Loading Dataset

In [2]:

```
from sklearn.datasets import fetch_20newsgroups
```

In [3]:

```
from pprint import pprint
pprint(list(fetch_20newsgroups(subset='train').target_names))
```

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

In [4]:

```
# Selecting categories
categories = [
    'alt.atheism',
    'talk.religion.misc',
    'comp.graphics',
    'sci.space'
]
```

In [5]:

```
fetch_20newsgroups(subset='train',categories=categories).target[:10]
```

Out[5]:

```
array([1, 3, 2, 0, 2, 0, 2, 1, 2, 1], dtype=int64)
```

In [6]:

```
fetch_20newsgroups(subset='train',categories=categories).filenames.shape
```

Out[6]:

```
(2034,)
```

In [7]:

```
data_train = fetch_20newsgroups(subset='train',categories=categories,shuffle=True,random_st
train_label = data_train.target
data_test = fetch_20newsgroups(subset='test',categories=categories,shuffle=True,random_stat
test_label = data_test.target
```

In [8]:

```
import numpy as np
np.unique(test_label)
```

Out[8]:

```
array([0, 1, 2, 3], dtype=int64)
```

In [9]:

```
#import re
```

In [10]:

```
#def clean(x):

    #remove all html tags from data
    #remove all numbers from data
    #remove all special chars from data
    #remove stop words
    #stemming
    #etc..
 #   s = re.sub('<.*?>',' ',x) #remove html tags

  #  s = re.sub('[^A-Za-z]',' ',s)
    #to replace everything except A-Z or a-z with ' '(single space)

   # s = re.sub('\s+',' ',s)
    #to replace more than one space's with single space only

    #s = s.strip()
    #remove spaces from either from beginning or end of string

    #return s.lower()    #return string in lower case
```

Getting error: TypeError: expected string or bytes-like object

In [11]:

```
# Cleaning the data
import nltk
nltk.download('names')
import nltk
nltk.download('wordnet')

from collections import defaultdict
from nltk.stem import WordNetLemmatizer
from nltk.corpus import names

all_names = names.words()
WNL = WordNetLemmatizer()
def clean(data):
    cleaned = defaultdict(list)
    count = 0
    for group in data:
        for words in group.split():
            if words.isalpha() and words not in all_names:
                cleaned[count].append(WNL.lemmatize(words.lower()))
        cleaned[count] = ' '.join(cleaned[count])
        count +=1
    return(list(cleaned.values()))
```

```
[nltk_data] Downloading package names to
[nltk_data]     C:\Users\Aniket\AppData\Roaming\nltk_data...
[nltk_data]   Package names is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Aniket\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [12]:

```
x_train = clean(data_train.data)
x_train[0]
```

Out[12]:

```
'where did all the texture rule noticed that if you only save a model all yo
ur mapping plane positioned to a file that when you reload it after restarti
ng they are given a default position and but if you save to a file their are
doe anyone know why this information is not stored in the nothing is explici
tly said in the manual about saving texture rule in the like to be able to r
ead the texture rule doe anyone have the format for the is the file format a
vailable from rych rycharde hawkes virtual environment laboratory of psychol
ogy tel of edinburgh fax'
```

In [13]:

```
len(x_train)
```

Out[13]:

```
2034
```

In [14]:

```
x_test = clean(data_test.data)
```

In [15]:

```
len(x_test)
```

Out[15]:

```
1353
```

In [16]:

```
# Converting to TF-IDF Format
from sklearn.feature_extraction.text import TfidfVectorizer

tf = TfidfVectorizer(stop_words='english', max_features=1000)
X_train = tf.fit_transform(x_train)
X_test = tf.transform(x_test)
```

In [17]:

```
X_train.shape
```

Out[17]:

```
(2034, 1000)
```

In [18]:

```
X_test.shape
```

Out[18]:

```
(1353, 1000)
```

MultimonialNB

In [19]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics

clf = MultinomialNB()
params = {'alpha' : (0.5,1,1.5,2)}
```

In [20]:

```
from sklearn.model_selection import GridSearchCV
grid_search1 = GridSearchCV(clf, param_grid=params,n_jobs=-1,cv=4)
grid_search1.fit(X_train,train_label)
```

Out[20]:

```
GridSearchCV(cv=4, estimator=MultinomialNB(), n_jobs=-1,
             param_grid={'alpha': (0.5, 1, 1.5, 2)})
```

In [21]:

```
grid_search1.best_params_
```

Out[21]:

```
{'alpha': 0.5}
```

In [22]:

```
grid_search1.best_score_
```

Out[22]:

```
0.8854409990254166
```

In [23]:

```
best1 = grid_search1.best_estimator_
accuracy1 = best1.score(X_test,test_label)
accuracy1
```

Out[23]:

```
0.8248337028824834
```

SVC

In [24]:

```
from sklearn.svm import SVC
```

In [25]:

```
svc_lib = SVC(kernel='linear')
params = {'C':(0.5,0.6,1.0,1.3,1.5)}
```

In [26]:

```
grid_search2 = GridSearchCV(svc_lib, param_grid=params, n_jobs=-1, cv=4)
```

In [27]:

```
grid_search2.fit(X_train, train_label)
```

Out[27]:

```
GridSearchCV(cv=4, estimator=SVC(kernel='linear'), n_jobs=-1,
             param_grid={'C': (0.5, 0.6, 1.0, 1.3, 1.5)})
```

In [28]:

```
grid_search2.best_score_
```

Out[28]:

```
0.8849488730411645
```

In [29]:

```
best2 = grid_search2.best_estimator_
```

In [30]:

```
accuracy2 = best2.score(X_test, test_label)
```

In [31]:

```
accuracy2
```

Out[31]:

```
0.7982261640798226
```

Linear SVC

In [34]:

```
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
```

In [35]:

```
# Model tuning in Linear SVC
pipeline = Pipeline([('tf_id', TfidfVectorizer(stop_words = "english")), ('svm_im', LinearS
pipeline

parameter = {'tf_id__max_features' : (100,1000, 2000, 8000),
             'tf_id__max_df' : (0.25, 0.5),
             'tf_id__smooth_idf' : (True, False),
             'tf_id__sublinear_tf' : (True, False)
}
```

In [36]:

```
grid_search = GridSearchCV(pipeline, parameter,cv = 3)
grid_search.fit(x_train, train_label)
```

Out[36]:

```
GridSearchCV(cv=3,
             estimator=Pipeline(steps=[('tf_id',
                                        TfidfVectorizer(stop_words='englis
h')),
                                       ('svm_im', LinearSVC())]),
             param_grid={'tf_id__max_df': (0.25, 0.5),
                         'tf_id__max_features': (100, 1000, 2000, 8000),
                         'tf_id__smooth_idf': (True, False),
                         'tf_id__sublinear_tf': (True, False)})
```

In [37]:

```
grid_search.best_params_
```

Out[37]:

```
{'tf_id__max_df': 0.25,
 'tf_id__max_features': 8000,
 'tf_id__smooth_idf': False,
 'tf_id__sublinear_tf': False}
```

In [39]:

```
grid_search.best_score_
```

Out[39]:

```
0.9410029498525074
```

Making a table for all the scores obtained so far

In [51]:

```
scores = {
    'MultinomialNB' : [0.824],
    'SVC' : [0.798],
    'LinearSVC' : [0.941]
}
```

In [54]:

```
import pandas as pd
index = pd.Index(['scores'])
score = pd.DataFrame(scores)
scores = score.set_index(index)
scores
```

Out[54]:

	MultinomialNB	SVC	LinearSVC
scores	0.824	0.798	0.941

From the above result we can easily see that, we are getting an accuracy of 94.1% approx when we are using LinearSVC model, in comparison to MultinomialNB and SVC

Hence, the dataset has been classified

In [ ]:

```

```