# Assignment 2


### Pranay Gheewala - B00826923

### Karan Thakkar - B00823819


In [4]:
```
from sklearn.datasets import fetch_20newsgroups
import nltk
import sklearn
import spacy
import string
import re
import pandas as pd
```

# Q1. Collocation extraction

### Selecting 4 newsgroup as a part of interest

In [6]:
```
#selecting 4 newsgroup as a part of interest
corpus = ['alt.atheism','talk.religion.misc','comp.graphics', 'sci.space']
newsgroups_train = fetch_20newsgroups(subset='all', categories=corpus,shuffle=Tru
```

In [3]:
```
newsgroups_train.filenames.shape
```

Out[3]: (3387,)

In [4]:
```
newsgroups_train.target.shape
```

Out[4]: (3387,)

In [5]:
```
#creating dataframe of selected 4 newsgroup dataset
df= pd.DataFrame([newsgroups_train.data,newsgroups_train.target.tolist()])
```

In [6]:
```
df.head()
```

Out[6]:

| | 0 | 1 | 2 | 3 | |
|---|---|---|---|---|---|
| **0** | existance. \n\n\n**********... | Forwarded from the Mars Observer Project\n\n ... | \n\tThe word "released" is loaded: until conv... | ========== | |
| **1** | 3 | 0 | 2 | 3 | |

2 rows × 3387 columns

In [7]:
```
df=df.T
df.head()
```

Out[7]:

| | 0 | 1 |
|---|---|---|
| **0** | | 3 |
| **1** | existance. \n\n\n**********... | 0 |
| **2** | Forwarded from the Mars Observer Project\n\n ... | 2 |
| **3** | \n\tThe word "released" is loaded: until conv... | 3 |
| **4** | ==========... | 2 |

In [8]:
```
print(df)
```

```
                                                      0  1
0                                                        3
1     existance.  \n\n\n**************************...  0
2     Forwarded from the Mars Observer Project\n\n  ...  2
3     \n\tThe word "released" is loaded:  until conv...  3
4     ==========================================...  2
5     \n\n\n  That's _Five Weeks In A Balloon_.  And...  2
6     \nWhy don't you ask the approx. two million Br...  0
7     This response originally fell into a bit bucke...  0
8     \nI realy like this idea, it would be wonderfu...  2
9     I've done a bit of looking, and havn't been ab...  1
10    \nA lot of people won't agree with me.  That's...  3
11    \n(about my reply)\n\n\nIt a society that is c...  0
12    CALL FOR PAPERS\n                                ...  1
13    \n\nWow.  I was beginning to think that I had ...  2
14    I saw an imaging program some time ago on an A...  1
15    \nOr have different classes of competetors.. a...  2
16    \n[Rest deleted.  Can anybody out in a.p.h hel...  3
17    Stuff deleted\n\nI reiterate that I would agre...  0
18                                                       1
19    ------------------------------------\n\t+ ......  1
20    \n  [snip]\n\nHonest, it just ended like that,...  0
21    I need some help.  We are upgrading our animat...  1
22    I understand the when one is in orbit, the inw...  2
23    I commend everybody to look at the FTP site 'f...  2
24    \nOf course, the list has to agree with the ni...  0
25    \n\nI and many others on a.a have described ho...  0
26    I'm making a customized paint program in DOS a...  1
27    \n\nI _know_ I shouldn't get involved, but... ...  0
28    \nAnyone from Alabama knows it should be:\n\nI...  0
29    \nPrecisely.\n\nThere's no objective medicine;...  3
...                                                  ... ..
3357  \nAbsolutely not true. Without religion - eith...  0
3358  : BULLSHIT!!! In the Gulf Massacre, 7% of all ...  0
3359  \nNo, this is far from clear.  We only have th...  3
3360  \nSounds as though you are confused between "w...  0
3361  /(emery)\n/The one single historic event that ...  3
3362  Hi, Everyone.\nI am  currently planning to wri...  1
3363  [ stuff deleted ]\n\nThe French SPOT is an exa...  2
3364  Hi,\n\n\nAnyone has a converter from BMP to an...  1
3365  \nLast night CNN reported that FBI has infrare...  3
3366  \n:>re: majority of users not readding from fl...  1
3367  For some reasons we humans think that it is ou...  2
3368  \nBill, I think you have a misunderstanding ab...  0
3369  \n\n\n\n\n\n\n\nThis isn't inherently bad.\n\n...  2
3370  : \n: \tWild and fanciful claims require great...  0
3371  I need a graphics display program that can tak...  1
3372  \nOnly if he doesn't spend more than a billion...  2
3373  \nMy apologies if this is a re-post - I submit...  2
3374  \nWell, here goes.\n\nThe first item of busine...  2
3375  \nWhat is "aluminium siding"?  I keep seeing r...  3
3376  Having read in the past about the fail-safe me...  2
3377  \nAnd that, of course, is the point.   You can...  0
3378  BIOLOGICAL ALCHEMY\n                             ...  2
```

```
3379  \nJonathan, interesting questions.  Some wonde...  2
3380  \n\nApparently not.\n\nIn response to his clai...  0
3381  Is the film from the "putt-putt" test vehicle ...  2
3382  \nYes, atheists tend to claim self control and...  0
3383  \n\n\nOf course not. I would think that would ...  0
3384  \n#I use xwd/xpr (from the X11R5 dist.) and va...  1
3385  \nI could give much the same testimonial about...  0
3386  \n\nThe gravity data is collected in real-time...  2

[3387 rows x 2 columns]
```

In [9]:
```
#Renaming the columns of the dataframe
df.rename(columns={0: "news", 1: "categories"},inplace=True)
```

In [10]:
```
print(df)
```

```
                                                   news categories
0                                                                 3
1     existance.  \n\n\n**************************...          0
2     Forwarded from the Mars Observer Project\n\n  ...          2
3     \n\tThe word "released" is loaded:  until conv...          3
4     ==========================================...          2
5     \n\n\n  That's _Five Weeks In A Balloon_.  And...          2
6     \nWhy don't you ask the approx. two million Br...          0
7     This response originally fell into a bit bucke...          0
8     \nI realy like this idea, it would be wonderfu...          2
9     I've done a bit of looking, and havn't been ab...          1
10    \nA lot of people won't agree with me.  That's...          3
11    \n(about my reply)\n\n\nIt a society that is c...          0
12    CALL FOR PAPERS\n                                ...          1
13    \n\nWow.  I was beginning to think that I had ...          2
14    I saw an imaging program some time ago on an A...          1
15    \nOr have different classes of competetors.. a...          2
16    \n[Rest deleted.  Can anybody out in a.p.h hel...          3
17    Stuff deleted\n\nI reiterate that I would agre...          0
18                                                                1
19    ------------------------------------\n\t+ ......          1
20    \n  [snip]\n\nHonest, it just ended like that,...          0
21    I need some help.  We are upgrading our animat...          1
22    I understand the when one is in orbit, the inw...          2
23    I commend everybody to look at the FTP site 'f...          2
24    \nOf course, the list has to agree with the ni...          0
25    \n\nI and many others on a.a have described ho...          0
26    I'm making a customized paint program in DOS a...          1
27    \n\nI _know_ I shouldn't get involved, but... ...          0
28    \nAnyone from Alabama knows it should be:\n\nI...          0
29    \nPrecisely.\n\nThere's no objective medicine;...          3
...                                                 ...        ...
3357  \nAbsolutely not true. Without religion - eith...          0
3358  : BULLSHIT!!! In the Gulf Massacre, 7% of all ...          0
3359  \nNo, this is far from clear.  We only have th...          3
3360  \nSounds as though you are confused between "w...          0
3361  /(emery)\n/The one single historic event that ...          3
3362  Hi, Everyone.\nI am  currently planning to wri...          1
3363  [ stuff deleted ]\n\nThe French SPOT is an exa...          2
3364  Hi,\n\n\nAnyone has a converter from BMP to an...          1
3365  \nLast night CNN reported that FBI has infrare...          3
3366  \n:>re: majority of users not readding from fl...          1
3367  For some reasons we humans think that it is ou...          2
3368  \nBill, I think you have a misunderstanding ab...          0
3369  \n\n\n\n\n\n\n\nThis isn't inherently bad.\n\n...          2
3370  : \n: \tWild and fanciful claims require great...          0
3371  I need a graphics display program that can tak...          1
3372  \nOnly if he doesn't spend more than a billion...          2
3373  \nMy apologies if this is a re-post - I submit...          2
3374  \nWell, here goes.\n\nThe first item of busine...          2
3375  \nWhat is "aluminium siding"?  I keep seeing r...          3
3376  Having read in the past about the fail-safe me...          2
3377  \nAnd that, of course, is the point.   You can...          0
3378  BIOLOGICAL ALCHEMY\n                             ...          2
```

```
3379  \nJonathan, interesting questions.  Some wonde...         2
3380  \n\nApparently not.\n\nIn response to his clai...         0
3381  Is the film from the "putt-putt" test vehicle ...         2
3382  \nYes, atheists tend to claim self control and...         0
3383  \n\n\nOf course not. I would think that would ...         0
3384  \n#I use xwd/xpr (from the X11R5 dist.) and va...         1
3385  \nI could give much the same testimonial about...         0
3386  \n\nThe gravity data is collected in real-time...         2

[3387 rows x 2 columns]
```

In [11]:
```
#cleaning all punctuation marks from the dataset
df['news']=df['news'].str.replace('[{}]'.format(string.punctuation), '')
df['news'].head()
```

Out[11]:
```
0
1    existance  \n\n\n\n\tI just thought it necessa...
2    Forwarded from the Mars Observer Project\n\n  ...
3    \n\tThe word released is loaded  until convict...
4    \nI am posting this for someone else  Please r...
Name: news, dtype: object
```

In [12]:
```
#removing unnecessary characters and making characters lowercase
df["news"] = df["news"].str.lower()
df["news"] = df["news"].str.replace(r"[^a-z]+"," ")
df["news"].head()
```

Out[12]:
```
0
1    existance i just thought it necessary to help ...
2    forwarded from the mars observer project mars ...
3     the word released is loaded until convicted i...
4     i am posting this for someone else please res...
Name: news, dtype: object
```

## a.) Tokenizing the dataset

In [13]:
```
df['tokenize_news'] = [ nltk.word_tokenize( str(sentence) ) for sentence in df['
print(df['tokenize_news'])
```

```
0                                                      []
1       [existance, i, just, thought, it, necessary, t...
2       [forwarded, from, the, mars, observer, project...
3       [the, word, released, is, loaded, until, convi...
4       [i, am, posting, this, for, someone, else, ple...
5       [thats, five, weeks, in, a, balloon, and, if, ...
6       [why, dont, you, ask, the, approx, two, millio...
7       [this, response, originally, fell, into, a, bi...
8       [i, realy, like, this, idea, it, would, be, wo...
9       [ive, done, a, bit, of, looking, and, havnt, b...
10      [a, lot, of, people, wont, agree, with, me, th...
11      [about, my, reply, it, a, society, that, is, c...
12      [call, for, papers, progress, in, neural, netw...
13      [wow, i, was, beginning, to, think, that, i, h...
14      [i, saw, an, imaging, program, some, time, ago...
15      [or, have, different, classes, of, competetors...
16      [rest, deleted, can, anybody, out, in, aph, he...
17      [stuff, deleted, i, reiterate, that, i, would,...
18                                                     []
19      [the, otis, project, the, operative, term, is,...
20      [snip, honest, it, just, ended, like, that, i,...
21      [i, need, some, help, we, are, upgrading, our,...
22      [i, understand, the, when, one, is, in, orbit,...
23      [i, commend, everybody, to, look, at, the, ftp...
24      [of, course, the, list, has, to, agree, with, ...
25      [i, and, many, others, on, aa, have, described...
26      [im, making, a, customized, paint, program, in...
27      [i, know, i, shouldnt, get, involved, but, bit...
28      [anyone, from, alabama, knows, it, should, be,...
29      [precisely, theres, no, objective, medicine, s...
                              ...
3357    [absolutely, not, true, without, religion, eit...
3358    [bullshit, in, the, gulf, massacre, of, all, o...
3359    [no, this, is, far, from, clear, we, only, hav...
3360    [sounds, as, though, you, are, confused, betwe...
3361    [emery, the, one, single, historic, event, tha...
3362    [hi, everyone, i, am, currently, planning, to,...
3363    [stuff, deleted, the, french, spot, is, an, ex...
3364    [hi, anyone, has, a, converter, from, bmp, to,...
3365    [last, night, cnn, reported, that, fbi, has, i...
3366    [re, majority, of, users, not, readding, from,...
3367    [for, some, reasons, we, humans, think, that, ...
3368    [bill, i, think, you, have, a, misunderstandin...
3369    [this, isnt, inherently, bad, this, isnt, real...
3370    [wild, and, fanciful, claims, require, greater...
3371    [i, need, a, graphics, display, program, that,...
3372    [only, if, he, doesnt, spend, more, than, a, b...
3373    [my, apologies, if, this, is, a, repost, i, su...
3374    [well, here, goes, the, first, item, of, busin...
3375    [what, is, aluminium, siding, i, keep, seeing,...
3376    [having, read, in, the, past, about, the, fail...
3377    [and, that, of, course, is, the, point, you, c...
3378    [biological, alchemy, another, form, of, cold,...
```

```
3379    [jonathan, interesting, questions, some, wonde...
3380    [apparently, not, in, response, to, his, claim...
3381    [is, the, film, from, the, puttputt, test, veh...
3382    [yes, atheists, tend, to, claim, self, control...
3383    [of, course, not, i, would, think, that, would...
3384    [i, use, xwdxpr, from, the, x, r, dist, and, v...
3385    [i, could, give, much, the, same, testimonial,...
3386    [the, gravity, data, is, collected, in, realti...
Name: tokenize_news, Length: 3387, dtype: object
```

## a.) Part of speech tagging on tokenized words

In [14]:
```
from nltk import pos_tag
tokenized_news =  df['tokenize_news'].tolist()
pos = []
for tokens in tokenized_news:
    pos +=tokens
tagged_news=(pos_tag(pos))
```

In [15]:
```
tagged_news
```

Out[15]:
```
[('existance', 'NN'),
 ('i', 'NN'),
 ('just', 'RB'),
 ('thought', 'VBD'),
 ('it', 'PRP'),
 ('necessary', 'JJ'),
 ('to', 'TO'),
 ('help', 'VB'),
 ('defend', 'VB'),
 ('the', 'DT'),
 ('point', 'NN'),
 ('that', 'WDT'),
 ('jesus', 'NN'),
 ('existed', 'VBD'),
 ('guys', 'NNS'),
 ('jesus', 'NN'),
 ('existed', 'VBD'),
 ('if', 'IN'),
 ('he', 'PRP'),
```

In [16]:
```
#removing stop words from the dataset
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
df['tokenize_news']=df['tokenize_news'].apply(lambda x: [item for item in x if i
```

In [17]:
```
df['tokenize_news']
```

Out[17]:
```
0                                                    []
1       [existance, thought, necessary, help, defend, ...
2       [forwarded, mars, observer, project, mars, obs...
3       [word, released, loaded, convicted, cxourt, ch...
4       [posting, someone, else, please, respond, addr...
5       [thats, five, weeks, balloon, anyone, tell, ge...
6       [dont, ask, approx, two, million, british, mus...
7       [response, originally, fell, bit, bucket, im, ...
8       [realy, like, idea, would, wonderfull, see, bi...
9       [ive, done, bit, looking, havnt, able, come, m...
10      [lot, people, wont, agree, thats, right, respe...
11      [reply, society, constantly, verge, flaming, u...
12      [call, papers, progress, neural, networks, spe...
13      [wow, beginning, think, made, remember, movie,...
14      [saw, imaging, program, time, ago, amiga, cros...
15      [different, classes, competetors, made, total,...
16      [rest, deleted, anybody, aph, help, find, nigh...
17      [stuff, deleted, reiterate, would, agree, litt...
18                                                   []
19      [otis, project, operative, term, stimulate, fi...
20      [snip, honest, ended, like, swear, hmmmmi, rec...
21      [need, help, upgrading, animationvideo, editin...
22      [understand, one, orbit, inward, force, gravit...
23      [commend, everybody, look, ftp, site, ftpcicbf...
24      [course, list, agree, nickname, laws, laid, gi...
25      [many, others, aa, described, tried, find, god...
26      [im, making, customized, paint, program, dos, ...
27      [know, shouldnt, get, involved, bit, deleted, ...
28      [anyone, alabama, knows, bear, catholic, pope,...
29      [precisely, theres, objective, medicine, peopl...
                              ...
3357    [absolutely, true, without, religion, either, ...
3358    [bullshit, gulf, massacre, ordnance, used, sma...
3359    [far, clear, word, fbi, spokepeople, survivor,...
3360    [sounds, though, confused, want, think, morall...
3361    [emery, one, single, historic, event, biggest,...
3362    [hi, everyone, currently, planning, write, pro...
3363    [stuff, deleted, french, spot, example, comes,...
3364    [hi, anyone, converter, bmp, format, xview, xv...
3365    [last, night, cnn, reported, fbi, infrared, pi...
3366    [majority, users, readding, floppy, well, us, ...
3367    [reasons, humans, think, place, control, every...
3368    [bill, think, misunderstanding, atheism, lack,...
3369    [isnt, inherently, bad, isnt, really, light, p...
3370    [wild, fanciful, claims, require, greater, evi...
3371    [need, graphics, display, program, take, param...
3372    [doesnt, spend, billion, dollars, since, prize...
3373    [apologies, repost, submitted, friday, got, me...
3374    [well, goes, first, item, business, establish,...
3375    [aluminium, siding, keep, seeing, references, ...
3376    [read, past, failsafe, mechanisms, spacecraft,...
3377    [course, point, cant, simply, divide, world, a...
3378    [biological, alchemy, another, form, cold, fus...
3379    [jonathan, interesting, questions, wonder, whe...
```

```
3380    [apparently, response, claim, terrifies, gay, ...
3381    [film, puttputt, test, vehicle, used, conventi...
3382    [yes, atheists, tend, claim, self, control, se...
3383    [course, would, think, would, great, fun, ever...
3384    [use, xwdxpr, x, r, dist, various, programs, p...
3385    [could, give, much, testimonial, experience, s...
3386    [gravity, data, collected, realtime, recorded,...
Name: tokenize_news, Length: 3387, dtype: object
```

In [18]:
```
# Stemming the dataset to reduce words to their root word or to chop off the der
from nltk.stem import PorterStemmer
ps = PorterStemmer()
df["tokenize_news"] = df["tokenize_news"].apply(lambda x: [ps.stem(y) for y in x
df['tokenize_news'].head()
```

Out[18]:
```
0                                                      []
1    [exist, thought, necessari, help, defend, poin...
2    [forward, mar, observ, project, mar, observ, s...
3      [word, releas, load, convict, cxourt, children]
4    [post, someon, els, pleas, respond, address, l...
Name: tokenize_news, dtype: object
```

In [19]:
```
#Lemmatizing the dataset to reduce words to their base word, which is linguistic
from nltk.stem.wordnet import WordNetLemmatizer
lemmatize_data = WordNetLemmatizer()
df["tokenize_news"] = df["tokenize_news"].apply(lambda a: [lemmatize_data.lemmat
df["tokenize_news"].head()
```

Out[19]:
```
0                                                      []
1    [exist, thought, necessari, help, defend, poin...
2    [forward, mar, observ, project, mar, observ, s...
3         [word, releas, load, convict, cxourt, child]
4    [post, someon, el, plea, respond, address, lis...
Name: tokenize_news, dtype: object
```

In [20]:
```
#turning all tokens to one single list
cleaned_news =  df['tokenize_news']
cleaned_words = []
for wordList in cleaned_news:
    cleaned_words +=wordList
```

In [21]:
```
cleaned_words
```

Out[21]:
```
['exist',
 'thought',
 'necessari',
 'help',
 'defend',
 'point',
 'jesu',
 'exist',
 'guy',
 'jesu',
 'exist',
 'didnt',
 'say',
 'socrat',
 'didnt',
 'exist',
 'cuz',
 'like',
 'jesu',
```

In [22]:
```
#finding frequency distribution of the words
from nltk.probability import FreqDist
fdist = FreqDist(cleaned_words)
print(fdist)
```

```
<FreqDist with 24633 samples and 360065 outcomes>
```

In [23]:
```
#most common used 3 words in the newsgroup
fdist.most_common(3)
```

Out[23]:
```
[('one', 2090), ('use', 2067), ('would', 2006)]
```

# b.) Applying techinques to extract Bigram collocation from the corpus

In [24]:
```
bigrams = nltk.collocations.BigramAssocMeasures()
bigramFinder = nltk.collocations.BigramCollocationFinder.from_words(cleaned_words
```

### Frequency with filter

In [25]:
```
#finding frequencies of each bigram
bigram_freq = bigramFinder.ngram_fd.items()
bigramFreqTable = pd.DataFrame(list(bigram_freq), columns=['bigram','freq']).sor
bigramFreqTable.head().reset_index(drop=True)
```

Out[25]:

| | bigram | freq |
|---|---|---|
| **0** | (dont, know) | 169 |
| **1** | (dont, think) | 136 |
| **2** | (would, like) | 130 |
| **3** | (anonym, ftp) | 124 |
| **4** | (imag, process) | 123 |

In [26]:
```
#Top 20 bigram having higher frequencies
bigramFreqTable[:20]
```

Out[26]:

| | bigram | freq |
|---|---|---|
| **3347** | (dont, know) | 169 |
| **1624** | (dont, think) | 136 |
| **349** | (would, like) | 130 |
| **2504** | (anonym, ftp) | 124 |
| **7403** | (imag, process) | 123 |
| **3861** | (comput, graphic) | 115 |
| **6260** | (im, sure) | 110 |
| **5966** | (space, station) | 92 |
| **5430** | (file, format) | 92 |
| **3977** | (anyon, know) | 91 |
| **135396** | (lord, jehovah) | 90 |
| **12302** | (solar, system) | 88 |
| **4017** | (mani, peopl) | 86 |
| **759** | (god, exist) | 83 |
| **3351** | (believ, god) | 82 |
| **15288** | (po, box) | 81 |
| **5103** | (sourc, code) | 80 |
| **21563** | (bit, imag) | 77 |
| **3303** | (year, ago) | 76 |
| **135162** | (god, elohim) | 75 |

In [27]:
```
#function to filter for ADJ/NN bigrams as pronouns/articles/prepositions come up
def rightTypes(ngram):
    if '-pron-' in ngram or '' in ngram or ' 'in ngram or 't' in ngram:
        return False
    acceptable_types = ('JJ', 'JJR', 'JJS', 'NN', 'NNS', 'NNP', 'NNPS')
    second_acceptable_type = ('NN', 'NNS', 'NNP', 'NNPS')
    tags = nltk.pos_tag(ngram)
    if tags[0][1] in acceptable_types and tags[1][1] in second_acceptable_type:
        return True
    else:
        return False
```

In [28]:
```
#filtered bigrams
filtered_bi = bigramFreqTable[bigramFreqTable.bigram.map(lambda x: rightTypes(x))
filtered_bi[:20]
```

Out[28]:

| | bigram | freq |
|---|---|---|
| **1624** | (dont, think) | 136 |
| **2504** | (anonym, ftp) | 124 |
| **7403** | (imag, process) | 123 |
| **3861** | (comput, graphic) | 115 |
| **6260** | (im, sure) | 110 |
| **5966** | (space, station) | 92 |
| **5430** | (file, format) | 92 |
| **135396** | (lord, jehovah) | 90 |
| **12302** | (solar, system) | 88 |
| **4017** | (mani, peopl) | 86 |
| **3351** | (believ, god) | 82 |
| **15288** | (po, box) | 81 |
| **5103** | (sourc, code) | 80 |
| **135162** | (god, elohim) | 75 |
| **3184** | (thank, advanc) | 75 |
| **27334** | (space, shuttl) | 74 |
| **444** | (x, x) | 70 |
| **953** | (jesu, christ) | 67 |
| **2490** | (ftp, site) | 67 |
| **34573** | (p, p) | 66 |

In [29]:
```
freq_bi = filtered_bi[:20].bigram.values
```

## Bigrams using PMI

**I have used PMI with frequency filter because PMI is very sensitive to rare or misspelled combination of words so it is important to use frequency filter**

In [30]:
```
#applying frequency filter
bigramFinder.apply_freq_filter(20)
```

In [31]:
```
#finding PMI of bigrams
bigramPMITable = pd.DataFrame(list(bigramFinder.score_ngrams(bigrams.pmi)), colu
```

In [32]:
```
#Top 20 bigrams having highest PMI
bigramPMITable[:20]
```

Out[32]:

| | bigram | PMI |
|---|---|---|
| 0 | (beauchain, bobbeviceicotekcom) | 13.698564 |
| 1 | (blew, bronx) | 13.650543 |
| 2 | (sank, manhattan) | 13.641981 |
| 3 | (manhattan, sea) | 12.757458 |
| 4 | (san, jose) | 12.503702 |
| 5 | (vertex, vertex) | 12.375436 |
| 6 | (mari, shafer) | 12.337361 |
| 7 | (ame, dryden) | 12.240836 |
| 8 | (joseph, smith) | 12.110941 |
| 9 | (bob, beauchain) | 12.026275 |
| 10 | (et, al) | 11.936297 |
| 11 | (stay, blew) | 11.913577 |
| 12 | (gammaray, burst) | 11.872935 |
| 13 | (physicist, dewey) | 11.872935 |
| 14 | (thou, shalt) | 11.629232 |
| 15 | (cheer, kent) | 11.614433 |
| 16 | (oort, cloud) | 11.576784 |
| 17 | (washington, dc) | 11.335801 |
| 18 | (mr, decenso) | 11.207599 |
| 19 | (po, box) | 11.159538 |

In [33]:
```
pmi_bi = bigramPMITable[:20].bigram.values
```

## Bigram using T-test with filter

In [34]:
```
#applying frequency filter
bigramFinder.apply_freq_filter(20)
```

In [35]:
```
#extracting bigrams using T-test
bigramTtable = pd.DataFrame(list(bigramFinder.score_ngrams(bigrams.student_t)),
bigramTtable.head()
```

Out[35]:

| | bigram | t |
|---|---|---|
| **0** | (dont, know) | 12.686487 |
| **1** | (dont, think) | 11.328072 |
| **2** | (anonym, ftp) | 11.122078 |
| **3** | (imag, process) | 10.899800 |
| **4** | (would, like) | 10.746015 |

**T-test are also sensitive to pairs with prepositions, pronouns, articles etc. so I have applied filter to consider the right types**

In [36]:
```
#filtering bigrams
filteredT_bi = bigramTtable[bigramTtable.bigram.map(lambda x: rightTypes(x))]
filteredT_bi[:20]
```

Out[36]:

| | bigram | t |
|---|---|---|
| 1 | (dont, think) | 11.328072 |
| 2 | (anonym, ftp) | 11.122078 |
| 3 | (imag, process) | 10.899800 |
| 5 | (comput, graphic) | 10.638477 |
| 6 | (im, sure) | 10.423276 |
| 7 | (space, station) | 9.522928 |
| 8 | (lord, jehovah) | 9.473173 |
| 10 | (file, format) | 9.363303 |
| 11 | (solar, system) | 9.301847 |
| 12 | (po, box) | 8.996066 |
| 13 | (mani, peopl) | 8.961285 |
| 14 | (sourc, code) | 8.896898 |
| 16 | (believ, god) | 8.711577 |
| 18 | (thank, advanc) | 8.635711 |
| 19 | (god, elohim) | 8.592764 |
| 20 | (space, shuttl) | 8.483513 |
| 23 | (x, x) | 8.241457 |
| 24 | (ftp, site) | 8.141646 |
| 25 | (jesu, christ) | 8.139612 |
| 26 | (p, p) | 8.057854 |

In [37]:
```
#top 20 bigrams haing highest T-test score
t_bi = filteredT_bi[:20].bigram.values
```

## Bigrams using Chi-Sq Test

In [38]:
```
#Finding bigrams using Chi-Sq Test
bigramChiTable = pd.DataFrame(list(bigramFinder.score_ngrams(bigrams.chi_sq)), c
```

In [39]:
```
bigramChiTable.head(20)
```

Out[39]:

| | bigram | chi-sq |
|---|---|---|
| 0 | (beauchain, bobbeviceicotekcom) | 319070.178270 |
| 1 | (blew, bronx) | 308623.714057 |
| 2 | (sank, manhattan) | 306797.254182 |
| 3 | (cheer, kent) | 194361.877991 |
| 4 | (po, box) | 185244.131557 |
| 5 | (vertex, vertex) | 180637.777414 |
| 6 | (san, jose) | 168402.063273 |
| 7 | (manhattan, sea) | 166169.999021 |
| 8 | (mari, shafer) | 134534.778178 |
| 9 | (washington, dc) | 118859.027014 |
| 10 | (ame, dryden) | 111304.633991 |
| 11 | (bob, beauchain) | 104263.599234 |
| 12 | (ray, tracer) | 102693.574853 |
| 13 | (anonym, ftp) | 102556.654940 |
| 14 | (kilomet, mile) | 101323.416620 |
| 15 | (oort, cloud) | 97720.396056 |
| 16 | (et, al) | 94035.394346 |
| 17 | (physicist, dewey) | 93744.790018 |
| 18 | (stay, blew) | 92567.912840 |
| 19 | (joseph, smith) | 88447.959577 |

In [40]:
```
#Top 20 bigrams with highest Chi-Sq test score
chi_bi = bigramChiTable[:20].bigram.values
```

## c.) Comparing all the results from the tests

In [41]:
```
#Making a dataframe of all tests results
compare = pd.DataFrame([freq_bi,pmi_bi,t_bi,chi_bi])
```

In [42]:
```
compare=compare.T
compare.rename(columns={0: "Frequnecy with Filter", 1: "PMI",2:"T-test with Filte
compare
```

Out[42]:

| | Frequnecy with Filter | PMI | T-test with Filter | Chi-Sq Test |
|---|---|---|---|---|
| 0 | (dont, think) | (beauchain, bobbeviceicotekcom) | (dont, think) | (beauchain, bobbeviceicotekcom) |
| 1 | (anonym, ftp) | (blew, bronx) | (anonym, ftp) | (blew, bronx) |
| 2 | (imag, process) | (sank, manhattan) | (imag, process) | (sank, manhattan) |
| 3 | (comput, graphic) | (manhattan, sea) | (comput, graphic) | (cheer, kent) |
| 4 | (im, sure) | (san, jose) | (im, sure) | (po, box) |
| 5 | (space, station) | (vertex, vertex) | (space, station) | (vertex, vertex) |
| 6 | (file, format) | (mari, shafer) | (lord, jehovah) | (san, jose) |
| 7 | (lord, jehovah) | (ame, dryden) | (file, format) | (manhattan, sea) |
| 8 | (solar, system) | (joseph, smith) | (solar, system) | (mari, shafer) |
| 9 | (mani, peopl) | (bob, beauchain) | (po, box) | (washington, dc) |
| 10 | (believ, god) | (et, al) | (mani, peopl) | (ame, dryden) |
| 11 | (po, box) | (stay, blew) | (sourc, code) | (bob, beauchain) |
| 12 | (sourc, code) | (gammaray, burst) | (believ, god) | (ray, tracer) |
| 13 | (god, elohim) | (physicist, dewey) | (thank, advanc) | (anonym, ftp) |
| 14 | (thank, advanc) | (thou, shalt) | (god, elohim) | (kilomet, mile) |
| 15 | (space, shuttl) | (cheer, kent) | (space, shuttl) | (oort, cloud) |
| 16 | (x, x) | (oort, cloud) | (x, x) | (et, al) |
| 17 | (jesu, christ) | (washington, dc) | (ftp, site) | (physicist, dewey) |
| 18 | (ftp, site) | (mr, decenso) | (jesu, christ) | (stay, blew) |
| 19 | (p, p) | (po, box) | (p, p) | (joseph, smith) |

## d.) How much overlap is there among the techniques? Do you think it makes sense to consider the union of the results?

- As we can see from the combined results Frequency with filter and T-test with filter results are having almost similar extractions of bigrams. These shows there is overlap among the techniques. Also PMI and Chi-Sq test results are having almost similar extraction of bigrams as it is seen from the results.
- As PMI is very sensitive to rare or misspelled combination of words so it is important to use in union frequency filter to produce better bigrams
- Also T-test are also sensitive to pairs with prepositions, pronouns, articles etc. so it would be beneficial to apply filter to extract bigrams of right types

- The result produced from all the test are almost similar so it would not make sense to take union of the results.

# Q2. SVM and NB for Text Classification

In [7]:
```
#creating a new dataframe of the corpus of interest for text classification
df_2= pd.DataFrame([newsgroups_train.data,newsgroups_train.target.tolist()])
```

In [8]:
```
df_2.head()
```

Out[8]:

| | 0 | 1 | 2 | 3 | |
|---|---|---|---|---|---|
| **0** | existance. \n\n\n**********... | Forwarded from the Mars Observer Project\n\n ... | \n\tThe word "released" is loaded: until conv... | ========== | |
| **1** | 3 | 0 | 2 | 3 | |

2 rows × 3387 columns

In [9]:
```
df_2=df_2.T
df_2.head()
```

Out[9]:

| | 0 | 1 |
|---|---|---|
| **0** | | 3 |
| **1** | existance. \n\n\n**********... | 0 |
| **2** | Forwarded from the Mars Observer Project\n\n ... | 2 |
| **3** | \n\tThe word "released" is loaded: until conv... | 3 |
| **4** | ==========... | 2 |

In [10]:
```
#renaming the column names
df_2.rename(columns={0: "news", 1: "categories"},inplace=True)
df_2.head()
```

Out[10]:

| | news | categories |
|---|---|---|
| 0 | | 3 |
| 1 | existance. \n\n\n********************... | 0 |
| 2 | Forwarded from the Mars Observer Project\n\n ... | 2 |
| 3 | \n\tThe word "released" is loaded: until conv... | 3 |
| 4 | ==========...| 2 |

# a.) Clean the text

## Removing numbers and other non - letter characters

In [11]:
```
df_2["news"] = df_2["news"].str.lower()
df_2["news"] = df_2["news"].str.replace(r"[^a-z]+"," ")
df_2['news']=df_2['news'].str.replace('[{}]'.format(string.punctuation), '')
df_2['news'] = df_2['news'].str.replace('\d+', '')
df_2["news"].head()
```

Out[11]:
```
0
1    existance i just thought it necessary to help ...
2    forwarded from the mars observer project mars ...
3     the word released is loaded until convicted i...
4     i am posting this for someone else please res...
Name: news, dtype: object
```

## Removing Stop Words

In [12]:
```
from nltk.corpus import stopwords
df_2['news']=df_2['news'].str.split()
stop_words = stopwords.words('english')
df_2['news']=df_2['news'].apply(lambda x: [item for item in x if item not in sto
df_2['news'].head()
```

Out[12]:
```
0                                                   []
1    [existance, thought, necessary, help, defend, ...
2    [forwarded, mars, observer, project, mars, obs...
3    [word, released, loaded, convicted, cxourt, ch...
4    [posting, someone, else, please, respond, addr...
Name: news, dtype: object
```

## Stemming the words

In [13]:
```
# Stemming the dataset to reduce words to their root word or to chop off the der
from nltk.stem import PorterStemmer
ps = PorterStemmer()
df_2["news"] = df_2["news"].apply(lambda x: [ps.stem(y) for y in x])
df_2['news'].head()
```

Out[13]:
```
0                                                   []
1    [exist, thought, necessari, help, defend, poin...
2    [forward, mar, observ, project, mar, observ, s...
3      [word, releas, load, convict, cxourt, children]
4    [post, someon, els, pleas, respond, address, l...
Name: news, dtype: object
```

In [14]:
```
df_2['news'] = df_2['news'].apply(' '.join)
df_2['news'].head()
```

Out[14]:
```
0
1    exist thought necessari help defend point jesu...
2    forward mar observ project mar observ statu re...
3              word releas load convict cxourt children
4    post someon els pleas respond address list ple...
Name: news, dtype: object
```

**b.) We need to convert the text files into numerical feature vectors to run our machine learning algorithms, so I have convert them into bag of words using CountVectorizer which counts number of times each word occur in a document and assign an integer ID. Using this method we can get the vocabulary dictionary.**

In [15]:
```
#Creating bag of words vector using CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer
corpus=(df_2['news'])
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(corpus)
X_train_counts.shape
```

Out[15]:
```
(3387, 20816)
```

**b.) To avoid giving more more weightage to longer documents than shorter documents and to reduce the weightage of more common words like (the, is, an etc) we use TF-IDF. I have used TfidfTransformer to create bag-of-words tf-idf weighted vector representation**

In [16]:
```
#bag-of-words tf-idf weighted vector representation
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts).toarray()
X_train_tfidf.shape
```

Out[16]:
```
(3387, 20816)
```

## c.) Splitting the data randomly into training and testing set (70-30 %).

In [17]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_train_tfidf, df_2['categor
```

## c.)Training Multinomial NB

In [18]:
```
from sklearn.naive_bayes import MultinomialNB
import numpy as np
clf = MultinomialNB().fit(X_train, y_train)
predictions_NB = clf.predict(X_test)
print("Naive Bayes Accuracy Score -> ",np.mean(predictions_NB==y_test))
```

```
Naive Bayes Accuracy Score ->  0.7626918536009445
```

## c.) Confusion matrix for Multinomial NB

In [19]:
```
from sklearn.metrics import confusion_matrix
print("Confusion Matrix ")
confusion_matrix(y_test, predictions_NB)
```

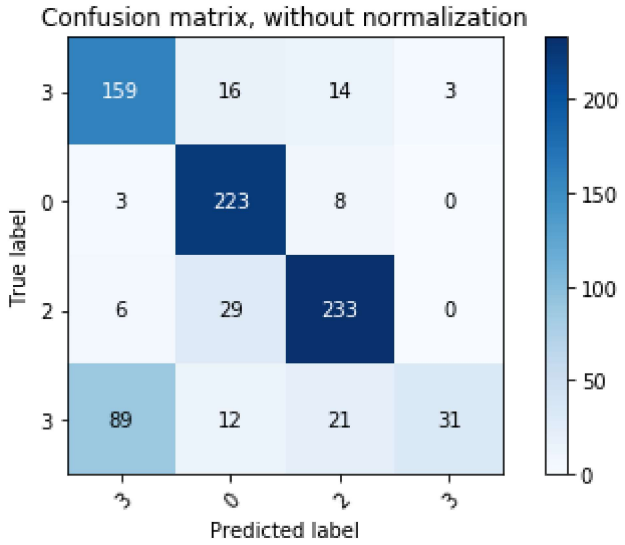
```
Confusion Matrix
```

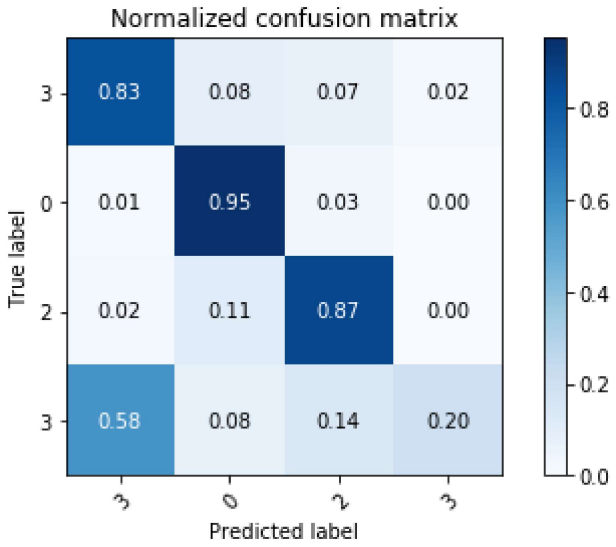
Out[19]:
```
array([[159,  16,  14,   3],
       [  3, 223,   8,   0],
       [  6,  29, 233,   0],
       [ 89,  12,  21,  31]], dtype=int64)
```

In [21]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_2['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                     ha="center", va="center",
                     color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predictions_NB, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predictions_NB, classes=class_names, normalize=Tru
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[159  16  14   3]
 [  3 223   8   0]
 [  6  29 233   0]
 [ 89  12  21  31]]
Normalized confusion matrix
[[0.83 0.08 0.07 0.02]
 [0.01 0.95 0.03 0.  ]
 [0.02 0.11 0.87 0.  ]
 [0.58 0.08 0.14 0.2 ]]
```






## c.) Training SVM model(SGDC Classifier)

In [24]:
```
from sklearn.linear_model import SGDClassifier
import numpy as np
text_clf_svm=SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3,  random_state
predicted_svm = text_clf_svm.predict(X_test)
print("SVM Accuracy Score -> ",np.mean(predicted_svm == y_test))
```

```
D:\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:166:
FutureWarning: max_iter and tol parameters have been added in SGDClassifier in
0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol i
s not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter wil
l be 1000, and default tol will be 1e-3.
  FutureWarning)

SVM Accuracy Score ->  0.8170011806375442
```

## c.) Confusion Matrix for SVM model

In [25]:
```
from sklearn.metrics import confusion_matrix
print("Confusion Matrix")
confusion_matrix(y_test, predicted_svm)
```

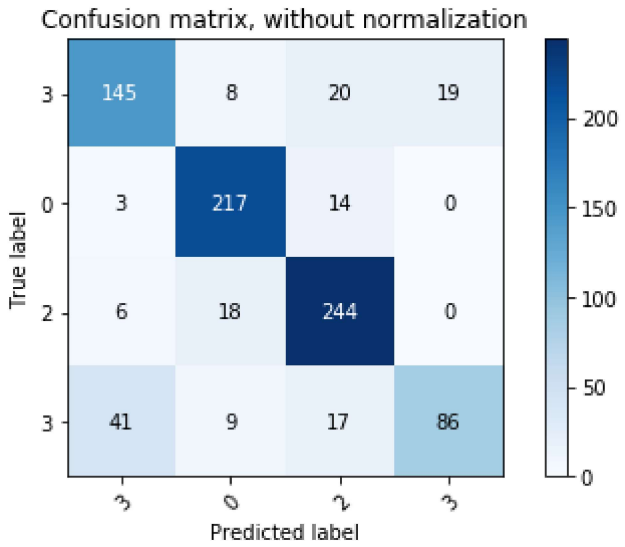
```
Confusion Matrix
```

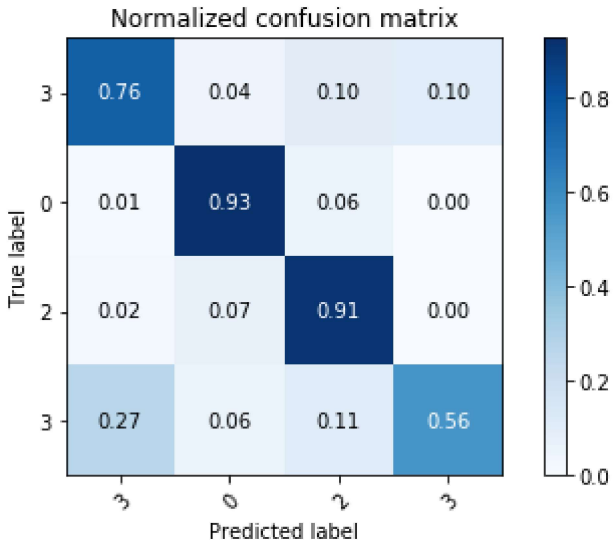
Out[25]:
```
array([[145,   8,  20,  19],
       [  3, 217,  14,   0],
       [  6,  18, 244,   0],
       [ 41,   9,  17,  86]], dtype=int64)
```

In [26]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_2['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                     ha="center", va="center",
                     color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predicted_svm, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predicted_svm, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[145   8  20  19]
 [  3 217  14   0]
 [  6  18 244   0]
 [ 41   9  17  86]]
Normalized confusion matrix
[[0.76 0.04 0.1  0.1 ]
 [0.01 0.93 0.06 0.  ]
 [0.02 0.07 0.91 0.  ]
 [0.27 0.06 0.11 0.56]]
```






- SVM alogrithm is having higher accuracy as compared to the MultinomailNB algorithm. SVM algorithm is having 81.7% accuracy whereas MultinomialNB is having 76.2% accuracy.
- SVM is having higher accuracy because SVM considers features by how much interatcion is there between them to a certain degree while MultinomailNB treats features as independent.
- Also SVM simple decision boundary and uses very sparse features which gives more accuracy.
- Moreover SVM have various hyperparamter such as kernel which take data as input and transform it into the required form, C which is penalty parameter and gamma which defines how much influence a single training example has, which gives more accuracy than MultinomialNB classifier which does not have all these hyperparameter which would produce more accuracy.

## c.) Training SVM using different kernels

## taking kernel='rbf' (Gaussian radial basis function)

In [40]:
```
from sklearn.svm import NuSVC
from sklearn.metrics import confusion_matrix
import numpy as np
clf_SVC_rbf = NuSVC(kernel='rbf').fit(X_train, y_train)
predictions_SVC_rbf = clf_SVC_rbf.predict(X_test)
print("Accuracy Score -> ",np.mean(predictions_SVC_rbf==y_test))
print("Confusion Matrix")
confusion_matrix(y_test, predictions_SVC_rbf)
```

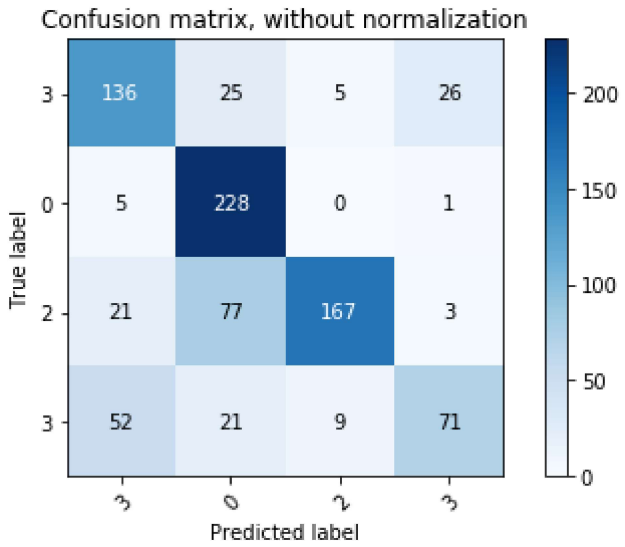
```
D:\Anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The defa
ult value of gamma will change from 'auto' to 'scale' in version 0.22 to accoun
t better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to av
oid this warning.
  "avoid this warning.", FutureWarning)

Accuracy Score ->  0.7107438016528925
Confusion Matrix
```

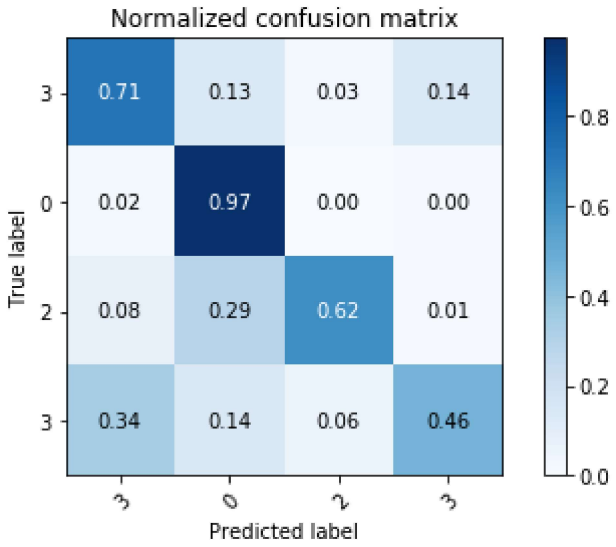
Out[40]:
```
array([[136,  25,   5,  26],
       [  5, 228,   0,   1],
       [ 21,  77, 167,   3],
       [ 52,  21,   9,  71]], dtype=int64)
```

In [41]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_2['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                     ha="center", va="center",
                     color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_rbf, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_rbf, classes=class_names, normaliz
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[136  25   5  26]
 [  5 228   0   1]
 [ 21  77 167   3]
 [ 52  21   9  71]]
Normalized confusion matrix
[[0.71 0.13 0.03 0.14]
 [0.02 0.97 0.   0.  ]
 [0.08 0.29 0.62 0.01]
 [0.34 0.14 0.06 0.46]]
```






## taking kernel='linear'

In [42]:
```
from sklearn.svm import NuSVC
from sklearn.metrics import confusion_matrix
import numpy as np
clf_SVC_linear = NuSVC(kernel='linear').fit(X_train, y_train)
predictions_SVC_linear = clf_SVC_linear.predict(X_test)
print("Accuracy Score -> ",np.mean(predictions_SVC_linear==y_test))
print("Confusion Matrix")
confusion_matrix(y_test, predictions_SVC_linear)
```

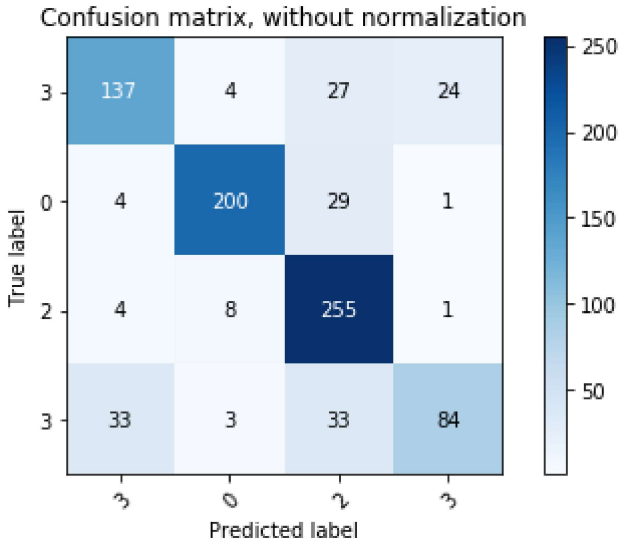
```
Accuracy Score ->  0.7981109799291618
Confusion Matrix
```

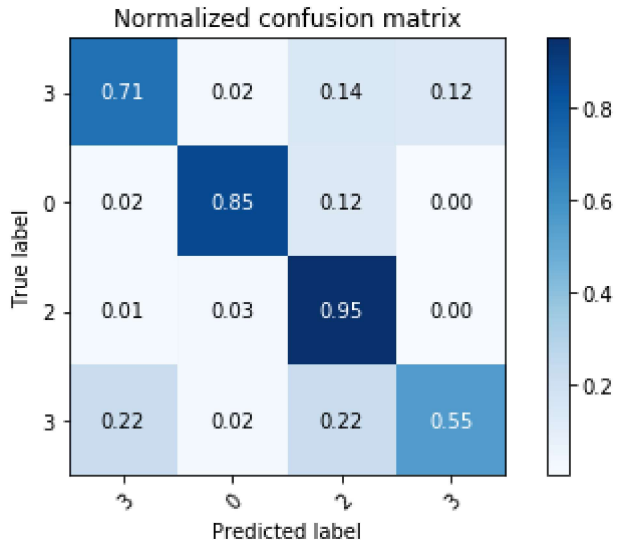
Out[42]:
```
array([[137,   4,  27,  24],
       [  4, 200,  29,   1],
       [  4,   8, 255,   1],
       [ 33,   3,  33,  84]], dtype=int64)
```

In [43]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_2['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_linear, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_linear, classes=class_names, norma
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[137   4  27  24]
 [  4 200  29   1]
 [  4   8 255   1]
 [ 33   3  33  84]]
Normalized confusion matrix
[[0.71 0.02 0.14 0.12]
 [0.02 0.85 0.12 0.  ]
 [0.01 0.03 0.95 0.  ]
 [0.22 0.02 0.22 0.55]]
```






## taking kernel='sigmoid' (sigmoid function)

In [44]:
```
from sklearn.svm import NuSVC
from sklearn.metrics import confusion_matrix
import numpy as np
clf_SVC_sigmoid = NuSVC(kernel='sigmoid').fit(X_train, y_train)
predictions_SVC_sigmoid = clf_SVC_sigmoid.predict(X_test)
print("Accuracy Score -> ",np.mean(predictions_SVC_sigmoid==y_test))
print("Confusion Matrix")
confusion_matrix(y_test, predictions_SVC_sigmoid)
```

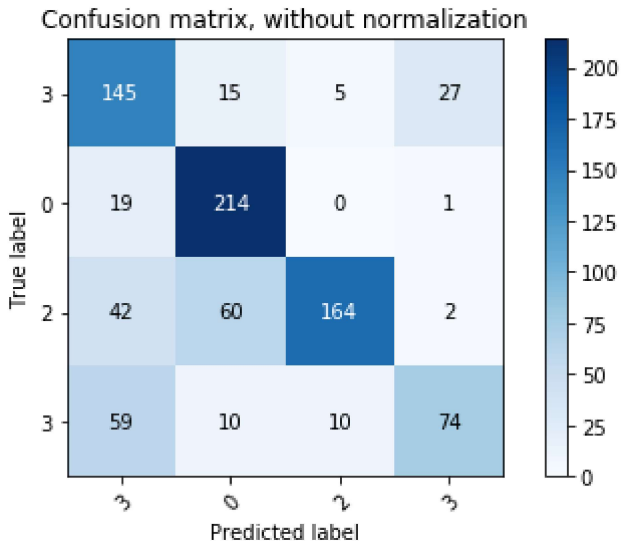
```
Accuracy Score ->  0.704840613931523
Confusion Matrix
```

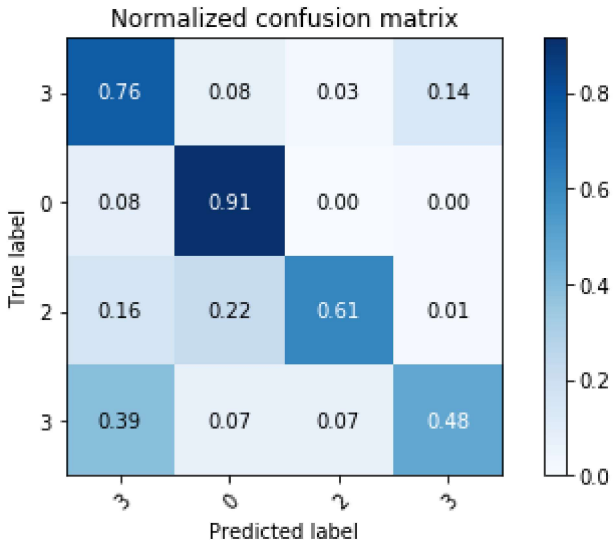
Out[44]:
```
array([[145,  15,   5,  27],
       [ 19, 214,   0,   1],
       [ 42,  60, 164,   2],
       [ 59,  10,  10,  74]], dtype=int64)
```

In [45]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_2['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_sigmoid, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_sigmoid, classes=class_names, norma
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[145  15   5  27]
 [ 19 214   0   1]
 [ 42  60 164   2]
 [ 59  10  10  74]]
Normalized confusion matrix
[[0.76 0.08 0.03 0.14]
 [0.08 0.91 0.   0.  ]
 [0.16 0.22 0.61 0.01]
 [0.39 0.07 0.07 0.48]]
```






## taking kernel='poly' (polynomial function)

In [46]:
```
from sklearn.svm import NuSVC
from sklearn.metrics import confusion_matrix
import numpy as np
clf_SVC_poly = NuSVC(kernel='poly').fit(X_train, y_train)
predictions_SVC_poly = clf_SVC_poly.predict(X_test)
print("Accuracy Score -> ",np.mean(predictions_SVC_poly==y_test))
print("Confusion Matrix")
confusion_matrix(y_test, predictions_SVC_poly)
```

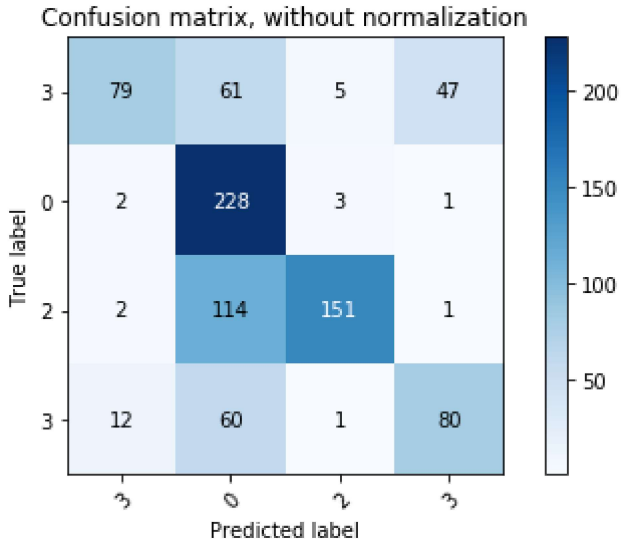
```
Accuracy Score ->  0.6351829988193625
Confusion Matrix
```

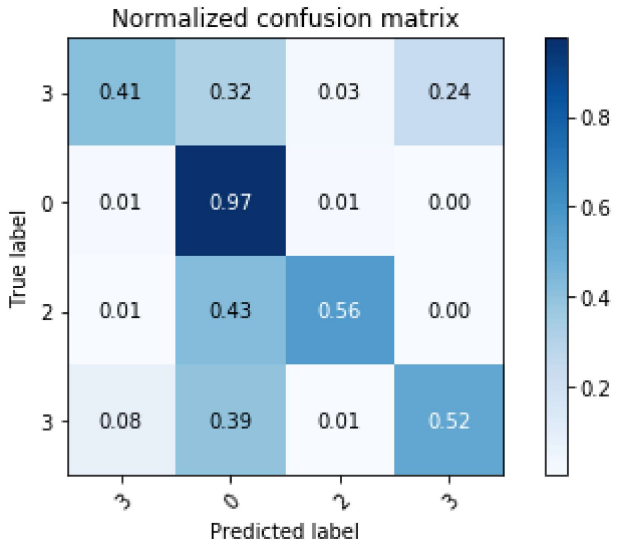
Out[46]:
```
array([[ 79,  61,   5,  47],
       [  2, 228,   3,   1],
       [  2, 114, 151,   1],
       [ 12,  60,   1,  80]], dtype=int64)
```

In [47]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_2['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_poly, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_poly, classes=class_names, normali
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[ 79  61   5  47]
 [  2 228   3   1]
 [  2 114 151   1]
 [ 12  60   1  80]]
Normalized confusion matrix
[[0.41 0.32 0.03 0.24]
 [0.01 0.97 0.01 0.  ]
 [0.01 0.43 0.56 0.  ]
 [0.08 0.39 0.01 0.52]]
```






## taking kernel='rbf' and taking penalty parameter C=1

In [48]:
```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
import numpy as np
clf_SVC_rbf_2 = SVC(C=1,kernel='rbf').fit(X_train, y_train)
predictions_SVC_rbf_2 = clf_SVC_rbf_2.predict(X_test)
print("Accuracy Score -> ",np.mean(predictions_SVC_rbf_2==y_test))
print("Confusion Matrix")
confusion_matrix(y_test, predictions_SVC_rbf_2)
```

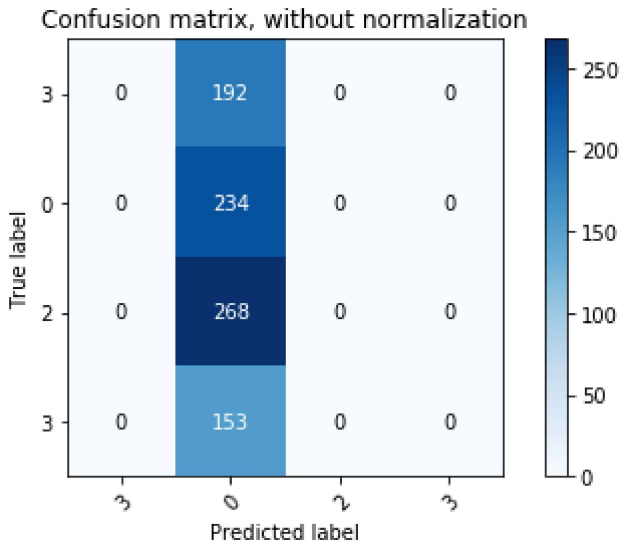
```
Accuracy Score ->  0.2762691853600944
Confusion Matrix
```

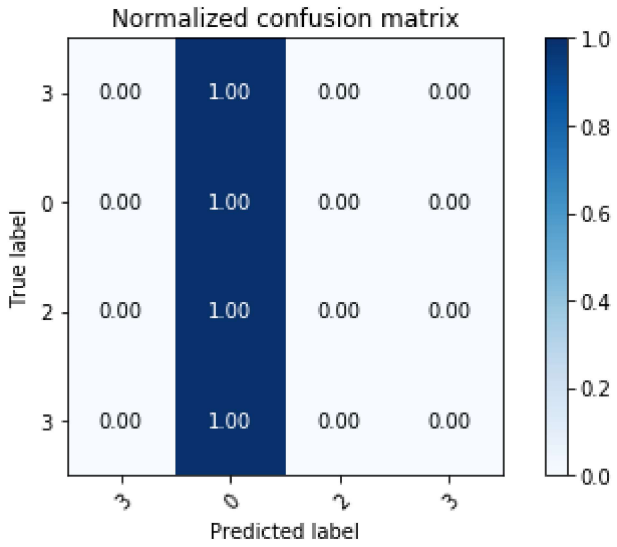
Out[48]:
```
array([[  0, 192,   0,   0],
       [  0, 234,   0,   0],
       [  0, 268,   0,   0],
       [  0, 153,   0,   0]], dtype=int64)
```

In [49]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_2['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                     ha="center", va="center",
                     color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_rbf_2, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_rbf_2, classes=class_names, normali
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[  0 192   0   0]
 [  0 234   0   0]
 [  0 268   0   0]
 [  0 153   0   0]]
Normalized confusion matrix
[[0. 1. 0. 0.]
 [0. 1. 0. 0.]
 [0. 1. 0. 0.]
 [0. 1. 0. 0.]]
```






## taking kernel='rbf' and taking penalty parameter C=1000

In [50]:
```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
import numpy as np
clf_SVC_rbf_2 = SVC(C=1000,kernel='rbf').fit(X_train, y_train)
predictions_SVC_rbf_2 = clf_SVC_rbf_2.predict(X_test)
print("Accuracy Score -> ",np.mean(predictions_SVC_rbf_2==y_test))
print("Confusion Matrix")
confusion_matrix(y_test, predictions_SVC_rbf_2)
```

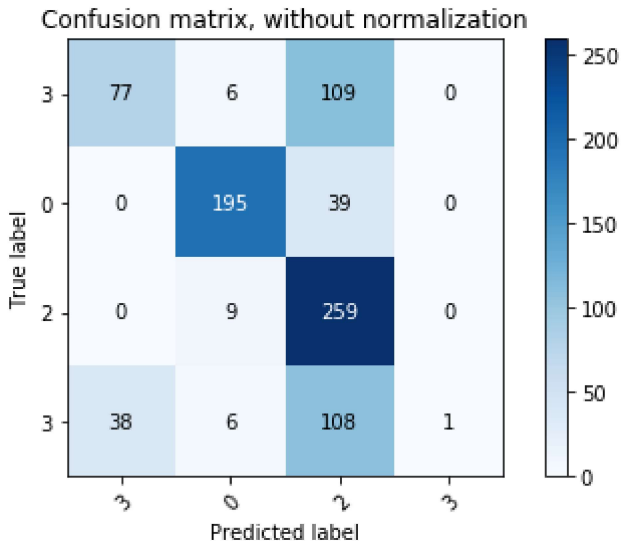
```
Accuracy Score ->  0.628099173553719
Confusion Matrix
```

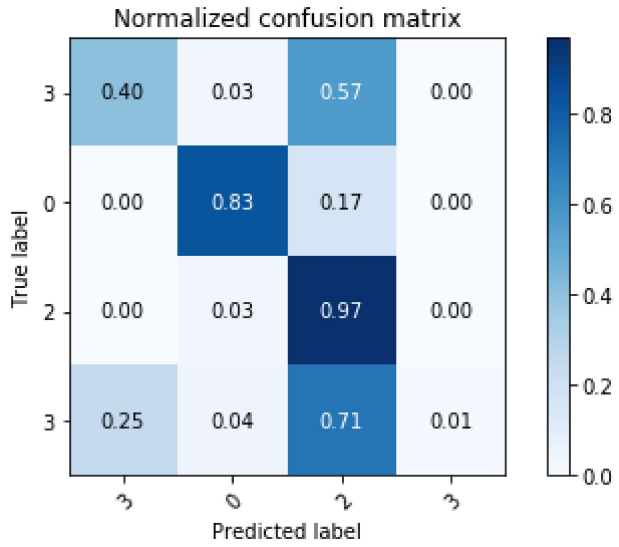
Out[50]:
```
array([[ 77,   6, 109,   0],
       [  0, 195,  39,   0],
       [  0,   9, 259,   0],
       [ 38,   6, 108,   1]], dtype=int64)
```

In [51]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_2['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                     ha="center", va="center",
                     color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_rbf_2, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_rbf_2, classes=class_names, normali
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[ 77   6 109   0]
 [  0 195  39   0]
 [  0   9 259   0]
 [ 38   6 108   1]]
Normalized confusion matrix
[[0.4  0.03 0.57 0.  ]
 [0.   0.83 0.17 0.  ]
 [0.   0.03 0.97 0.  ]
 [0.25 0.04 0.71 0.01]]
```






## taking kernel='rbf' and taking penalty parameter C=1000 and gamma='scale' which defines how much influence a single training example has

In [52]:
```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
import numpy as np
clf_SVC_rbf_2 = SVC(C=1000,kernel='rbf',gamma='scale').fit(X_train, y_train)
predictions_SVC_rbf_2 = clf_SVC_rbf_2.predict(X_test)
print("Accuracy Score -> ",np.mean(predictions_SVC_rbf_2==y_test))
print("Confusion Matrix")
confusion_matrix(y_test, predictions_SVC_rbf_2)
```

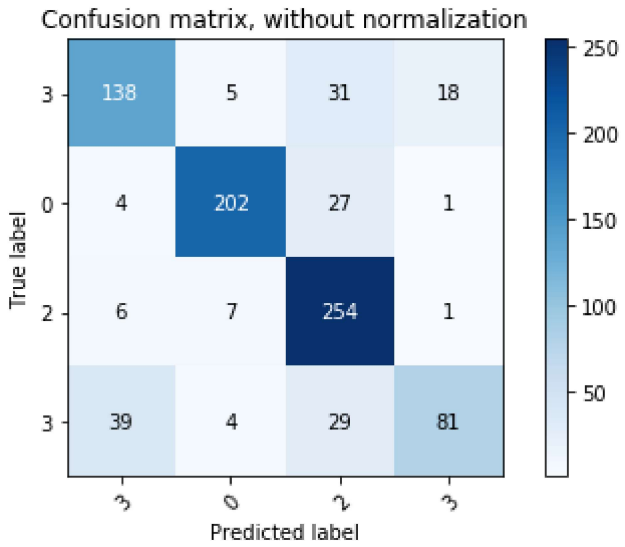
```
Accuracy Score ->  0.7969303423848878
Confusion Matrix
```

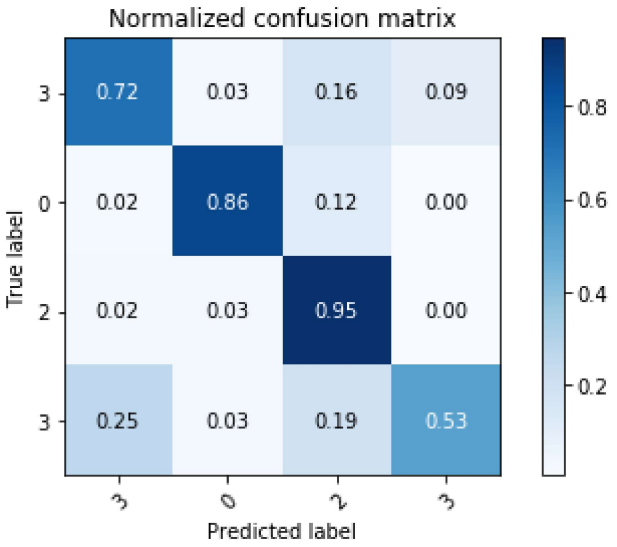
Out[52]:
```
array([[138,   5,  31,  18],
       [  4, 202,  27,   1],
       [  6,   7, 254,   1],
       [ 39,   4,  29,  81]], dtype=int64)
```

In [53]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_2['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_rbf_2, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_rbf_2, classes=class_names, normali
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[138   5  31  18]
 [  4 202  27   1]
 [  6   7 254   1]
 [ 39   4  29  81]]
Normalized confusion matrix
[[0.72 0.03 0.16 0.09]
 [0.02 0.86 0.12 0.  ]
 [0.02 0.03 0.95 0.  ]
 [0.25 0.03 0.19 0.53]]
```






By applying different hyperparameter to SVM model we find different results as above.

- When applying rbf(Gaussian radial basis function) and linear and we got 77.3% and 79.8% accuracy respectively.
- However, when we tried to apply polynomial and sigmoid kernel we got only 67.6% and 70.4% accuracy.
- From, these results it can be concluded that after applying linear and rbf kernels on the data in SVM models it gives more accuracy because it fits the data correctly and,the data are clustered in a way that they fit in linear models but while applying polynomial and sigmoid kernel it dosent fit the data as other kernels. The data given can easily be linearly seprable and trying to represent them in higher dimensional using polynomial and sigmoid kernel is not a good idea as it will decrease accuracy while separating datapoints in various classes in higher dimensions.
- The difference in accuracy is not high for linear and rbf kernels.
- Also along with kernel parameter I have also experimented with C and gamma hyperparameter of the SVM model. The C parameter is the penalty parameter which penalize datapoint for being on the wrong side. When the value of C is small such as C=1.0 which is default in SVM model, it allows to ignore points close to boundary and increase the margin, while a large value of C(C=1000) a large penalty is assigned to errors and margin errors. As C decreases, the hyperplane's orientation is also changed to maximize the margin for the rest of the data. So with C=1.0 I got only 27% accuracy while having C=1000 I got 64% accuracy because the model tries to classify data points correctly by penalizing the wrong datapoints. But this tends to overfit the training data.
- When we used gamma value as 'auto' which is default for SVM model, model showed only 27% accuracy while changing gamma to 'scale' the accuracy changed to 81% which was quite interesting. Gamma basically defines how much influence a single training example has, so when gamma is used as auto the value it takes is 1/n_features while when gamma is set to scale it take 1/n_features*X.var(), thus when we use scale for gamma the resulting model will behave similarly to a linear model with a set of hyperplanes that separate the centers of high density of any pair of two classes.

## d.) Perform POS on raw data and extract only nouns

In [54]:
```
#creating a new dataframe
df_without_cleaning= pd.DataFrame([newsgroups_train.data,newsgroups_train.target
df_without_cleaning=df_without_cleaning.T
df_without_cleaning.rename(columns={0: "news", 1: "categories"},inplace=True)
df_without_cleaning.head()
df_filtered=df_without_cleaning
```

In [55]:
```
#removing unnessary characters,letters and punctuations
df_without_cleaning["news"] = df_without_cleaning["news"].str.lower()
df_without_cleaning["news"] = df_without_cleaning["news"].str.replace(r"[^a-z]+"
df_without_cleaning['news']=df_without_cleaning['news'].str.replace('[{}]'.forma
df_without_cleaning['news'] = df_without_cleaning['news'].str.replace('\d+', '')
```

In [56]:
```
#tokenizing and tagging the tokenize
from nltk.tag import pos_tag
from nltk import word_tokenize
df_without_cleaning['news'] = df_without_cleaning.apply(lambda row: nltk.word_tol
df_without_cleaning['news'] = df_without_cleaning.apply(lambda row: nltk.pos_tag
df_without_cleaning['news'].head()
```

Out[56]:
```
0                                                    []
1    [(existance, NN), (i, NN), (just, RB), (though...
2    [(forwarded, VBN), (from, IN), (the, DT), (mar...
3    [(the, DT), (word, NN), (released, VBN), (is, ...
4    [(i, NN), (am, VBP), (posting, VBG), (this, DT...
Name: news, dtype: object
```

## d.) Extracting noun tags from the pos tagged words

In [57]:
```
noun_tags = ["NN","NNS","NP","NPS"]
noun_data = []
for i in range(0,df_without_cleaning['news'].size-1):
    noun = []
    for tags in df_without_cleaning['news'][i]:
        if tags[1] in noun_tags:
            noun.append(tags)
    noun_data.append(noun)
df_without_cleaning['Nouns_extracted_tags']=pd.Series(noun_data)
df_without_cleaning['Nouns_extracted_tags'].head()
```

Out[57]:
```
0                                                    []
1    [(existance, NN), (i, NN), (point, NN), (jesus...
2    [(mars, NNS), (project, NN), (mars, NNS), (sta...
3          [(word, NN), (cxourt, NN), (children, NNS)]
4    [(i, NN), (someone, NN), (respond, NN), (addre...
Name: Nouns_extracted_tags, dtype: object
```

## d.) Extracting noun words from the noun tagged words

In [58]:
```
noun_data = []
for i in range(0,df_without_cleaning['Nouns_extracted_tags'].size-1):
    nouns = []
    for tags in df_without_cleaning['Nouns_extracted_tags'][i]:
        nouns.append(tags[0])
    nouns=' '.join(nouns)
    noun_data.append(nouns)
df_without_cleaning['Nouns_extracted_words']=pd.Series(noun_data)
df_without_cleaning['Nouns_extracted_words'].head()
```

Out[58]:
```
0
1    existance i point jesus guys jesus exist cuz j...
2    mars project mars status report flight sequenc...
3                                  word cxourt children
4    i someone respond address please duplication m...
Name: Nouns_extracted_words, dtype: object
```

## d.) Obtain a bag-of-words tf-idf weighted vector representation using only the nouns

In [59]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
corpus=(df_without_cleaning['Nouns_extracted_words'].values.astype('U'))
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(corpus)
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
X_train_tfidf.shape
```

Out[59]:
```
(3387, 19382)
```

In [60]:
```
#splitting data for train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_train_tfidf, df_without_cl
```

## d.) Training MultiNomialNB

In [34]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
import numpy as np
clf = MultinomialNB().fit(X_train, y_train)
predictions_NB = clf.predict(X_test)
print("Naive Bayes Accuracy Score -> ",np.mean(predictions_NB==y_test))
print("Confusion Matrix")
confusion_matrix(y_test, predictions_NB)
```

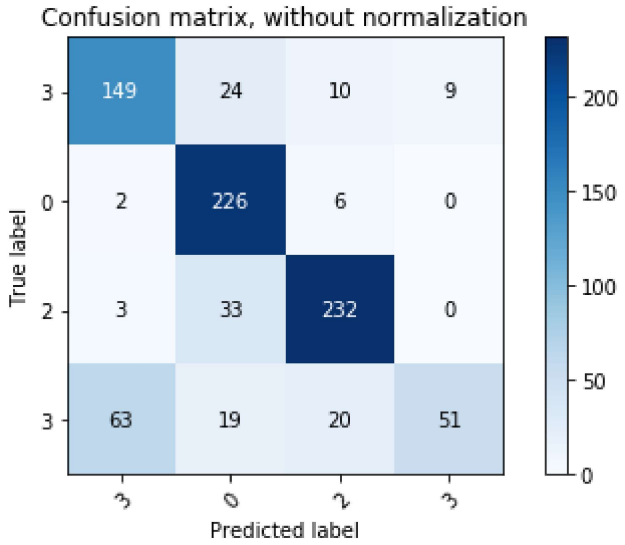
```
Naive Bayes Accuracy Score ->  0.7768595041322314
Confusion Matrix
```

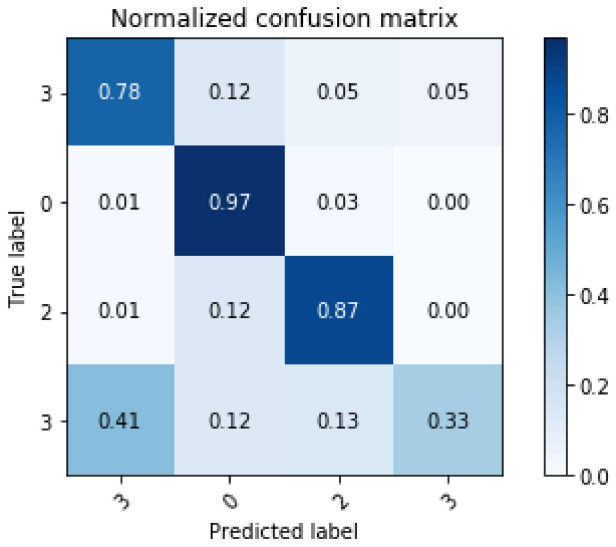
Out[34]:
```
array([[149,  24,  10,   9],
       [  2, 226,   6,   0],
       [  3,  33, 232,   0],
       [ 63,  19,  20,  51]], dtype=int64)
```

In [39]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_without_cleaning['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predictions_NB, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predictions_NB, classes=class_names, normalize=Tru
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[149  24  10   9]
 [  2 226   6   0]
 [  3  33 232   0]
 [ 63  19  20  51]]
Normalized confusion matrix
[[0.78 0.12 0.05 0.05]
 [0.01 0.97 0.03 0.  ]
 [0.01 0.12 0.87 0.  ]
 [0.41 0.12 0.13 0.33]]
```






## d.) Training SVM without kernel

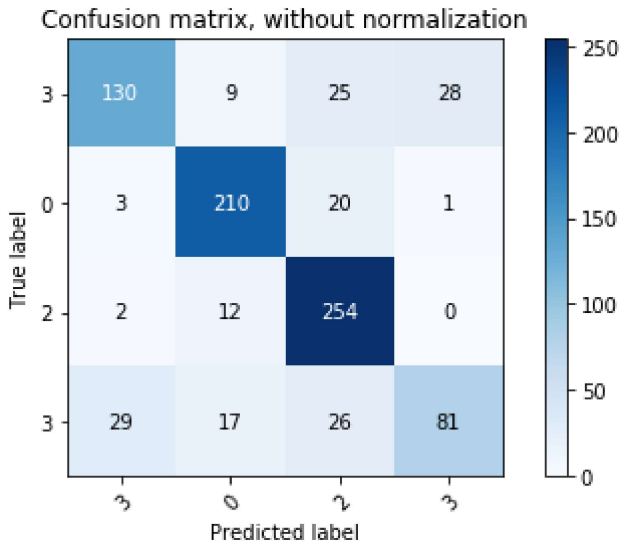
In [36]:
```
from sklearn.linear_model import SGDClassifier
import numpy as np
text_clf_svm=SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3,  random_state
predicted_svm = text_clf_svm.predict(X_test)
print("SVM Accuracy Score -> ",np.mean(predicted_svm == y_test))
```

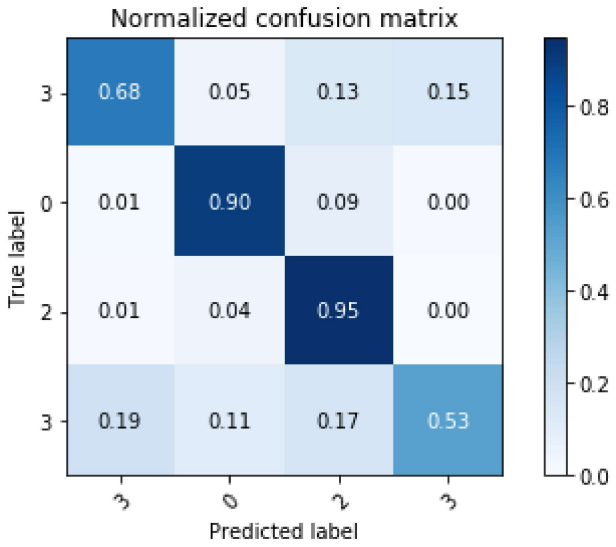
```
D:\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:166:
FutureWarning: max_iter and tol parameters have been added in SGDClassifier in
0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol i
s not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter wil
l be 1000, and default tol will be 1e-3.
  FutureWarning)

SVM Accuracy Score ->  0.7969303423848878
```

In [38]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_without_cleaning['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predicted_svm, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predicted_svm, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[130   9  25  28]
 [  3 210  20   1]
 [  2  12 254   0]
 [ 29  17  26  81]]
Normalized confusion matrix
[[0.68 0.05 0.13 0.15]
 [0.01 0.9  0.09 0.  ]
 [0.01 0.04 0.95 0.  ]
 [0.19 0.11 0.17 0.53]]
```






## d.) Training SVM using different kernels

## taking kernel='linear'

In [61]:
```
from sklearn.svm import NuSVC
from sklearn.metrics import confusion_matrix
import numpy as np
clf_SVC_linear =NuSVC(kernel='linear').fit(X_train, y_train)
predictions_SVC_linear = clf_SVC_linear.predict(X_test)
print("SVM Accuracy Score -> ",np.mean(predictions_SVC_linear==y_test))
print("Confusion Matrix")
confusion_matrix(y_test, predictions_SVC_linear)
```

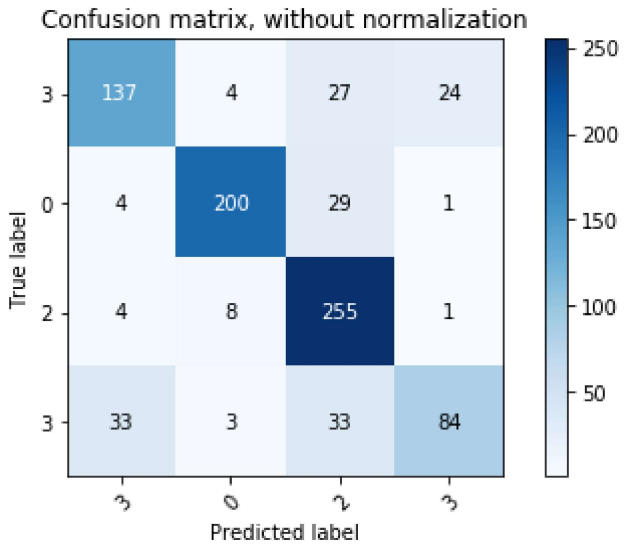
```
SVM Accuracy Score ->  0.7981109799291618
Confusion Matrix
```

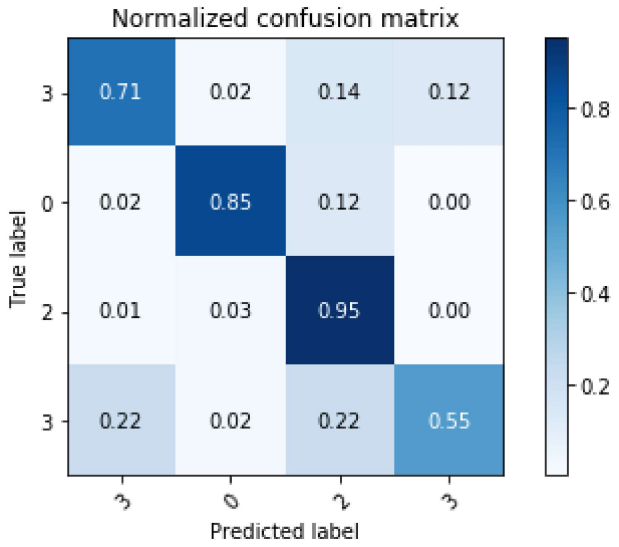
Out[61]:
```
array([[137,   4,  27,  24],
       [  4, 200,  29,   1],
       [  4,   8, 255,   1],
       [ 33,   3,  33,  84]], dtype=int64)
```

In [62]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_without_cleaning['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                     ha="center", va="center",
                     color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_linear, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_linear, classes=class_names, norma
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[137   4  27  24]
 [  4 200  29   1]
 [  4   8 255   1]
 [ 33   3  33  84]]
Normalized confusion matrix
[[0.71 0.02 0.14 0.12]
 [0.02 0.85 0.12 0.  ]
 [0.01 0.03 0.95 0.  ]
 [0.22 0.02 0.22 0.55]]
```






## taking kernel='rbf' (Gaussian radial basis function)

In [63]:
```
from sklearn.svm import NuSVC
from sklearn.metrics import confusion_matrix
import numpy as np
clf_SVC_rbf = NuSVC(kernel='rbf').fit(X_train, y_train)
predictions_SVC_rbf = clf_SVC_rbf.predict(X_test)
print("Accuracy Score -> ",np.mean(predictions_SVC_rbf==y_test))
print("Confusion Matrix")
confusion_matrix(y_test, predictions_SVC_rbf)
```

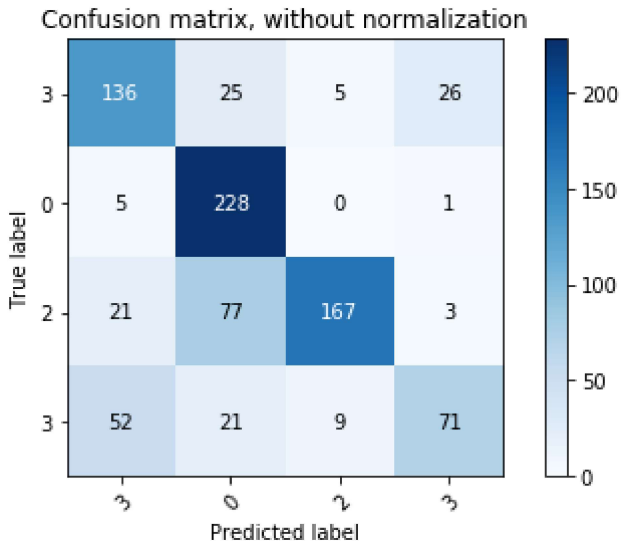
```
Accuracy Score ->  0.7107438016528925
Confusion Matrix
```

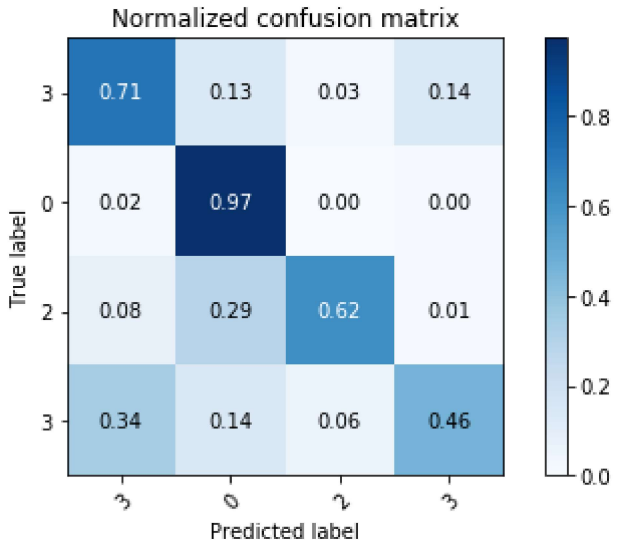
Out[63]:
```
array([[136,  25,   5,  26],
       [  5, 228,   0,   1],
       [ 21,  77, 167,   3],
       [ 52,  21,   9,  71]], dtype=int64)
```

In [64]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_without_cleaning['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                     ha="center", va="center",
                     color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_rbf, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_rbf, classes=class_names, normaliz
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[136  25   5  26]
 [  5 228   0   1]
 [ 21  77 167   3]
 [ 52  21   9  71]]
Normalized confusion matrix
[[0.71 0.13 0.03 0.14]
 [0.02 0.97 0.   0.  ]
 [0.08 0.29 0.62 0.01]
 [0.34 0.14 0.06 0.46]]
```






## taking kernel='poly'

In [65]:
```
from sklearn.svm import NuSVC
from sklearn.metrics import confusion_matrix
import numpy as np
clf_SVC_poly = NuSVC(kernel='poly').fit(X_train, y_train)
predictions_SVC_poly = clf_SVC_poly.predict(X_test)
print("Accuracy Score -> ",np.mean(predictions_SVC_poly==y_test))
print("Confusion Matrix")
confusion_matrix(y_test, predictions_SVC_poly)
```

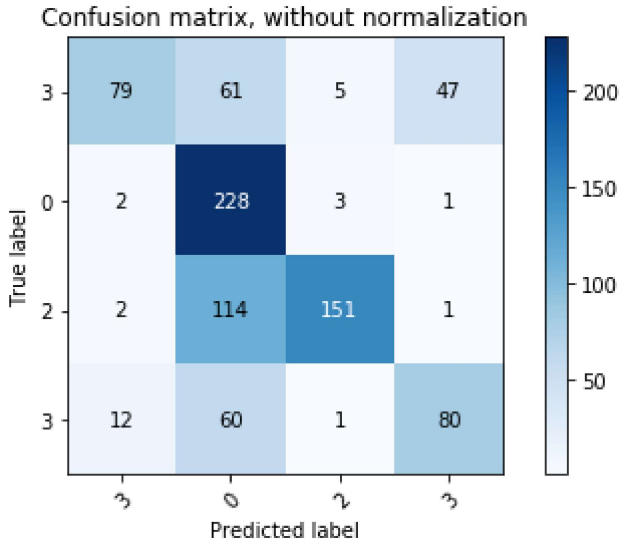
```
Accuracy Score ->  0.6351829988193625
Confusion Matrix
```

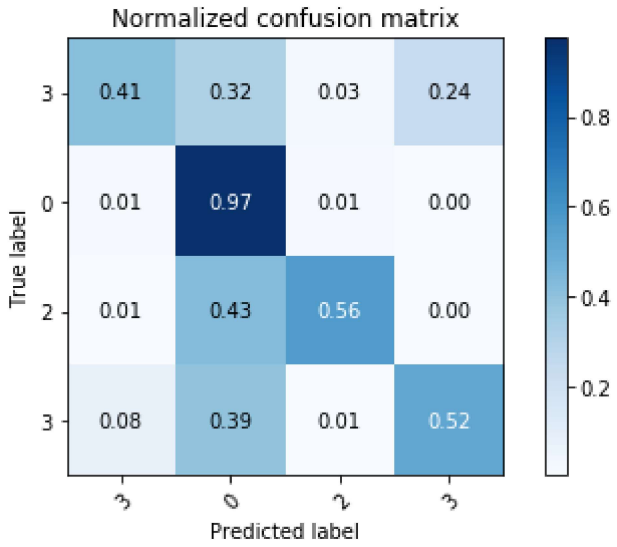
Out[65]:
```
array([[ 79,  61,   5,  47],
       [  2, 228,   3,   1],
       [  2, 114, 151,   1],
       [ 12,  60,   1,  80]], dtype=int64)
```

In [66]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_without_cleaning['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_poly, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_poly, classes=class_names, normali
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[ 79  61   5  47]
 [  2 228   3   1]
 [  2 114 151   1]
 [ 12  60   1  80]]
Normalized confusion matrix
[[0.41 0.32 0.03 0.24]
 [0.01 0.97 0.01 0.  ]
 [0.01 0.43 0.56 0.  ]
 [0.08 0.39 0.01 0.52]]
```






## taking kernel='sigmoid'

In [68]:
```
from sklearn.svm import NuSVC
from sklearn.metrics import confusion_matrix
import numpy as np
clf_SVC_sigmoid = NuSVC(kernel='sigmoid').fit(X_train, y_train)
predictions_SVC_sigmoid = clf_SVC_sigmoid.predict(X_test)
print("Accuracy Score -> ",np.mean(predictions_SVC_sigmoid==y_test))
print("Confusion Matrix")
confusion_matrix(y_test, predictions_SVC_sigmoid)
```

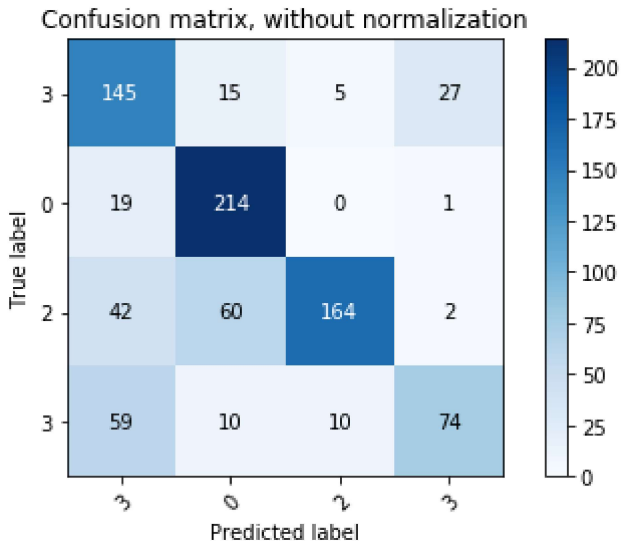
```
Accuracy Score ->  0.704840613931523
Confusion Matrix
```

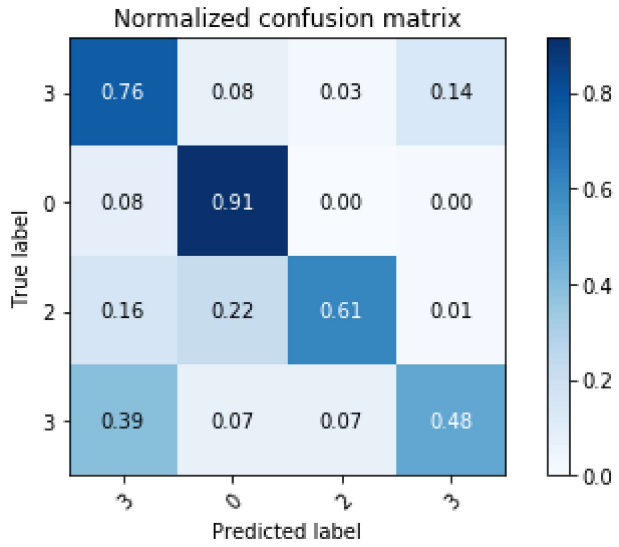
Out[68]:
```
array([[145,  15,   5,  27],
       [ 19, 214,   0,   1],
       [ 42,  60, 164,   2],
       [ 59,  10,  10,  74]], dtype=int64)
```

In [69]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
class_names = df_without_cleaning['categories']
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
```

```
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_sigmoid, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(y_test, predictions_SVC_sigmoid, classes=class_names, norma
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[145  15   5  27]
 [ 19 214   0   1]
 [ 42  60 164   2]
 [ 59  10  10  74]]
Normalized confusion matrix
[[0.76 0.08 0.03 0.14]
 [0.08 0.91 0.   0.  ]
 [0.16 0.22 0.61 0.01]
 [0.39 0.07 0.07 0.48]]
```






## taking kernel='rbf' and taking penalty parameter C=1000 and gamma='scale' which defines how much influence a single training example has

In [72]:
```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
import numpy as np
clf_SVC_rbf = SVC(kernel='rbf',C=1000,gamma='scale').fit(X_train, y_train)
predictions_SVC_rbf = clf_SVC_rbf.predict(X_test)
print("Accuracy Score -> ",np.mean(predictions_SVC_rbf==y_test))
print("Confusion Matrix")
confusion_matrix(y_test, predictions_SVC_rbf)
```

```
Accuracy Score ->  0.7969303423848878
Confusion Matrix
```

Out[72]:
```
array([[138,   5,  31,  18],
       [  4, 202,  27,   1],
       [  6,   7, 254,   1],
       [ 39,   4,  29,  81]], dtype=int64)
```

- The accuracy for both MultinomailNB and SVM has increased when considered only nouns as the vocabulary.
- Also for every kernel and other hyperparameter the accuracy was increased than the models of c.)
- The vocabulary generated using the countvectorizer decreased as we have only considered nouns but the decrease was not a huge but this new vocabulary of nouns have increased the accuracy of the model because it did not contained any other unncessary types such as prepositions, artices which skews the model results.

## References

- The Effects of Hyperparameters in SVM. (2019). Retrieved 16 July 2019, from https://yunhaocsblog.wordpress.com/2014/07/27/the-effects-of-hyperparameters-in-svm/ (https://yunhaocsblog.wordpress.com/2014/07/27/the-effects-of-hyperparameters-in-svm/)
- RBF SVM parameters — scikit-learn 0.21.2 documentation. (2019). Retrieved 16 July 2019, from https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html (https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html)
- 1.4. Support Vector Machines — scikit-learn 0.21.2 documentation. (2019). Retrieved 16 July 2019, from https://scikit-learn.org/stable/modules/svm.html#svm-kernels (https://scikit-learn.org/stable/modules/svm.html#svm-kernels)
- data, N., Coman, H., & Agarwal, P. (2019). Naive Bayes vs. SVM for classifying text data. Retrieved 16 July 2019, from https://stackoverflow.com/questions/35360081/naive-bayes-vs-svm-for-classifying-text-data (https://stackoverflow.com/questions/35360081/naive-bayes-vs-svm-for-classifying-text-data)
- sklearn.svm.SVC — scikit-learn 0.21.2 documentation. (2019). Retrieved 16 July 2019, from https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC)
- sklearn.linear_model.SGDClassifier — scikit-learn 0.21.2 documentation. (2019). Retrieved 16 July 2019, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_mo (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_mo
- sklearn.svm.SVC — scikit-learn 0.21.2 documentation. (2019). Retrieved 16 July 2019, from https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC)
- sklearn.svm.NuSVC — scikit-learn 0.21.2 documentation. (2019). Retrieved 16 July 2019, from https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html (https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html)
- Google Colaboratory. (2019). Retrieved 16 July 2019, from https://colab.research.google.com/drive/17LMCbDOnny8h1KTqqoX3smy6US6z1vki (https://colab.research.google.com/drive/17LMCbDOnny8h1KTqqoX3smy6US6z1vki)
- (2019). Retrieved 16 July 2019, from https://dal.brightspace.com/d2l/le/content/98749/viewContent/1340763/View (https://dal.brightspace.com/d2l/le/content/98749/viewContent/1340763/View)
- M. Learning and U. Engineers, "Ultimate guide to deal with Text Data (using Python) - for Data Scientists and Engineers", Analytics Vidhya, 2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/ (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/). [Accessed: 16- Jul- 2019].
- "Have messy text data? Clean it with simple lambda functions.", Medium, 2019. [Online]. Available: https://medium.com/@chaimgluck1/have-messy-text-data-clean-it-with-simple-lambda-functions-645918fcc2fc (https://medium.com/@chaimgluck1/have-messy-text-data-clean-it-with-simple-lambda-functions-645918fcc2fc). [Accessed: 16- Jul- 2019].