TEXT ANALYSIS ASSESSMENT: WEB SCRAPING AND CLASSIFICATION USING PYTHON

Suk Jing Lim B00100201@mytudublin.ie

Department of Informatics,
School of Informatics and Engineering,
Technological University Dublin – Blanchardstown Campus

Submitted to Technological University Dublin in partial fulfilment of the requirements for the degree of

Bachelor of Science (Honours) in Computing

Lecturer:

Dr. Aurelia Power

05 April 2020

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, except where otherwise stated. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.

I understand that plagiarism, collusion, and copying are grave and serious offences and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I acknowledge that copying someone else's assignment, or part of it, is wrong, and that submitting identical work to others constitutes a form of plagiarism. I have read and understood the colleges plagiarism policy 3AS08 (available here).

This material, or any part of it, has not been previously submitted for assessment for an academic purpose at this or any other academic institution.

I have not allowed anyone to copy my work with the intention of passing it off as their own work.

Name: _____Suk Jing Lim_____ Dated: _____05/04/2020___.

Table of Contents

1. Introduction

1.1 Background

Digital connectivity has transformed society in many different aspects, especially the way people interact and communicate with each other to obtain information. Due to the portability of smartphones and ease of getting access to the Internet, people will most likely turn to review sites such as IMDb[1] and TripAdvisor[2] with the intent to answer their immediate needs for opinions. According to the United Nations World Tourism Organization[3], UNWTO, urban tourism is more relevant now than ever as such industry plays a pivotal role in determining the economic development of a country. With more consumers relying on reviews to help make their purchase decisions, stakeholders involved in entertainment and leisure businesses should harness the power of data analytics and customer reviews to develop a marketing strategy that can be adapted quickly to stay relevant in today's fast-moving business environment.

1.2 Business Objective

- To classify user-submitted reviews into their respective categories and obtain a more concise representation of consumer opinions by utilizing text analysis techniques.

1.3 Text Mining Objective

- To classify reviews into "movie" and "non-movie" reviews using Support Vector Machine (SVM) and Naïve Bayes (NB) algorithms, given a set of highly unstructured data extracted from multiple websites.
- To identify which specific terms are the most promising predictor of movie reviews.
- To improve the performance of the classification model in terms of accuracy, precision and recall by applying feature selection/reduction techniques.

1 https://www.imdb.com/

2 https://www.tripadvisor.ie/

3 https://www.unwto.org/

2. Data Preparation

2.1 Overview

This is a type of binary classification problem where a review is either classified as **movie_review** or **not_movie_review**. A total of 200 documents representing the positive and negative class are scraped from two main review websites, with the topics and their descriptions summarized in **Table 1**. Rotten Tomatoes[4] is a popular website for cinephiles that allows verified movie critics to submit reviews and ratings, which are then converted into what is known as the "Tomatometer score" to rank a film. On the other hand, The World's 50 Best Restaurants[5] is a website featuring the best places to eat voted annually by 1000 culinary professionals from all around the world.

Table 1: Description of topics

Topics	Class Label	Description	Instances
Movie	Positive: movie_review	Films in a wide variety of genres. Data scraped from Rotten Tomatoes	100
Documentary	Negative: not_movie_review	Non-fictional films that illustrate the reality of today's society. Data scraped from Rotten Tomatoes	50
Restaurant	Negative: not_movie_review	Different types of restaurants and their famous dishes. Data scraped from The World's 50 Best Restaurants	50

2.2 Data Pre-processing

As text data scraped from websites are highly unstructured and polarized, basic cleaning such as discarding **HTML tags** that contain irrelevant texts, as well as removing **control characters** should be performed before writing the data to a comma-separated values (CSV) file. This pre-processing step is essential as it removes noise and organizes data into a format that makes it

4 https://www.rottentomatoes.com/

5 https://www.theworlds50best.com/about

easier to be processed in later steps. As the structure of each website varies significantly, different HTML tags are removed and they are detailed in the tables below. Further data preparation techniques (**section 2.6**) will be applied after the baseline model (**section 2.5**) is built.

```
for soup in (pos_soup + neg_soup):
    for div in soup.find_all('div', id=['movies_sidebar']):
        div.decompose()
    for div in soup.find_all('div', class_=['modal-dialog', 'rate-and-review-widget__module']):
        div.decompose()
    for section in soup.find_all('section', class_=['mop-ratings-wrap__row js-scoreboard-container', 'panel panel-rt panel-box']):
        section.decompose()

pos_tags_to_remove = ['span', 'button', 'script', 'noscript', 'style', 'input', 'table', 'img', 'form', 'nav', 'header', 'footer']
neg_tags_to_remove = ['span', 'button', 'script', 'noscript', 'style', 'input', 'table', 'img', 'form', 'nav', 'header', 'footer', 'iframe', 'hr', 'br']
```

Positive Class

Tags	Usage	Discarded
span	Contain all the film ratings – not predictive and specific to Rotten Tomatoes	Yes
button, nav, header, footer	Contain action verbs such as login, signup	Yes
script, noscript, style	Contain JavaScript source code or point to an external client-side script file	Yes
input, form	Contain input prompts that allow users to enter personal information	Yes
table	Display movies and their respective ratings in a tabular form, irrelevant to the mining goals	Yes
img	Define an image or graphic source	Yes
p, i, h2, em	Contain title, synopsis, and cast/crew information of a movie – high information value	No
div, section	Used as a container that encapsulates other HTML tags that contain a lot of information – only certain divs/sections are removed (e.g. news and videos section) by specifying their class names	No
ul, li, a	The and tag are used together to create an unordered list that encapsulate all the links <a> containing keywords specific to the mining goals such as DVD and TV	No

Negative Class

Tags	Usage	Discarded
span	Group names of the actors and actresses, but also contain the ratings and social media tags – not predictive and specific to the websites	Yes
button, nav, header, footer	Contain action verbs such as login, signup, subscribe	Yes
script, noscript, style	Contain JavaScript source code or point to an external client-side script file	Yes
input, form	Contain input prompts that allow users to enter personal information	Yes
table	Display movies and their respective ratings in a tabular form, irrelevant to the mining goals	Yes
img	Define an image or graphic source	Yes
p, i, h1, h2, h3, em, strong	Contains name and information of a documentary/restaurant – high information value	No
div, section	Used as a container that encapsulates other HTML tags that contain a lot of information – only certain divs/sections are removed (e.g. news and videos section) by specifying their class names	No
ul, li, a	The and tag are used together to create an unordered list that encapsulate all the links <a> containing keywords specific to the mining goals such as Documentary	No
iframe	Display a frame within a browser – typically for video-based ads	Yes
hr, br	Used for adding a break or separating contents	Yes

2.3 Describe the Corpus

Based on the summary of the descriptive statistics generated by data.describe(), the corpus is made up of two columns, i.e. **document** and **label**. The corpus used contains 200 observations on three different review topics, namely positive **movie** class and negative **documentary/restaurant** class, indicated by the **count** attribute. **Label** has two possible values (**movie_review** and **not_movie_review**), whereas **document** is made up of 200 unique values. The modes, given by the **top** attribute, are picked at random for both columns as the document is multimodal, i.e. the document is made up of unique instances that only occur once each and there are 100 instances for each class label.

	document	label
count	200	200
unique	200	2
top	Steirereck | The World's 50 Best Restaurants 2...	not_movie_review
freq	1	100

2.4 Describe the WordCloud

WordCloud is generated as an efficient way to visualize the corpus obtained in order to identify potential stop words. The size of each word shown in the plot indicates its frequency in the corpus, where frequently occurring terms are represented in larger font sizes and vice versa.

```
plt.figure(figsize=(10, 5)).canvas.set_window_title('WordCloud')
for i in range(len(docs)):
    cloud = wordcloud.WordCloud(width=800, height=800, background_color='black', min_font_size=10).generate(docs[i])
    plt.subplot(1, 2, i + 1)
    plt.title(labels[i])
    plt.imshow(cloud)
    plt.axis('off')
plt.show()
```

As shown in the WordCloud, it can be concluded that words such as **rent**, **buy**, **super**, **reviewer**, **binge**, **central**, **rotten** and **tomatoes** are common words across both classes and they will be removed in **section 2.6**. Besides that, some promising predictors can also be gathered from the plot: **movie, character**, **users**, **documentary**, **audience**, **chef**, **restaurant** and **menu**.

2.5 Baseline Models

Before any further cleaning/normalisation is performed (except for pre-processing in **section 2.2**), baseline matrices of three types are established: **simple frequency counts**, **TF** and **TF-IDF**. Each matrix contains 200 observations, 11942 regular attributes representing features extracted from the corpus, and one special feature functioning as the class label [200 rows x 11943 columns]. Two types of supervised machine learning algorithms, i.e. the Support Vector Machine (SVM) and Naïve Bayes (NB) algorithm are used to train the baseline models (detailed in **section 4.1**). The accuracy, recall and precision for the baseline matrices trained/tested using **Split Validation** are summarized in the table below:

Model	**Baseline SVM – Split Validation**			**Baseline NB – Split Validation**		
Matrix	**Count**	**TF**	**TF-IDF**	**Count**	**TF**	**TF-IDF**
Accuracy	0.90	0.45	0.88	0.72	0.70	0.70
Precision	0.89	0.45	0.78	0.62	0.60	0.60
Recall	0.89	1.00	1.00	1.00	1.00	1.00

As shown in **Figure 1**, high **recall** for all three matrices indicates that the models are returning a majority of all positive results. **NB** is a more balanced model overall, even though **SVM** generally has higher accuracy and precision. Low **precision** indicates that there are a high number of false positives, i.e. documentary/restaurant review labelled as movie review. The corpus has more than 10000 features, and as a result the classifiers tend to overfit to training sets and cause generalization issue. The **SVM** model using the **TF matrix** performs poorly as it has low accuracy and precision, which can be fixed by tuning the **kernel** parameter in **section 4.3**.

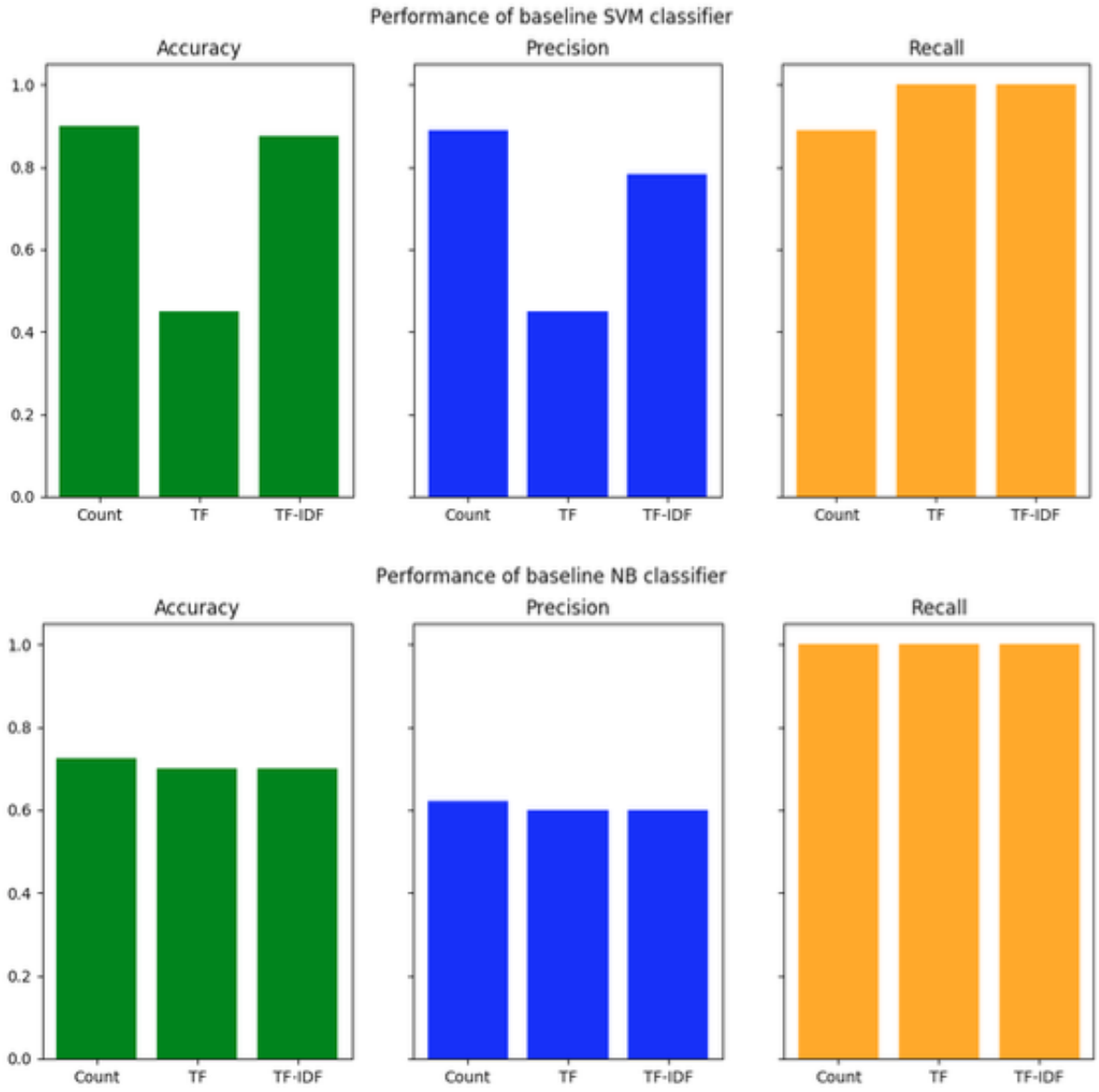


Figure 1: Performance of baseline classifiers

2.6 Cleaning/Normalization

As the corpus contains a massive number of columns, features that are not optimal for analysis should be replaced or removed, i.e. normalized. A few of the Natural Language Processing (NLP) techniques are used repeatedly throughout this process and they are as follows:

- **Tokenization:** split a document into a list of tokens.

```
tokenized_docs = [nltk.word_tokenize(doc) for doc in all_docs]
```

- **Part-of-Speech (POS) Tagging:** assign specific lexical categories to words based on the contexts in which they occur.

```
tagged_docs = [nltk.pos_tag(doc) for doc in tokenized_docs]
```

- **Named-Entity Recognition (NER):** identify proper nouns (NNP) in certain categories such as person names and locations.

```
tokenized_docs = utility.remove_pos_tags(tagged_docs)
```

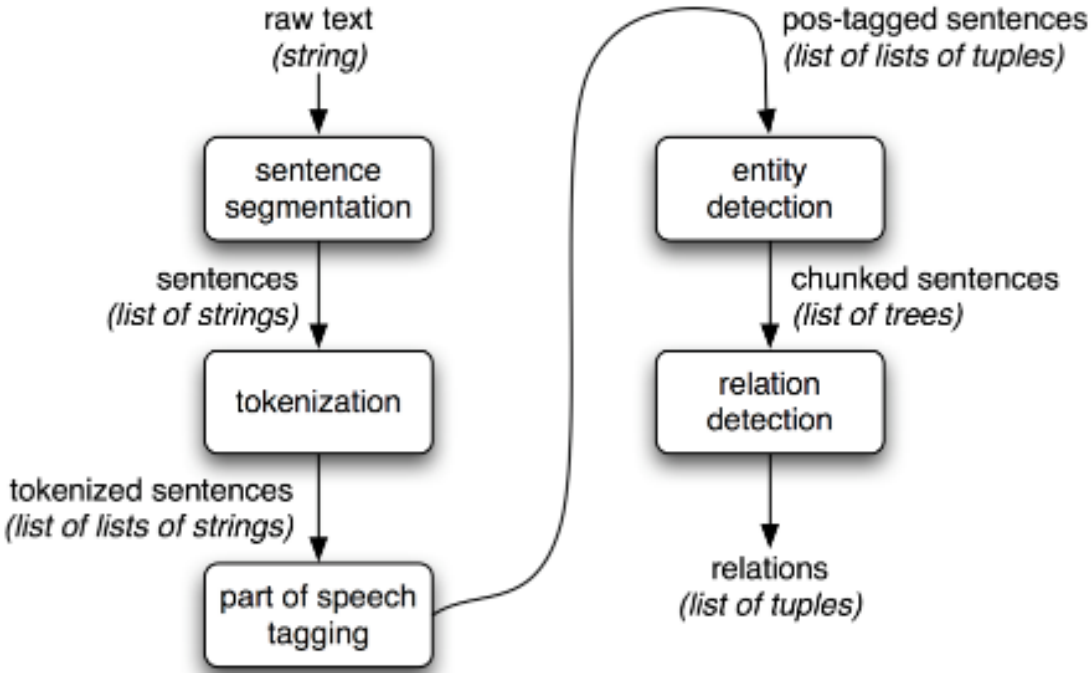


Figure 2: Flow chart of text processing

Digits and Punctuations

- A quick glance at the **sorted attributes** reveals that there is a significant amount of **numbers**. Further verification using a frequency counter shows that there is a total of 527 numbers.

```
sorted_attributes = sorted(baseline_matrix_list[0].columns)
```

```
'00', '000', '0020', '005', '0062', '01', '0110', '012', '0173', '019',
'020', '022', '024', '026', '03', '033', '036', '037', '038', '04',
'042', '0460', '049', '0498', '05', '052', '054', '058', '06', '0607',
'06500', '0799', '08', '080', '08015', '08036', '084', '09', '097',
'10', '100', '10010', '10019', '101', '10120', '102', '103', '1030',
```

```
count = 0
for attribute in sorted_attributes:
    if re.match(r'\d+.*', attribute):
        count += 1
```

- **Replace 4-digit numbers** that are valid years (either 19xx or 20xx) with "**year**", the reason being that they serve as a very promising predictor of the movie_review class, i.e. year of release. **Remove** the remaining numbers because they contain no useful information.

```
docs = [re.sub(r'\(*(19|20)\d{2}\)*', 'YEAR', doc) for doc in docs]
docs = [re.sub(r'\d+(.|\w*)', '', doc) for doc in docs]
```

Operation	Original	Cleaned
Replace 4-digit numbers	Once Upon a Time in Hollywood (2019)	Once Upon a Time in Hollywood year
Remove other numbers	The percentage of users who rated this 3.5 stars or higher.	The percentage of users who rated this stars or higher.

- **Remove punctuations** and **replace multiple spaces** with a **single space** because they are meaningless. The characters included in the punctuation regex are given by **string.punctuation**: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~)

```
docs = [re.sub(r'[!"#$%&\'()*+,-./:;<=>?@[\]^_`{|}~]+', '', doc) for doc in docs
docs = [re.sub(r'\s{2,}', ' ', doc) for doc in docs]
```

Operation	Original	Cleaned
Remove punctuations	authorities decided to "…let the fire burn."	authorities decided to let the fire burn
Replace multiple spaces	uncover a troubling and still deeply resonant chapter	uncover a troubling and still deeply resonant chapter

Stop Words

- After tokenizing the corpus, **remove** a token if it exists in the list of stop words imported from the **NLTK.corpus module**. Printing out the list of stop words verifies that it mostly consists of **function words**, such as pronouns, articles, auxiliary verbs and quantifiers; these types of words play a significant role in sentiment analysis, but not in this particular classification task as they add very little information value, if any, to improve the classification model.

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd",
 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers',
 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which',
 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been',
 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if',
 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between',
 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why',
 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',
 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
 "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't",
 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

- Looking at the WordCloud generated in **section 2.4**, the documents belonging to the positive **movie** and negative **documentary** have a lot of words in common, largely due to the fact that they are both crawled from the same website. **Remove a custom list of stop words** such as **watch**, **rent** and **buy** that occur too frequently in both documents as they will hinder the ability of the model to distinguish topics from one another.

```
stop_words = stopwords.words('english')  # NLTK
custom_stop_words = ['rotten', 'tomatoes', 'harry', 'potter', 'ranked', 'worst', 'best', 'tomatometer', 'view',
                     'guides', 'binge', 'central', 'springsummer', 'tv', 'guide', 'rt', 'news', 'indie', 'fresh',
                     'list', 'women', 'fateful', 'journeys', 'aspiring', 'sommelier', 'holy', 'schnikes',
                     'tommy', 'boy', 'still', 'hilarious', 'years', 'later', 'tickets', 'showtimes', 'trending',
                     'netflix', 'series', 'free', 'add', 'article', 'super', 'reviewer', 'ratings', 'reviews',
                     'explanation', 'watch', 'rent', 'buy', 'online', 'disney', 'see', 'discstreaming']  # custom
docs = [[token for token in doc if token.lower() not in (stop_words + custom_stop_words)] for doc in tokenized_docs]
```

Operation	Original	Cleaned
Remove NLTK stop words	[…'unlike', 'some', 'of', 'its', 'neighbouring', 'restaurants'…]	[…'unlike', 'neighbouring', 'restaurants'…]
Remove custom stop words	[…'watch', 'Rent', 'Buy', 'Movie', 'Info'…]	[…'Movie', 'Info'…]

Part-of-Speech (POS) Tags

- **Filter tokens** tagged as Proper Noun (NNP) and **concatenate** two of the consecutive tokens using an underscore. Proper nouns have to be dealt with because there is plenty of **named entities** in the documents, such as **actors/actresses** (movie), **film titles** (documentary), and **country/city names** (restaurant).

```
for doc in tagged_docs:
    groups = groupby(doc, key=lambda x: x[1])  # group by tags
    # named entities e.g. film titles and city names
    entities = [[t for t, _ in tokens if t != ''] for tag, tokens in groups if tag == 'NNP']
    entities = [('_'.join(entity), 'NNP') for entity in entities if len(entity) == 2]
    lists = [(token, tag) for (token, tag) in doc if tag != 'NNP'] + entities
    new_tagged_docs.append(lists)
```

Operation	Original	Cleaned
Concatenate names of actors/actresses	[…'Leonardo, 'DiCaprio', 'Brad', 'Pitt' …]	[…'Leonardo_DiCaprio', 'Brad_Pitt' …]
Concatenate film titles	[…'Toy', 'Story'…]	[…'Toy_Story'…]
Concatenate country/city names	[…'Los', 'Angeles…]	[…'Los_Angeles'…]

- **Remove tokens** listed in the table below because they are not relevant to the classification task. Verbs are removed because nouns (especially Proper Nouns) have a greater effect on the end task, i.e. produce better accuracy.

```
remove_list = ['CC', 'CD', 'DT', 'IN', 'MD', 'POS', 'PRP', 'PRP$', 'RP', 'TO', 'WDT', 'WP', 'WP$', 'WRB']
new_tagged_docs = [[token for (token, tag) in doc if tag not in remove_list] for doc in new_tagged_docs]
```

Tags	Part of Speech	Examples of Tokens
CC, CD	Coordinating Conjunction, Cardinal Number	either, one, six
DT	Determiner	another, every
IN	Preposition	around, although, onto
MD	Modal / Auxiliary Verb	could, may, might
POS, PRP, PRP$	Possessive Ending, Personal and Possessive Pronoun	', us, h, Godfather (mislabeled)
RP	Particle	away, back
TO	To	to, gonna (tokenized into 'gon', 'na')
VB, VBD, VBG, VBN, VBP, VBZ	Verb: Base Form, Past Tense, Gerund/Present Participle, Past Participle, Non-3rd/3rd Person Singular Present	make, kept, struggling, given, force, knows
WDT, WP, WP$, WRB	Wh-determiner, Wh-pronoun, Possessive Wh-pronoun, Wh-adverb	whatever, wouldnt, whose, well-worn (mislabeled)

Filter: Token Length

- **Remove** remaining tokens that are **shorter than 4 characters** because they are not likely to be predictive. Only keep tokens longer than 15 characters if they are newly concatenated tokens from the previous step to reduce the probability of having misspelt words in the corpus.

```
tokenized_docs = [[token for token in doc if 4 <= len(token) <= 15 or (len(token) > 15 and '_' in token)]
                  for doc in tokenized_docs]
```

Token longer than 15 characters that are removed:

```
['africanamericans', 'antiestablishment', 'characterisation', 'characterization',
 'characterizations', 'chefentrepreneurs', 'climatecontrolled', 'collectionimages',
 'comingofagehighschool', 'counterarguments', 'craftprojectturnedtoy', 'designerturnedchef',
 'differentiations', 'disturbingviolent', 'dramaticthriller', 'firsttimedirector',
 'gnatsattentionspan', 'httpcinephilecrocodileblogspotcoukyearhitchcocktruffautdirkentjonesyearhtml
 ', 'importerexporter', 'improbabilicious', 'ingredientdriven', 'ingredientfocused', 'label',
 'lowermiddleclass', 'millennialappropriate', 'mistakenidentity', 'multigenerational',
 'noholdsbarredzeroexception', 'peruvianjapanese', 'sensationalistic', 'situationslanguage',
 'southoftheborder', 'strangerthanfiction', 'thoughtprovoking', 'tormentorinterrogator',
 'underdevelopment', 'underrepresented', 'underwhelmingness', 'wellchoreographed',
 'willyoufeedthishungrychild']
```

3. Vectorization

3.1 Feature Extraction

As one of the algorithms used is the **Support Vector Machine**, it is essential to convert the corpus into a two-dimensional feature space (e.g. Pandas DataFrame[6] where instances are represented as rows and features represented as columns) as the model only accept numerical data. This process of representing a corpus as vectors is known as feature extraction; it is an essential step in text analysis as it reduces the dimensionality of the data without losing important information.

- **Simple Frequency Count:** a matrix of token counts generated using CountVectorizer. All NaN values are replaced with 0.

```
vectorizer = CountVectorizer(decode_error='replace', strip_accents='unicode', lowercase=True)
X = vectorizer.fit_transform(docs)  # transform the data and return the term-document matrix
features = list(vectorizer.get_feature_names())
count_matrix = pd.DataFrame(X.toarray(), columns=features)
return count_matrix.fillna(0)
```

- **Normalised Count:** a matrix of normalised frequencies generated by applying **Term-Frequency (TF) weighing** to each cell, where frequent terms are given greater weight. The formula is listed below:

$$TF = \frac{number\ of time\ a\ term\ occus\ in\ document\ A}{total\ number\ of\ terms\ in\ document\ A}$$

```
def compute_doc_length(matrix):
    return np.sum(matrix, axis=1)  # horizontally across columns


def generate_tf_matrix(docs):
    count_matrix = generate_count_matrix(docs)
    doc_length = compute_doc_length(count_matrix)
    return count_matrix.divide(doc_length, axis=0)  # vertically across rows
```

- **TF-IDF:** a term-document matrix generated by applying TF-IDF weighing to each cell. Instead of looking at a single document, **Inverse Document Frequency (IDF) weighting** takes into consideration the entire corpus, where infrequent terms are given greater weight because they might be predictive of a certain class. The formulas for IDF and TF-IDF are listed below:

6 https://pandas.pydata.org/pandas-docs/stable/getting_started/dsintro.html

$$IDF = log\left(\frac{total\ number\ of documents\ in\ the\ corpus}{number\ of\ documents\ where\ term\ A\ occurs}\right)$$

$$TF - IDF = TF\ x\ IDF$$

```
def compute_idf(matrix):
    doc_frequency = matrix[matrix > 0].count(axis=0)  # vertically across rows
    return np.log2(len(matrix) / doc_frequency)


def generate_tfidf_matrix(docs):
    vectorizer = TfidfVectorizer(decode_error='replace', strip_accents='unicode', lowercase=True)
    X = vectorizer.fit_transform(docs)  # transform the data and return the term-document matrix
    features = list(vectorizer.get_feature_names())
    tfidf_matrix = pd.DataFrame(X.toarray(), columns=features)
    return tfidf_matrix.fillna(0)
```

Notice in both vectorizers all characters are converted to lowercases before tokenization. The number of features after each operation is summarized in the table below:

Operation	Number of Features
Baseline	11943 features
Cleaning/Normalizing	7604 features
Feature Reduction	7417 features
Feature Selection	12 features

3.2 Feature Reduction

Filter: Sorted Attributes

- Reinspect the **sorted attributes** to identify suitable processing techniques/tasks after vectorisation. **Discard columns** that contain newly concatenated tokens from **section 2.6** that are invalid, or misspelt words from the corpus.

```
sorted_attributes = sorted(matrix_list[0].columns)
```

```
attribute_list = ['_focusing', 'a_la', 'academyaward_r', 'africanamerican', 'americanitalian', 'arpege_', 'cove_oscar', 'c_zeitgeist',
                  'emotiona', 'heaven_im', 'hypersustainable_septime', 'japaneseindian', 'lebaneseportuguese_za', 'menton_italianfrench',
                  'mexican_japanesemediterranean', 'morocco_im', 'movies_movies', 'otooles', 'oh_diana', 'oh_im', 'olivia_wide',
                  'p_spiderman', 'part_journal', 'rating_pgfor', 'regardless_id', 'samantha_boyhood', 'studio_aabout', 'vietnam_throughout',
                  'wwi_diana', 'wwi_hey', 'wwi_im', 'yeah_im', 'yes_holland', 'yet_pinocchio', 'aboveearth', 'acrosstheboard', 'againwhich',
                  'allnew', 'answersand', 'armsbonus', 'asthetic', 'bodyshazam', 'brotherstoby', 'btoob', 'caviarwatch', 'centerfont',
                  'centurys', 'challengemade', 'charachters', 'childrens', 'citys', 'civilizationat', 'coalitionsact', 'communitys',
                  'companys', 'crisisthe', 'dissapointong', 'emotionnal', 'endyet', 'escalatedand', 'everythingi', 'facessome', 'familys',
                  'filmatms', 'forebearsa', 'fourtime', 'franchiseand', 'friendsmedgar', 'fthe', 'fullon', 'genrey', 'goingson',
                  'halfcomanche', 'handson', 'hattrickalbeit', 'hattyin', 'hcentury', 'hehi', 'hfloor', 'historyc', 'husbandsatm',
                  'ighostbustersi', 'ilaurai', 'imuhammad', 'istar', 'ithe', 'kamebish', 'mens', 'migrationan', 'neighborsharon',
                  'noreservations', 'nostalgiatinged', 'oftforgotten', 'particualrly', 'peformance', 'personailties', 'philosphy',
                  'plushjoin', 'procedral', 'pwace', 'pyschiatrists', 'schoolthe', 'sciencey', 'selo', 'shhhhh', 'shitzillion',
                  'showstealingly', 'silouhette', 'sizefacecentury', 'slavesugh', 'sohurray', 'standardstobe', 'standardstobe', 'suitand',
                  'suitwhy', 'teame', 'teenagerboth', 'threemichelin', 'tidbits', 'tmeteri', 'togethera', 'tosend', 'transistion',
                  'turbulant', 'unfortuante', 'villians', 'warfareno', 'warhow', 'wayas', 'wery', 'weve', 'wifes', 'wordshazamthis',
                  'worldwith', 'yearor', 'yearyear', 'yearyearish', 'yuba', 'consensus_capturing', 'consensus_favourite',
                  'consensus_moonlight', 'consensus_spotlight', 'contact_building', 'contact_franko', 'contact_heumarkt', 'contact_largo',
                  'contact_plaza', 'diy_blt', 'doc_ock', 'el_bulliinfluenced']
matrices = [matrix.drop(attribute_list, axis=1) for matrix in matrices]
```

Operation	Number of Features Before Reduction	Number of Features After Reduction
Discard invalid / misspelt features	7604	7455

Correlation Matrix

- Given that the corpus is large, **correlation matrices** are particularly useful when it comes to removing irrelevant features from the dataset; highly correlated attributes are typically removed as they are similar in behaviour. A positive correlation coefficient indicates that both attributes change in the same direction, whereas a negative correlation coefficient indicates that both attributes change in opposite directions. Correlation value ranges from -1.0 to +1.0, the higher the values the stronger the correlations. Features that have total correlations of more than 0.70 to the term **movie**, **documentary**, and **restaurant** are removed.

```
corr = matrices[0].corr().abs()  # correlation matrix
corr_m = corr['movie']
corr_d = corr['documentary']
corr_r = corr['restaurant']
correlated_features = corr_m[corr_m > 0.7] + corr_d[corr_d > 0.7] + corr_r[corr_r > 0.7]
matrices = [matrix.drop(correlated_features, axis=1) for matrix in matrices]
```

As shown in **Figure 3**, it can be concluded that **abstract** is perfectly correlated to **accuracy** and **accept** to **accidentally**. For demonstration purposes, the correlation matrix of the first 40 features is visualized using **seaborn.heatmap**[7] library:

```
demo_matrix = matrices[0].iloc[:, :40]
plt.figure(figsize=(10, 8)).canvas.set_window_title('Correlation Matrix')
plt.title('Correlation Matrix of First 40 Features')
demo_corr = demo_matrix.corr()
sns.heatmap(demo_corr, cmap="YlGnBu", linewidths=0.1)
plt.show()
```

[7] https://seaborn.pydata.org/generated/seaborn.heatmap.html

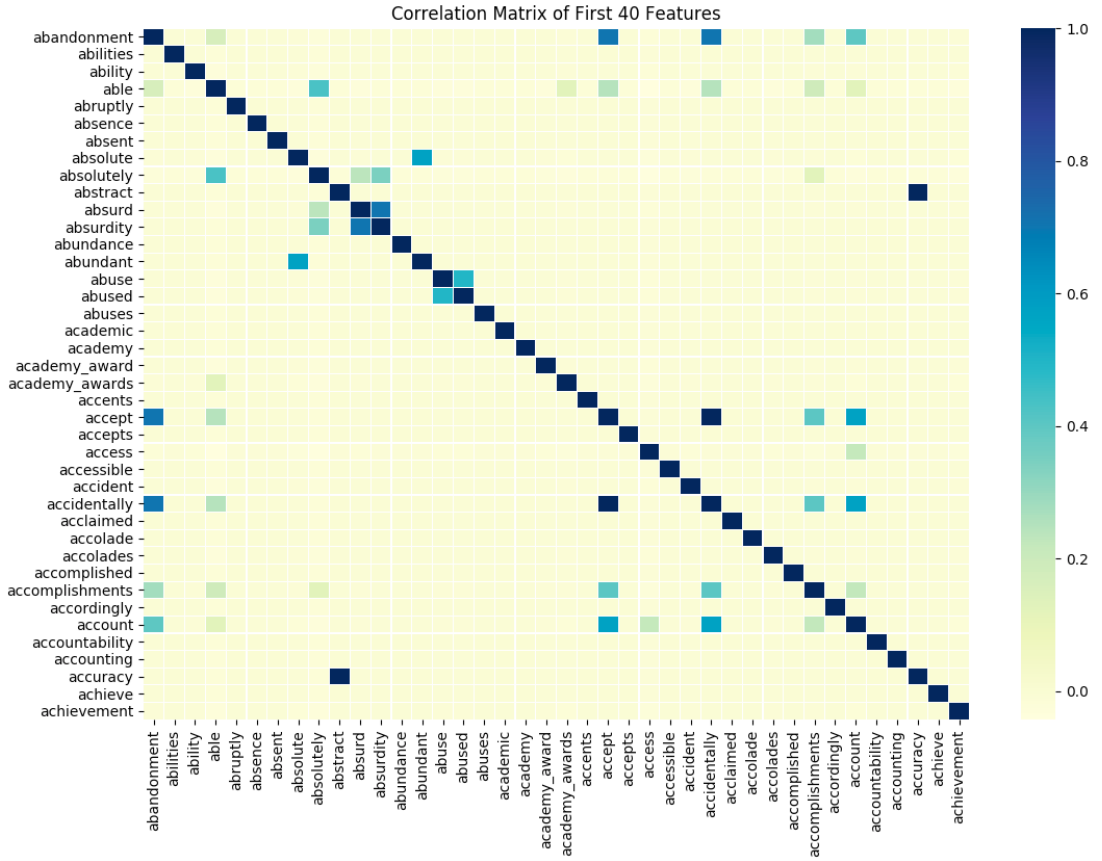


Figure 3: Correlation Matrix

Operation	Number of Features Before Reduction	Number of Features After Reduction
Discard highly correlated features	7455	7417

Feature Selection

- Given that test runs of other feature reduction techniques do not show significant performance improvement, Decision tree is used to transform the corpus into a subset of 12 best features (detailed in **section 4.4**).

```
dtc = DecisionTreeClassifier(random_state=1)
dtc = dtc.fit(X, y)
model = SelectFromModel(dtc, prefit=True)
X_reduced = model.transform(X)
```

Operation	Number of Features Before Selection	Number of Features After Selection
Selects best features	7417	12

4. Text Mining / Modelling

4.1 Select Modelling Techniques

As the text mining objective is to classify consumer reviews, a supervised classification model is required to assign each review to one of the predefined classes, i.e. **movie_review** (positive) or **not_movie_review** (negative), based on the selected features. As the output data type is binomial, the **Support Vector Machine (SVM)** and **Naïve Bayes (NB)** appear to be the most suitable algorithms, as they generally work well with text classification problems. In order to maximize the performance of the models, irrelevant and redundant features have been removed, the corpus is represented as count, TF and TF-IDF matrices, which are then further reduced to a subset of 12 using feature selection as detailed in **section 2.2**, **2.6**, and **3.2.**

4.2 Generate Test Design

The dataset has to be split into two subsets—one subset is used for training the model, while the other subset is used to evaluate the performance of the classification model in terms of its **accuracy**, **precision** and **recall**. Supervised learning algorithms will be used to discover patterns from the **training data set**, produce a classification model, before moving on to apply the model to the **test data set** to assign predefined labels to each instance. When it comes to generating a training/test dataset, the **Cross Validation** method appears to be a better approach as it allows the model to train recursively on multiple train/test splits, as opposed to the **Split Validation** method which relies on a single split. The steps taken to generate the training/test data set are summarized as follows:

- Split each matrix into features (X) and label (y) column.

```
Xs, ys = utility.split_feature_label(matrix_list)
for X, y in zip(Xs, ys):
    print(X.shape, y.shape)
```

- Further split each X/y matrix into training and test sets: **X_train** [0], **X_test** [1], **y_train** [2], **y_test** [3]. A test_size of 0.2 will randomly select 80% of the instances for training (160), and the remaining 20% for testing (40).

```
sets = []
for X, y in zip(X_list, y_list):
    sets.append(train_test_split(X, y, test_size=0.2, random_state=1))
return sets
```

- Use **Split Validation** to train the baseline and final model.

```
scores = []
for i in range(len(matrices)):
    model.fit(sets[i][0], sets[i][2])  # fit the model according to the given training sets (X_train, y_train)
    y_pred = model.predict(sets[i][1])  # perform classification on the given test set (X_test)
    accuracy = acc(y_true=sets[i][3], y_pred=y_pred)
    precision = pr(y_true=sets[i][3], y_pred=y_pred, pos_label=pos_label)
    recall = rec(y_true=sets[i][3], y_pred=y_pred, pos_label=pos_label)
    scores.append((accuracy, precision, recall))
    if print_:
        print('------------', matrices[i], '-------------')
        print('Split Validation Accuracy: %0.2f' % accuracy)
        print('Split Validation Precision: %0.2f' % precision)
        print('Split Validation Recall: %0.2f' % recall)
return scores
```

- Use **Cross Validation** to repeatedly (set to 5 folds) train the algorithm on the training set in order to optimize the model.

```
scores = []
for i in range(len(matrices)):
    model.fit(sets[i][0], sets[i][2])  # fit the model according to the given training sets (X_train, y_train)
    # perform classification on the given test set (5 iterations)
    accuracy = xval(model, sets[i][0], sets[i][2], cv=folds).mean()
    precision = xval(model, sets[i][0], sets[i][2], cv=folds, scoring='precision_macro').mean()
    recall = xval(model, sets[i][0], sets[i][2], cv=folds, scoring='recall_macro').mean()
    scores.append((accuracy, precision, recall))
    if print_:
        print('------------', matrices[i], '-------------')
        print('Cross Validation Accuracy: %0.2f' % accuracy)
        print('Cross Validation Precision: %0.2f' % precision)
        print('Cross Validation Recall: %0.2f' % recall)
return scores
```

4.3 Build, Assess and Tune the Model

Support Vector Machine:

The **Support Vector Classifier (SVC)** from the scikit-learn's SVM[8] library is utilized to build the SVM model. To begin with, set the **kernel** parameter to linear, **C** to 1.0, **random_state** to 1 and **gamma** to scale. Such a combination gives an accuracy of 51% when the TF and TF-IDF matrices are used, which is not optimal as the data might not be linearly separable. One

8 https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

way to improve the performance is to set the **kernel** parameter to radial basis function (rbf) to transform the data into a non-linear format. The accuracy, recall and precision for the matrices before and after parameter tuning (**trained/tested using Cross Validation**) are summarized in the table below:

Model	**Reduced SVM – Cross Validation**			**Tuned SVM – Cross Validation**		
Matrix	**Count**	**TF**	**TF-IDF**	**Count**	**TF**	**TF-IDF**
Accuracy	0.93	0.51	0.51	0.96	0.87	0.92
Precision	0.94	0.26	0.26	0.96	0.87	0.92
Recall	0.93	0.50	0.50	0.96	0.87	0.92

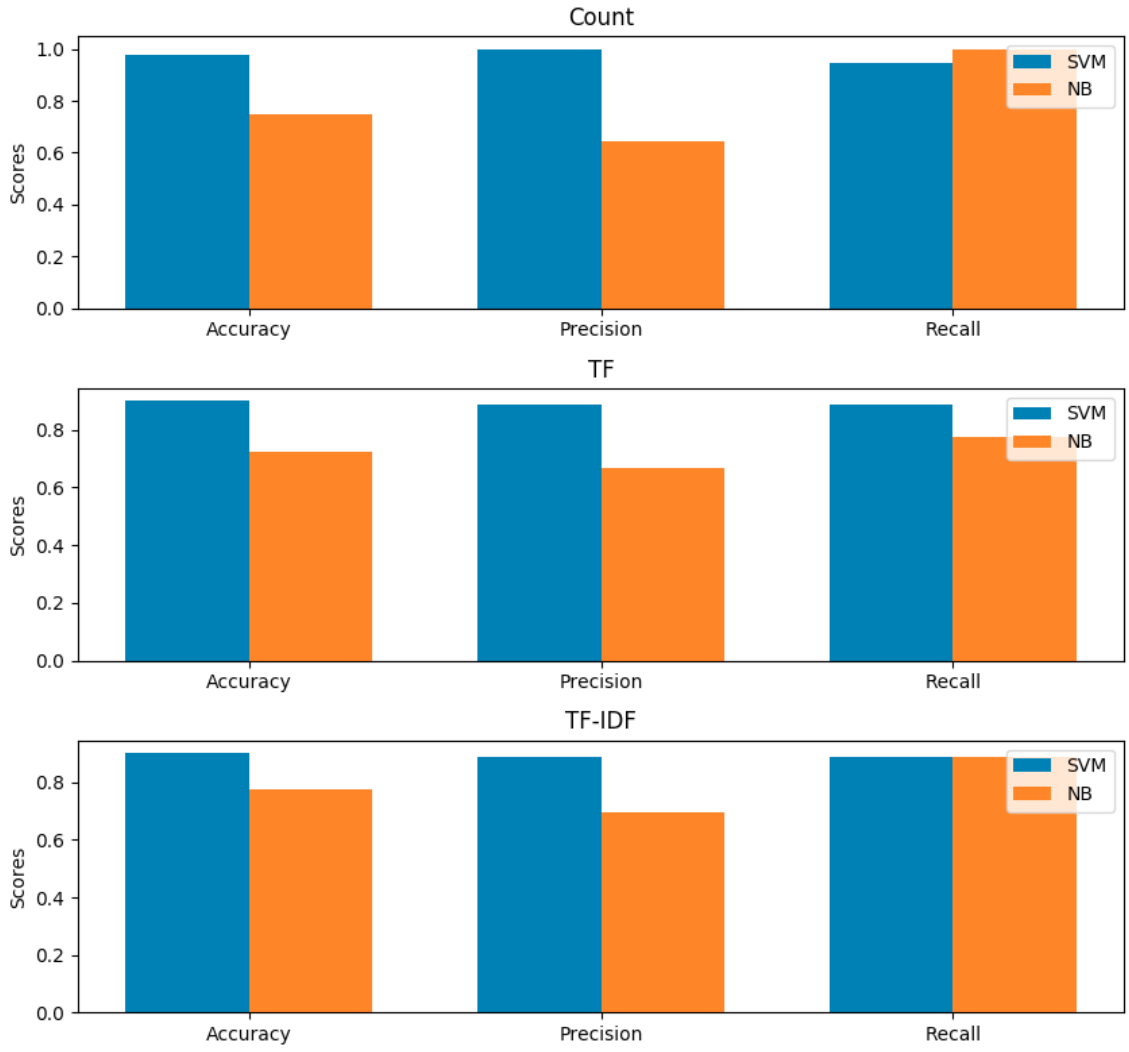
Naïve Bayes:

The **Complement Naive Bayes (ComplimentNB) Classifier** from the scikit-learn's naïve_bayes[9] library is utilized to build the NB model. To begin with, set the **alpha** parameter to 0.0, **fit_prior** to True, **class_prior** to None and **norm** to False. Such a combination increase the precision from baseline 60% to 80% when the TF and TF-IDF matrices are used, which is optimal as the number of false positives decreases. Smoothing the NB model by setting the **alpha** parameter to 1.0 does not seem to increase the performance. The accuracy, recall and precision for the matrices before and after parameter tuning (**trained/tested using Cross Validation**) are summarized in the table below:

Model	**Reduced NB – Cross Validation**			**Tuned NB - Cross Validation**		
Matrix	**Count**	**TF**	**TF-IDF**	**Count**	**TF**	**TF-IDF**
Accuracy	0.70	0.78	0.78	0.68	0.68	0.74
Precision	0.79	0.80	0.81	0.77	0.81	0.82
Recall	0.69	0.77	0.77	0.67	0.67	0.74

9 https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html

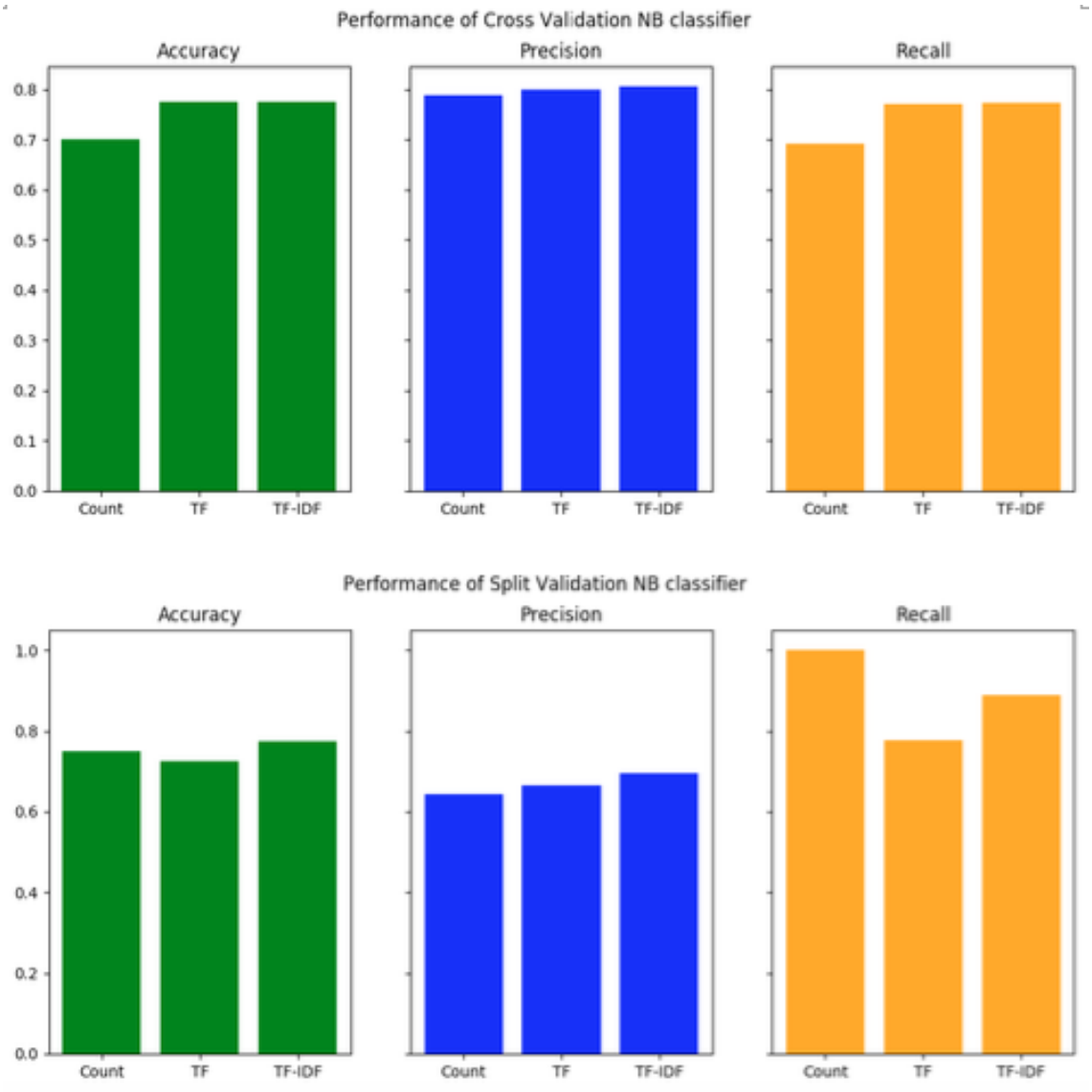
Evaluating the **accuracy** is often enough for classification model that predict classes of equal importance; however, some of the classes have a greater impact than the other and it might be beneficial to take into account both **precision** and **recall** for more effective performance evaluation. In this particular classification task where **movie_review** is the positive class and **not_movie_review** is the negative class, a **true positive** represents the outcome where the model correctly predicts a movie review, whereas a **true negative** represents the outcome where the model correctly predicts a documentary/restaurant review. It is often a trade-off between precision and recall; higher recall indicates a higher number of true positives, whereas a higher precision indicates a higher number of true negatives.



After normalisation and features reduction/selection, performances of both models increase significantly from the baseline, especially **SVM**.

Model	**Baseline SVM – Split Validation**			**Baseline NB – Split Validation**		
Matrix	**Count**	**TF**	**TF-IDF**	**Count**	**TF**	**TF-IDF**
Accuracy	0.90	0.45	0.88	0.72	0.70	0.70
Precision	0.89	0.45	0.78	0.62	0.60	0.60
Recall	0.89	1.00	1.00	1.00	1.00	1.00
Model	**Final SVM – Split Validation**			**Final NB - Split Validation**		
Matrix	**Count**	**TF**	**TF-IDF**	**Count**	**TF**	**TF-IDF**
Accuracy	0.97	0.90	0.90	0.75	0.72	0.78
Precision	1.00	0.89	0.89	0.64	0.67	0.70
Recall	0.94	0.89	0.89	1.00	0.78	0.89

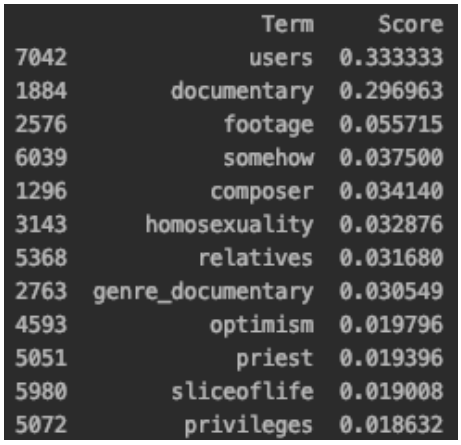
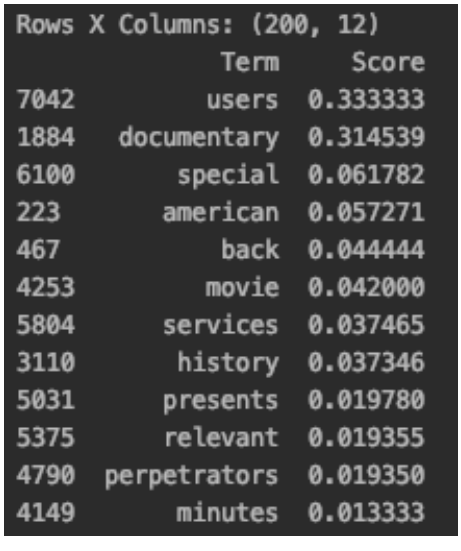
Bar charts are used to visualise the classification results for the **NB model** using **Cross** and **Split Validation**. As can be seen, the model is totally overfitting; it performs well on the training sets but does not generalize well on the test sets.



4.4 Feature Importance / Weights

Decision tree is used to transform the corpus into a subset of 12 best features. Some of the features predicted using WordCloud in **section 2.4** are present in the list, such as **movie** and **users** (positive class), as well as **documentary** (negative class).

```
dtc = DecisionTreeClassifier(random_state=1)
dtc = dtc.fit(X, y)
model = SelectFromModel(dtc, prefit=True)
X_reduced = model.transform(X)
```

Matrix	Best Features
Count	
TF	
TF-IDF	

	Term	Score
7042	users	0.333333
1884	documentary	0.296963
2576	footage	0.055715
6039	somehow	0.037500
1296	composer	0.034140
3143	homosexuality	0.032876
5368	relatives	0.031680
2763	genre_documentary	0.030549
4593	optimism	0.019796
5051	priest	0.019396
5980	sliceoflife	0.019008
5072	privileges	0.018632

	Term	Score
4253	movie	0.342282
1884	documentary	0.305612
6100	special	0.087463
3110	history	0.057617
223	american	0.044291
6779	trailers	0.040000
5804	services	0.037568
6652	threats	0.019789
2598	forever	0.019377
5483	revolutionary	0.019333
3862	love	0.016000
82	adolescence	0.010667

	Term	Score
7042	users	0.333333
1884	documentary	0.314539
6100	special	0.061782
223	american	0.057271
467	back	0.044444
4253	movie	0.042000
5804	services	0.037465
3110	history	0.037346
5031	presents	0.019780
5375	relevant	0.019355
4790	perpetrators	0.019350
4149	minutes	0.013333

5. Evaluation

5.1 Vector

The baseline **SVM** model using the **TF matrix** performs poorly (45% accuracy) but after processing all three matrices work well with the model (in the high 90% range). On the other hand, the **NB model** using the **TF-IDF matrix** has the best and most balanced performance (in the high 80% range).

5.2 Classification Algorithms

The **SVM** algorithm performs noticeably better as its accuracy, precision and recall across all three matrices are all in the high 90% range. The **NB** algorithm has high generalization errors due to overfitting issues. Differences between two of the algorithms are summarized in the table below:

Algorithm	SVM	NB
Type	Supervised learner	Supervised learner
Input	Numerical data	All type of data (preferably categorical data)
Output	Binary label	Categorical label
Outliers	Sensitive to outliers in training sets	Yes, might lead to zero probabilities if a word does not occur in the training sets
Training Time	Slow training time as it is relative to the size of dataset	Fast training time as the same probability values can be reused
Advantage	The SVM model can be scaled easily, and both linear and RBF kernel work well with high-dimensional data	The NB model make probabilistic predictions that work well with text classification problems

Based on the predictive features selected by the **Decision Tree** model, it can be deduced that **movie, users** are good predictors of the positive movie_review class, whereas **documentary** is very predictive of the negative not_movie_review class.