coursework_1

December 1, 2017

1 NLE Coursework 1

1.1 Important Information

Submission format:
You should submit just one file that should be a Jupyter notebook.

Due date:
Your notebook should be submitted on Study Direct before 4pm on Thursday 30th November. The standard late penalties will apply.

Return date:
Marks and feedback will be provided on Sussex Direct on Friday December 21st for all submissions that are submitted by the due date.

Weighting:
This assessment contributes 50% of the mark for the module.

Word limit:
Your Jupyter notebook should contain no more than 3000 words. The standard penalties apply for deviations from this limit (see below). You *must* specify the number of words in your report.

1.1.1 Failure to observe limits of length

The following is taken from the Examination and Assessment Regulations Handbook

> The maximum length for each assessment is publicised to students. The limits as stated include quotations in the text, but do not include the bibliography, footnotes/endnotes, appendices, abstracts, maps, illustrations, transcriptions of linguistic data, or tabulations of numerical or linguistic data and their captions. Any excess in length should not confer an advantage over other students who have adhered to the guidance. Students are requested to state the word count on submission. Where a student has marginally (within 10%) exceeded the word length the Marker should penalise the work where the student would gain an unfair advantage by exceeding the word limit. In excessive cases (>10%) the Marker need only consider work up to the designated word count, and discount any excessive word length beyond that to ensure equity across the cohort. Where an assessment is submitted and falls significantly short (>10%) of the word length, the Marker must consider in assigning a mark, if the argument has been sufficiently developed and is sufficiently supported and not assign the full marks allocation where this is not the case.

Note that code and the content of cell outputs are excluded from the word count.

In [46]:
```
import sys
# sys.path.append(r'\\ad.susx.ac.uk\ITS\TeachingResources\Departments\Info
sys.path.append(r'/Users/warrenboult/Documents/MSC/nlp/resources/')
import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import collections
from collections import defaultdict,Counter
from itertools import zip_longest
from IPython.display import display
from random import seed
get_ipython().magic('matplotlib inline')
import random
import math
import nltk
import matplotlib.pylab as pylab
%matplotlib inline
params = {'legend.fontsize': 'large',
          'figure.figsize': (15, 5),
         'axes.labelsize': 'large',
         'axes.titlesize':'large',
         'xtick.labelsize':'large',
         'ytick.labelsize':'large'}
pylab.rcParams.update(params)
from pylab import rcParams
from operator import itemgetter, attrgetter, methodcaller
import matplotlib.pyplot as plt
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
import seaborn as sns
import csv
```

In [8]:
```
import spacy
from sussex_nltk.corpus_readers import AmazonReviewCorpusReader

nlp = spacy.load('en')
dvd_reviews = [review for review in AmazonReviewCorpusReader().category("dv
print("The dvd review dataset contains {} reviews".format(len(dvd_reviews))
parsed_reviews = [nlp(review) for review in dvd_reviews]
print('done')
```

```
Sussex NLTK root directory is /Users/warrenboult/Documents/MSC/nlp/resources
The dvd review dataset contains 5491 reviews
done
```

1.2 Overview

For this assignment, you are asked to write a report on some of the activities covered in the notebooks for Topic 3 (week 5) and Topic 6 (week 8).

- Topic 3 concerns the classification of Amazon product reviews according to whether the overall sentiment being expressed is positive or negative. In the lab exercises you explored factors that have an impact on classifier accuracy.
- Topic 6 involves extending an opinion extractor function so that it deals with a variety of ways that opinions are expressed in Amazon DVD reviews. This work is covered in the lab classes in week 8.

For this report you should create a Jupyter notebook containing the following sections (further details regarding the content of each section are given below).

Section 1:
A report on what you discovered while undertaking some of the exercises in the Topic 3 notebook.

Section 2:
The python code for your opinion extractor and a description of how it works. This section should also include a demonstration that your opinion extractor produces the correct output for the example sentences that we have given for each of the required extensions.

Section 3:
An assessment of the limitations of your extractor based on applying it to your own personalised random sample of 100 sentences from the corpus of DVD amazon reviews (see below for details of how to obtain your personalised random sample).

1.3 Details of Requirements

Your submission will be marked out of 100. Please read the following guidelines carefully.

1.4 Section 1: Document Level Sentiment Analysis (30 marks)

There are 30 marks available for this section. Section 1 should include the following four subsections.

1.4.1 Section 1.1 (7 marks)

You investigated the differences in accuracy between the wordlist classifier performance and the Naïve Bayes classifier performance. In this subsection, describe what you found, and, by looking at classifier performance on a sample of product reviews (it is up to you to decide how many), discuss the reasons behind the differences in performance of the methods being compared.

1.4.2 Section 1.2 (7 marks)

For the methods that require training data, you investigated the impact on classifier accuracy when the amount of training data is varied. In this subsection you should discuss your findings, and then attempt to predict what the accuracy of the Naïve Bayes classifier would be if you massively increased the amount of training data.

1.4.3 Section 1.3 (8 marks)

You investigated the impact on classifier accuracy of training a classifier with data in one domain (the source domain), and testing the same classifier on data from a different domain (the target domain). In this subsection you should describe what you found, and then investigate the reasons why some source to target changes in domain have more impact on classifier accuracy than others.

1.4.4 Section 1.4 (8 marks)

In this subsection you should discuss what you found when investigating the impact on classifier accuracy of various feature extraction methods.

1.4.5 Things to note

- Quality of your experimental method
- make sure that you make an appropriate division of labelled data for training and testing
- make sure that you have repeated experiments with different random samples and presented averaged results
- Presentation of results
- graphs or tables should be used when presenting your empirical findings
- when you are making a comparison between two or more results, you should display them together on the same graph so that the comparison can be seen directly.
- in cases where your empirical investigation is not complete, be explicit as to what you have not managed to achieve.
- Quality of analysis
- do not just present tables of numbers and graphs without any discussion. You need to put forward a reasonable explanation as to why you have observed the results that you have obtained.
- Always back up claims with evidence. Be careful not to make bold conclusions from small-scale testing.

```
In [70]: # Section 1.1
         from classification_utils import *
         from random import sample

         reader = AmazonReviewCorpusReader().category("dvd")

         word_list_size = 100
         av_WL_accuracy = 0
         av_NB_accuracy = 0
         for i in range(10):
             pos_train,neg_train,pos_test,neg_test = get_train_test_data(reader)
             sub_pos_test = sample(pos_test, 100)
             sub_neg_test = sample(neg_test, 100)
             av_WL_accuracy += run_WL(pos_train,neg_train,sub_pos_test,sub_neg_test
             av_NB_accuracy += run_NB(pos_train,neg_train,sub_pos_test,sub_neg_test
         av_WL_accuracy = av_WL_accuracy / 10
```

```
        av_NB_accuracy = av_NB_accuracy / 10
        print('Example Run')
        print(av_WL_accuracy)
        print(av_NB_accuracy)
        # print(list(sub_pos_test))
        # print(list(sub_neg_test))
```

```
Example Run
0.633
0.7924999999999999
```

1.4.6 Section 1.1

First of all, the performance of the Word List classifier was compared with the performance of the Naive Bayes classifier on the 'dvd' review category, averaged over 10 runs on test set sizes of 100 positive and 100 negative. The results are shown in the figure below; the Naive Bayes classifier performs significantly better than the Word List, as one might expect, with accuracies of 82% and 62% respectively.

If we look at the a small section of the wordlists used by the WL classifier, for the positive and negative data, we can see why its performance is so low. Looking at the 10 most frequent words in each class, for positive we have:

['I', 'The', 'movie', 'film', 'one', 'It', 'This', 'like', 'DVD', 'great']

and for negative we have:

['I', 'movie', 'The', 'film', 'one', 'like', 'It', 'would', 'This', 'DVD']

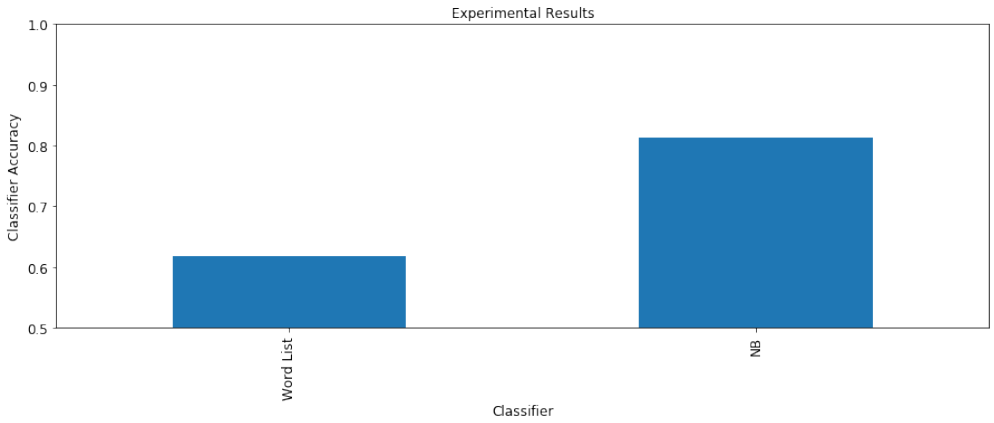
This just gives a taste of the overall similarity of the word lists, and thus the difficult task the classifier would have in choosing between the two classes; a popular word list would need some sensible filtering to make the words more indicative of the classes they are representing. In fact, it is often the rarer words in a class that have more impact on which class a document would belong to, as the inverse document frequency measure for document classification goes to show.

Meanwhile, the Naive Bayes classifier obtains its probability scores by taking the product of probabilities of all words found in a document of a class. Therefore the NB classifier has a better chance of picking up on the less-popular, more-informative words in a review, and can thus better classify the documents. The Word List classifier can be seen as a somewhat simplified version of the Naive Bayes classifier.

1.4.7 Section 1.2

Next, the effect of varying the size of the training data set was analysed. The size of the training data, for each class, was varied between 1 and 700 reviews, and the classifiers were then tested on each of the Amazon review categories. The results are presented in the figures below. In all cases, we can see similar behaviour, of each classifier's performance being about random when the training size is 1 sample, and steadily increasing to an accuracy value similar to the baselines found above. Clearly, when the amount of training data is low, both classifiers highly bias their training towards these few samples.

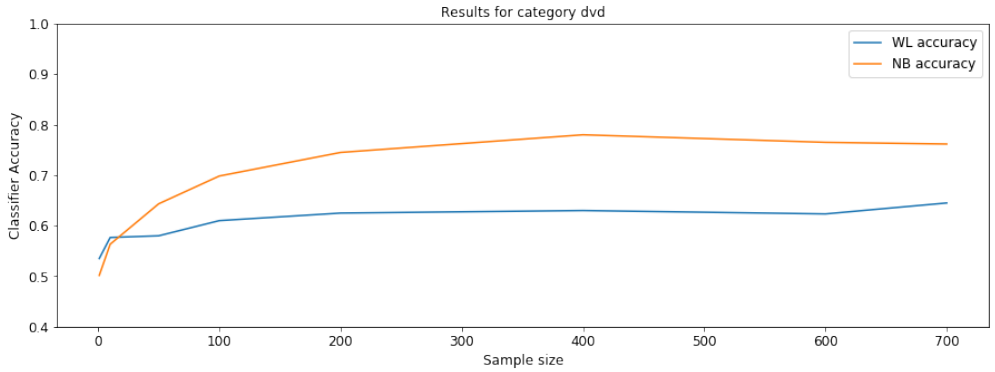
For each classifier, their set of words to train from is small and so they can't generalise as well to other reviews containing different words. As the sample sizes increase, the WL classifier builds a slightly better top list of words, but it will generally still contain uninformative words in its list

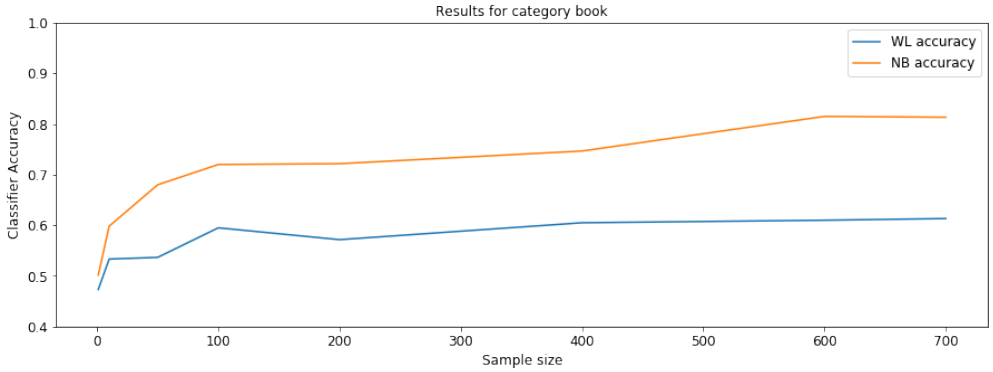


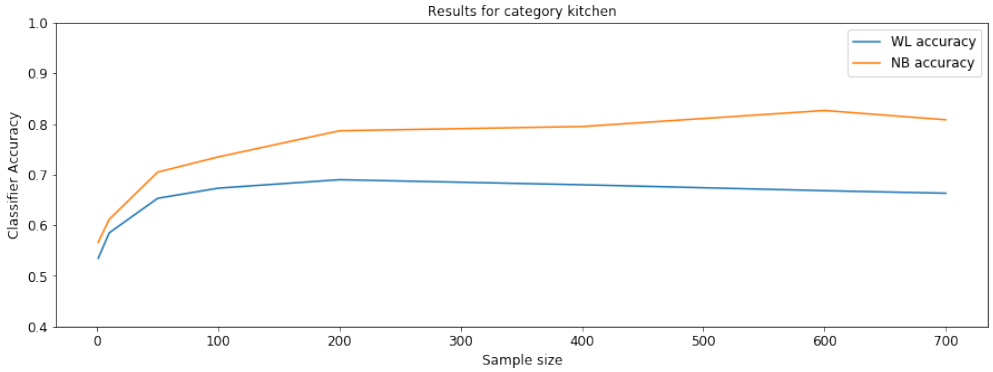
alt text

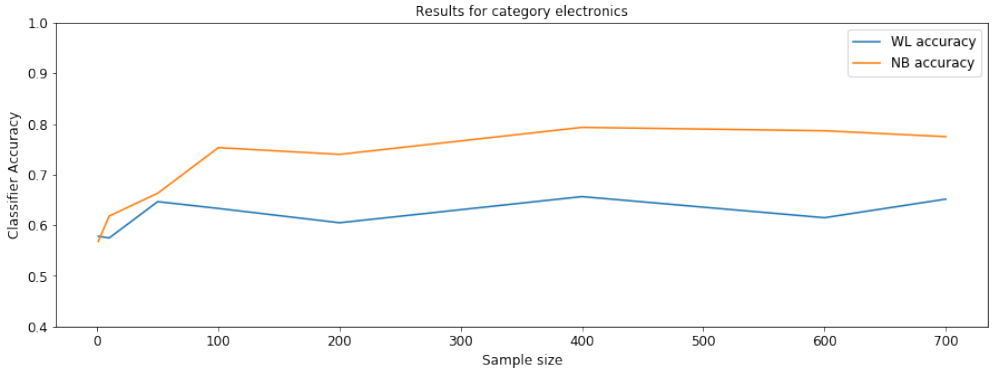
since these are far more common than the informative, sentiment-displaying words in a review. For NB, increasing the sample sizes leads to many more informative words being included for training, and so it leaps up almost to its max value quickly in each case, at about a size of 100-200 reviews. This shows the power of the NB over the WL, even a modest training set size can produce a decent quality classifier.

As the size of the training set massively increases however, we wouldn't expect to get much more of an increase in performance from the NB classifier without modifying its functionality more. This is due to the fact that while more informative words would be included in the larger training set, these would likely be offset by the noisy words present, and there is also a limit on how informative a bag of words approach to classification can be. There will always be information missed by only taking words as individual features. The figures below seem to support a decline in the performance improvements of the NB classifier, with the curves levelling out largely as the sample sizes are increased over 300-400.







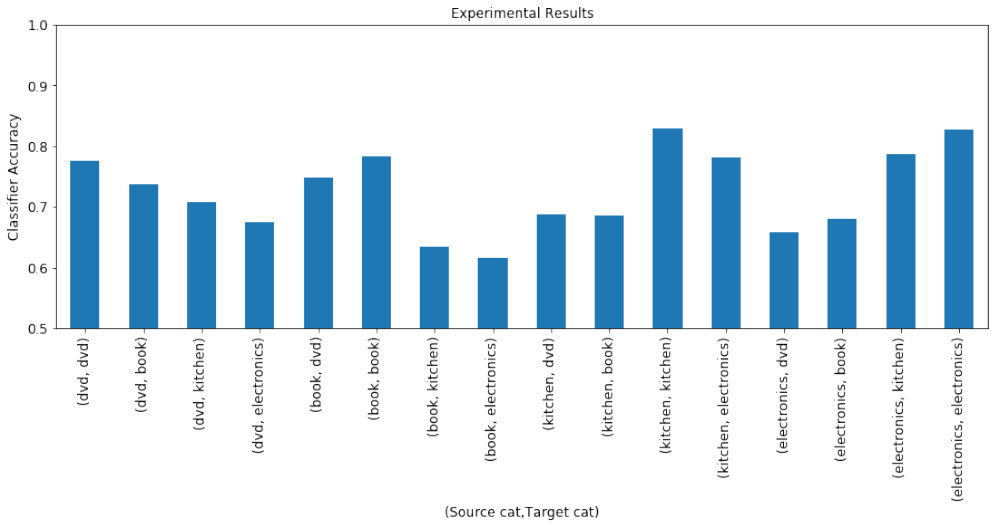


1.4.8 Section 1.3

The figure below shows the accuracy results for training the NB classifier in one domain, and testing it in another. Clearly, the accuracy scores are lower when using different domains to train and test, with the lowest score being between books and electronics, and the highest being between electronics and kitchen. Not all accuracy scores are as low or high as each other however, and this is down to the content of the reviews for each category. Firstly, the words describing the book category are going to be very different than the words describing the electronics, or kitchen category; book reviews will contain evaluations of the quality of the plot, or characters, how interesting the story is; meanwhile for electronics and kitchen reviews, people will be reviewing physical items, commenting on the build quality, how it feels to use, if it works correctly, etc. This serves to explain both why book-electronics cross domain analysis was so poor, and why kitchen-electronics analysis was almost as good as single domain.

Similarly, dvd-book, and book-dvd analysis performed nearly as well as single domain, and this is because people will be mentioning similar things in these reviews, like the qual-

ity of the plot, the development of the characters; of course there are slight differences in what people will be mentioning, but it will be mostly similar. Furthermore, it could be said that book-dvd analysis outperforms dvd-book due to the book training set containing lots of content generally reviewing 'stories', and so most of its content will apply to the dvd test set, while the dvd training set will containg more specific content relating to the visual aspects of a movie, making it slightly less geared towards testing on books.



1.4.9 Section 1.4

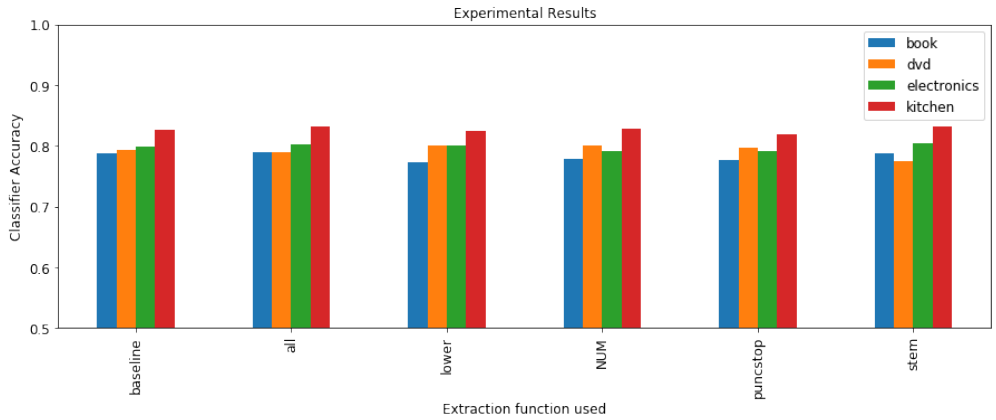
The Naive Bayes classifer was extended with a variety of different simple feature extractors. The 5 feature extractors used were: FE_all, which returns all the words in a review; FE_lower, which returns all words as lowercase; FE_num, which returns numbers as NUM and all other words as they are; FE_puncstop, which filters out stopwords and punctuation; and FE_stem, which stems and returns all the words.

The performance of each feature extractor was tested across each review category and compared with a baseline, averaging results across 5 runs. The results are displayed in the figure below. Firstly, since these feature extractors aren't altering the content of the reviews wildly, the accuracy of each is quite similar. In addition, the FE_all extractor classifier is essentially the same as the baseline classifier, hence their results *should* be almost identical.

Seemingly, the category most susceptible to a change in accuracy was the 'dvd' category. We can see that the lower, num and puncstop feature extractors all give a slight increase in accuracy over the baseline, showing that the normalisation of the tokens can have a beneficial effect on performance; in contrast the stem feature extractor performs worse on the 'dvd' category, showing that perhaps overly normalising the data can be detrimental in certain situations.

On the flipside of this, the 'book' category saw slight decreases in accuracy from lower, num and puncstop, showing the variability of these methods. We can posit that perhaps the lower extractor is removing some important proper nouns from the review content, or that the num extractor is removing any information gained by the size of the number used, but realistically these miniscule differences in the accuracy scores are likely down to variance, so it is hard to read too much into them.

Interestingly though, one might expect the puncstop and stem extractors to give a significant bump in performance, since they are doing more wholesale normalisation, yet they showed no noticeable increase. Part of this could be down to the fact these are internet reviews, and so lack



alt text

the formality of language use of a proper print review; this means that stemming and removing words will be less effective, with incorrect spellings and uncommon words more frequent.

1.5 Section 2: Opinion Extractor (30 marks)

There are 30 marks available for this section. Section 2 should include the following subsections.

1.5.1 Section 2.1 (20 marks)

In this subsection you should present the code defining your opinion extractor together with a clear explanation as to how it works. You only need to cover the five required extensions described in the Topic 6 notebook.

1.5.2 Section 2.2 (10 marks)

In this subsection you should show that for each of the five required extensions, A-E, your opinion extractor produces the desired output when given the corresponding `core` sentence(s), as defined here:

```
core = [("A.1","It has an exciting fresh plot.",set(["fresh", "exciting"])),
        ("B.1","The plot was dull.",set(["dull"])),
        ("C.1","It has an excessively dull plot.",set(["excessively-dull"])),
        ("C.2","The plot was excessively dull.",set(["excessively-dull"])),
        ("D.1","The plot wasn't dull.",set(["not-dull"])),
        ("D.2","It wasn't an exciting fresh plot.",set(["not-exciting", "not-fresh"
        ("D.3","The plot wasn't excessively dull.",set(["not-excessively-dull"])),
        ("E.1","The plot was cheesy, but fun and inspiring.",set(["cheesy", "fun",
        ("E.2","The plot was really cheesy and not particularly special.",set(["rea
       ]
```

1.5.3 Things to note

- Do not write separate opinion extractor functions for each of the different extension. It is important that all of the different extensions are integrated, so that a single opinion extractor

function deals with all of the cases you are covering.

- You should state explicitly whether or not you have been successful in extending your opinion extractor in the ways required.
- Do not forget to include comments in your code where necessary. Split long functions into smaller coherent sub-functions.

```
In [4]: core = [("A.1","It has an exciting fresh plot.",set(["fresh", "exciting"]))
                ("B.1","The plot was dull.",set(["dull"])),
                ("C.1","It has an excessively dull plot.",set(["excessively-dull"])
                ("C.2","The plot was excessively dull.",set(["excessively-dull"])),
                ("D.1","The plot wasn't dull.",set(["not-dull"])),
                ("D.2","It wasn't an exciting fresh plot.",set(["not-exciting", "no
                ("D.3","The plot wasn't excessively dull.",set(["not-excessively-du
                ("E.1","The plot was cheesy, but fun and inspiring.",set(["cheesy",
                ("E.2","The plot was really cheesy and not particularly special.",s
               ]

        optional = [("A","The script and plot are utterly excellent.",set(["utterly
                    ("B","The script and plot were unoriginal and boring.",set(["un
                    ("C","The plot wasn't lacking.",set(["not-lacking"])),
                    ("D","The plot is full of holes.",set(["full-of-holes"])),
                    ("E","There was no logical plot to this story.",set(["no-logica
                    ("F","I loved the plot.",set(["loved"])),
                    ("G","I didn't mind the plot.",set(["not-mind"]))
                   ]
```

```
In [28]: # Section 2.1

         # Main opinion extraction function
         def opinion_extractor(aspect_token,parsed_sentence):
             AFlag = BFlag = CFlag = DFlag = EFlag = 0
             opinions = []
             for token in parsed_sentence:
                 # if token is one of the aspect_words continue
                 if token.pos_ == 'NOUN' and token.text == aspect_token:
                     # decide whether the sentence is a negation, if so build a 'no
                     negation, DFlag = build_negation_string(token, DFlag)
                     # if there are adjectival modifiers on the aspect_word, add th
                     for child in token.children:
                         if child.dep_ == 'amod':
                             AFlag  = 1
                             advmod, CFlag = build_advmod_string(child, CFlag)
                             opinions.append(negation+advmod+child.text)
                     # if aspect_word is part of a verb phrase, work out what has b
                     if token.dep_ == 'nsubj':
```

```
                filterList = set(filter(lambda x: x.dep_ == 'acomp' or x.d
#                   print(list(filterList))
                for x in filterList:
                    BFlag = 1
                    advmod, CFlag = build_advmod_string(x, CFlag)
                    opinions.append(negation+advmod+x.text)
                    EFlag, CFlag = add_conj_opinions(opinions,x,negation,
#     print(opinions)
    return opinions, {'ext A': AFlag, 'ext B': BFlag, 'ext C': CFlag, 'ext

# builds an opinion string from an aspect_word and its adjectives
def build_advmod_string(token, CFlag):
    advmod = ''
    notString = ''
    for child in token.children:
        if child.dep_ == 'advmod':
            CFlag = 1
            advmod += child.text + '-'
        if child.dep_ == 'neg':
            notString = 'not-'
    return notString + advmod, CFlag

# Adds a negation to the start of opinions if the sentence is a negation
def build_negation_string(token, DFlag):
    negation = ''
    if token.dep_ == 'nsubj' or token.dep_ == 'attr':
        for child in token.head.children:
            if child.dep_ == 'neg':
                DFlag = 1
                negation += 'not-'
                break
    return negation, DFlag

# Adds opinions given by conjugation, eg 'the film was funny AND interesti
def add_conj_opinions(opinions, token, negation, EFlag, CFlag):
    for child in token.children:
        if child.dep_ == 'conj':
            EFlag = 1
            advmod, CFlag = build_advmod_string(child, CFlag)
            opinions.append(negation+advmod+child.text)
            EFlag, CFlag = add_conj_opinions(opinions, child, negation, EF
    return EFlag, CFlag
```

My opinion extractor works as follows:

First, it finds one of the aspect words in a sentence. Then, it verifies if the opinion is going to be a negated one, by checking if the aspect word is part of a verb phrase, and then seeing if the root verb of the phrase was negated. After that, the extractor checks if the given aspect token has any adjectival modifiers on it, and if so, it tries to find any additional adverbial modifiers it has using

the build_advmod_string, eg 'the ridiculously stupid plot', building a hyphenated opinion string of 'ridiculously-stupid'. The build_advmod_string function also checks if a negation is a child of the amod dependent, since sometimes the dependency tree places a 'not' as a child of the amod rather than the verb itself. It will then add any of these opinion strings to the set.

Then, the extractor checks to see if the given aspect word was the subject of a verb phrase, eg 'the plot was good', and if so, it finds the amod child of the root verb, again building an advmod string if there are any advmods, eg 'the plot was ridiculously good'. Alongside these checks, the extractor checks if there any any conj dependents of the root verb here, since sometimes conjugations can stem from here, such as 'fun' in E1 'The plot was cheesy, but fun and inspiring'.

Finally, on these verb phrases the extractor checks for any nested conjugations, of the type found in E2: 'The plot was really cheesy and not particularly special', where 'special' is a conj child of 'cheesy'. It finds any of these adjectives nested in 'ands' or other conjugations via recursion, and then again builds a hyphenated string from any advmods present.

The opinion extractor was successfully extended to cover each of the extensions.

```
In [9]: sents = [nlp(entry[1]) for entry in core]
        # print(sents)

        for index,sen in enumerate(sents):
            opinions = opinion_extractor("plot",sen)
            print(opinions)
            if set(opinions) == core[index][2]:
                print('Test set {} passed\n'.format(index+1))
```

```
['exciting', 'fresh']
Test set 1 passed

['dull']
Test set 2 passed

['excessively-dull']
Test set 3 passed

['excessively-dull']
Test set 4 passed

['not-dull']
Test set 5 passed

['not-exciting', 'not-fresh']
Test set 6 passed

['not-excessively-dull']
Test set 7 passed

['fun', 'inspiring', 'cheesy']
Test set 8 passed
```

```
['really-cheesy', 'not-particularly-special']
Test set 9 passed
```

My opinion extractor successfully extracted the opinions for all extensions.

1.6 Section 3: Assessment of Opinion Extractor Performance (40 marks)

There are 40 marks available for this section. Section 3 should include the following subsections.

1.6.1 Section 3.1 (5 marks)

In Section 3, you will be making an assessment of the effectiveness of your opinion extractor on your own personalised random sample of 100 DVD reviews. In this subsection you are asked to give details of the makeup of your personalised sample.

The following code cell contains code that you should use to produce your personalised sample.

NB: you *must* change the first line of the code in this cell so that your own candidate number replaces the number `12345678`

If you do not use your own candidate number to generate the sample then you will receive zero marks for Section 3 of the coursework, i.e. a loss of 40 marks!

Note that the code cell below presumes that the first two code cells of the notebook for Topic 6 have been run giving an appropriate value for `dvd_reviews`.

Once you have inserted your candidate number and then run the cell below, `my_sample` will be a list of 100 randomly chosen sentences, each of which contains at least one of the specified target tokens.

In this subsection, you should report on how many of the sentences in `my_sample` contain each of the target tokens, `"plot"`, `"characters"`, `"cinematography"`, and `"dialogue"`.

```
In [15]: seed(21718984) # you ***MUST*** replace '12345678' with your candidate nu

         num_of_each_type = {"plot":0,"characters":0,"cinematography":0,"dialogue":
         def target_sentence(sentence,target_tokens):
             for token in sentence:
                 if token.orth_ in target_tokens:
                     num_of_each_type[token.orth_] += 1
                     return True
             return False

         target_tokens = {"plot","characters","cinematography","dialogue"}
         sample_size = 100
         my_sample = []
         num_found = 0
         while num_found < sample_size:
             review = random.choice(dvd_reviews)
             parsed_review = nlp(review)
             sentence = random.choice(list(parsed_review.sents))
```

```
            if target_sentence(sentence,target_tokens):
                my_sample.append(sentence)
                num_found += 1

        print(num_of_each_type)
```

```
{'plot': 40, 'characters': 34, 'cinematography': 4, 'dialogue': 22}
```

1.6.2 Section 3.2 (30 marks)

In this subsection, you should present an assessment as to how well your opinion extractor performs on the 100 sentences in `my_sample`.

- You should identify the number of sentences where each of your five extensions applied. Give illustrative examples taken from `my_sample`.
- You should indicate what proportion of the sentences were correctly analysed by the opinion extractor.
  Give illustrative examples from `my_sample`.
- You should report on those cases where the opinion extractor does not produce a desirable output, identifying the reason for the error. Examples of possible reasons are that it is due to a deficiency (that you should describe) in your opinion extractor algorithm, an error or errors made by the part-of-speech tagger, or an error or errors made by the dependency parser.
  Do not restrict yourself to just these types of errors: identify other types of errors as needed. Illustrate each kind of error with examples taken from `my_sample`.

1.6.3 Section 3.3 (5 marks)

In this subsection, you should briefly discuss ways in which your opinion extractor might be improved to overcome the problems that you have observed in Subsection 3.2.

```
In [71]: # Section 3.2
         extension_grand_totals = {'ext A': 0, 'ext B': 0, 'ext C': 0, 'ext D': 0,
         for index,sen in enumerate(my_sample):
             opinion_set = []
             extension_totals = {'ext A': 0, 'ext B': 0, 'ext C': 0, 'ext D': 0, 'e
             for aspect in target_tokens:
                 opinions, extensions = opinion_extractor(aspect,sen)
                 extension_totals = { k: extension_totals.get(k, 0) + extensions.ge
                 opinion_set.append((aspect,opinions))
             extension_grand_totals = { k: extension_grand_totals.get(k, 0) + exter
         #     print('----------------')
         #     print(sen)
         #     print(opinion_set)
         #     print(sorted(extension_totals.items()))
```

```
#     print('----------------\n')
print('*** Total number of sentences hitting each extension: ')
print(sorted(extension_grand_totals.items()))
```

```
*** Total number of sentences hitting each extension:
[('ext A', 36), ('ext B', 11), ('ext C', 9), ('ext D', 2), ('ext E', 2)]
```

1.6.4 Section 3.2

My opinion extractor utilised each of the 5 specified extensions, and below is a summary of how many times across the my_sample test set each extension was used:

[('ext A', 36), ('ext B', 11), ('ext C', 9), ('ext D', 2), ('ext E', 2)]

Naturally, extension A had the most hits, since this is the most common format in which someone would add opinion to a noun, by adding an adjective to it. One such sentence illustrative of this is:

"*special award* for expressionless characters." ('characters', ['expressionless'])

Furthermore, there is mostly a steady decrease in the number of sentences hitting each extension from A-E, as these features are less commonly used in opinion forming, or used in a more complex fashion than my extractor can find. Below is an example of the extractor successfully using extensions B and D to extract an opinion:

"The dialogue isn't hard-boiled except for a few lines delivered by Vicky." ('dialogue', ['not-hard-boiled'])

It correctly works out that hard-boiled is speaking of the dialogue, and then that there is a negation. An example of the extractor successfully extracting an opinion based on extension E is as follows:

"The plot is stupid, the characters are dull and insipid, and the whole movie leaves you asking, as the credits run" ('characters', ['dull', 'insipid', 'leaves'])

While 'leaves' was extracted incorrectly, both dull and insipid were correctly obtained. The incorrect 'leaves' speaks to an overfitting of my extractor, particularly in my construction of extension E, as the 'leaves' verb has a conj dependency from 'was', thus being caught by the extractor.

On the whole my_sample test set, my extractor correctly analysed about 70% of sentences. This figure should be taken with a grain of salt however, since my measure of success was a holistic one, and furthermore many sentences didn't include an opinion, somewhat skewing the data.

Some examples of a mostly correct opinion analysed:

"The story is filled with big laughs–from crude humor to some of the most intelligent dialogue I've probably ever heard." [('dialogue', ['most-intelligent']), ('characters', []), ('cinematography', []), ('plot', [])]

This opinion may be open to debate, as some may feel that 'most' need not be included in the opinion, however I do feel that 'most-intelligent' is more informative and so a success for the extractor.

"I think the acting in the movie was outstanding, the direction was genius, and the plot was amazingly horrific." [('dialogue', []), ('characters', []), ('cinematography', []), ('plot', ['amazingly-horrific'])]

Here the extractor successfully picked out amazingly-horrific as a single opinion, making use of extensions A and B. This example is one that can be easily found by this simple opinion extractor, but at the same time it is definitely an informative opinion to find.

There were many examples however of the opinion extractor not successfully picking up on opinions, for a variety of different reasons; some were deficiencies of the spacy package, others were directly issues with the opinion extractor's implementation.

An example of a problem with the dependency tree built by spacy:

"It has great fun dialogue, is very technically modern for its time…" [('dialogue', ['great']), ('characters', []), ('cinematography', []), ('plot', [])] TODO: get tree from displacy to sew what's wrong here

In this sentence, 'fun' is missed, and this is due to the dependency parser giving 'fun' a dependency tag of compound from the head 'dialogue'. This appears to be a problem with the dependency parser, incorrectly thinking 'fun dialogue' is a compound phrase, when in fact 'fun' should just also be an amod.

An example of a lack of features of the opinion extractor:

"Facing Windows displays several symbolic ideas through terrific mise-en-scene and cinematography…" [('dialogue', []), ('characters', []), ('cinematography', []), ('plot', [])]

Here, the reviewer has expressed that the cinematography was terrific, however the noun phrase terrific is depending on is 'mise-en-scene and cinematography', so the extractor doesn't pick up this opinion. This extension can be implemented without too much trouble and is explained further in section 3.3 below.

Another example:

"There's just no central drive or consistent storyline, and too many non-essential characters." [('dialogue', []), ('characters', ['too-many', 'essential']), ('cinematography', []), ('plot', [])]

In this example, aside from whether spacy dependencies should have grouped 'too many non-essential' into one amod phrase, we see that the extractor did not pick up 'non-' in front of essential. This is due to spacy parsing these as two separate nodes, with 'non-' dependent on 'essential' as an amod. This is another situation that could be easily handled by the extractor with an update.

Another example:

"The plot of the movie is interesting and keeps you wondering what is gonna happen next." [('dialogue', []), ('characters', []), ('cinematography', []), ('plot', ['interesting', 'keeps'])]

In this case, the extractor successfully obtains 'interesting', but it also includes 'keeps'; this is a deficiency with extension E of my extractor. The dependency tag on 'keeps' from the root 'is' is a conj, and so it is picked up by my extractor. This was overfitting on my part to the particular sentences I built the extractor based upon, the conj dependency only showed up on noun phrases that I looked at, but here it is a verb phrase and would need to be handled accordingly, possibly returning 'keeps-you-wondering'.

An example of a deficiency of the part-of-speech tagger influencing the dependency parser:

"The dialogue seems forced, funnier on the script than on the screen and most scenes feel posed and sterile." [('dialogue', []), ('characters', []), ('cinematography', []), ('plot', [])]

In this instance, the use of 'forced' as an adjective is not picked up by the dependency parser as an acomp but an oprd, since 'forced' is tagged by the POS tagger as a verb, in turn meaning that the opinion extractor doesn't find it. This error may be fixed by including searching for oprd dependencies, but there could be other cases like that that haven't been seen, with different dependencies.

Finally, an example of a homonym of the sense of 'plot' that we are interested in:

"All he needs to complete this is a small plot of land." [('dialogue', []), ('characters', []), ('cinematography', []), ('plot', ['small'])]

Here the opinion extractor has found a noun 'plot' to analyse, but it is the wrong sense of 'plot' instead referring to the 'piece of land' meaning of the word. This issue would be harder to account

for, since there needs to be a check on whether the word being searched for is the correct meaning of that word, requiring analysis of its context.

There are many other particular examples of the opinion extractor failing for some reason, but they are too numerous to list and the above gives a good selection. The opinion extractor is far from perfect at this stage, and it has certainly been overfitted somewhat to pass the tests of the example sentences, with slightly specific conditions to find opinions.

1.6.5 Section 3.3

Firstly, there are frequent instances of one of the aspect words being part of a larger conjugated noun phrase, as in the 'terrific mise-en-scene and cinematography' example. It is commonplace to see such phrasology, for example 'the characters and plot were great'; this is not too difficult to account for, by checking if the given word is part of a conjugation, and then seeing what adjective is modifying the whole conjugated phrase. This would most likely be the first improvement I would make to the extractor, since it is simple and covers a lot of cases.

Secondly, noun phrases containing the aspect word, wherein the adjective describing the word has itself dependents (commonly found as hyphenated adjectives), can be caught quite easily. The phrase 'non-essential characters' from above is an example of this; here the 'non-' is missed out, but if the extractor also found the amod dependents of the original amod, i.e 'essential' for the example, it could catch these issues.

A final improvement I might make involves complex conjugate phrases, such as in the example 'The plot of the movie is interesting and keeps you wondering what is gonna happen next'. The 'keeps' token forms the root of a conj dependency from the 'is' root verb, but currently the opinion extractor only finds 'keeps' as the opinion. This issue could be fixed by updating the extractor to construct an opinion from the conjugate phrase and its children, i.e 'keeps-you-wondering' for the example above. One could debate about whether to include the whole 'keeps you wondering what is gonna happen next' as the opinion, however this would be a bit of a long opinion to use for further analysis, and its meaning is mostly captured by 'keeps-you-wondering'.

I used roughly 3,008 words in this report, however this is including my examples of hitting extensions, eg '[('ext A', 36), ('ext B', 11), ('ext C', 9), ('ext D', 2), ('ext E', 2)]' in section 3.2, which aren't really words.

In [ ]: