CSCI - 5901 - The Process of Data Science - Summer 2019 Assignment 2

Nirav Solanki(B00808427)

Aditya Gadhvi(B00809664)

Dataset

For this particular assignment, the fetch20_newsgroups has been used. It contains around 18000 newsgroups post divided into 20 different topics. We are only going to use newsgroups associated with only 4 topics: alt.atheism, talk.religion.misc, comp.graphics, sci.space.

Collocation extraction

Fetching the data

We will be utilizing the entire dataset associated with the library. We will filter the data to only include the data associated with the 4 newsgroups described in above. We will also discard headers, footers and quotes.

In [1]:
```
from sklearn.datasets import fetch_20newsgroups
categories = ['alt.atheism', 'talk.religion.misc', 'comp.graphics', 'sci.space']
newsgroups = fetch_20newsgroups(categories=categories)
```

To easily manipulate the dataset, we will store the data inside a pandas dataframe, generate proper shape, and give the columns appropriate names.

In [2]:
```
import pandas as pd
df = pd.DataFrame([newsgroups.data,newsgroups.target.tolist()])
df = df.T #Transposing the dataframe to create proper shape.
df.columns = ["news","categories"]
df.head()
```

Out[2]:

	news	categories
0	From: rych@festival.ed.ac.uk (R Hawkes)\nSubje...	1
1	Subject: Re: Biblical Backing of Koresh's 3-02...	3
2	From: Mark.Perew@p201.f208.n103.z1.fidonet.org...	2
3	From: dpw@sei.cmu.edu (David Wood)\nSubject: R...	0
4	From: prb@access.digex.com (Pat)\nSubject: Con...	2

Tokenizing the data

In [3]:
```
import nltk
#nltk.download('punkt')
from nltk.tokenize import word_tokenize

df["tokenized_news"] = df["news"].apply(word_tokenize).tolist()
df["tokenized_news"].head()
```

Out[3]:
```
0    [From, :, rych, @, festival.ed.ac.uk, (, R, Ha...
1    [Subject, :, Re, :, Biblical, Backing, of, Kor...
2    [From, :, Mark.Perew, @, p201.f208.n103.z1.fid...
3    [From, :, dpw, @, sei.cmu.edu, (, David, Wood,...
4    [From, :, prb, @, access.digex.com, (, Pat, ),...
Name: tokenized_news, dtype: object
```

Performing POS(Part-Of-Speech) Tagging

In [4]:
```
from nltk import pos_tag_sents
#nltk.download('averaged_perceptron_tagger')
df['POS'] = pos_tag_sents(df["tokenized_news"])
df['POS'].head()
```

Out[4]:
```
0    [(From, IN), (:, :), (rych, NN), (@, NN), (fes...
1    [(Subject, JJ), (:, :), (Re, NN), (:, :), (Bib...
2    [(From, IN), (:, :), (Mark.Perew, NNP), (@, NN...
3    [(From, IN), (:, :), (dpw, NN), (@, NN), (sei....
4    [(From, IN), (:, :), (prb, NN), (@, NN), (acce...
Name: POS, dtype: object
```

Calculating Frequency Distribution on the tokens

The FreqDist() was not accepting the dataframe as its parameter, so we converted the dataframe into a list and then calculated the frequency distribution

In [5]:
```
tokenized_news =  df['tokenized_news']
words = []
for wordList in tokenized_news:
    words += wordList
```

In [6]:
```
from nltk.probability import FreqDist
fdist = FreqDist(words)
fdist.most_common(3)
```

Out[6]:
```
[(',', 30129), ('.', 27573), ('the', 23477)]
```

In [7]:
```
fdist.most_common(3)
```

Out[7]:
```
[(',', 30129), ('.', 27573), ('the', 23477)]
```

Applying techniques such as Frequency with filter, PMI, T-test with filter, Chi-Sq test to extract bigram collocations from the corpus

In order to extract meaningful bigrams, we have cleaned the data first and then applied all of the techniques. Without cleaning, irrelevant and unmeaningful bigrams were being extracted

Applying cleaning techniques:

In [8]:
```
import re
cleanlist=[re.sub('[^a-zA-Z]+', '', _) for _ in words]
```

(1) Frequency with filter

In [9]:
```
# The following code prints the bigrams without applying any kind of filter.
import nltk
bigrams = nltk.collocations.BigramAssocMeasures()
bigramFinder = nltk.collocations.BigramCollocationFinder.from_words(cleanlist)
#bigrams
bigram_freq = bigramFinder.ngram_fd.items()
bigramFreqTable = pd.DataFrame(list(bigram_freq), columns=['bigram','freq']).sort
bigramFreqTable.head(20)
```

Out[9]:

	bigram	freq
22	(,)	61931
25	(, I)	4863
441	(, and)	3629
482	(of, the)	3085
170	(, The)	2842
0	(From,)	2137
9	(Subject,)	2087
21	(Lines,)	2049
169	(Organization,)	1962
8	(, Subject)	1896
889	(, the)	1819
86	(in, the)	1740
831	(, but)	1696
148	(Re,)	1608
188	(writes,)	1594
176	(, In)	1570
147	(, Re)	1517
187	(, writes)	1392
160	(, From)	1225
178	(article,)	1211

In [10]:
```
# The following code performs cleaning such as removing stop words etc. From this
nltk.download('averaged_perceptron_tagger')
from nltk.corpus import stopwords
#get english stopwords
en_stopwords = set(stopwords.words('english'))

#function to filter for ADJ/NN bigrams
def rightTypes(ngram):
    if '-pron-' in ngram or '' in ngram or ' 'in ngram or 't' in ngram:
        return False
    for word in ngram:
        if word in en_stopwords:
            return False
    acceptable_types = ('JJ', 'JJR', 'JJS', 'NN', 'NNS', 'NNP', 'NNPS')
    second_type = ('NN', 'NNS', 'NNP', 'NNPS')
    tags = nltk.pos_tag(ngram)
    if tags[0][1] in acceptable_types and tags[1][1] in second_type:
        return True
    else:
        return False
```

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

In [11]:
```
#filter bigrams
# We get meaningfull bigrams after apllying filter.
filtered_bi = bigramFreqTable[bigramFreqTable.bigram.map(lambda x: rightTypes(x)
filtered_bi.head(20)
```

Out[11]:

	bigram	freq
1562	(world, NNTPPostingHost)	128
1324	(Henry, Spencer)	96
7127	(Jon, Livesey)	93
7147	(Keith, Allan)	89
7148	(Allan, Schneider)	89
733	(Computer, Science)	87
13988	(University, Lines)	85
5878	(Kent, Sandvik)	78
7133	(Political, Atheists)	76
15083	(California, Institute)	70
1333	(Toronto, Zoology)	67
13258	(State, University)	65
1569	(Ron, Baalke)	64
668	(USA, Lines)	61
15084	(Pasadena, Lines)	60
6418	(Inc, Lines)	59
9917	(Jesus, Christ)	59
13411	(Frank, ODwyer)	58
29765	(Space, Shuttle)	58
5656	(VAXVMS, VNEWS)	57

In [12]:
```
frequency_bi = filtered_bi[:20].bigram.values
```

(2) PMI (Pointwise Mutual Information)

In [13]:
```
bigramFinder.apply_freq_filter(20)
```

In [14]:
```
bigramPMITable = pd.DataFrame(list(bigramFinder.score_ngrams(bigrams.pmi)), colur
```

In [15]:
```
bigramPMITable.head(20)
```

Out[15]:

	bigram	PMI
0	(comme, aucun)	15.265952
1	(Steinn, Sigurdsson)	15.195563
2	(ISLAMIC, LAW)	15.128448
3	(fait, comme)	15.064318
4	(sank, Manhattan)	14.997204
5	(Carnegie, Mellon)	14.832993
6	(Beam, Jockey)	14.729899
7	(Bake, Timmons)	14.707462
8	(Frequently, Asked)	14.633684
9	(Chapel, Hill)	14.619589
10	(Brigham, Young)	14.479356
11	(Los, Angeles)	14.417955
12	(MY, REPLY)	14.394467
13	(Cookamunga, Tourist)	14.378427
14	(Resource, Listing)	14.370649
15	(fred, j)	14.278891
16	(Asked, Questions)	14.265952
17	(Alaska, Fairbanks)	14.230328
18	(Virtual, Reality)	14.148945
19	(McDonnell, Douglas)	14.033291

In [16]:
```
pmi_bi=bigramPMITable[:20].bigram.values
```

(3) T-test with filter

In [17]:
```
bigramTtable = pd.DataFrame(list(bigramFinder.score_ngrams(bigrams.student_t)), 
bigramTtable.head(20)
```

Out[17]:

	bigram	t
0	(,)	54.074340
1	(of, the)	47.884251
2	(, I)	40.172893
3	(, The)	39.116069
4	(in, the)	36.425290
5	(From,)	34.338498
6	(Subject,)	34.336979
7	(Lines,)	34.022641
8	(In, article)	33.641848
9	(Organization,)	33.258201
10	(do, nt)	31.807164
11	(, Subject)	31.638553
12	(Re,)	30.154340
13	(writes,)	29.879905
14	(to, be)	29.542314
15	(, but)	28.812437
16	(, Re)	28.709193
17	(, In)	27.974413
18	(on, the)	27.859579
19	(, writes)	26.560332

In [18]:
```
filteredT_bi = bigramTtable[bigramTtable.bigram.map(lambda x: rightTypes(x))]
filteredT_bi.head(20)
```

Out[18]:

	bigram	t
171	(world, NNTPPostingHost)	11.281002
241	(Henry, Spencer)	9.796435
246	(Jon, Livesey)	9.642051
266	(Allan, Schneider)	9.432706
267	(Keith, Allan)	9.431556
276	(Computer, Science)	9.319078
304	(University, Lines)	8.954049
318	(Kent, Sandvik)	8.830147
321	(Political, Atheists)	8.716659
356	(California, Institute)	8.362592
379	(Toronto, Zoology)	8.184418
400	(State, University)	8.050273
411	(Ron, Baalke)	7.999178
450	(USA, Lines)	7.758238
462	(Pasadena, Lines)	7.712686
470	(Jesus, Christ)	7.666401
476	(Frank, ODwyer)	7.614650
478	(Inc, Lines)	7.608259
481	(Space, Shuttle)	7.602255
488	(VAXVMS, VNEWS)	7.549269

In [19]:
```
ttest_bi=filteredT_bi[:20].bigram.values
```

(4) Chi-Sq test

In [20]:
```
bigramChiTable = pd.DataFrame(list(bigramFinder.score_ngrams(bigrams.chi_sq)), co
bigramChiTable.head(20)
```

Out[20]:

	bigram	chi-sq
0	(Carnegie, Mellon)	788024.000000
1	(Cookamunga, Tourist)	788024.000000
2	(Steinn, Sigurdsson)	788024.000000
3	(comme, aucun)	788024.000000
4	(VAXVMS, VNEWS)	761309.389691
5	(ISLAMIC, LAW)	752203.772702
6	(Frequently, Asked)	737181.870899
7	(Los, Angeles)	722352.583218
8	(Beam, Jockey)	706501.586118
9	(sank, Manhattan)	686792.839845
10	(fait, comme)	685235.652108
11	(Allan, Schneider)	658695.225256
12	(Ron, Baalke)	622622.813724
13	(Tourist, Bureau)	620351.446439
14	(Henry, Spencer)	617216.551988
15	(Jet, Propulsion)	608092.486339
16	(Bake, Timmons)	588579.919579
17	(Toronto, Zoology)	586622.965211
18	(Political, Atheists)	581954.474364
19	(Chapel, Hill)	578972.419253

In [21]:
```
chi_bi=bigramChiTable[:20].bigram.values
```

Comparing the top 20 bigram results of all the applied techniques:

In [22]:
```
bigrams_compare_results = pd.DataFrame([frequency_bi, pmi_bi, ttest_bi, chi_bi])
```

In [23]:
```
bigrams_compare_results.columns = ['Frequency With Filter', 'PMI', 'T-test With F
```

```
In [24]: bigrams_compare_results
```

Out[24]:

	Frequency With Filter	PMI	T-test With Filter	Chi-Sq Test
0	(world, NNTPPostingHost)	(comme, aucun)	(world, NNTPPostingHost)	(Carnegie, Mellon)
1	(Henry, Spencer)	(Steinn, Sigurdsson)	(Henry, Spencer)	(Cookamunga, Tourist)
2	(Jon, Livesey)	(ISLAMIC, LAW)	(Jon, Livesey)	(Steinn, Sigurdsson)
3	(Keith, Allan)	(fait, comme)	(Allan, Schneider)	(comme, aucun)
4	(Allan, Schneider)	(sank, Manhattan)	(Keith, Allan)	(VAXVMS, VNEWS)
5	(Computer, Science)	(Carnegie, Mellon)	(Computer, Science)	(ISLAMIC, LAW)
6	(University, Lines)	(Beam, Jockey)	(University, Lines)	(Frequently, Asked)
7	(Kent, Sandvik)	(Bake, Timmons)	(Kent, Sandvik)	(Los, Angeles)
8	(Political, Atheists)	(Frequently, Asked)	(Political, Atheists)	(Beam, Jockey)
9	(California, Institute)	(Chapel, Hill)	(California, Institute)	(sank, Manhattan)
10	(Toronto, Zoology)	(Brigham, Young)	(Toronto, Zoology)	(fait, comme)
11	(State, University)	(Los, Angeles)	(State, University)	(Allan, Schneider)
12	(Ron, Baalke)	(MY, REPLY)	(Ron, Baalke)	(Ron, Baalke)
13	(USA, Lines)	(Cookamunga, Tourist)	(USA, Lines)	(Tourist, Bureau)
14	(Pasadena, Lines)	(Resource, Listing)	(Pasadena, Lines)	(Henry, Spencer)
15	(Inc, Lines)	(fred, j)	(Jesus, Christ)	(Jet, Propulsion)
16	(Jesus, Christ)	(Asked, Questions)	(Frank, ODwyer)	(Bake, Timmons)
17	(Frank, ODwyer)	(Alaska, Fairbanks)	(Inc, Lines)	(Toronto, Zoology)
18	(Space, Shuttle)	(Virtual, Reality)	(Space, Shuttle)	(Political, Atheists)
19	(VAXVMS, VNEWS)	(McDonnell, Douglas)	(VAXVMS, VNEWS)	(Chapel, Hill)

Overlap among the techniques:

From the above results it can be clearly seen that there is an overlap among the techniques. There are many bigrams that are common among the techniques. By carefully looking at the bigrams of Frequency with filter and T-test with filter, it is observed that almost all of the bigrams are common among these two techniques. As there is a high overlap among the techniques, it makes sense to consider the union of the results. There are many bigrams which are common among the techniques, hence we should union the common results and not keep them seperate.

SVM and NB for Text Classification

Data Cleaning :

Remove stop words

Remove numbers and other non-letter characters

Stem the words

In [25]:
```
df["cleaned_news"] = df["news"].str.lower()
df["cleaned_news"] = df["cleaned_news"].str.replace(r"[^a-z]+"," ")
df["cleaned_news"].head()
```

Out[25]:
```
0    from rych festival ed ac uk r hawkes subject d...
1    subject re biblical backing of koresh s tape c...
2    from mark perew p f n z fidonet org subject re...
3    from dpw sei cmu edu david wood subject reques...
4    from prb access digex com pat subject conferen...
Name: cleaned_news, dtype: object
```

In [26]:
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

stop_words = set(stopwords.words('english'))

df["cleaned_news"] = df["cleaned_news"].apply(lambda x: ' '.join([word for word :
df["cleaned_news"].head()
```

Out[26]:
```
0    rych festival ed ac uk r hawkes subject ds tex...
1    subject biblical backing koresh tape cites enc...
2    mark perew p f n z fidonet org subject comet t...
3    dpw sei cmu edu david wood subject request sup...
4    prb access digex com pat subject conference ma...
Name: cleaned_news, dtype: object
```

In [27]:
```
df["tokenized_news"] = df["cleaned_news"].apply(word_tokenize).tolist()
df["tokenized_news"].head()
```

Out[27]:
```
0    [rych, festival, ed, ac, uk, r, hawkes, subjec...
1    [subject, biblical, backing, koresh, tape, cit...
2    [mark, perew, p, f, n, z, fidonet, org, subjec...
3    [dpw, sei, cmu, edu, david, wood, subject, req...
4    [prb, access, digex, com, pat, subject, confer...
Name: tokenized_news, dtype: object
```

In [28]:
```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
df["tokenized_news"] = df["tokenized_news"].apply(lambda x: [ps.stem(y) for y in
df['tokenized_news'].head()
```

Out[28]:
```
0    [rych, festiv, ed, ac, uk, r, hawk, subject, d...
1    [subject, biblic, back, koresh, tape, cite, en...
2    [mark, perew, p, f, n, z, fidonet, org, subjec...
3    [dpw, sei, cmu, edu, david, wood, subject, req...
4    [prb, access, digex, com, pat, subject, confer...
Name: tokenized_news, dtype: object
```

In [29]:
```
df["cleaned_news"] = df['tokenized_news'].apply(' '.join)
df["cleaned_news"].head()
```

Out[29]:
```
0    rych festiv ed ac uk r hawk subject ds textur ...
1    subject biblic back koresh tape cite enclos km...
2    mark perew p f n z fidonet org subject comet t...
3    dpw sei cmu edu david wood subject request sup...
4    prb access digex com pat subject confer man lu...
Name: cleaned_news, dtype: object
```

Creating TFIDF vectors

Machine learning algorithms only accepts float or integer values to make model. So we find the TFIDF values associated with words in the dataframe.

In [57]:
```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

corpus = (df['cleaned_news'])
vectorizer = CountVectorizer()
X_counts = vectorizer.fit_transform(corpus)
tfidf_transformer = TfidfTransformer()
X_tfidf = tfidf_transformer.fit_transform(X_counts).toarray()
#Vocabulary size of entire data
X_tfidf.shape
```

Out[57]:
```
(2034, 21051)
```

Splitting the data into train and test sets.

The TFIDF values are taken as features and the categories are taken as targets. The Size of train dataset is kept as 70% and test dataset is taken as 30%.

In [31]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, df["categories"].as
```

We will utilize two models : nusvc and NB for Text Classification

In [32]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
import numpy as np
M_Classifier = MultinomialNB().fit(X_train, y_train)

pred_1 = M_Classifier.predict(X_test)

print("Accuracy:",np.mean(pred_1 == y_test))
print("Confusion Matrix:\n",confusion_matrix(y_test, pred_1))
```

```
Accuracy: 0.8887070376432079
Confusion Matrix:
 [[136   0   2   1]
 [  0 161   3   0]
 [  0   6 182   0]
 [ 43   3  10  64]]
```

In [33]:
```
# Here kernel is taken by default as rbf
from sklearn.svm import NuSVC
S_Classifier = NuSVC().fit(X_train, y_train)

pred_2 = S_Classifier.predict(X_test)

print("Accuracy:",np.mean(pred_2 == y_test))
print("Confusion Matrix:\n",confusion_matrix(y_test, pred_2))
```

```
C:\Users\adity\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarn
ing: The default value of gamma will change from 'auto' to 'scale' in version
0.22 to account better for unscaled features. Set gamma explicitly to 'auto' o
r 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

Accuracy: 0.8265139116202946
Confusion Matrix:
 [[101   4   0  34]
 [  0 162   2   0]
 [  4  44 140   0]
 [  2  15   1 102]]
```

In [34]:
```
# Here kernel is taken as linear
from sklearn.svm import NuSVC
S_Classifier = NuSVC(kernel="linear").fit(X_train, y_train)

pred_2 = S_Classifier.predict(X_test)

print("Accuracy:",np.mean(pred_2 == y_test))
print("Confusion Matrix:\n",confusion_matrix(y_test, pred_2))
```

```
Accuracy: 0.9410801963993454
Confusion Matrix:
 [[134   2   1   2]
 [  0 163   1   0]
 [  0   9 179   0]
 [  5  14   2  99]]
```

In [35]:
```
# Here kernel is taken as poly
from sklearn.svm import NuSVC
S_Classifier = NuSVC(kernel="poly").fit(X_train, y_train)

pred_2 = S_Classifier.predict(X_test)

print("Accuracy:",np.mean(pred_2 == y_test))
print("Confusion Matrix:\n",confusion_matrix(y_test, pred_2))
```

```
C:\Users\adity\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarn
ing: The default value of gamma will change from 'auto' to 'scale' in version
0.22 to account better for unscaled features. Set gamma explicitly to 'auto' o
r 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

Accuracy: 0.6579378068739771
Confusion Matrix:
 [[ 76  51   0  12]
 [  0 161   1   2]
 [  0 118  70   0]
 [  5  20   0  95]]
```

In [36]:
```
# Here kernel is taken as sigmoid
from sklearn.svm import NuSVC
S_Classifier = NuSVC(kernel="sigmoid").fit(X_train, y_train)

pred_2 = S_Classifier.predict(X_test)

print("Accuracy:",np.mean(pred_2 == y_test))
print("Confusion Matrix:\n",confusion_matrix(y_test, pred_2))
```

```
C:\Users\adity\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarn
ing: The default value of gamma will change from 'auto' to 'scale' in version
0.22 to account better for unscaled features. Set gamma explicitly to 'auto' o
r 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

Accuracy: 0.8297872340425532
Confusion Matrix:
 [[101   4   0  34]
 [  0 162   2   0]
 [  2  44 142   0]
 [  2  15   1 102]]
```

Comparing the accuracy of both algorithms: MultinomialNB and NuSVC:

MultinomialNB algorithm gives an accuracy of 88%, while NuSVC algorithm(default kernel rbf) gives an accuracy of 82%.

MultinomialNB algorithm is giving higher accuracy than NuSVC algorithm. The reason behind this is that MultinomialNB algorithm generally works better with text data in which same words occur more frequently. In addition to this, it also works better with discrete count of text data. It is a type of algorithm that is considered as best one to use when the text data has higher frequency count of words. Due to all of these reasons, MultinomialNB is giving higher accuracy than NuSVC.

Changing the kernel of NuSVC:

The accuracy of NuSVC on default kernel rbf is 82.65%

The accuracy of NuSVC on kernel linear is 94%

The accuracy of NuSVC on kernel poly is 65%

The accuracy of NuSVC on kernel sigmoid is 82.78%

By comparing the NuSVC accuracy with different kernels, it can be clearly observed that changing the kernel does affect the accuracy. From the results, it can be seen that linear kernel gives the highest accuracy while poly kernel gives the lowest accuracy. The rbf and sigmoid kernel pretty much gives the same accuracy. So from this, we can say that kernel plays an important role in SVM and it can clearly increase as well as decrease the accuracy.

Creating the model using noun tags.

First we perform pos-tagging and fetch the nouns present in the dataset.

In [37]:
```
import nltk
#nltk.download('punkt')
from nltk.tokenize import word_tokenize

df["tokenized_news"] = df["news"].apply(word_tokenize).tolist()
df["tokenized_news"].head()
```

Out[37]:
```
0    [From, :, rych, @, festival.ed.ac.uk, (, R, Ha...
1    [Subject, :, Re, :, Biblical, Backing, of, Kor...
2    [From, :, Mark.Perew, @, p201.f208.n103.z1.fid...
3    [From, :, dpw, @, sei.cmu.edu, (, David, Wood,...
4    [From, :, prb, @, access.digex.com, (, Pat, ),...
Name: tokenized_news, dtype: object
```

In [38]:
```
df['POS'] = pos_tag_sents(df["tokenized_news"])
df['POS'].head()
```

Out[38]:
```
0    [(From, IN), (:, :), (rych, NN), (@, NN), (fes...
1    [(Subject, JJ), (:, :), (Re, NN), (:, :), (Bib...
2    [(From, IN), (:, :), (Mark.Perew, NNP), (@, NN...
3    [(From, IN), (:, :), (dpw, NN), (@, NN), (sei....
4    [(From, IN), (:, :), (prb, NN), (@, NN), (acce...
Name: POS, dtype: object
```

In [39]:
```
noun_tags = ["NN","NNS","NP","NPS"]
noun_data = []
for i in range(0,df["POS"].size-1):
    noun = []
    for tags in df["POS"][i]:
        if tags[1] in noun_tags:
            noun.append(tags)
    noun_data.append(noun)
# print(noun_data)
```

In [40]:
```
df["noun_tags"] = pd.Series(noun_data)
df["noun_tags"]
```

Out[40]:
```
0       [(rych, NN), (@, NN), (festival.ed.ac.uk, NN),...
1       [(Re, NN), (Tape, NN), (kmcvay, NN), (@, NN), ...
2       [(p201.f208.n103.z1.fidonet.org, NN), (Subject...
3       [(dpw, NN), (@, NN), (sei.cmu.edu, NN), (Subje...
4       [(prb, NN), (@, NN), (access.digex.com, NN), (...
5       [(andrew.cmu.edu, NN), (>, NN), (Re, NN), (acc...
6       [(ofa123.fidonet.org, NN), (Subject, NN), (Re,...
7       [(mjw19, NN), (@, NN), (cl.cam.ac.uk, NN), (Su...
8       [(henry, NN), (@, NN), (zoo.toronto.edu, NN), ...
9       [(hendrix, NN), (@, NN), (oasys.dt.navy.mil, N...
10      [(prb, NN), (@, NN), (access.digex.com, NN), (...
11      [(zemcik, NN), (@, NN), (ls, NN), (Subject, NN...
12      [(ddeciacco, NN), (@, NN), (cix.compulink.co.u...
13      [(pgf, NN), (@, NN), (srl03.cacs.usl.edu, NN),...
14      [(kph2q, NN), (@, NN), (onyx.cs.Virginia.EDU, ...
15      [(pyron, NN), (@, NN), (skndiv.dseg.ti.com, NN...
16      [(batman.bmd.trw.com, NN), (Subject, NN), (Re,...
17      [(dpage, NN), (@, NN), (ra.csc.ti.com, NN), (S...
18      [(henry, NN), (@, NN), (zoo.toronto.edu, NN), ...
19      [(edm, NN), (@, NN), (twisto.compaq.com, NN), ...
20      [(@, NN), (iti.org, NN), (Subject, NN), (Re, N...
21      [(neideck, NN), (@, NN), (nestvx.enet.dec.com,...
22      [(jmeritt, NN), (mental.mitre.org, NN), (Subje...
23      [(kilroy, NN), (@, NN), (gboro.rowan.edu, NN),...
24      [(rws, NN), (@, NN), (cs.arizona.edu, NN), (Su...
25      [(tonyo, NN), (@, NN), (pendragon.CNA.TEK.COM,...
26      [(baalke, NN), (@, NN), (kelvin.jpl.nasa.gov, ...
27      [(tp892275, NN), (@, NN), (vine.canberra.edu.a...
28      [(brody, NN), (eos.arc.nasa.gov, NN), (Subject...
29      [(ch981, NN), (@, NN), (cleveland.Freenet.Edu,...
                              ...                        
2004    [(pl1u+, NN), (andrew.cmu.edu, NN), (>, NN), (...
2005    [(bosch, NN), (rz.uni-karlsruhe.de, NN), (Subj...
2006    [(harder.ccr-p.ida.org, NN), (Subject, NN), (R...
2007    [(sandvik, NN), (@, NN), (newton.apple.com, NN...
2008    [(sgoldste, NN), (@, NN), (aludra.usc.edu, NN)...
2009    [(ednclark, NN), (@, NN), (kraken.itc.gu.edu.a...
2010    [(jkatz, NN), (@, NN), (access.digex.com, NN),...
2011    [(cup.portal.com, NN), (Subject, NN), (Organiz...
2012    [(sandvik, NN), (@, NN), (newton.apple.com, NN...
2013    [(Re, NN), (shafer, NN), (rigel.dfrf.nasa.gov,...
2014    [(cocoa, NN), (netcom.com, NN), (Subject, NN),...
2015    [(Re, NN), (authority, NN), (women, NNS), (liv...
2016    [(andrew.cmu.edu, NN), (>, NN), (Re, NN), (fau...
2017    [(@, NNS), (bohr.physics.purdue.edu, NN), (Sub...
2018    [(craig, NN), (toontown.ColumbiaSC.NCR.COM, NN...
2019    [(henry, NN), (@, NN), (zoo.toronto.edu, NN), ...
2020    [(@, NN), (sunvax.sun.ac.za, NN), (query, NN),...
2021    [(darice, NN), (@, NN), (yoyo.cc.monash.edu.au...
2022    [(mathew, NN), (mathew, NN), (mantis.co.uk, NN...
2023    [(shag, NN), (@, NN), (aero.org, NN), (Subject...
2024    [(rouben, NN), (@, NN), (math9.math.umbc.edu, ...
2025    [(bil, NN), (@, NN), (okcforum.osrhe.edu, NN),...
2026    [(edm, NN), (@, NN), (twisto.compaq.com, NN), ...
2027    [(rogers, NNS), (>, NN), (Subject, NN), (Re, N...
2028    [(ykim, NN), (@, NN), (cs.columbia.edu, NN), (...
2029    [(arthurc, NN), (@, NN), (sfsuvax1.sfsu.edu, N...
2030    [(schaefer, NN), (sal-sun121.usc.edu, NN), (Su...
2031    [(jamie, NN), (@, NN), (genesis.MCS.COM, NN), ...
2032    [(@, NN), (pooh.bears, NNS), (Subject, NN), (I...
2033                                                  NaN
Name: noun_tags, Length: 2034, dtype: object
```

In [41]:

```python
noun_data = []
for i in range(0,df["noun_tags"].size-1):
    nouns = []
    for tags in df["noun_tags"][i]:
        nouns.append(tags[0])
    noun_data.append(nouns)
print(noun_data)
```

```
'science', 'bombs', 'percentage', 'casualties', 'mistakes', 'targets', 'i
nnocents', 'war', 'way', 'rulers', 'interests', 'regard', 'lives', 'peopl
e', 'way', 'war', 'people', 'reason', 'rulers', 'selfish', 'interests',
'allies', 'fuss', 'couple', 'weeks', '>', 'claim', 'effort', 'policy', 'w
ar', 'portion', 'oil', 'supply', 'war', 'effort', 'part', 'decision', 're
sponse', '>', 'sarcasm', 'idiot', 'feel', 'intent', 'intent', 'job', 'cou
ntries', 'mass', 'slaughter', 'peoples', 'dominion', 'thing', 'people',
'mind', 'conquest', 'war', 'appeasement', 'war', 'eh', '>', 'appeasemen
t', 'situation', 'lessons', 'world', 'alliance', 'war', '>', '>', 'thin
k', 'civilians', 'Germans', 'quest', 'consequences', 'Tyrants', 'kind',
'diplomacy', 'point', 'gun', 'regret', '>', 'roll', '>', 'power', 'peopl
e', 'sacrifice', 'mention', '>', 'comments', '>', 'jury', 'trial', 'verdi
ct', 'things', 'lot', 'people', 'jury', '*policemen*', 'trial', 'law', 'j
ury', 'peers', 'officers', 'jurors', 'point', 'motivations', '>', 'mout
h', 'blood', 'policemen', 'video', 'tape', '>', 'jury', 'evidence', 'evid
ence', '>', '>', 'bunch', 'policemen', 'someone', 'defenceless', 'groun
d', 'evidence', 'Argument', 'acceptance', 'author', 'Argument', 'revisio
n', 'evidence', 'everyone', 'everyone', '>', 'beating', 'prosecution', 'd
efense', 'way', 'silence', 'principals', 'country', 'rights', 'criminal
s', 'cops', 'doubt', 'guilt', 'jury', 'position', '>', '>', 'evidence',
```

In [42]:
```
df["nouns"] = pd.Series(noun_data)
df["nouns"]
```

Out[42]:
```
0       [rych, @, festival.ed.ac.uk, Subject, texture,...
1       [Re, Tape, kmcvay, @, oneb.almanac.bc.ca, Orga...
2       [p201.f208.n103.z1.fidonet.org, Subject, Re, X...
3       [dpw, @, sei.cmu.edu, Subject, Request, reques...
4       [prb, @, access.digex.com, Subject, Conference...
5       [andrew.cmu.edu, >, Re, account, NNTP-Posting-...
6       [ofa123.fidonet.org, Subject, Re, mission, nam...
7       [mjw19, @, cl.cam.ac.uk, Subject, Re, Keywords...
8       [henry, @, zoo.toronto.edu, Subject, Re, moon,...
9       [hendrix, @, oasys.dt.navy.mil, Subject, Proce...
10      [prb, @, access.digex.com, Subject, Re, Organi...
11      [zemcik, @, ls, Subject, pixel, clock, pixel, ...
12      [ddeciacco, @, cix.compulink.co.uk, Subject, R...
13      [pgf, @, srl03.cacs.usl.edu, Subject, Re, Orga...
14      [kph2q, @, onyx.cs.Virginia.EDU, Subject, vend...
15      [pyron, @, skndiv.dseg.ti.com, Subject, Re, wo...
16      [batman.bmd.trw.com, Subject, Re, article, man...
17      [dpage, @, ra.csc.ti.com, Subject, Re, Quaint,...
18      [henry, @, zoo.toronto.edu, Subject, Re, Orion...
19      [edm, @, twisto.compaq.com, Subject, Re, In-Re...
20      [@, iti.org, Subject, Re, Conference, Lines, a...
21      [neideck, @, nestvx.enet.dec.com, Subject, Re,...
22      [jmeritt, mental.mitre.org, Subject, promise, ...
23      [kilroy, @, gboro.rowan.edu, Subject, Re, Food...
24      [rws, @, cs.arizona.edu, Subject, outlining, s...
25      [tonyo, @, pendragon.CNA.TEK.COM, Subject, Nee...
26      [baalke, @, kelvin.jpl.nasa.gov, Subject, Re, ...
27      [tp892275, @, vine.canberra.edu.au, Subject, O...
28      [brody, eos.arc.nasa.gov, Subject, Re, Confere...
29      [ch981, @, cleveland.Freenet.Edu, Subject, Re,...
                              ...
2004    [pl1u+, andrew.cmu.edu, >, Subject, fault, Org...
2005    [bosch, rz.uni-karlsruhe.de, Subject, Re, Dist...
2006    [harder.ccr-p.ida.org, Subject, Re, positions,...
2007    [sandvik, @, newton.apple.com, Subject, Re, au...
2008    [sgoldste, @, aludra.usc.edu, Subject, Re, rea...
2009    [ednclark, @, kraken.itc.gu.edu.au, Subject, R...
2010    [jkatz, @, access.digex.com, Subject, Distribu...
2011    [cup.portal.com, Subject, Organization, Distri...
2012    [sandvik, @, newton.apple.com, Subject, Re, Or...
2013    [Re, shafer, rigel.dfrf.nasa.gov, t, @, aurora...
2014    [cocoa, netcom.com, Subject, Re, history, ques...
2015    [Re, authority, women, livesey, solntze.wpd.sg...
2016    [andrew.cmu.edu, >, Re, fault, Organization, a...
2017    [@, bohr.physics.purdue.edu, Subject, Re, time...
2018    [craig, toontown.ColumbiaSC.NCR.COM, Subject, ...
2019    [henry, @, zoo.toronto.edu, Subject, Re, moon,...
2020    [@, sunvax.sun.ac.za, query, Organization, che...
2021    [darice, @, yoyo.cc.monash.edu.au, Subject, Re...
2022    [mathew, mathew, mantis.co.uk, >, Re, Organiza...
2023    [shag, @, aero.org, Subject, Re, space, food, ...
2024    [rouben, @, math9.math.umbc.edu, Subject, Re, ...
```

```
2025    [bil, @, okcforum.osrhe.edu, Subject, Re, auth...
2026    [edm, @, twisto.compaq.com, Subject, Re, Victi...
2027    [rogers, >, Subject, Re, Life, X-Xxdate, X-Use...
2028    [ykim, @, cs.columbia.edu, Subject, wireframe,...
2029    [arthurc, @, sfsuvax1.sfsu.edu, Subject, space...
2030    [schaefer, sal-sun121.usc.edu, Subject, Re, mo...
2031    [jamie, @, genesis.MCS.COM, Subject, FTP, Dist...
2032    [@, pooh.bears, Subject, Islam, Reply-To, @, p...
2033                                                 NaN
Name: nouns, Length: 2034, dtype: object
```

In [43]:

```
df["nouns"] = df['tokenized_news'].apply(' '.join)
df["nouns"].head()
```

Out[43]:

```
0    From : rych @ festival.ed.ac.uk ( R Hawkes ) S...
1    Subject : Re : Biblical Backing of Koresh 's 3...
2    From : Mark.Perew @ p201.f208.n103.z1.fidonet....
3    From : dpw @ sei.cmu.edu ( David Wood ) Subjec...
4    From : prb @ access.digex.com ( Pat ) Subject ...
Name: nouns, dtype: object
```

We now again create TFIDF vectors, divide data into test and train, and build models.

In [58]:

```
corpus = (df['nouns'])
vectorizer = CountVectorizer()
X_counts = vectorizer.fit_transform(corpus)
tfidf_transformer = TfidfTransformer()
X_tfidf = tfidf_transformer.fit_transform(X_counts).toarray()
#Vocabulary size of just noun data
X_tfidf.shape
```

Out[58]:

```
(2034, 34112)
```

In [45]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, df["categories"].as
```

In [46]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
import numpy as np
M_Classifier = MultinomialNB().fit(X_train, y_train)

pred_1 = M_Classifier.predict(X_test)

print("Accuracy:",np.mean(pred_1 == y_test))
print("Confusion Matrix:\n",confusion_matrix(y_test, pred_1))
```

```
Accuracy: 0.8494271685761048
Confusion Matrix:
 [[137   0   1   1]
 [  3 156   5   0]
 [  0   6 182   0]
 [ 66   1   9  44]]
```

In [48]:
```
from sklearn.svm import NuSVC
S_Classifier = NuSVC().fit(X_train, y_train)

pred_2 = S_Classifier.predict(X_test)

print("Accuracy:",np.mean(pred_2 == y_test))
print("Confusion Matrix:\n",confusion_matrix(y_test, pred_2))
```

```
C:\Users\adity\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarn
ing: The default value of gamma will change from 'auto' to 'scale' in version
0.22 to account better for unscaled features. Set gamma explicitly to 'auto' o
r 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

Accuracy: 0.7446808510638298
Confusion Matrix:
 [[ 79  30   0  30]
 [  0 158   1   5]
 [  5  46 129   8]
 [  2  28   1  89]]
```

In [49]:
```
from sklearn.svm import NuSVC
S_Classifier = NuSVC(kernel="linear").fit(X_train, y_train)

pred_2 = S_Classifier.predict(X_test)

print("Accuracy:",np.mean(pred_2 == y_test))
print("Confusion Matrix:\n",confusion_matrix(y_test, pred_2))
```

```
Accuracy: 0.9198036006546645
Confusion Matrix:
 [[131   3   1   4]
 [  1 161   1   1]
 [  0  13 175   0]
 [  6  16   3  95]]
```

In [50]:
```
from sklearn.svm import NuSVC
S_Classifier = NuSVC(kernel="poly").fit(X_train, y_train)

pred_2 = S_Classifier.predict(X_test)

print("Accuracy:",np.mean(pred_2 == y_test))
print("Confusion Matrix:\n",confusion_matrix(y_test, pred_2))
```

```
C:\Users\adity\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarn
ing: The default value of gamma will change from 'auto' to 'scale' in version
0.22 to account better for unscaled features. Set gamma explicitly to 'auto' o
r 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

Accuracy: 0.6333878887070377
Confusion Matrix:
 [[ 66  61   0  12]
 [  0 161   1   2]
 [  0  92  84  12]
 [  4  40   0  76]]
```

In [51]:
```
from sklearn.svm import NuSVC
S_Classifier = NuSVC(kernel="sigmoid").fit(X_train, y_train)

pred_2 = S_Classifier.predict(X_test)

print("Accuracy:",np.mean(pred_2 == y_test))
print("Confusion Matrix:\n",confusion_matrix(y_test, pred_2))
```

```
C:\Users\adity\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarn
ing: The default value of gamma will change from 'auto' to 'scale' in version
0.22 to account better for unscaled features. Set gamma explicitly to 'auto' o
r 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

Accuracy: 0.7414075286415712
Confusion Matrix:
 [[ 78  30   0  31]
 [  0 158   1   5]
 [  5  46 128   9]
 [  2  28   1  89]]
```

Comparing the accuracy of both algorithms when performed only on Nouns:

MultinomialNB and NuSVC:

MultinomialNB algorithm gives an accuracy of 84%, while NuSVC algorithm(default kernel rbf) gives an accuracy of 74%. MultinomialNB algorithm is giving higher accuracy than NuSVC algorithm also with Noun data. The reason behind this is that MultinomialNB algorithm generally works better with text data in which same words occur more frequently. In addition to this, it also works better with discrete count of text data. It is a type of algorithm that is considered as best one to use when the text data has higher frequency count of words. Due to all of these reasons, MultinomialNB is giving higher accuracy than NuSVC.

In both the cases, entire text data and just noun data, the MultinomialNB algorithm is more effective algorithm and gives higher accuracy than SVM(NuSVC)

Changing the kernel of NuSVC when performed on just Nouns:

The accuracy of NuSVC on default kernel rbf is 74.46%

The accuracy of NuSVC on kernel linear is 91%

The accuracy of NuSVC on kernel poly is 63%

The accuracy of NuSVC on kernel sigmoid is 74.14%

By comparing the NuSVC accuracy with different kernels, it can be clearly observed that changing the kernel does affect the accuracy. From the results, it can be seen that linear kernel gives the highest accuracy while poly kernel gives the lowest accuracy. The rbf and sigmoid kernel pretty much gives the same accuracy. So from this, we can say that kernel plays an important role in SVM and it can clearly increase as well as decrease the accuracy.

Comparision of accuracies (of entire data and just nouns)

MultinomialNB algorithm accuracy, when performed on entire data is 88%, while when performed on just noun data, it is 84%. There is a reduction in its accuracy when the data just contains nouns.

NuSVC algorithm(kernel='linear') accuracy, when performed on entire data is 94%, while when performed on just noun data, it is 91%. There is a reduction in its accuracy when the data just contains nouns.

By observing the accuracies of both the algorithms when performed on both entire data as well as noun data, it is clear that there is a reduction in accuracy when just noun data is considered.

Comparision of vocabulary size of entire data and just noun data:

The output of shape function for the entire data(part-c) is (2034, 21051)

The output of shape function for the noun data(part-d) is (2034, 34112)

It can be observed from the results that the number of rows remain same, while the number of columns gets increased when comparing both the entire data and just

noun data.

Implementation of shape functions for both types of data can be found in the code(TFIDF code) above cells.

References:

We have used some of the code from Tutorial-6, more specifically the filtering code and we are providing its reference below. Apart from this, we are also taking code from stackoverflow as reference and using it in our code(the cleaning code used in performing t-test etc.). We are providing all of the references below:

[1]"Google Colaboratory", Colab.research.google.com, 2019. [Online]. Available: https://colab.research.google.com/drive/17LMCbDOnny8h1KTqqoX3smy6 7zBNgSJx (https://colab.research.google.com/drive/17LMCbDOnny8h1KTqqoX3smy 7zBNgSJx). [Accessed: 17- Jul- 2019].[Tutorial-6]

[2]r. [closed], M. Karia, M. Pieters and V. Peer, "removing special characters from a list of items in python", Stack Overflow, 2019. [Online]. Available: https://stackoverflow.com/questions/47301795/removing-special-characters-from-a-list-of-items-in-python (https://stackoverflow.com/questions/47301795/removing-special-characters-from-a-list-of-items-in-python). [Accessed: 17- Jul- 2019].

[3]"nicharuc/Collocations", GitHub, 2019. [Online]. Available: https://github.com/nicharuc/Collocations/blob/master/Collocations.ipynb (https://github.com/nicharuc/Collocations/blob/master/Collocations.ipynb [Accessed: 17- Jul- 2019].

[4]"sklearn.svm.NuSVC — scikit-learn 0.21.2 documentation", Scikit-learn.org, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html (https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html). [Accessed: 17- Jul- 2019].

[5]"sklearn.naive_bayes.MultinomialNB — scikit-learn 0.21.2 documentation", Scikit-learn.org, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.h (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.h [Accessed: 17- Jul- 2019].

[6]"5.6.2. The 20 newsgroups text dataset — scikit-learn 0.19.2 documentation", Scikit-learn.org, 2019. [Online]. Available: https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html (https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html). [Accessed: 17- Jul- 2019].

[7]"5.2. Feature extraction — scikit-learn 0.21.2 documentation", Scikit-learn.org, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction (https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction). [Accessed: 17- Jul- 2019].

[8]"Natural Language Toolkit — NLTK 3.4.4 documentation", Nltk.org, 2019. [Online]. Available: http://www.nltk.org/ (http://www.nltk.org/). [Accessed: 17- Jul- 2019].

[9] Lab_Text Processing.ipynb (CSCI-5901 Lab-6 Lab_Text Processing) [Accessed: 17- Jul- 2019].

In [ ]: