Assignment 2 : CSCI 5901

Submitted by:

1. Gaganpreet Singh (B00819217)
2. Shruthi Kalasapura Ramesh (B00822766)

Question 1) Collection and Extraction

In [1]:
```
#Importing Libraries and dataset
from sklearn.datasets import fetch_20newsgroups
from pprint import pprint
import pandas as pd
%pprint
```

```
Pretty printing has been turned OFF
```

Exploring the dataset.

In [2]:
```
cat = ['alt.atheism',
       'talk.religion.misc',
       'comp.graphics',
       'sci.space']

allArticles = fetch_20newsgroups(subset='all',
                                 categories=cat)

# # # # # # # # # # # # # # # # # # # # # # # # # # # #
# #
# # Exploring and Analysing the dataset
# # Uncomment the below Code to understand the Dataset
# #
# # # # # # # # # # # # # # # # # # # # # # # # # # # #
# print("Lets see what does article have:")
# pprint(list(allArticles))
# print("************************************")
# print("What is in data?")
# pprint((allArticles.data[0]))
# print("************************************")
# print("What is in filenames?")
# pprint((allArticles.filenames[0]))
# print("************************************")
# print("What is in target_names?")
# pprint((allArticles.target_names))
# print("************************************")
# print("What is in target for 1st Article?")
# pprint((allArticles.target_names[allArticles.target[0]]))
# print("************************************")
# print("What is in DESCR for 1st Article?")
# print((allArticles.DESCR))
# # # # # # # # # # # # # # # # # # # # # # # # # # # #
```

Remove articles that are not in english language.

In [3]:

```
# Removing articles that are not in English Language
# !pip install langdetect
from langdetect import detect_langs

def isEnglish(string):
    res = detect_langs(string)
    for item in res:
        if item.lang == "en":
            return 1
    print(string)
    return 0

engArticles = []
for document in allArticles.data:
    if isEnglish(document):
        engArticles.append(document)
```

From: lochem@fys.ruu.nl (Gert-Jan van Lochem)
Subject: Dutch: symposium compacte objecten
Summary: U wordt uitgenodigd voor het symposium compacte objecten 26-4-93
Keywords: compacte objecten, symposium
Organization: Physics Department, University of Utrecht, The Netherlands
Lines: 122

Sterrenkundig symposium 'Compacte Objecten'
op 26 april 1993

In het jaar 1643, zeven jaar na de oprichting van de Universiteit van Utrecht, benoemde de universiteit haar eerste sterrenkundige waarnemer. Hiermee ontstond de tweede universiteitssterrenwacht ter wereld. Aert Jansz, de eerste waarnemer, en zijn opvolgers voerden de Utrechtse sterrenkunde in de daaropvolgende jaren, decennia en eeuwen naar de voorhoede van het astronomisch onderzoek. Dit jaar is het 350 jaar geleden dat deze historische benoeming plaatsvond.

De huidige generatie Utrechtse sterrenkundigen en studenten sterrenkunde, verenigd in het Sterrekundig Instituut Utrecht, vieren de benoeming van hun 'oervader' middels een breed scala aan feestelijke activiteiten. Zo is er voor scholieren een planetenproject, programmeert de Studium Generale een aantal voordrachten met een sterrenkundig thema en wordt op de Dies Natalis aan een astronoom een eredoctoraat uitgereikt. Er staat echter meer op stapel.

Studenten natuur- en sterrenkunde kunnen op 26 april aan een sterrenkundesymposium deelnemen. De onderwerpen van het symposium zijn opgebouwd rond een van de zwaartepunten van het huidige Utrechtse onderzoek: het onderzoek aan de zogeheten 'compacte objecten', de eindstadia in de evolutie van sterren. Bij de samenstelling van het programma is getracht de deelnemer een zo aktueel en breed mogelijk beeld te geven van de stand van zaken in het onderzoek aan deze eindstadia. In de eerste, inleidende lezing zal dagvoorzitter prof. Lamers een beknopt overzicht geven van de evolutie van zware sterren, waarna de zeven overige sprekers in lezingen van telkens een half uur nader op de specifieke evolutionaire eindprodukten zullen ingaan. Na afloop van elke lezing is er gelegenheid tot het stellen van vragen. Het dagprogramma staat afgedrukt op een apart vel.
Het niveau van de lezingen is afgestemd op tweedejaars studenten natuur- en sterrenkunde. OOK ANDERE BELANGSTELLENDEN ZIJN VAN HARTE WELKOM!

Tijdens de lezing van prof. Kuijpers zullen, als alles goed gaat, de veertien radioteleskopen van de Radiosterrenwacht Westerbork worden ingezet om via een directe verbinding tussen het heelal, Westerbork en Utrecht het zwakke radiosignaal van een snel roterende kosmische vuurtoren, een zogeheten pulsar, in de symposiumzaal door te geven en te audiovisualiseren.
Prof. Kuijpers zal de binnenkomende signalen (elkaar snel

opvolgende scherp gepiekte pulsen radiostraling) bespreken en trachten te verklaren.
Het slagen van dit unieke experiment staat en valt met de technische haalbaarheid ervan. De op te vangen signalen zijn namelijk zo zwak, dat pas na een waarnemingsperiode van 10 miljoen jaar genoeg energie is opgevangen om een lamp van 30 Watt een seconde te laten branden! Tijdens het symposium zal er niet zo lang gewacht hoeven te worden: de hedendaagse technologie stelt ons in staat live het heelal te beluisteren.

Deelname aan het symposium kost f 4,- (exclusief lunch) en f 16,- (inclusief lunch). Inschrijving geschiedt door het verschuldigde bedrag over te maken op ABN-AMRO rekening 44.46.97.713 t.n.v. stichting 350 JUS. Het gironummer van de ABN-AMRO bank Utrecht is 2900. Bij de inschrijving dient te worden aangegeven of men lid is van de NNV. Na inschrijving wordt de symposiummap toegestuurd. Bij inschrijving na 31 maart vervalt de mogelijkheid een lunch te reserveren.

Het symposium vindt plaats in Transitorium I, Universiteit Utrecht.

Voor meer informatie over het symposium kan men terecht bij Henrik Spoon, p/a S.R.O.N., Sorbonnelaan 2, 3584 CA Utrecht. Tel.: 030-535722. E-mail: henriks@sron.ruu.nl.

******* DAGPROGRAMMA ***************************************

9:30 ONTVANGST MET KOFFIE & THEE

10:00 Opening
Prof. dr. H.J.G.L.M. Lamers (Utrecht)

10:10 Dubbelster evolutie
Prof. dr. H.J.G.L.M. Lamers

10:25 Radiopulsars
Prof. dr. J.M.E. Kuijpers (Utrecht)

11:00 Pulsars in dubbelster systemen
Prof. dr. F. Verbunt (Utrecht)

11:50 Massa & straal van neutronensterren
Prof. dr. J. van Paradijs (Amsterdam)

12:25 Theorie van accretieschijven
Drs. R.F. van Oss (Utrecht)

13:00 LUNCH

14:00 Hoe zien accretieschijven er werkelijk uit?
Dr. R.G.M. Rutten (Amsterdam)

14:35 Snelle fluktuaties bij accretie op neutronensterren

```
         en zwarte gaten
            Dr. M. van der Klis (Amsterdam)

15:10    THEE & KOFFIE

15:30    Zwarte gaten: knippen en plakken met ruimte en tijd
            Prof. dr. V. Icke (leiden)

16:05    afsluiting

16:25    BORREL

--
Gert-Jan van Lochem             \\         "What is it?"
Fysische informatica             \\        "Something blue"
Universiteit Utrecht              \\       "Shapes, I need shapes!"
030-532803                         \\                       - HHGG -
```

In [4]:
```python
print("Number of Articles Dropped:",len(allArticles.data)-len(engArticles))
```

```
Number of Articles Dropped: 1
```

Checking if there are any duplicate articles and deleting them.

In [5]:
```python
# to remove any duplicate articles, if they exist
engArticles = list(dict.fromkeys(engArticles))
```

a) Tokenize the corpus and perform part-of-speech tagging

Tokenizing the articles into list of tokens.

In [6]:
```python
# !pip install gensim
import gensim
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS

# Convert each document into a list of lowercase tokens,
# ignoring tokens that are too short.
tokens=[]
for document in engArticles:
    tokens.append(gensim.utils.simple_preprocess(document,min_len=2))

len(tokens)
```

Out[6]:
```
3386
```

Performing Lemmatization before POS tagging.

In [7]:
```
from textblob import TextBlob, Word
import nltk
nltk.download('wordnet')
lemmatizedTokens = []
for line in tokens:
    lemmatizedwords = []
    for word in line:
        w = Word(word)

        lemmatizedwords.append(w.lemmatize())
    lemmatizedTokens.append(list(lemmatizedwords))
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/gaganpree99/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [8]:
```
#Verify that we lemmatized all documents
len(lemmatizedTokens)
```

Out[8]: 3386

POS Tagging

In [9]:
```
nltk.download('averaged_perceptron_tagger')
tags = []
for doc in lemmatizedTokens:
  tags.append(nltk.pos_tag(doc))
print(len(tags))
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/gaganpree99/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!

3386
```

In [10]:

```
# confirming tagging by checking 1st document
tags[0]
```

```
Out[10]: [('from', 'IN'), ('healta', 'JJ'), ('saturn', 'NN'), ('wwc', 'NN'), ('e
du', 'NN'), ('tammy', 'NN'), ('healy', 'NN'), ('subject', 'JJ'), ('re',
'NN'), ('who', 'WP'), ('are', 'VBP'), ('we', 'PRP'), ('to', 'TO'), ('ju
dge', 'VB'), ('bobby', 'NN'), ('line', 'NN'), ('organization', 'NN'),
('walla', 'NN'), ('walla', 'NN'), ('college', 'NN'), ('line', 'NN'),
('in', 'IN'), ('article', 'NN'), ('apr', 'NN'), ('ultb', 'JJ'), ('isc',
'NN'), ('rit', 'NN'), ('edu', 'NN'), ('snm', 'NN'), ('ultb', 'JJ'), ('i
sc', 'NN'), ('rit', 'NN'), ('edu', 'NN'), ('mozumder', 'NN'), ('write
s', 'VBZ'), ('from', 'IN'), ('snm', 'NN'), ('ultb', 'JJ'), ('isc', 'N
N'), ('rit', 'NN'), ('edu', 'NN'), ('mozumder', 'NN'), ('subject', 'J
J'), ('re', 'NN'), ('who', 'WP'), ('are', 'VBP'), ('we', 'PRP'), ('to',
'TO'), ('judge', 'VB'), ('bobby', 'NN'), ('date', 'NN'), ('wed', 'VB
D'), ('apr', 'JJ'), ('gmt', 'NN'), ('in', 'IN'), ('article', 'NN'), ('h
ealta', 'NN'), ('saturn', 'NN'), ('wwc', 'NN'), ('edu', 'NN'), ('healt
a', 'NN'), ('saturn', 'VBP'), ('wwc', 'JJ'), ('edu', 'NN'), ('tammy',
'NN'), ('healy', 'NN'), ('writes', 'VBZ'), ('bobby', 'RB'), ('would',
'MD'), ('like', 'VB'), ('to', 'TO'), ('take', 'VB'), ('the', 'DT'), ('l
iberty', 'NN'), ('to', 'TO'), ('quote', 'VB'), ('from', 'IN'), ('christ
ian', 'JJ'), ('writer', 'NN'), ('named', 'VBN'), ('ellen', 'IN'), ('whi
te', 'JJ'), ('hope', 'NN'), ('that', 'IN'), ('what', 'WP'), ('she', 'PR
P'), ('said', 'VBD'), ('will', 'MD'), ('help', 'VB'), ('you', 'PRP'),
('to', 'TO'), ('edit', 'VB'), ('your', 'PRP$'), ('remark', 'NN'), ('i
n', 'IN'), ('this', 'DT'), ('group', 'NN'), ('in', 'IN'), ('the', 'D
T'), ('future', 'NN'), ('do', 'VBP'), ('not', 'RB'), ('set', 'VB'), ('y
ourself', 'PRP'), ('a', 'DT'), ('standard', 'NN'), ('do', 'VBP'), ('no
t', 'RB'), ('make', 'VB'), ('your', 'PRP$'), ('opinion', 'NN'), ('you
r', 'PRP$'), ('view', 'NN'), ('of', 'IN'), ('duty', 'NN'), ('your', 'PR
P$'), ('interpretation', 'NN'), ('of', 'IN'), ('scripture', 'NN'), ('cr
iterion', 'NN'), ('for', 'IN'), ('others', 'NNS'), ('and', 'CC'), ('i
n', 'IN'), ('your', 'PRP$'), ('heart', 'NN'), ('condemn', 'VB'), ('the
m', 'PRP'), ('if', 'IN'), ('they', 'PRP'), ('do', 'VBP'), ('not', 'R
B'), ('come', 'VB'), ('up', 'RP'), ('to', 'TO'), ('your', 'PRP$'), ('id
eal', 'JJ'), ('thought', 'JJ'), ('fromthe', 'NN'), ('mount', 'NN'), ('o
f', 'IN'), ('blessing', 'VBG'), ('hope', 'NN'), ('quoting', 'VBG'), ('t
his', 'DT'), ('doesn', 'NN'), ('make', 'VBP'), ('the', 'DT'), ('atheis
t', 'NN'), ('gag', 'NN'), ('but', 'CC'), ('think', 'VBP'), ('ellen', 'V
BN'), ('white', 'JJ'), ('put', 'VBD'), ('it', 'PRP'), ('better', 'JJ
R'), ('than', 'IN'), ('could', 'MD'), ('tammy', 'VB'), ('point', 'VB'),
('peace', 'NN'), ('bobby', 'NN'), ('mozumder', 'NN'), ('my', 'PRP$'),
('point', 'NN'), ('is', 'VBZ'), ('that', 'IN'), ('you', 'PRP'), ('set',
'VBP'), ('up', 'RP'), ('your', 'PRP$'), ('view', 'NN'), ('a', 'DT'),
('the', 'DT'), ('only', 'JJ'), ('way', 'NN'), ('to', 'TO'), ('believe',
'VB'), ('saying', 'VBG'), ('that', 'IN'), ('all', 'DT'), ('eveil', 'N
N'), ('in', 'IN'), ('this', 'DT'), ('world', 'NN'), ('is', 'VBZ'), ('ca
used', 'VBN'), ('by', 'IN'), ('atheism', 'NN'), ('is', 'VBZ'), ('ridicu
lous', 'JJ'), ('and', 'CC'), ('to', 'TO'), ('dialogue', 'VB'), ('in',
'IN'), ('this', 'DT'), ('newsgroups', 'NNS'), ('see', 'VBP'), ('in', 'I
N'), ('your', 'PRP$'), ('post', 'NN'), ('spirit', 'NN'), ('of', 'IN'),
('condemnation', 'NN'), ('of', 'IN'), ('the', 'DT'), ('atheist', 'NN'),
('in', 'IN'), ('this', 'DT'), ('newsgroup', 'NN'), ('bacause', 'IN'),
('they', 'PRP'), ('don', 'VBP'), ('believe', 'VBP'), ('exactly', 'RB'),
('a', 'DT'), ('you', 'PRP'), ('do', 'VBP'), ('if', 'IN'), ('you', 'PR
P'), ('re', 'VBP'), ('here', 'RB'), ('to', 'TO'), ('try', 'VB'), ('to',
'TO'), ('convert', 'VB'), ('the', 'DT'), ('atheist', 'NN'), ('here', 'R
B'), ('you', 'PRP'), ('re', 'VBP'), ('failing', 'VBG'), ('miserably',
'RB'), ('who', 'WP'), ('want', 'VBP'), ('to', 'TO'), ('be', 'VB'), ('i
n', 'IN'), ('position', 'NN'), ('of', 'IN'), ('constantly', 'RB'), ('de
```

```
fending', 'VBG'), ('themselves', 'PRP'), ('agaist', 'JJ'), ('insultin
g', 'JJ'), ('attack', 'NN'), ('like', 'IN'), ('you', 'PRP'), ('seem',
'VBP'), ('to', 'TO'), ('like', 'VB'), ('to', 'TO'), ('do', 'VB'), ('sor
ry', 'VB'), ('you', 'PRP'), ('re', 'VBP'), ('so', 'RB'), ('blind', 'I
N'), ('that', 'IN'), ('you', 'PRP'), ('didn', 'VBP'), ('get', 'VB'),
('the', 'DT'), ('messgae', 'NN'), ('in', 'IN'), ('the', 'DT'), ('quot
e', 'NN'), ('everyone', 'NN'), ('else', 'RB'), ('ha', 'NN'), ('seemed',
'VBD'), ('to', 'TO'), ('tammy', 'VB')]
```

b) Apply the techniques described in tutorial 6.

Bigram on the basis of Frequency Distribution

In [11]:

```
import itertools, nltk

def ngrams_wrapper(sent):
    return list(nltk.ngrams(sent, 2))

flat_list = [item for sublist in lemmatizedTokens for item in sublist]
bigrams = map(ngrams_wrapper, lemmatizedTokens)
bigram = list(itertools.chain.from_iterable(bigrams))
freq_dist = nltk.FreqDist(bigram)

freqTable = pd.DataFrame({'Collocation': [v for v in freq_dist.keys()],
                          'Frequency': [k for k in freq_dist.values()]})

freqTable = freqTable.sort_values(by='Frequency', ascending=False).reset
_index(drop=True)
```

In [12]:
```
freqTable[:20]
```

Out[12]:

	Collocation	Frequency
0	(of, the)	5499
1	(in, the)	3321
2	(subject, re)	2449
3	(in, article)	1970
4	(to, the)	1888
5	(it, is)	1875
6	(on, the)	1787
7	(to, be)	1727
8	(nntp, posting)	1496
9	(posting, host)	1493
10	(if, you)	1430
11	(for, the)	1366
12	(that, the)	1321
13	(and, the)	1205
14	(is, the)	1120
15	(this, is)	1105
16	(line, in)	1046
17	(from, the)	1002
18	(is, not)	979
19	(there, is)	950

This output is not meaningfull since it contains pronouns and Stop words. Let us try and remove these stop words.

Also, 2 word collocation should be of the format Nouns/Adjective, Noun. The following function will filter out the bigrams that follow this rule.

```
In [13]: from nltk.corpus import words as nltk_words
nltk.download('words')
# Function to filter for ADJ/NN bigrams
# The function also filters stop words
dictionary = set(nltk_words.words())

def filterNounAdj(bigram):
    for word in bigram:
        if word in gensim.parsing.preprocessing.STOPWORDS or word not in
dictionary:
            return False

    ## A collocation may have Noun/Adjective in 1st word
    nounOrAdj = ('JJ',
                 'JJR',
                 'JJS',
                 'NN',
                 'NNS',
                 'NNP',
                 'NNPS')

    ## The second word can only be noun
    nouns = ('NN',
             'NNS',
             'NNP',
             'NNPS')

    tags = nltk.pos_tag(bigram)

    if tags[0][1] in nounOrAdj and tags[1][1] in nouns:
        return True
    else:
        return False
```

```
[nltk_data] Downloading package words to
[nltk_data]     /Users/gaganpree99/nltk_data...
[nltk_data]   Package words is already up-to-date!
```

```
In [14]: filteredFreqDist = {}
for tuple in freq_dist:
    if filterNounAdj(tuple):
            filteredFreqDist[tuple] = freq_dist[tuple]
```

Sorting Biagrams on the basis of frequency and displaying top 20 results.

In [15]:
```
filt_FreqTable = pd.DataFrame({'Collocation': [v for v in filteredFreqDist.keys()],
                                'Frequency': [k for k in filteredFreqDist.values()]})

filt_FreqTable = filt_FreqTable.sort_values(by='Frequency',
                                            ascending=False).reset_index(drop=True)
filt_FreqTable[:20]
```

Out[15]:

	Collocation	Frequency
0	(organization, university)	495
1	(distribution, world)	372
2	(line, distribution)	316
3	(newton, apple)	211
4	(newsreader, tin)	194
5	(henry, spencer)	180
6	(tin, version)	164
7	(henry, zoo)	162
8	(university, line)	153
9	(space, station)	139
10	(state, university)	133
11	(gamma, ray)	129
12	(alt, atheism)	127
13	(space, shuttle)	127
14	(bit, image)	116
15	(file, format)	109
16	(allan, schneider)	102
17	(political, atheist)	92
18	(jet, propulsion)	92
19	(new, york)	89

Bigram on the basis of PMI

In [16]:
```
bigram_measures = nltk.collocations.BigramAssocMeasures()

finder = nltk.collocations.BigramCollocationFinder.from_words(flat_list)
finder.apply_word_filter(lambda w: len(w) < 3 or word not in gensim.pars
ing.preprocessing.STOPWORDS)
finder.apply_freq_filter(20)
```

In [17]:
```
bigramPMITable = pd.DataFrame(list(finder.score_ngrams(bigram_measures.p
mi)),
                              columns=['Collocation','PMI']).sort_values
(by='PMI', ascending=False)
```

Calling the function `filterNounAdj` to filter biagrams and then displaying top 20 results.

In [20]:
```
filt_PMITable = bigramPMITable[bigramPMITable.Collocation.map(lambda x:
filterNounAdj(x))]
filt_PMITable[:20]
```

Out[20]:

	Collocation	PMI
6	(duck, pond)	15.238298
21	(maple, circa)	14.676250
46	(beam, jockey)	14.290164
62	(chapel, hill)	13.976427
88	(advisory, committee)	13.593477
103	(tourist, bureau)	13.404011
109	(ann, miller)	13.294462
113	(philosophical, significance)	13.219255
114	(cup, portal)	13.187206
125	(allan, schneider)	13.011130
129	(thou, shalt)	12.951775
133	(nominal, fee)	12.890952
135	(jeff, cook)	12.888788
139	(delta, clipper)	12.864839
141	(eternal, damnation)	12.853668
144	(red, herring)	12.837760
149	(pooh, bear)	12.798829
159	(lick, observatory)	12.667835
171	(tear, gas)	12.570973
191	(vertex, vertex)	12.357031

Bigram on the basis of t-test

In [21]:
```
bigramTtestTable = pd.DataFrame(list(finder.score_ngrams(bigram_measures
.student_t)),
                                                          columns=['Collocation','t-test']).sort_val
ues(by='t-test', ascending=False)
```

Calling the function `filterNounAdj` to filter biagrams and then displaying top 20 results.

In [22]:
```
filt_tTable = bigramTtestTable[bigramTtestTable.Collocation.map(lambda x
: filterNounAdj(x))]
filt_tTable[:20]
```

Out[22]:

	Collocation	t-test
14	(organization, university)	21.996450
19	(distribution, world)	19.245885
27	(line, distribution)	17.608997
56	(newton, apple)	14.520278
70	(newsreader, tin)	13.924899
77	(henry, spencer)	13.410081
87	(tin, version)	12.794010
90	(henry, zoo)	12.720916
118	(university, line)	11.840785
122	(space, station)	11.738172
130	(state, university)	11.456894
138	(gamma, ray)	11.349625
140	(alt, atheism)	11.258630
142	(space, shuttle)	11.179041
171	(bit, image)	10.573228
180	(file, format)	10.342956
204	(allan, schneider)	10.098282
236	(jet, propulsion)	9.589307
238	(political, atheist)	9.577927
253	(new, york)	9.422737

Bigram on the basis of chi-square

In [23]:
```
bigramChiTable = pd.DataFrame(list(finder.score_ngrams(bigram_measures.c
hi_sq)),
                              columns=['Collocation','chi_sq']).sort_val
ues(by='chi_sq', ascending=False)
```

Calling the function `filterNounAdj` to filter biagrams and then displaying top 20 results.

In [24]:
```
filt_chiTable = bigramChiTable[bigramChiTable.Collocation.map(lambda x:
filterNounAdj(x))]
filt_chiTable[:20]
```

Out[24]:

	Collocation	chi_sq
32	(allan, schneider)	842037.561326
36	(duck, pond)	811711.510816
41	(newsreader, tin)	774354.405216
49	(tourist, bureau)	715387.639844
60	(beam, jockey)	641076.521562
80	(newton, apple)	551056.588877
81	(maple, circa)	549796.473883
93	(chapel, hill)	499656.025147
118	(henry, spencer)	381574.790466
122	(jet, propulsion)	374425.678153
128	(western, reserve)	357316.082583
150	(pooh, bear)	313501.258486
154	(navy, mil)	301475.396372
155	(henry, zoo)	294172.659316
160	(thou, shalt)	285188.586752
177	(philosophical, significance)	247924.883390
179	(advisory, committee)	247196.385453
181	(cup, portal)	242479.408279
194	(loss, timer)	213457.226950
195	(ann, miller)	210961.910733

c) How much overlap is there among the techniques? Do you think it makes sense to considerthe union of the results?

Bigram Comparison

In [26]:
```
compareBiagrams = pd.DataFrame([filt_FreqTable[:20].Collocation.values,
                                filt_PMITable[:20].Collocation.values,
                                filt_tTable[:20].Collocation.values,
                                filt_chiTable[:20].Collocation.values]).T
compareBiagrams.columns = ['basedOnFreq','basedOnPMI','basedOnt-Test','b
asedOnChi-Square']
compareBiagrams
```

Out[26]:

	basedOnFreq	basedOnPMI	basedOnt-Test	basedOnChi-Square
0	(organization, university)	(duck, pond)	(organization, university)	(allan, schneider)
1	(distribution, world)	(maple, circa)	(distribution, world)	(duck, pond)
2	(line, distribution)	(beam, jockey)	(line, distribution)	(newsreader, tin)
3	(newton, apple)	(chapel, hill)	(newton, apple)	(tourist, bureau)
4	(newsreader, tin)	(advisory, committee)	(newsreader, tin)	(beam, jockey)
5	(henry, spencer)	(tourist, bureau)	(henry, spencer)	(newton, apple)
6	(tin, version)	(ann, miller)	(tin, version)	(maple, circa)
7	(henry, zoo)	(philosophical, significance)	(henry, zoo)	(chapel, hill)
8	(university, line)	(cup, portal)	(university, line)	(henry, spencer)
9	(space, station)	(allan, schneider)	(space, station)	(jet, propulsion)
10	(state, university)	(thou, shalt)	(state, university)	(western, reserve)
11	(gamma, ray)	(nominal, fee)	(gamma, ray)	(pooh, bear)
12	(alt, atheism)	(jeff, cook)	(alt, atheism)	(navy, mil)
13	(space, shuttle)	(delta, clipper)	(space, shuttle)	(henry, zoo)
14	(bit, image)	(eternal, damnation)	(bit, image)	(thou, shalt)
15	(file, format)	(red, herring)	(file, format)	(philosophical, significance)
16	(allan, schneider)	(pooh, bear)	(allan, schneider)	(advisory, committee)
17	(political, atheist)	(lick, observatory)	(jet, propulsion)	(cup, portal)
18	(jet, propulsion)	(tear, gas)	(political, atheist)	(loss, timer)
19	(new, york)	(vertex, vertex)	(new, york)	(ann, miller)

As we can see here, the results of frequency distribution are almost similar with that of t-test. Also, the results from Chi-Square test and PMI are more accurate even without applying the filters.

We can definitley combine the results of these techniques to obtain better results.

Since PMI tells us the measure of how likely the words co-occur if they were dependent it is more sesnitive towards rare combination of words. This means that a random biagram is given high significance when checked through PMI. So it makes more sense to combine the results of PMI and frequency distribution as they will give us more practical results.

Question 2

a) Cleaning the text

The following code performs the following:

1. Tokenize the corpus
2. Remove the stop words
3. Remove numbers, punctuation, special characters etc.
4. Stemming of tokens

In [27]:
```
allArticles = fetch_20newsgroups(subset='all',categories=cat)

tokens=[]
for document in allArticles.data:
    tokens.append(gensim.utils.simple_preprocess(document,min_len=2))
```

In [28]:
```
from nltk.stem import WordNetLemmatizer, SnowballStemmer
stemmer = SnowballStemmer("english")

stemmedTokens1 = []
for line in tokens:
    stemmedwords = []
    for word in line:
        if word not in gensim.parsing.preprocessing.STOPWORDS:
            stemmedwords.append(stemmer.stem(WordNetLemmatizer().lemmati
ze(word, pos='v')))
    stemmedTokens1.append(list(stemmedwords))

stemmedTokens1[:2]
```

Out[28]:
```
[['healta', 'saturn', 'wwc', 'edu', 'tammi', 'heali', 'subject', 'jud
g', 'bobbi', 'line', 'organ', 'walla', 'walla', 'colleg', 'line', 'arti
cl', 'apr', 'ultb', 'isc', 'rit', 'edu', 'snm', 'ultb', 'isc', 'rit',
'edu', 'mozumd', 'write', 'snm', 'ultb', 'isc', 'rit', 'edu', 'mozumd',
'subject', 'judg', 'bobbi', 'date', 'wed', 'apr', 'gmt', 'articl', 'hea
lta', 'saturn', 'wwc', 'edu', 'healta', 'saturn', 'wwc', 'edu', 'tamm
i', 'heali', 'write', 'bobbi', 'like', 'liberti', 'quot', 'christian',
'writer', 'name', 'ellen', 'white', 'hope', 'say', 'help', 'edit', 'rem
ark', 'group', 'futur', 'set', 'standard', 'opinion', 'view', 'duti',
'interpret', 'scriptur', 'criterion', 'heart', 'condemn', 'come', 'idea
l', 'thought', 'fromth', 'mount', 'bless', 'hope', 'quot', 'atheist',
'gag', 'think', 'ellen', 'white', 'better', 'tammi', 'point', 'peac',
'bobbi', 'mozumd', 'point', 'set', 'view', 'way', 'believ', 'say', 'eve
il', 'world', 'caus', 'atheism', 'ridicul', 'dialogu', 'newsgroup', 'po
st', 'spirit', 'condemn', 'atheist', 'newsgroup', 'bacaus', 'believ',
'exact', 'tri', 'convert', 'atheist', 'fail', 'miser', 'want', 'posit',
'constant', 'defend', 'agaist', 'insult', 'attack', 'like', 'like', 'so
rri', 'blind', 'messga', 'quot', 'tammi'], ['jk', 'lehtori', 'cc', 'tu
t', 'fi', 'kouhia', 'juhana', 'subject', 'gray', 'level', 'screen', 'or
gan', 'tamper', 'univers', 'technolog', 'line', 'distribut', 'inet', 'n
ntp', 'post', 'host', 'cc', 'tut', 'fi', 'articl', 'apr', 'cis', 'uab',
'edu', 'sloan', 'cis', 'uab', 'edu', 'kenneth', 'sloan', 'write', 'crea
t', 'grey', 'level', 'imag', 'display', 'time', 'slice', 'grey', 'leve
l', 'imag', 'mean', 'item', 'bite', 'imag', 'work', 'work', 'bite', 'sc
reen', 'screen', 'intens', 'non', 'linear', 'bite', 'pixel', 'c_', 'c
_', 'time', 'give', 'level', 'linear', 'screen', 'intens', 'linear', 'c
_', 'c_', 'work', 'best', 'compin', 'level', 'chois', 'best', 'choos',
'differ', 'compin', 'level', 'vari', 'bite', 'level', 'keep', 'order',
'reader', 'verifi', 'write', 'juhana', 'kouhia']]
```

b) Bag-of-words tf-idf weighted vector representation

The cleaned text is now used to perform TF-IDF weighted vector representation

In [29]:
```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
feature_counts = count_vect.fit_transform([' '.join(x) for x in stemmedT
okens1])
```

```
In [30]: from sklearn.feature_extraction.text import TfidfTransformer
         tfidf_transformer = TfidfTransformer()
         feature_tfidf = tfidf_transformer.fit_transform(feature_counts)
         feature_tfidf.shape

Out[30]: (3387, 26312)
```

c) Split the data randomly into training and testing set (70-30 %)

The following code splits the TF-IDF representation into train and testing set:

```
In [31]: from sklearn.metrics import confusion_matrix
         from sklearn.model_selection import train_test_split
         import time


         X_train, X_test, y_train, y_test = train_test_split(feature_tfidf,
                                                             allArticles.target,
                                                             test_size=0.3,
                                                             random_state=11)
```

Train Multinomial NB and report confusion matrix

```
In [32]: from sklearn.naive_bayes import MultinomialNB
         clf = MultinomialNB().fit(X_train, y_train)
         print("Accuracy: " + str(round(100*(clf.score(X_test, y_test)),2))+"%")

         Accuracy: 90.76%
```

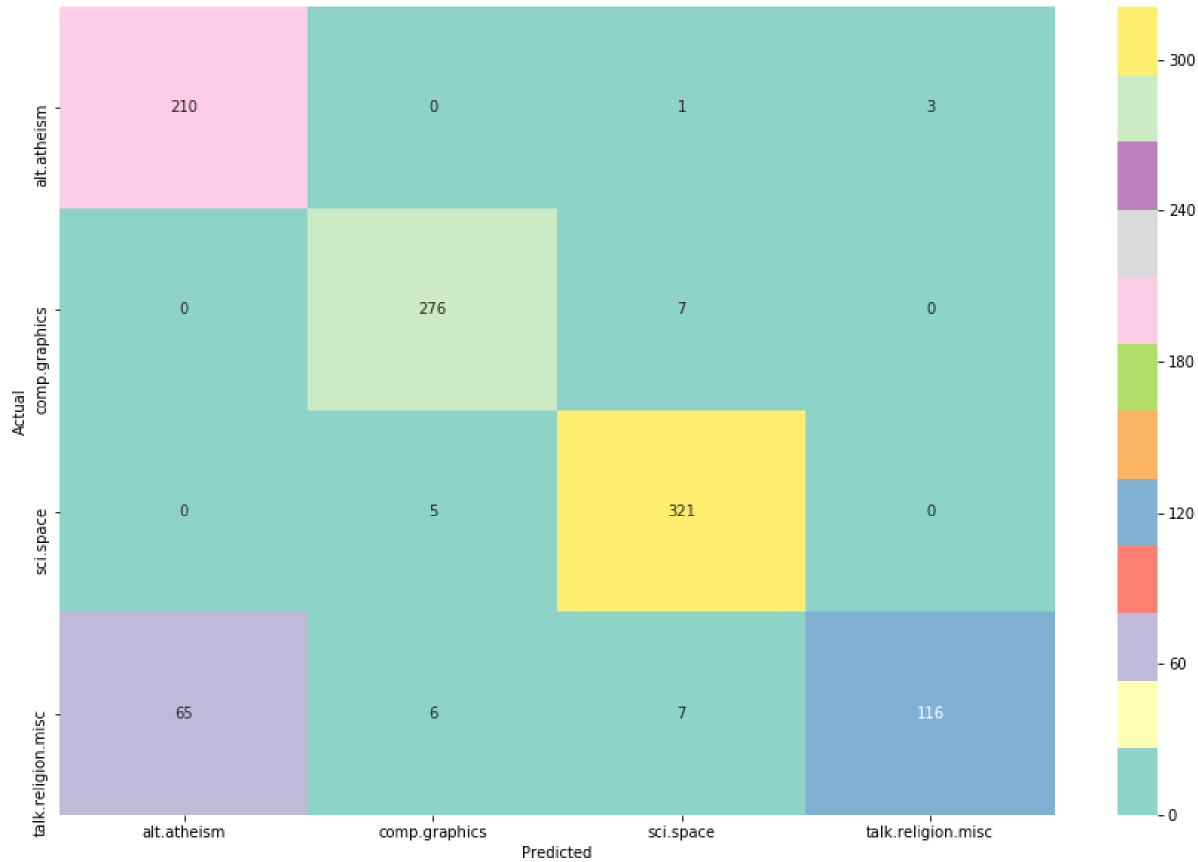
Following function is used to plot confusion matrix of a given model.

```
In [33]: import matplotlib.pyplot as plt
         import seaborn as sns

         def plotConfusionMatrix(model):
             y_pred = model.predict(X_test)
             conf_mat = confusion_matrix(y_test, y_pred)

             labels=allArticles.target_names
             # Plot confusion_matrix
             fig, ax = plt.subplots(figsize=(15, 10))
             sns.heatmap(conf_mat, annot=True, cmap = "Set3", fmt ="d", xticklabe
ls=labels, yticklabels=labels)
             plt.ylabel('Actual')
             plt.xlabel('Predicted')
             plt.show()
```

In [34]:
```
#Plot Confusion matrix for Multinomial Naive Based Model
plotConfusionMatrix(clf)
```

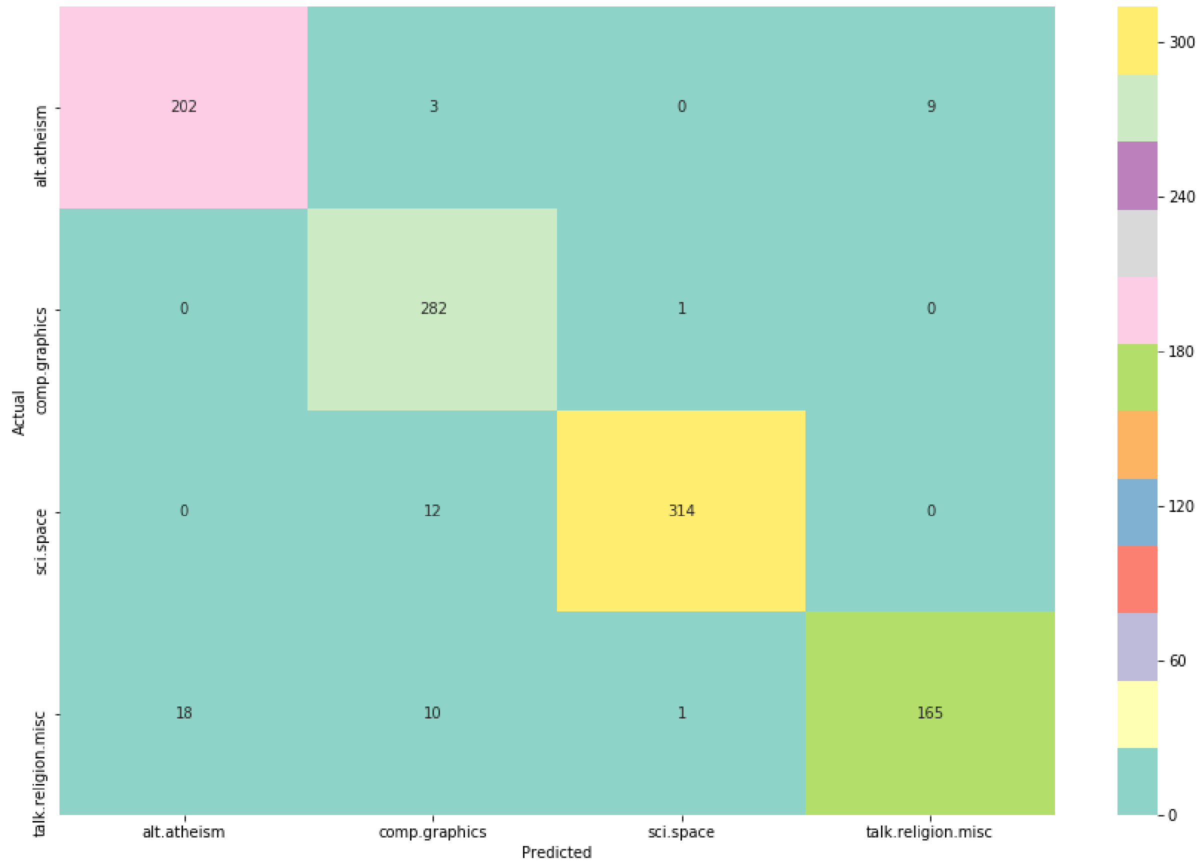


Train SVM and report confusion matrix.

In [35]:
```
from sklearn.svm import SVC
clf2 = SVC(gamma='scale').fit(X_train, y_train)
print("Accuracy: " + str(round(100*(clf2.score(X_test, y_test)),2))+"%")
```

```
Accuracy: 94.69%
```

In [36]:
```
#Plot Confusion matrix for SVM
plotConfusionMatrix(clf2)
```



Which algorithm has higher accuracy and why?

SVM has a slight higher acuracy as compared to Naive Bayes. This can be explained as Naive Bayes treats the features points as independent, however SVM looks at the interaction between the points.

In our case since the feature points that are the tokens extracted from the document have relation among each other, hence SVM tends to perform slighly better.

Also, MNB tends to perform better on snippets than on long documents. An important point to note here is that although SVM provides a better accuracy, the execution time for MNB is way shorter than SVM. Hence, to get a accuracy improvement of ~3% we have to make a trade-off in time domain.

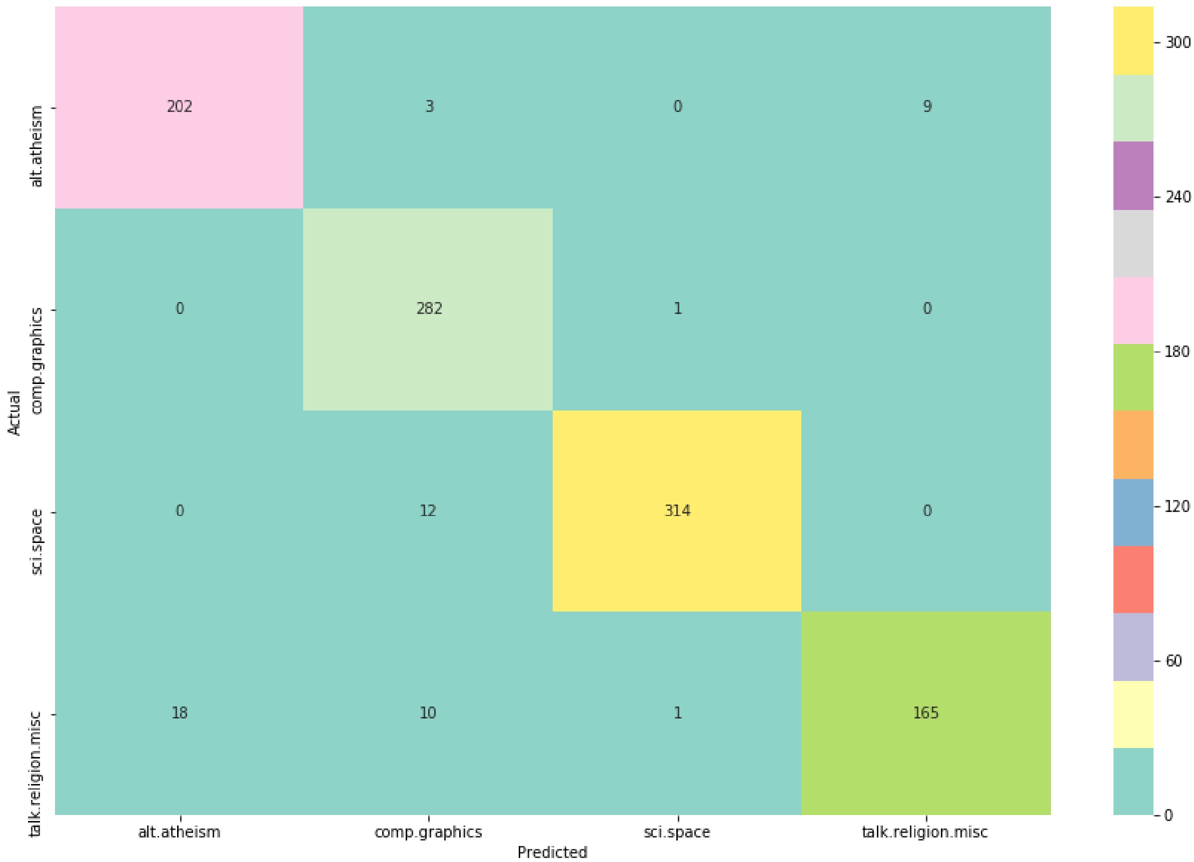
Changing Kernels to see if we get better results

1) with `RBF` Kernel

In [37]:
```
clf3 = SVC(kernel='rbf', gamma='scale').fit(X_train, y_train)
print("Accuracy: " + str(round(100*(clf3.score(X_test, y_test)),2))+"%")
```

```
Accuracy: 94.69%
```

In [38]:
```python
#Plot Confusion matrix with RBF kernel
plotConfusionMatrix(clf3)
```

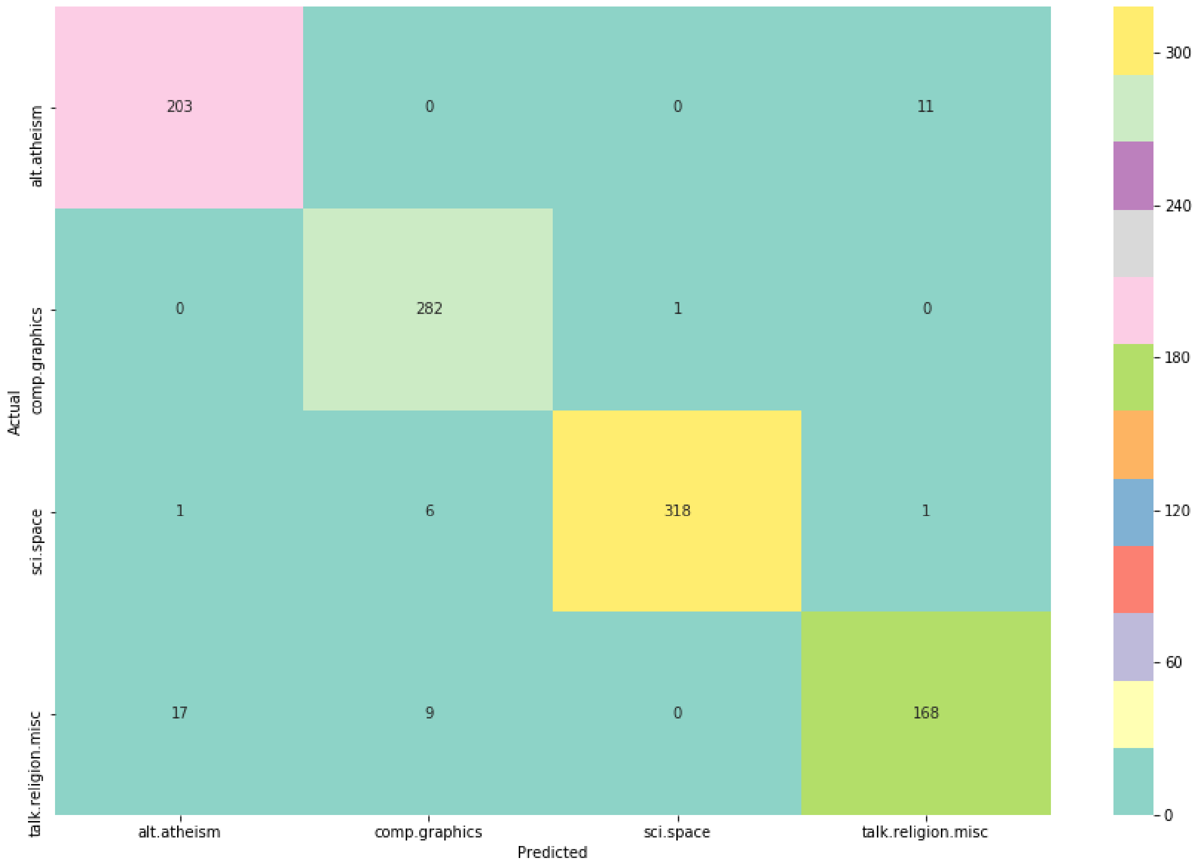


2) with `linear` kernel

In [39]:
```python
clf4 = SVC(kernel='linear', gamma='scale').fit(X_train, y_train)
print("Accuracy: " + str(round(100*(clf4.score(X_test, y_test)),2))+"%")
```

```
Accuracy: 95.48%
```

```
In [40]: #Plot Confusion matrix with Linear kernel
         plotConfusionMatrix(clf4)
```

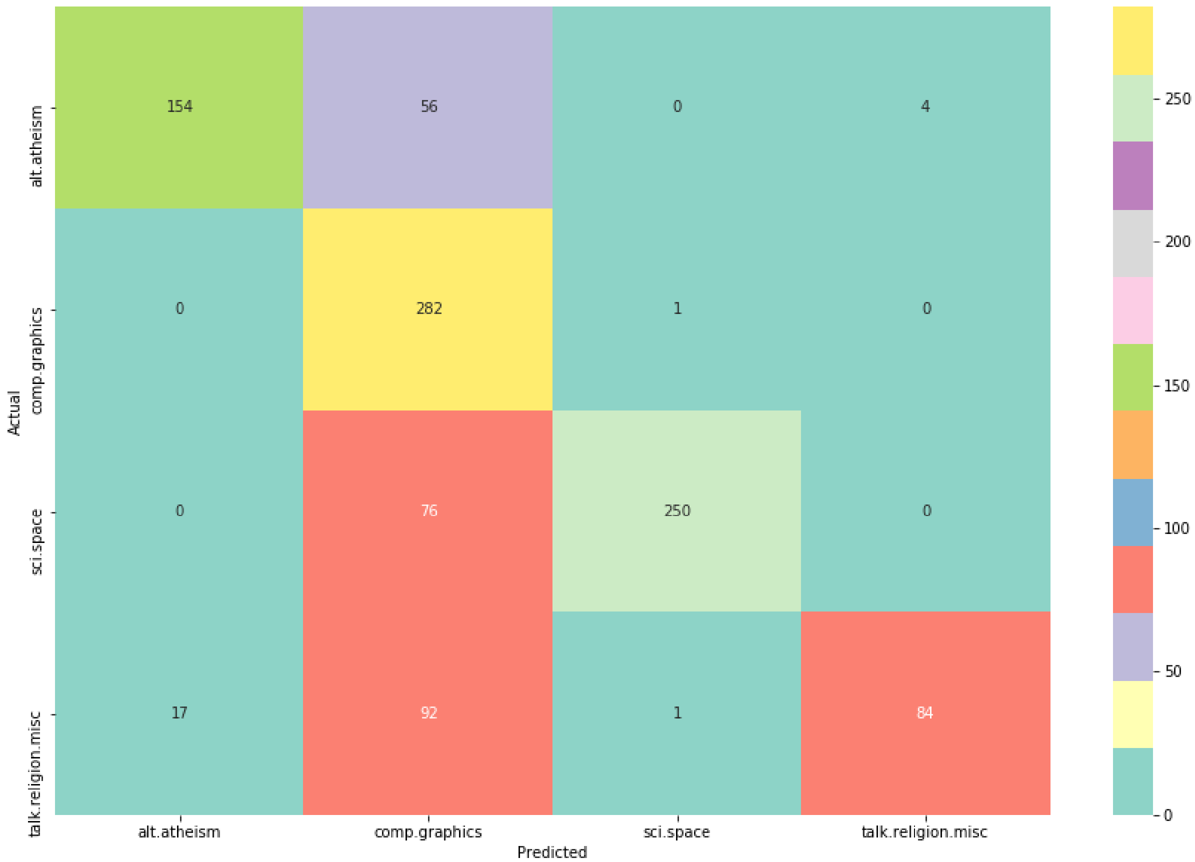


3) with `poly` kernel

```
In [41]: clf5 = SVC(kernel='poly', gamma='scale').fit(X_train, y_train)
         print("Accuracy: " + str(round(100*(clf5.score(X_test, y_test)),2))+"%")
```

```
Accuracy: 75.71%
```

In [42]:
```
#Plot Confusion matrix with poly kernel
plotConfusionMatrix(clf5)
```

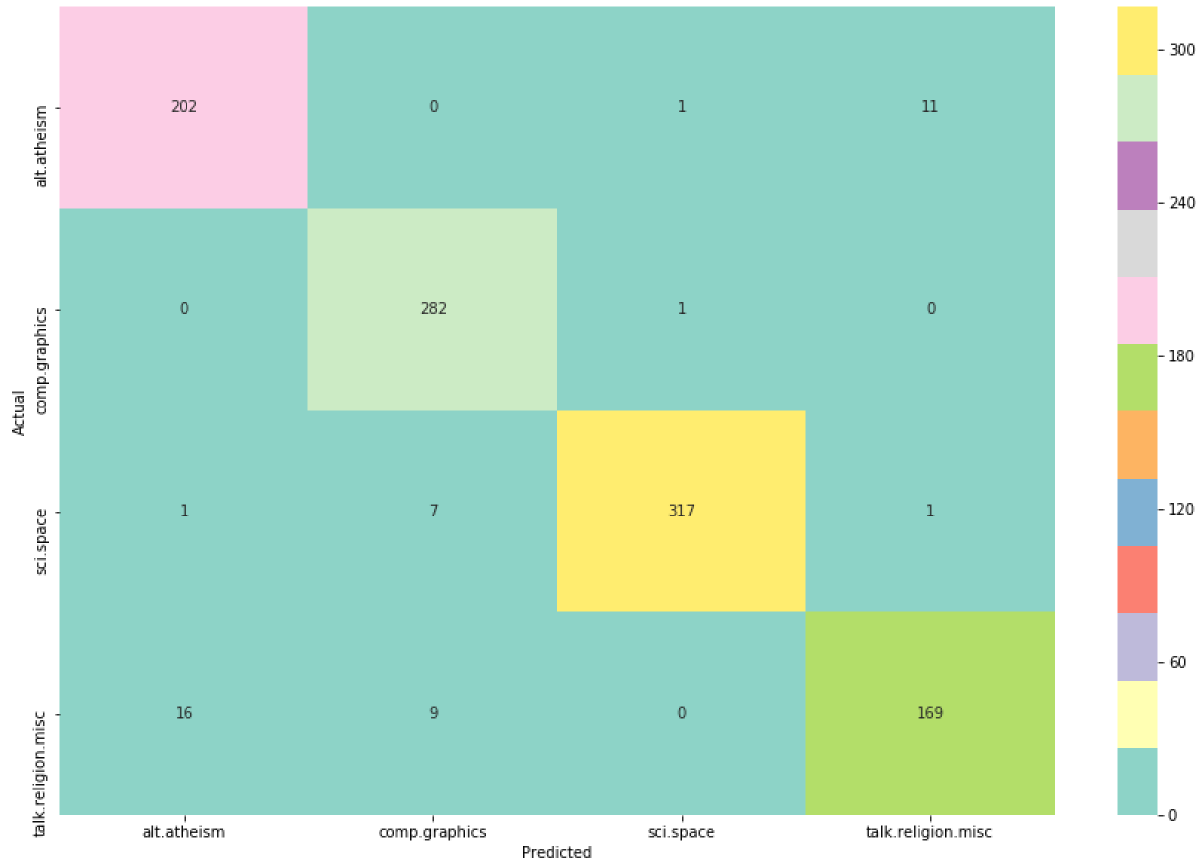


4) with `sigmoid` kernel

In [43]:
```
clf6 = SVC(kernel='sigmoid', gamma='scale').fit(X_train, y_train)
print("Accuracy: " + str(round(100*(clf6.score(X_test, y_test)),2))+"%")
```

```
Accuracy: 95.38%
```

```
In [44]: #Plot Confusion matrix with sigmoid kernel
         plotConfusionMatrix(clf6)
```



As we can see, the best accuracy is achieved with `linear` kernel and has a accuracy of 95.48%. The Confusion matrix for `linear` kernel has least number of misclassifications.

Hence, `linear` kernel has the best results in comparison to other non-linear kernels.

---

d) Perform POS tagging and perform clasification again

Fetching raw data from given dataset

```
In [45]: allArticles = fetch_20newsgroups(subset='all',categories=cat)
         tokens=[]
         for document in allArticles.data:
             tokens.append(gensim.utils.simple_preprocess(document,min_len=2))
```

Perform POS Tagging on raw dataset

In [46]:
```
taggedTokens = []
for doc in tokens:
    tags = nltk.pos_tag(doc)
    taggedTokens.append(tags)
```

Extract nouns from the tagged data

In [47]:
```
nouns = ('NN',
         'NNS',
         'NNP',
         'NNPS')

nounTokens = []
for document in taggedTokens:
    nounsFound = []
    for taggedToken in document:
        if taggedToken[1] in nouns:
            nounsFound.append(taggedToken[0])
    nounTokens.append(nounsFound)
```

Repeat the steps 2(a), 2(b) and 2(c)

In [48]:
```
from nltk.stem import WordNetLemmatizer, SnowballStemmer
stemmer = SnowballStemmer("english")

stemmedTokens2 = []
for line in nounTokens:
    stemmedwords = []
    for word in line:
        if word not in gensim.parsing.preprocessing.STOPWORDS:
            stemmedwords.append(stemmer.stem(WordNetLemmatizer().lemmati
ze(word, pos='v')))
    stemmedTokens2.append(list(stemmedwords))
```

In [49]:
```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
feature_counts = count_vect.fit_transform([' '.join(x) for x in nounToke
ns])
```

In [50]:
```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
feature_tfidf = tfidf_transformer.fit_transform(feature_counts)
feature_tfidf.shape
```

Out[50]:
```
(3387, 24713)
```

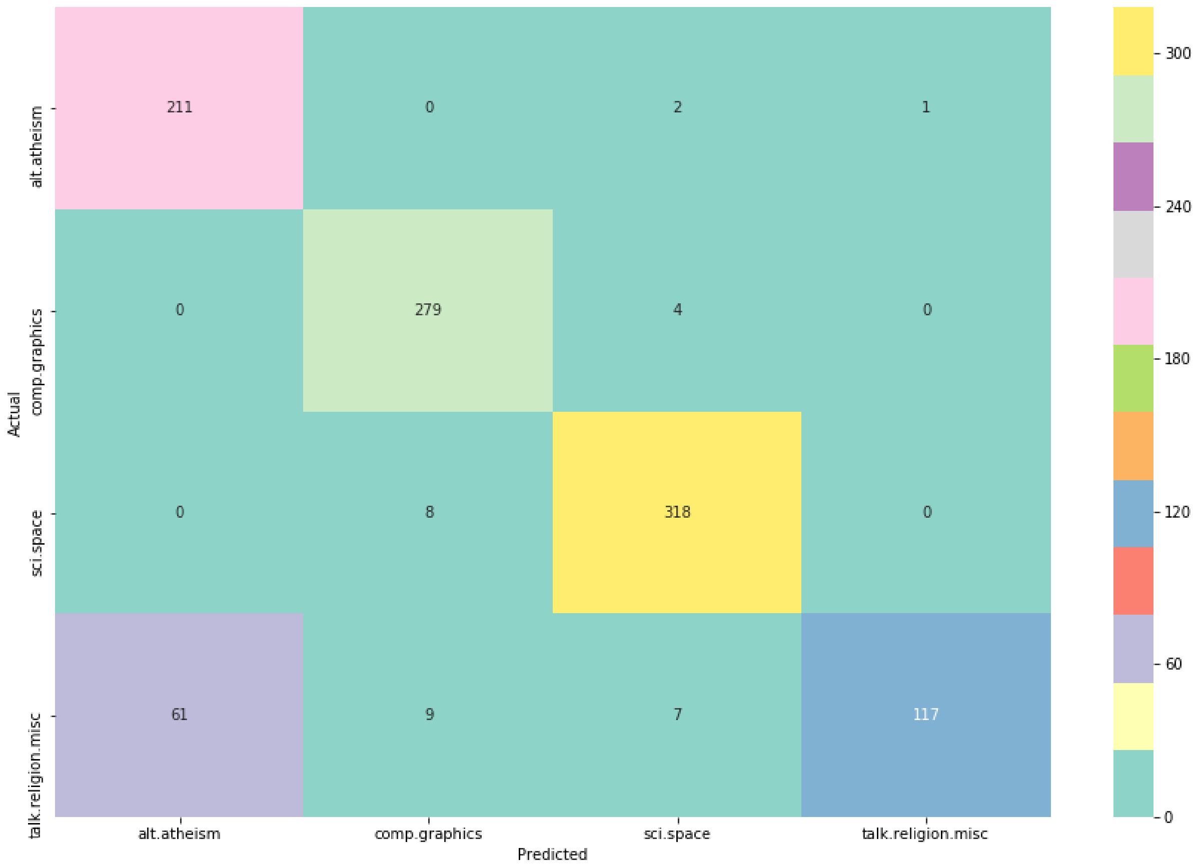
In [51]:
```
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import time


X_train, X_test, y_train, y_test = train_test_split(feature_tfidf,
                                                    allArticles.target,
                                                    test_size=0.3,
                                                    random_state=11)
```

In [52]:
```
from sklearn.naive_bayes import MultinomialNB
clf7 = MultinomialNB().fit(X_train, y_train)
print("Accuracy: " + str(round(100*(clf7.score(X_test, y_test)),2)))
```

```
Accuracy: 90.95
```

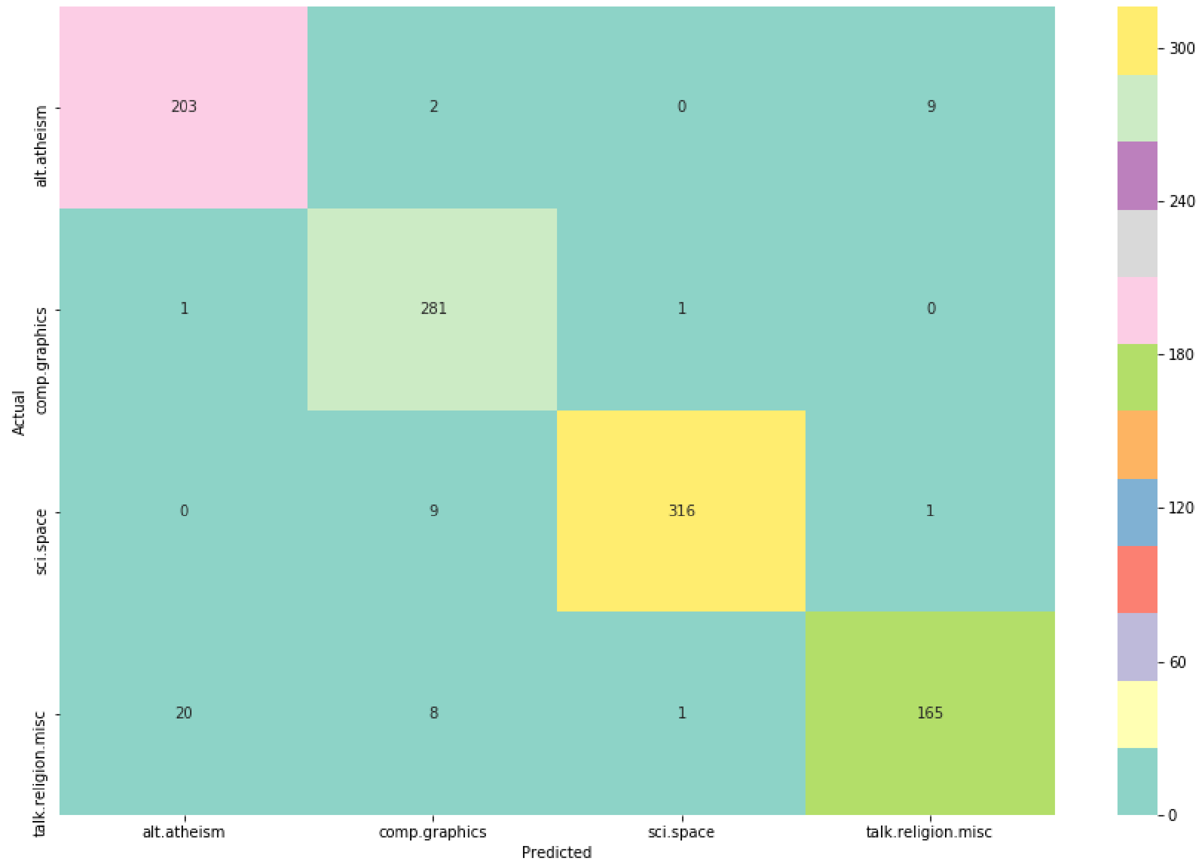
In [53]:
```
plotConfusionMatrix(clf7)
```



In [54]:
```
clf8 = SVC(kernel='linear', gamma='scale').fit(X_train, y_train)
print("Accuracy: " + str(round(100*(clf8.score(X_test, y_test)),2)))
```

```
Accuracy: 94.89
```

```
In [55]: plotConfusionMatrix(clf8)
```



As we can see there is almost no change in the accuracy after POS tagging.

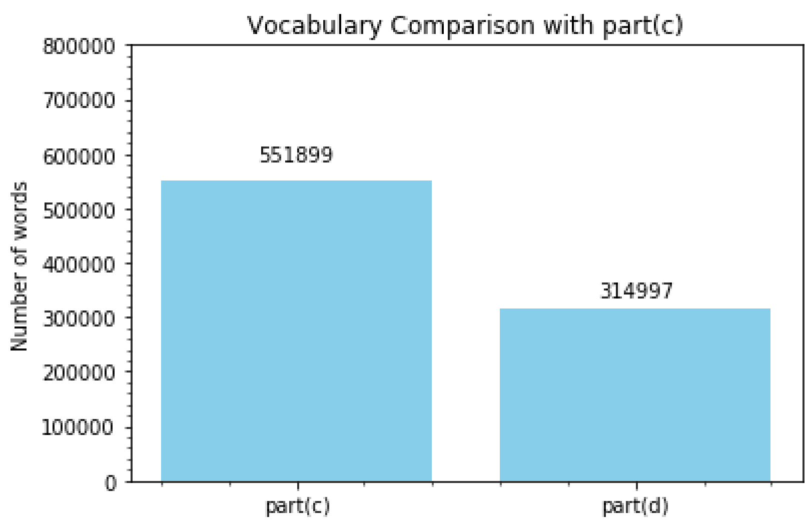
How does the size of the vocabulary compare with that of part (c)

```
In [59]: # Vocabulary without POS tagging
count1=0
for list in stemmedTokens1:
    count1+=len(list)
```

```
In [60]: # Vocabularywith POS tagging
count2=0
for list in stemmedTokens2:
    count2+=len(list)
```

In [58]:
```
import numpy as np
import matplotlib.pyplot as plt
objects = ('part(c)', 'part(d)')
y_pos = np.arange(len(objects))
performance = [count1,count2]
bar = plt.bar(y_pos, performance, align='center',color='skyblue')
plt.xticks(y_pos, objects)
plt.ylabel('Number of words')
plt.title('Vocabulary Comparison with part(c)')
plt.minorticks_on()
plt.ylim(0,800000)
for rect in bar:
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width()/2., 1.05*height,'%d' % int(
height), ha='center', va='bottom')

plt.show()
```



It is clear from the above bar plot that we have reduced the vocabulary to almost half in comparison to part(c). There's a drop in accuracy by ~0.4%, however this is not much significant.

Since, the number of words after POS tagging are quite less, hence the model becomes quite faster.

References

[1] NLP with the 20 Newsgroups Dataset - Rox S - Medium. (2019). Retrieved 17 July 2019, from https://medium.com/@siyao_sui/nlp-with-the-20-newsgroups-dataset-ab35cd0ea902 (https://medium.com/@siyao_sui/nlp-with-the-20-newsgroups-dataset-ab35cd0ea902)

[2] priya-dwivedi/Deep-Learning. (2019). Retrieved 17 July 2019, from https://github.com/priya-dwivedi/Deep-Learning/blob/master/topic_modeling/LDA_Newsgroup.ipynb (https://github.com/priya-dwivedi/Deep-Learning/blob/master/topic_modeling/LDA_Newsgroup.ipynb)

In [ ]: