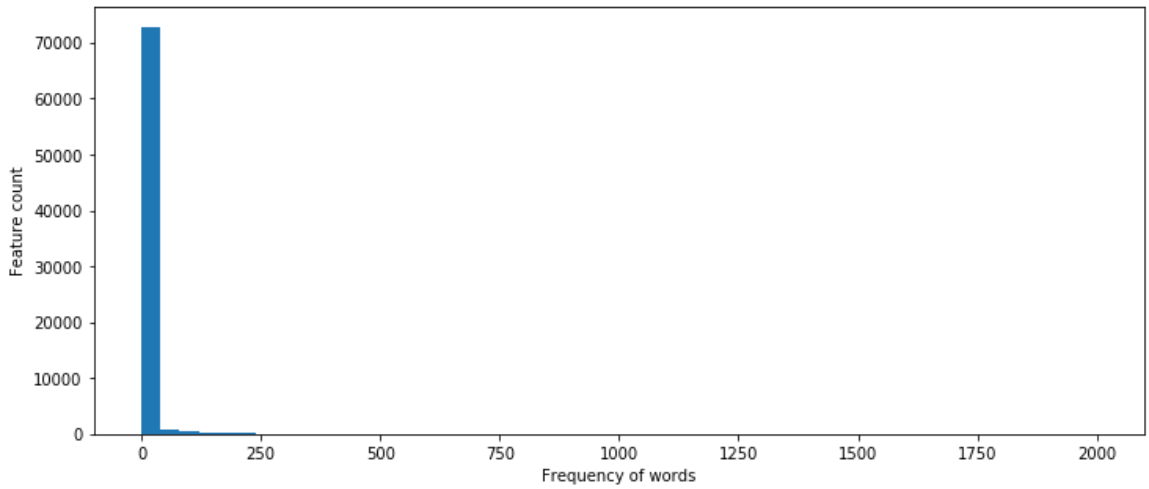
	features	counts
0	__	19
1	____	14
2	_____	25
3	______	1
4	_________	1
...	...	...
75101	zz_	1
75102	zzbjye	1
75103	zzx	1
75104	zzz	9
75105	zzzjpg	1

75106 rows × 2 columns

Frequency of each word can be analyzed by histogram.

In [54]:

```
plt.figure( figsize=(12,5))
plt.hist(feature_counts_df.counts, bins=50, range = (0, 2000));
plt.xlabel( 'Frequency of words' )
plt.ylabel( 'Feature count' );
```



Large number of features have very less occurances. So lets find out those rare words.

In [55]:

```
len(feature_counts_df[feature_counts_df.counts == 1])
```

Out[55]:

40232

In [56]:

```
len(feature_counts_df[feature_counts_df.counts == 2])
```

Out[56]:

13181

In [57]:

```
len(feature_counts_df[feature_counts_df.counts == 3])
```

Out[57]:

5655

We can see that features which are appearing only once are 40232, appearing twice are 13181, appearing thrice are 5655 and so on.

3.4.5 Prepare count vector by restrciting features

Lets restrcit ourself to only 5000 features.

In [60]:

```
# Initialize the CountVectorizer
count_vectorizer = CountVectorizer(max_features=5000)

# Create the dictionary from the corpus
feature_vector = count_vectorizer.fit( text_data.post )

# Get the feature names
features = feature_vector.get_feature_names()

# Transform the document into vectors
train_ds_features = count_vectorizer.transform( text_data.post )

# Count the frequency of the features
features_counts = np.sum( train_ds_features.toarray(), axis = 0 )
feature_counts = pd.DataFrame( dict( features = features,
counts = features_counts ) )
```

In [61]:

```
feature_counts.sort_values('counts', ascending = False)[0:15]
```

Out[61]:

	features	counts
4442	the	42697
4519	to	30458
2276	is	18451
2131	in	17885
192	and	15477
3012	of	12308
4461	this	11840
2287	it	11687
2088	if	9970
1755	for	9721
4440	that	8638
1967	have	7737
2804	my	6959
766	class	6752
605	but	6630

If we look at these top 15 features, we can see that many words are too common and does not do a value addition, so makes sense to remove them out.

3.4.6 Locate stop words

sklearn provides a commonly appearing stop words in English language so lets use it to remove the stop words - i.e. commonly appearning irrelevant words.

In [62]:

```
# import the stop words list
from sklearn.feature_extraction import text
my_stop_words = text.ENGLISH_STOP_WORDS

#Printing first few stop words
print("Few stop words: ", list(my_stop_words)[0:10])
```

```
Few stop words:  ['always', 'mostly', 'to', 're', 'neither', 'thus',
'alone', 'therefore', 'all', 'however']
```

3.4.6 Prepare count vector by removing stop wards

In [63]:

```
# Setting stop words list
count_vectorizer = CountVectorizer( stop_words = my_stop_words, max_features = 5
000 )
feature_vector = count_vectorizer.fit( text_data.post )
train_ds_features = count_vectorizer.transform( text_data.post )

features = feature_vector.get_feature_names()
features_counts = np.sum( train_ds_features.toarray(), axis = 0 )
feature_counts = pd.DataFrame( dict( features = features,
counts = features_counts ) )
```

In [64]:

```
feature_counts.sort_values('counts', ascending = False)[0:15]
```

Out[64]:

	features	counts
756	class	6752
1306	div	5860
2228	int	5457
811	code	4935
1817	function	4660
2887	new	4638
2058	id	4421
4231	string	4288
2473	like	3872
4750	using	3675
4772	value	3671
4846	want	3661
1653	file	3539
3736	return	3532
3464	public	3030

Now the stop words have been removed. But there is a possibility that the word is apperning in different formats like "wait", "waiting" which will increase feature count. Lets try to resolve it.

3.4.7 Stemming and Lemmatization

Stemming and lemmmatization are two popular techniques which helps us to identify the similar words.
Stemming - This removes the differences between the inflected forms of a word to reduce each word to its root form, usually done by chopping off the end of words. The root form may not be a real word.
PostStemmer and LancasterStemmer are two popular alogs for the same.
Lemmatization - This takes morpholigical analysis of word into consideration, also uses English dictionary into consideration.
NLTK - Natural Language toolkit and consits of highly optimized libraries which helps users to carry out common text operations easily.

NLTK supports PorterStemmer, EnglishStemmer and LancasterStemmer for stemming and WordNetLemmatizer for lemmatization. These features can be used in CountVectorizer. Lets create a utility method that

- **takes the documents**
- **tokenizes it to create words**
- **stem the words**
- **remove the stop words and return the final set of words**

In [66]:

```
from nltk.stem.snowball import PorterStemmer

stemmer = PorterStemmer()
analyzer = CountVectorizer().build_analyzer()

#Custom function for stemming and stop word removal
def stem_words(doc):
    ### Stemming of words
    stemmed_words = (stemmer.stem(w) for w in analyzer(doc))

    ### Remove the words in stop words list
    non_stop_words = [ word for word in list(set(stemmed_words) - set(my_stop_wo
rds)) ]
    return non_stop_words
```

3.4.8 Prepare count vector using stemming

Lets prepare a CountVectorizer using the stemming

In [67]:

```
count_vectorizer = CountVectorizer( analyzer=stem_words, max_features = 5000)
feature_vector = count_vectorizer.fit( text_data.post )
train_ds_features = count_vectorizer.transform( text_data.post )

features = feature_vector.get_feature_names()
features_counts = np.sum( train_ds_features.toarray(), axis = 0 )
feature_counts = pd.DataFrame( dict( features = features,
counts = features_counts ) )
feature_counts.sort_values( "counts", ascending = False )[0:15]
```

Out[67]:

	features	counts
4416	thi	5922
4709	use	4177
818	code	3242
4824	want	2893
2475	like	2681
4530	tri	2666
4910	work	2496
241	ani	2298
3684	return	1998
765	class	1971
1778	function	1945
4407	thank	1905
2873	new	1903
4745	valu	1851
2866	need	1782

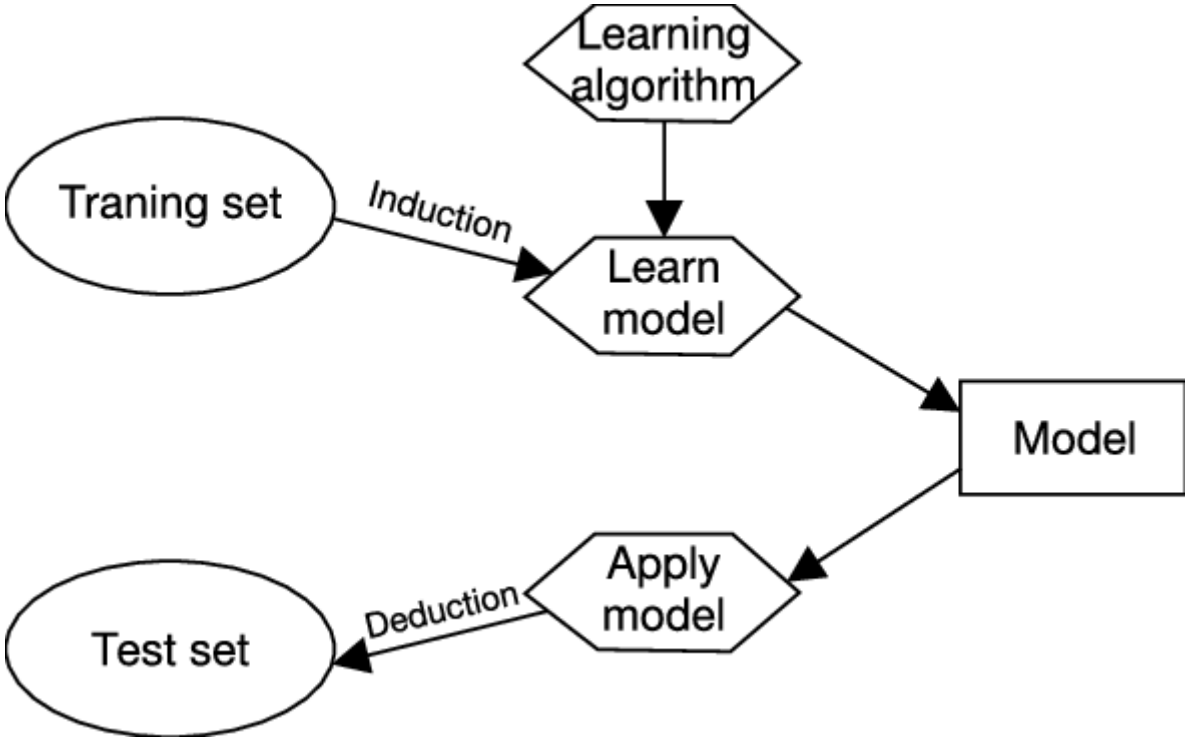
In [68]:

```
feature_counts.shape
```

Out[68]:

```
(5000, 2)
```

4 Naive-bayes model for post classification



In statistics, Naïve Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. We will use it to build a classifier to predict the tags. Its widely used in the Natural Language Processing and proved to give better results. More details can be found here and here.

4.1. Split the dataset

Lets split the dataset into training and testing datasetsin ratio 70:30%.

In [73]:

```
from sklearn.model_selection import train_test_split

train_X, test_X, train_y, test_y = train_test_split( train_ds_features,
text_data.tags,
test_size = 0.3,
random_state = 123 )
```

4.2. Build the model

Lest build a model using training dataset

In [72]:

```
from sklearn.naive_bayes import BernoulliNB
nb_clf = BernoulliNB()
nb_clf.fit( train_X.toarray(), train_y )
```

Out[72]:

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=Tru
e)
```

4.3. Make predictions on test data

In [78]:

```
test_ds_predicted = nb_clf.predict( test_X.toarray() )
```

Out[78]:

```
array(['objective-c', '.net', 'html', ..., 'sql', 'android', 'htm
l'],
      dtype='<U13')
```

4.4. Obtain Model accuracy

In [79]:

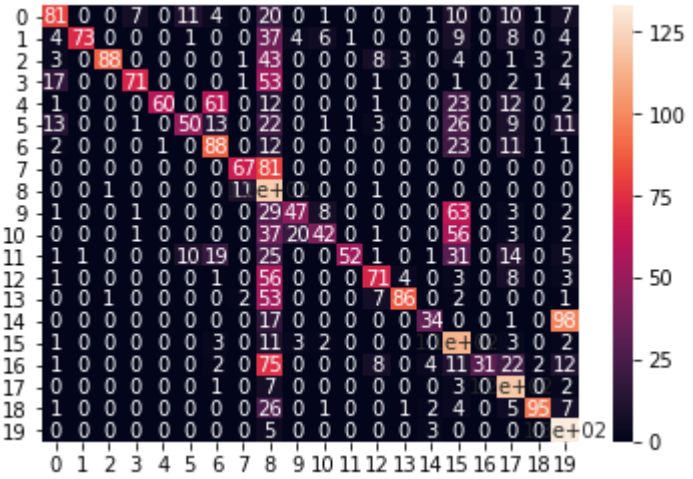
```
from sklearn import metrics
print('Accuracy: ',accuracy_score(test_y, test_ds_predicted)*100,'%')
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
Accuracy:  51.06666666666667 %
              precision    recall  f1-score   support

        .net       0.64      0.53      0.58       153
     android       0.99      0.50      0.66       147
   angularjs       0.98      0.56      0.72       156
     asp.net       0.88      0.47      0.61       151
           c       0.98      0.35      0.52       172
          c#       0.69      0.33      0.45       150
         c++       0.46      0.63      0.53       139
         css       0.82      0.45      0.58       148
        html       0.16      0.90      0.28       135
         ios       0.64      0.31      0.41       154
      iphone       0.69      0.26      0.38       162
        java       0.96      0.33      0.49       160
  javascript       0.70      0.48      0.57       147
      jquery       0.91      0.57      0.70       152
       mysql       0.76      0.23      0.35       150
 objective-c       0.30      0.82      0.44       139
         php       1.00      0.18      0.31       168
      python       0.52      0.90      0.66       134
ruby-on-rails       0.92      0.67      0.78       142
         sql       0.45      0.94      0.61       141

   micro avg       0.51      0.51      0.51      3000
   macro avg       0.72      0.52      0.53      3000
weighted avg       0.73      0.51      0.53      3000
```

Around 73% accuracy is shown by the model. Lets see the confusion matrix for the same results.

In [82]:

```
#!pin install seaborn
import seaborn as sn
from sklearn import metrics
cm = metrics.confusion_matrix( test_y, test_ds_predicted )
sn.heatmap(cm, annot=True);
```



Lets see if we can improve upon this model further.

4.4. Prepare model using TF-IDF vectors

TF-IDF Vectors

TF-IDF is an improvement upon the classical Bag of Words Technique. The disadvantage of Bag of Words technique is that it cannot stress on the context of words. TF-IDF overcomes this problem by introducing IDF or Inverse Document Frequency which compares the words used in a sentence with the words used in the entire document.

Term Frequency(TF) = ratio of number of times a word occurred in a document to the total number of words in the document.

Inverse Document Frequency(IDF) = logarithm of (total number of documents divided by number of documents containing the word).

TF-IDF is the product of these two terms

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t.$$

Therefore, if a dataset is about say, tigers. The word 'tiger' will appear across all documents (mostly) with high frequency. Thus, the weightage of this word will be reduced due to the IDF term, even if the Term Frequency is high.

The TF-IDF Vector is formed when the value of the product of TF and IDF is assigned to the word. That value becomes the vector representation.

In [85]:

```
from sklearn.model_selection import train_test_split
X = text_data.post
y = text_data.tags
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
state = 123)

#classes
my_tags = ['java','html','asp.net','c#','ruby-on-rails','jquery','mysql','php',
'ios','javascript','python','c','css','android','iphone','sql','objective-c','c+
+','angularjs','.net']
```

In [87]:

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, confusion_matrix

nb = Pipeline([('vect', TfidfVectorizer()),
               ('tfidf', TfidfTransformer()),
               ('clf', MultinomialNB()),
              ])
nb.fit(X_train, y_train)

from sklearn.metrics import classification_report
y_pred = nb.predict(X_test)

print('Accuracy: ',accuracy_score(y_pred, y_test)*100,'%')
print(classification_report(y_test, y_pred,target_names=my_tags))
```

```
Accuracy:  67.53333333333333 %
               precision    recall  f1-score   support

         java       0.56      0.55      0.56       153
         html       0.86      0.82      0.84       147
      asp.net       0.93      0.88      0.90       156
           c#       0.71      0.68      0.69       151
ruby-on-rails       0.80      0.81      0.80       172
       jquery       0.60      0.47      0.53       150
        mysql       0.64      0.68      0.66       139
          php       0.62      0.90      0.73       148
          ios       0.40      0.70      0.51       135
   javascript       0.55      0.53      0.54       154
       python       0.62      0.36      0.46       162
            c       0.79      0.64      0.71       160
          css       0.72      0.51      0.60       147
      android       0.73      0.64      0.69       152
       iphone       0.65      0.67      0.66       150
          sql       0.55      0.63      0.59       139
  objective-c       0.88      0.57      0.69       168
          c++       0.73      0.87      0.79       134
    angularjs       0.82      0.89      0.85       142
         .net       0.59      0.77      0.67       141

    micro avg       0.68      0.68      0.68      3000
    macro avg       0.69      0.68      0.67      3000
 weighted avg       0.69      0.68      0.67      3000
```

Accuracy is less than earlier built model. Lets use stemming in the Vectorizer

In [88]:

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, confusion_matrix

nb = Pipeline([('vect', TfidfVectorizer(analyzer=stem_words, max_features=5000
)),
               ('tfidf', TfidfTransformer()),
               ('clf', MultinomialNB()),
              ])
nb.fit(X_train, y_train)

from sklearn.metrics import classification_report
y_pred = nb.predict(X_test)

print('Accuracy: ',accuracy_score(y_pred, y_test)*100,'%')
print(classification_report(y_test, y_pred,target_names=my_tags))
```

```
Accuracy:  72.03333333333333 %
               precision    recall  f1-score   support

         java       0.58      0.59      0.58       153
         html       0.94      0.82      0.88       147
      asp.net       0.92      0.93      0.92       156
           c#       0.78      0.70      0.73       151
ruby-on-rails       0.82      0.80      0.81       172
       jquery       0.66      0.51      0.57       150
        mysql       0.59      0.71      0.64       139
          php       0.71      0.91      0.80       148
          ios       0.56      0.59      0.57       135
   javascript       0.55      0.48      0.51       154
       python       0.63      0.50      0.56       162
            c       0.83      0.72      0.77       160
          css       0.71      0.62      0.66       147
      android       0.73      0.85      0.79       152
       iphone       0.68      0.75      0.71       150
          sql       0.59      0.66      0.63       139
  objective-c       0.91      0.68      0.78       168
          c++       0.74      0.90      0.81       134
    angularjs       0.90      0.91      0.90       142
         .net       0.65      0.83      0.73       141

    micro avg       0.72      0.72      0.72      3000
    macro avg       0.72      0.72      0.72      3000
 weighted avg       0.73      0.72      0.72      3000
```

Lets try using another variant for NB.

In [91]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer( analyzer=stem_words, max_features = 5000)
feature_vector = tfidf_vectorizer.fit( text_data.post )
train_ds_features = tfidf_vectorizer.transform( text_data.post )
features = feature_vector.get_feature_names()
```

In [92]:

```
from sklearn.naive_bayes import GaussianNB
train_X, test_X, train_y, test_y = train_test_split( train_ds_features,
text_data.tags,
test_size = 0.3,
random_state = 123 )
nb_clf = GaussianNB()
nb_clf.fit( train_X.toarray(), train_y )
```

Out[92]:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [ ]:

```
test_ds_predicted = nb_clf.predict( test_X.toarray() )
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

In [ ]:

```

```