INFORMATION EXTRACTION FROM TV SERIES SCRIPTS FOR UPTAKE PREDICTION

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF COMPUTER AND INFORMATION SCIENCES

Supervisors

Dr. Parma Nand

Dr. Muhammad Asif Naeem

August 2017

By

Junshu Wang

School of Engineering, Computer and Mathematical Sciences

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Junshu Wang

Signature of candidate

Acknowledgements

It is my pleasure to acknowledge all the individuals who have directly or indirectly helped me in the journey of this research.

Most importantly, I express my deepest gratitude to all the members of my family, from the bottom of my heart. With their support, I made the decision to come to AUT for this degree and this thesis.

I would like to thank Dr Parma Nand. As my primary supervisor, he helped me in the early stage of selecting this topic and guided me throughout the entire course of this work. Moreover, I learnt a lot of knowledge from two papers in which Dr Parma was the lecturer. I also thank Dr Muhammad Asif Naeem. As my secondary supervisor, he has offered valuable guidance in the brainstorming sessions.
I would also like to thank the company Parrot Analytic for initiating this research topic and providing guidance along the way.

I would like to extend my appreciation to another two people in the School of Engineering, Computer, and Mathematical Sciences at AUT, for their advice and assistance throughout the course of this thesis. They are: Dr. William Liu, the Programme Leader of MCIS; and Sharda Mujoo, the Academic Administrator.

Abstract

The script of a movie, or of an episode of a television series, describes the setting, the storyline, and the scene changes. It also details the movement, actions, non-oral expression, and dialogues of the characters.

The script is assessed by potential investors. If it is considered to be qualified, a decision is made to arrange funds and other resources to create the real product, i.e. a movie or a television series. This action of approving the project is known as green-lighting.

Many studies have been conducted on building models to predict the success of movies. However, the majority of these studies exploit factors which only become known after the decision of green-lighting, or after the release of the products. Only a few studies have focused on predictive models based on pre-greenlighting factors, which are available before the decision of green-lighting.

In comparison, there are even less models that forecast the performance of television series exploiting pre-greenlighting factors.

This study aims to extract features from scripts of pilot episodes, which are the first episodes of television series. These features will be exploited to construct predictive models for uptake of the television series.

Three data sources were employed, including the IMDB, the OpenSubtitles2016

corpus, and television series scripts retrieved from multiple websites. The scripts were then parsed, and the structures were analysed. Subsequently, features were extracted and data matrices were generated. These features and data matrices were used in classification algorithms for training and construction of predictive models. The output from the prediction models was then used for prediction of the uptake. However, the results were not as compelling as expected.

The present research was compared with previous studies on the same topic. The evaluation results are discussed, and suggestions for future work are given.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The storyline of a movie or of an episode of a television series is described by a script. The script details the movement, actions, non-oral expression, and dialogues of the characters. It also describes the setting of the story and the changes of the scenes.

After accomplishment of the script writing, a script (or a collection of scripts for a television series consisting of many episodes) is presented to the potential investors. If the script (or the collection of scripts) is considered to be qualified, then a decision will be made to arrange funds to recruit the director, the actors, and other necessary teams, to create the real product, i.e. the movie or the television series. The action of approving the project is also known as green-lighting.

For a movie, after the accomplishment of the product, it will be brought to the market. There are many possible metrics that could be used to measure the success of a movie. Two typical metrics are the financial result (the so-called box office) and the ratings given by review websites.

As for a multi-seasonal television series, it is commonly planned on a seasonal basis. The processes of script writing, episode capturing, and broadcasting, are continuous.

A common practice is that, scripts for only one season, or less, are prepared before shooting, and only a few (that is three, two, or even one) episodes are completed in advance.

Practicable metrics to measure the performance of a television series include the size of the audience and ratings given by reviewers. Another two recently adopted metrics include whether a series survives for one or two seasons (Hunter & Breen, 2017a) and the number of episodes in a single season (Hunter & Breen, 2017b).

A pilot episode is the test episode of an intended television series. It helps investors to make decision on whether to fund the television series.

If the television series is successfully released, the pilot episode will commonly be aired as the very first episode.

1.2 Scripts

1.2.1 Elements in scripts

A script consists of a sequence of consecutive *scenes*.

Each *scene* comprises a *slug line* (the heading) and a list of *blocks*.

The *slug line* consists of three segments. The first states whether the scene is interior (INT.), exterior (EXT.), or both. The second segment introduces the venue of the scene, such as in a meeting room or on roadside. Unlike these two segments which give spatial information, the optional third segment expresses the time information, for instance, morning or sunset.

There are two types of blocks, which are named as *desc-block* and *talk-block* respectively in the current study. A *talk-block* represents a piece of spoken content. The headline of a talk-block, which is always in uppercase, denotes the speaker. The

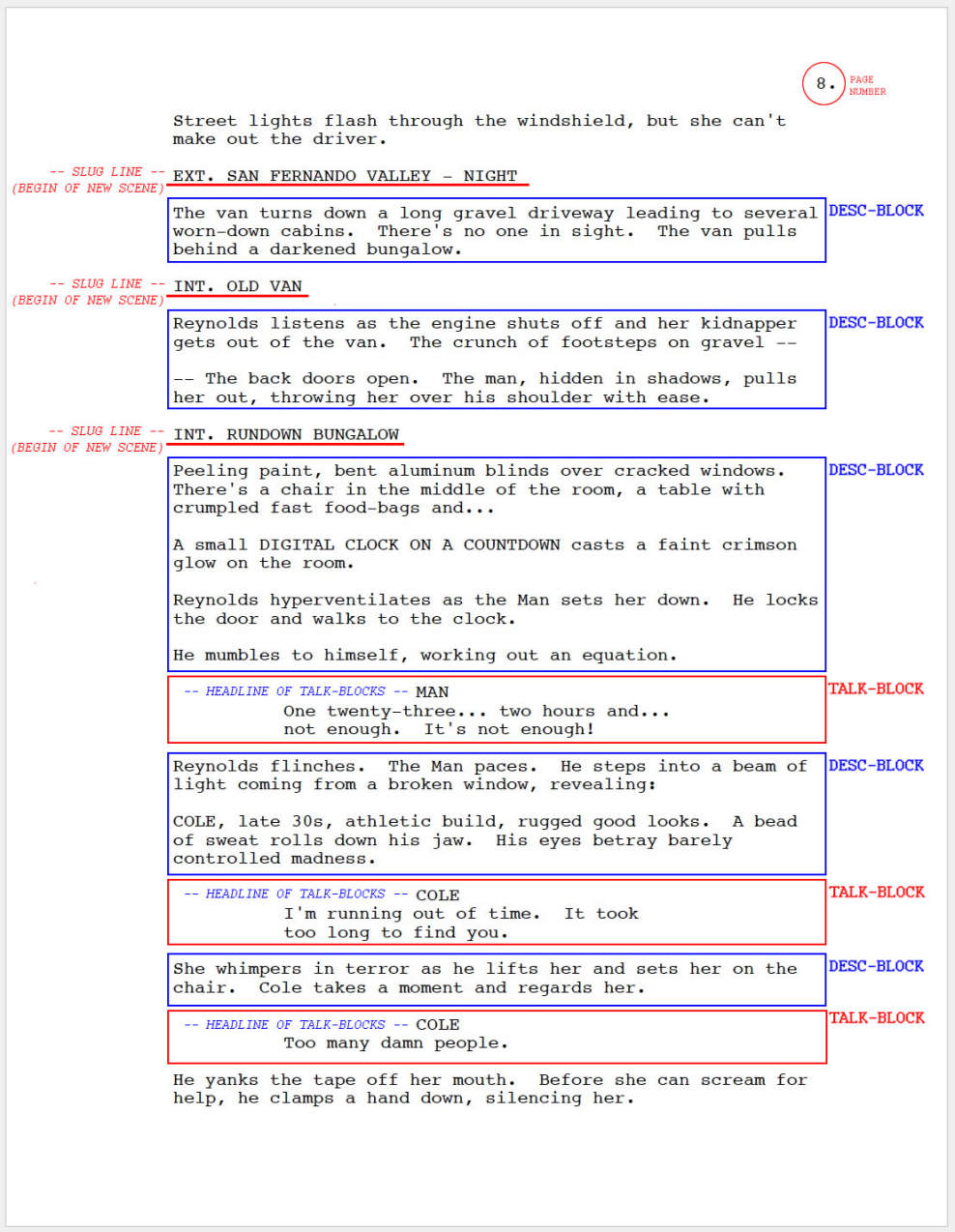


Figure 1.1: Script Page - Lines and Blocks

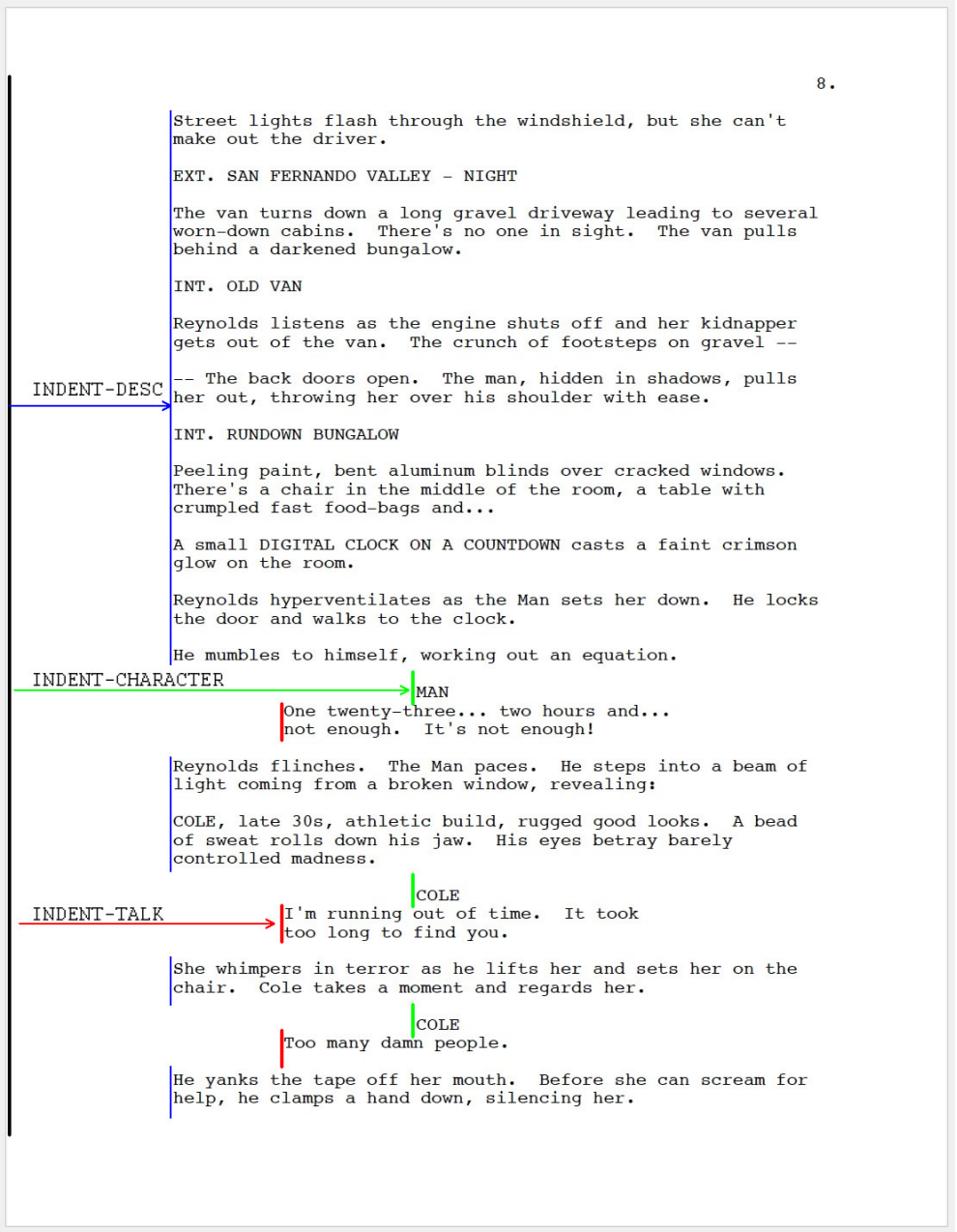


Figure 1.2: Scirpt Page - Indents

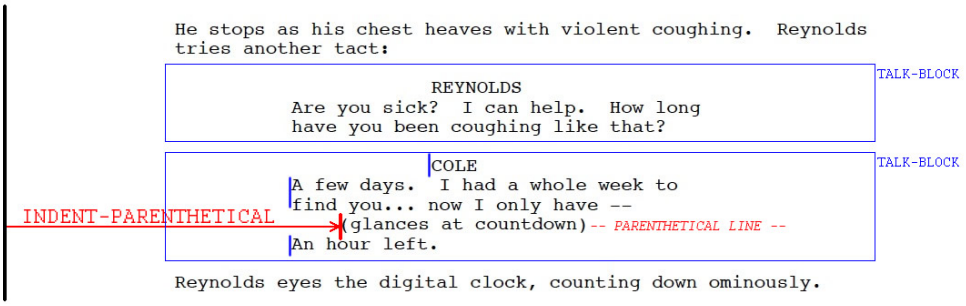


Figure 1.3: Script Page - Parenthetical

headline may contain extra information such as V.O. (voice over), O.S. (off screen), O.C. (off camera), or CONT'D (continued). Every other line in a talk-block represents a part of the spoken content if it is not parenthesized. Otherwise, the line is used to express accompanying action or attitude of the speaker. A *desc-block* describes settings of the scene, movement and actions of characters, events, as well as any other information that should be conveyed besides of spoken content.

All the scripts collected for the present work are in PDF format.

Content of PDF files is organised in pages. A script commonly, but not necessarily begins with a cover page, which introduces the title of the movie/episode, editors of the script, and other information such as revision and release date etc. Except for the first page, other pages representing the scenario all have *page number*s and might have *page-header*s which express the name of the movie/episode, the revision, and the release date. These pages could also have *page-footer*s, which might contain the "(CONTINUED)" instruction, or other information.

Multiple revisions of a script could be released, including early drafts, the revision for green-lighting, the revision for shooting, and the final revision which is documented after the shooting. Within the later revisions, scenes might be organised into a list of *act*s, each of which is composed of several scenes. Also within the later revisions, there could be other information introduced before the first scene, e.g. the cast list.

Within a script, elements related to the page structure are considered to be meaningless to the present research, thus, they will be ignored. Additionally, only a minor proportion of the collected scripts organises their scenes into lists of acts. Hence, the concept of *act* is not involved.

The elements such as scenes, slug-lines, and blocks are related to the storyline of the episodes. They will be extracted from the PDF files and made easy to access. In order to extract these elements, the lines in scripts are classified into these types:

- *slug line*
- *line of disc-blocks*
- *headline of talk-blocks*
- *line of spoken content in talk-blocks*
- *parenthetical line in talk-blocks*

1.2.2 PDF as the Script Carrier

It is speculated that the collected PDF files, which carry the scripts, are generated from different approaches. Consequently, the files could be divided into three groups.

- Files of the first group contain text objects. These files are generated by script-writing applications or text-editing software.
- Files of the second group also contain text objects. The text is recognised from images which are produced by scanning physical papers.
- In a different manner, files of the third group contain image objects. These images are also produced by scanning, but the process of text recognition has not been applied.

Files that belong to the third group will be ignored in the present work since processing of them would consume too much of the time budget.

1.2.3 Slices and Lines

The text entity described by the PDF file is not a text line. Instead, it is the text slice. The information provided for each slice includes text content, font size, the *coordinates*, etc.

The text content of a slice could be as much as one line, and as little as one letter. From another perspective, each line in PDF file could consist of one single slice or a few slices.

Each page in PDF file could be considered as a plane with bi-dimensional Cartesian coordinate system, where the origin is usually the bottom-left point, the x-axis orients from left to right, and the y-axis orients from bottom to top. For every text slice, a rectangle on the plane is required to display its content. The bottom-left point of the rectangle is denoted by the *coordinates*. The width and height of the rectangle could be calculated from letters in the text slice and setting of the font.

In a PDF page, the slices with the same y coordinate usually could be combined to create a text line.

The *indent* of a line is defined as the x coordinate for the leftmost slice within the line. Indents of the lines which belong to a particular category are often same. Moreover, in a certain PDF file, the indent of slug lines and the indent of the desc-block lines are also identical.

Accordingly, four indents could be outlined for the five line categories:

- **indent-desc** for slug-lines and the lines in desc-block
- **indent-talk** for lines of spoken content in talk-block

- **indent-character** for headline of the talk-block
- **indent-parenthetical** for parenthetical lines in talk-block

1.3 The Present Work

The present study aims to extract features from the scripts of television series only, and then to exploit these features to establish models to predict the performance of intended television series.

Chapter 2 firstly reviews previous research, which focus on building predictive models for the performance of movies and television series, especially those that employ factors available before green-lighting. It then inspects previous studies on extracting character networks from multiple sources, especially from movie scripts. Lastly, works related to script parsing are also examined.

The reviews of studies related to distributed representations and several NLP (natural language processing) techniques are in Chapter 5 and Chapter 6.

Chapter 3 introduces data sources, collecting and processing of original material, including script parsing, and IMDB data organisation, etc. Chapter 4 introduces extraction of features from basic statistics and character networks extracted from the scripts. Chapter 5 introduces extraction of features by adopting the distributed representation techniques. Chapter 6 presents feature extractions via typical NLP techniques, including the bag-of-words language model, named entity recognition, dependency parsing, and relation extraction.

Chapter 7 contains two parts. In the first part, genre classifiers are constructed. The genre information of television series, which is adopted as predictive features, could

be supplied as additional information of an intended television series, or be predicted by genre classifiers which take only the scripts as input. The second part demonstrates establishment of the predictive models. The data matrices corresponding to the extracted features in the previous chapters are concatenated, and multiple datasets are prepared. These datasets are then used to construct and test the predictive models. The accuracies of these predictive models are also presented in the second part.

The methodology adopted by the current study and the intermediate results are demonstrated in chapters 3, 4, 5, 6, and 7. Final results are given in Chapter 7.

Chapter 8 presents the discussion.

Chapter 9 concludes this thesis with suggestions for future research.

Chapter 2

Literature Review

2.1 Predictive Models

Several studies have focused on building models that predict the success of movies. However, the predictive features exploited by these studies are mainly post-greenlighting factors, which become known only after the plan of a movie is published, or the product has been completed or released. Mestyán, Yasseri and Kertész (2013) predicted the movie box office based on activity data from related Wikipedia pages before the release of a movie. Asur and Huberman (2010) forecasted the box office performance using data collected from multiple social media and web sources including Twitter, IMDB, and YouTube. Sawhney and Eliashberg (1996) forecasted long-term revenue by exploiting box office of early weeks.

Only a few studies that employ pre-greenlighting factors to do prediction have been conducted.

Eliashberg, Hui and Zhang (2007) extracted information from spoilers of movies and built a model to forecast the ROI (Return On Investment) of movies. The researchers intended to implement their approach with movie scripts. However, they failed to collect

enough scripts in electronic format. Thus a compromise was made, and the researchers' attention was restricted to spoilers. Movie spoilers are written by viewers after they watch a movie, and a well-written spoiler can extensively summarise the storyline of the movie. The information extracted from spoilers includes (1) *semantics* (produced by functions of Microsoft Word, and include *Number of characters, Number of words, Number of sentences, Passive sentences, Characters per word*, all of which are actually basic statistics), (2) *bag-of-words*, (3) *genre and content analysis*. The first two are low-level characteristics and are available without understanding the storyline. The collection of these two is completed automatically. The third point presents high-level aspects of the storyline, and is therefore collected by human experts with cinema training. The technique chosen to establish the predictive model is the Bag-CART (Bootstrap Aggregated Classification and Regression Tree). The main reason this model was chosen was that factors were thought to interact with each other in a nonlinear, complex fashion. Moreover, the adopted methodology was expected to be easy to interpret and lead to intuitive insights.

Gil, Kuenzel and Caroline (2011) extracted character networks from movie scripts and then calculated values of several metrics of the networks. These values were later employed to build predictive models for many aspects, including the scale of the audience. However, the performance of the audience classifier was poor. Gil et al. (2011) concluded that "there does not seem to be a strong link between the plot structure of a movie and its rating or reception".

Eliashberg, Hui and Zhang (2014) developed a methodology to predict the potential revenue at the time of green-lighting. From a database of 300 movie scripts, three layers of information were extracted. These are *semantics* (some statistics, including *number of scenes, percentage of interior scenes, average length of dialogues*, etc.), *bag-of-words*, and *genre/content*. During this process, screenwriting domain knowledge, human input, and NLP techniques were adopted. Subsequently, by adopting the kernel-based

approaches, two slightly different models were established. Results of experiments showed that the proposed approach outperforms several benchmark methods, including regression and tree-based methods (including Bag-CART, which was employed by Eliashberg et al. (2007)), and others. The results loosely reflected practical results of movie studios (Eliashberg et al., 2014). For future work, it was suggested to extract a wider set of textual variables and features from scripts. Eliashberg et al. (2014) stated that to the best of their knowledge, the study is the first one that collected and analysed actual movie scripts.

Hunter, Smith and Singh (2016) conducted a study that was both comparable and complementary to the work of Eliashberg et al. (2014). In this study, the size of the text network of the scripts was used as one of the predictive features. The steps to calculate the value are:

- First, the MMCs (multi-morphemic compounds) were identified.
- Second, each MMC was decomposed into its constituent morphemes, and each morpheme was "assigned to a conceptual category defined by its most remote etymological root" (Hunter, Smith & Singh, 2016).
- Third, for each screenplay, a symmetrical matrix where "rows and column labels were the etymological roots associated with all MMCs" (Hunter, Smith & Singh, 2016) was created.
- Finally, sizes of the resulting networks were calculated.

Subsequently, the ordinary-least squares (OLS) regression analysis was adopted to build the model. The results demonstrate strong support for the hypothesis that "the size of the text network is positively associated with box office performance" (Hunter, Smith & Singh, 2016).

For television series, there were even less studies that adopted pre-greenlighting information to predict the performance.

To the best of the knowledge of the present study, at the beginning of this work, the only two published studies on the topic were: Hunter, Chinta, Smith, Shamim and Bawazir (2016); Hunter, Smith and Chinta (2016). It seems that these two studies were conducted by the same team in the same year. The two works exploit "three previously untested" predictive factors (Hunter, Chinta et al., 2016), all of which could be collected before a decision of green-lighting. These three factors are (1) originality of the concept of the show, (2) the track record of success of the creators, and (3) the size of the conceptual network generated from the script. While the first two factors demand extra inputs, the third factor only requires the scripts. The target feature in these two studies is the size of the audience of the episodes. The results showed that the collected features are both positive and significant.

2.2 Character Network

Several studies have been conducted on extracting character networks, which are also named social graphs, from text works, such as 19th-century literary (Elson, Dames & McKeown, 2010), Shakespeare's work (Moretti, 2011), and Marvel Comics (Alberich, Miro-Julia & Rosselló, 2002). The extracted character networks were used to improve the understanding of the text works.

An early study on constructing character networks for movies and television series was conducted by Weng, Chu and Wu (2009). The co-occurrences of the characters in scenes were exploited to support the process of connection establishing, which is the main activity of network establishing. However, the co-occurrences are determined by analysing the video contents, but not by analysing scripts, which is the focus of the

present work. Later, the character networks were used to promote the understanding of the scenario, by identifying lead roles and detecting communities.

Park, Oh and Jo (2012) employed characters' co-participating in dialogues, instead of their co-occurrences in scenes, to establish connections between characters for construction of networks. In this study, the video, the script, and the subtitle of a movie were employed to identify characters and detect the length of dialogues. Specifically, the dialogue time was extracted via the technique of *script-subtitle alignment*, which is explained by Ronfard and Thuong (2003); Turetsky and Dimitrova (2004), according to Park et al. (2012). The degree centrality metric of the network was used to categorise characters into major, minor, and extra.

Park and You (2012) proposed a procedure that provides recommendations for the green-lighting decision. Firstly, a character network was built from the script. Secondly, metrics were extracted by employing social network analysis techniques. Lastly, these metrics of a being examined script were compared with metrics of past successful scripts. During this procedure, the collection of emotional words, identified with help of WordNet (Miller, 1995; Fellbaum, 2010), was also involved. However, the methods for character network construction and the comparison, as well as how a recommendation is presented, were not detailed. Considering that newly written scripts will be involved in the proposed procedure, it can be speculated that character networks can only be built based on the scripts.

From movie scripts, Jung, You and Park (2013) employed characters' co-participating in dialogues to extract character networks. Further, by adopting emotional factors, the characters were categorised into protagonist (main characters), tritagonist (supporters of main characters), and antagonist (opponent of main characters).

Gil et al. (2011) extracted character networks from movie scripts and texts of plays

(such as The Complete Words of William Shakespeare). The approach used in this work to detect a connection between two characters is named the Closeness approach, wherein a connection is established if two characters speak nearby lines in the same scene. The purpose of this logic is to overcome the issue caused by very long scenes. Subsequently, values of several metrics of the character networks were computed. These metrics include the *number of characters, clustering coefficient, character centrality, betweenness centrality*, etc., which were then exploited to build predictors to forecast many aspects which include the audience.

A recent study conducted by Makris and Vikatos (2016) exploited the co-occurrences of characters in scenes to construct character networks. The movie scripts used in this study were crawled from the IMSDB (Internet Movie Script Database)[1]. The co-occurrences were detected mainly in three steps. Firstly, the Wikipedia pages of the movies were accessed for detecting characters of each movie. Secondly, the lines that contain *EXT.* or *INT.* were detected from the movie scripts, and subsequently, each script was cut into scenes by using these lines as boundaries. Lastly, co-occurrences of characters in the scenes were recognised.

2.3 Script Parsing

To the best of the knowledge of the present work, the only study focusing on the parsing of movie scripts was conducted by Agarwal, Balasubramanian, Zheng and Dash (2014). Based on NLP (Natural Language Processing) and ML (Machine Learning) techniques, their approach categorises each line into one of these classes: **S** for scene boundary; **N** for scene description; **C** for character name; **D** for dialogue; and **M** for meta-data. Subsequently, the structure of a script could be understood, and co-occurrences of

[1] http://www.imsdb.com/

characters could be detected.

The main challenge in the early stage of this study (Agarwal et al., 2014) was the absence of training data. A methodology was adopted to create training data from well-structured scripts, wherein some factors could be employed to give class tags to the lines. One significant factor is the indentation of lines. Another factor is the tags of *EXT* and *INT* which indicate boundaries of scenes.

In the study by Makris and Vikatos (2016), the method to cut a script into scenes was briefly mentioned. That is using the lines which contain *EXT* or *INT* as boundaries of scenes.

2.4 Other Aspects

The techniques of **distributed representation** are capable of representing a word or a piece of text with a relatively low-dimensional, high-dense real number vector. Two renowned techniques of them are the *word2vec* (Mikolov, Chen, Corrado & Dean, 2013; Goldberg & Levy, 2014) and *doc2vec* (Le & Mikolov, 2014; Lau & Baldwin, 2016). More details about distributed representation are introduced in Chapter 5.

A named entity (NE) is a physical existence or an abstract existence with a specific name. Some typical types of NE include Location, Organisation, and Person. The process of **named entity recognition** (NER) detects mentions of NEs from sentences. More details about NER are introduced in Section 6.2.

The relations between words of a sentence are asymmetrical, and these relations are named the **dependencies**. More details about dependencies, such as the representation, parsing, as well as its applications are introduced in Section 6.3 and Section 6.4.

Chapter 3

Data Sources

3.1 Raw Data Sources

3.1.1 Internet Movie Database

The Internet Movie Database [1], commonly abbreviated as IMDb or IMDB, is one of the most extensive, comprehensive, authoritative, and famous online databases focusing on movies and television series. It provides information about many aspects including language, running time, genres, rating, directors, writers, stars (actors and actresses), user reviews, etc.

The IMDB has been exploited by lots of research. The attribute "rating" which is on a scale from 0 to 10, is the quantification of user opinion towards the movie or television series. It is demonstrated as a decent indicator for predicting box office success (Hsu, Shen & Xie, 2014; Bae, Lee & Park, 2014), and is also connected to the popularity of the video products (Fraile & Guerri, 2014). The attribute "genre" was adopted by Kabinsingha, Chindasorn and Chantrapornchai (2012) as an predictive feature to forecast the ratings. Other employed attributes include "user reviews" (Mesnil,

[1] http://www.imdb.com

Mikolov, Ranzato & Bengio, 2014) and "connections" (Canet, Valero & Codina, 2016).

Besides the ordinary web pages, IMDB provides alternative interfaces to access their data [2]. Via the plain-text-file interface, data was retrieved in a mass manner, then the information which is useful for later work was extracted and subsequently organised and stored in the JSON format.

The present study uses the rating information of television series as the target feature and adopts the genre information as part of the predictive features in the phase of establishing predictive models.

Another two sources, TMDB [3] and tv.com [4], also provide data about ratings and genres. However, the amounts of the collected television series of these two sources are much less compared to IMDB. Therefore, the two alternatives are not involved.

3.1.2 OpenSubtitles2016

Maintained by the OPUS (open parallel corpus) [5] project, the collection *OpenSubtitles* is compiled from a huge database of subtitles of movie and television series. As a parallel corpus, the collection contains the same contents represented in multiple languages. It is adopted by several studies focusing on Machine Translation (Tiedemann, 2016) and other subjects (Lison & Meena, 2016).

The most recent major release of the corpus is *OpenSubtitles2016* (Lison & Tiedemann, 2016). The present study utilises the English part of the corpus, which contains more than three million sentences and more than 1.8 billion words in total. The data is adopted as a *domain corpus* to train models that generate *distributed representations* of the pilot scrips.

[2] http://www.imdb.com/interfaces
[3] https://www.themoviedb.org
[4] http://www.tv.com/shows/
[5] http://opus.lingfil.uu.se

3.1.3 Scripts of TV Series

A *script* is a document that details the scenario of a movie or an episode of a television series. It narrates the movement, actions, expression, and dialogues of the characters. It also explains the settings and changes of the scenes. A script should describe visual and aural, as well as behavioural and lingual information considering the movies and television series are rich media. The script is also known as *screenplay* or specifically *teleplay* provided it is for a television episode.

A single source which could provide sufficient eligible teleplays for the current study has not been found. The website IMSDb [6] is claimed as the "largest script resource" on the web. However, it hosts only movie scripts, which does not satisfy the requirement. The corpus CATS (Corpus of American Television Series) (Dose, 2013) is compiled from contemporary American television series. Unfortunately, it only retains the spoken language.

Therefore, television series scripts are collected from multiple websites, which include[7]:

- http://www.bbc.co.uk/writersroom/scripts
- http://www.zen134237.zen.co.uk/
- http://leethomson.myzen.co.uk/
- http://scripts.tv-calling.com/

There are 2992 scripts collected in total, all of which are in PDF format. Among them, less than half, 1414 to be precise, are scripts of the pilot episodes.

[6] http://www.imsdb.com/

[7] Two other websites,*tvwriting"* (https://sites.google.com/site/tvwriting/) and *simplyscripts* (http://www.simplyscripts.com/tv.html), introduce lots of scripts, which, however, are hosted by websites in the list.

A pilot episode is the first episode of the first season of a television series. Within the pilot episode, main characters and core cast are introduced, and the world view of the television series is briefly presented to the audience. As a test episode, the performance of the pilot will help the investor to determine whether to take the intended series into a capturing phase or not.

The pilot scripts are selected for further parsing, and others are omitted. The reason is that the pilot episodes are more worthy to study compared with other episodes because a pilot is captured before the launch of the potential new entertainment product. Moreover, ranks of the pilots would not be influenced by a pre-sequence.

3.2 Parsing PDF

A few steps are involved in the process of parsing the scripts which are in PDF format.

Step-I adopts the Python library *pdfminer* [8], a tool for processing PDF documents. Its low-level interface allows invoking of registered functions for events of *page_begin*, *page_end*, and *text_slice*.

By taking the PDF documents as input, this step intends to generate intermediate result files in plain-text form to describe the pages and text slices. For the *page_begin* event, a line of "## PAGE-BEGIN" is produced. For the *page_end* event, a line of "## PAGE-END" is produced. For the *text_slice* event, two lines are appended to the output file. The first line begins with "## SLICE", and embodies the *coordinates* information. The second line is the text content of the slice.

The PDF files which store content in images are not eligible for this step. This fact

[8] https://pypi.python.org/pypi/pdfminer/

further reduces the size of the script collection.

This step successfully delivers 1095 plain-text files.

Step-II analyses output of Step-I to detect values of the four previously discussed indents.

Some facts concerning the indents are:

- The indent of the slug-lines is **indent-desc**, while slug lines could be easily determined since they always begin with **INT.** or **EXT.**
- The presences of these outlined indents (except desc-parenthetical) are usually frequent than others.
- The four indents could be sorted by their values from small to large as:
  - **indent-desc**
  - **indent-talk**
  - **indent-parenthetical**
  - **indent-character**
- Content of a *parenthetical line* is always enclosed in parentheses.
- Only a minor proportion of the collected scrips contain parenthetical lines.

The intermediate result files generated by Step-I are iterated for multiple passes in this step. The first pass seeks slices that begin with **INT.** or **EXT.**, which help to determine **indent-desc**. The second pass finds out the frequency distribution of the indent values. Subsequently, the top three most frequent indents are picked out, which commonly include indent-desc, indent-talk, and indent-character. A third pass tries to detect desc-parenthetical.

Desired indents are eventually detected from 952 scripts during this step.

Step-III takes the output of Step-I and Step-II as input and delivers recognised structures of the scripts. Numerous slices are identified as noises and excluded from the structure. These noises include the page numbers, page headers, page footers, scene numbers appeared in some scripts, etc.

In this step, lines are created by either using a single slice or combining slices with the same y coordinate. Furthermore, lines are classified into five categories. A *character-line* indicates the beginning of a new *talk-block*, which will be ended by a line that is neither *talk-line* nor *parenthetical-line*. A list of continuing *desc-lines* is combined as a *desc-block*. A *slug line* causes a new scene.

Finally, a sequence of scenes is recognised for each script, and this output is stored in JSON format.

3.3 Matching with IMDB data

The raw IMDB data are kept in several plain-text files. Each of these files stores one single aspect of the information. For instance, the file *genres.list* only contains the genre information of the movies and television series.

In each of these files, there are descriptive texts at the beginning and the ending. The informative data lines comprise the meaningful main body, whose boundaries could be manually detected. Then by parsing these informative lines, data concerning *language*, *rating*, and *genre* are extracted and stored in JSON format. [9]

A television series as an entirety and all of its episodes could be given a rating value unless IMDB has not gathered sufficient votes from the audience.

[9] [http://www.imdb.com/help/show_leaf?usedatasoftware] Creation of a database is only permitted for *individual personal* use.

A television series could belong to multiple genres. For instance, the "Agents of SHIELD" is tagged as *action, drama*, and *sci-fi*.

Preliminarily, contents and structures are successfully extracted from pilot scripts of 952 television series. The language of these television series are checked in this step, and all are in English, which is not surprising since all of them are taken from English websites.

Ratings (ranks) of these pilots are then sought from the structured JSON storage. The ratings will be used as the target feature in the phase of building predictive models, thus only the pilots with a rating are worthy of remaining in the working collection. After this step, the working collection is significantly smaller and retains 512 pilot scripts.

Lastly, genres of these 512 television series are extracted from the JSON storage. The genre tags given to these series include drama, comedy, sci-fi, and so on.

3.4 Summary

The dataset is built from three data sources, including television series scripts, IMDB data, and OpenSubtitles2016.

From multiple websites, nearly 3000 television series scripts are downloaded. Among them, circa 1400 are pilot scripts, which are selected for further work. Contents and structures of these pilot scripts are extracted via three steps: (I) select objects of text slices from PDF files; (II) detect values of key indents; (III) combine slices into lines, and build blocks and scenes from lines.

The IMDB data provides rating, genre, and other information of a great quantity of

movies and television series.

A group of pilot scripts is eventually selected as the working collection. They have rating and genre information provided by IMDB data. Besides, contents and structures of these scripts are successfully extracted from them. The number of the selected scripts is 512.

The English part of the corpus OpenSubtitles2016 contains more than 1.8 billion words. It is adopted as a domain corpus to train models which generate distributed representations for the script documents.

Chapter 4

Features from Statistic and Character Network

4.1 Basic Statistical Features

A television series script consists of an array of scenes, each of which could be interior, exterior, or both. Every scene consists of one or more blocks, each of which is either *desc-block* or *talk-block*.

From the knowledge of the structure of a script, some statistical feature sets could be directly drawn out.

- ***FS_num_scene*** the number of scenes
- ***FS_num_scene_ext*** the number of exterior scenes
- ***FS_ratio_word_t2a*** the ratio of number of spoken words to number of all words
- ***FS_ratio_scene_ext2all*** a list of four values, which are calculated by:

1. divide the scene list into four sub lists, whose lengths are approximately same;
2. for every sub list, calculate the ratio of number of exterior scenes to number of all scenes.

Comparing with a single value *ratio_ext2all*, four values could additionally express the trend of ratio change.

By counting the number of blocks within every scene, a list of values could be composed. However, since scene numbers of different episode scripts are different, the generated list could not be directly used as features. Instead, the *arithmetic mean* and *standard deviation (**SD**)* of the values are adopted.

For a list of values $L = (x_1, x_2, ..., x_n)$, the equations to calculate its *arithmetic mean* (denoted as $mean_{(L)}$, also known as $\bar{x}$) and ***SD*** (denoted as $SD_{(L)}$) are:

$$\bar{x} = mean_{(L)} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i \tag{4.1}$$

$$SD_{(L)} = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{4.2}$$

Similarly, numbers of *sentences, words, talk-blocks, talk-sentences, and talk-words,* as well as *the ratio of spoken words to all words* of the scenes, could be used to extract features by computing the mean and standard deviation of corresponding lists. These feature sets are listed below:

- ***FS_scene_num_block***
- ***FS_scene_num_sentence***
- ***FS_scene_num_word***

- ***FS_scene_num_talk_block***
- ***FS_scene_num_talk_sentence***
- ***FS_scene_num_talk_word***
- ***FS_scene_ratio_word_t2a***

Comparably, by counting numbers of sentences and words within every talk-block to compose lists of values, and then calculating the mean and SD of the lists, these feature sets could be extracted:

- ***FS_tb_num_sentence***
- ***FS_tb_num_word***

4.2 Character Related Features

4.2.1 Identify Characters

A headline of a talk-block indicates the speaker of the following spoken content. In most cases, the name of the speaker is directly given. In the second category of cases, instead of names, descriptive words are adopted to refer to the speaker, mainly for the reason that the character is unimportant or an introduction of the name is not necessary. Some examples of the descriptive words include the policeman, the receptionist, the blond, and the president. These words are then chosen to represent the speaker. In a rare third category of instances, only parenthesized information is given, and no name or descriptive noun is presented. In this case, if the indicator *CONT'D* (which means continued) or *MORE* is given, the speaker of the last talk-block will be determined as the speaker of the present talk-block.

By identifying speakers of the talk-blocks, characters who appear in the episode are recognised.

There could be a situation that a character never speaks, wherein the character cannot be identified by the described method. However, this situation is assumed to be very rare and is thus ignored by this study.

Another potential method to identify characters is to employ the Named Entity Recognition (NER) technique and select the PERSON entities. However, a name's appearance in the script cannot guarantee that it is a character in the story. Moreover, a character might be referred to by words other than a name, such as *the policemen*, as previously described. Therefore, the NER technique is not involved in this step.

4.2.2 Feature sets

For each character, numbers of his/her *talk-blocks, spoken sentences,* and *spoken words* could be counted. Furthermore, these numbers of all the characters in the entire script compose three lists, whose mean value and SD are used as these feature sets:

- ***FS_role_num_talk_block***
- ***FS_role_num_talk_sentence***
- ***FS_role_num_talk_word***

By counting the number of speakers in a scene, the number of the characters who appear in the scene is determined. Actually, there might be a character who appears in a scene but doesn't say anything. However, this situation is assumed to be very rare and is therefore ignored. Additionally, a list of such numbers of all the scenes is composed, then its mean value and SD are used as the following feature set:

- ***FS_scene_num_role***

4.3 Character Network

In the context of television series scripts and film scripts, the terms *character network* and *role network* are used as aliases of the term *social network*.

The present study only focuses on undirected graphs.

4.3.1 A Brief Introduction to Graph

A graph **G** can be described by a set of vertices **V** and a set of edges **E**.

An *edge* is an association between two vertices, both of which are elements of **V**. An edge could have a specific direction from one vertex to the other. An edge could be assigned a numeric value named *weight* as a measurement of the association. Between two vertices, there could be no edge, only one edge, or even many edges. The last case could also be considered as having one edge with a weight larger than one.

If a graph permits multiple edges between a pair of vertices, this graph is referred to as a *multi-graph*. Otherwise, if only one edge is permitted, it is referred to as an *uni-graph* in this study.

An extreme example of a graph comprises many vertices, and has edges for every pair of vertices. Another extreme example of a graph is a single vertex with no edge.

If a vertex could be reached by another vertex via one edge or several edges, this two vertices are *connected*, and the list of ordered edge(s) is called a *path*. Between a pair of vertices, there could be more than one path, among which length of the shortest one is defined as the *distance* between the two vertices. If any pair of vertices in a graph is connected, this graph is referred to as a *connected graph*. Furthermore, if any pair of vertices is directly connected by an unique edge, it is a *complete graph*.

A *component* of a graph is defined as either a connected subgraph or an isolated single vertex. A graph could contain (1) only one component, in which case it is a

connected graph itself or comprises only one vertex; or (2) multiple components. An extreme example of the second case is a graph which consists of many vertices but without a single edge.

Some graph-level and vertex-level metrics are:

- The **density** of a graph is the ratio of the number of existing edges to the number of all possible edges.
- The **eccentricity** of a vertex is its maximum distance to any other vertices.
- The **diameter** of a graph is the maximum eccentricities of all its nodes.
- The **radius** of a graph is the minimum eccentricities of all its nodes.
- The **degree** of a vertex is the number of edges adjacent to it. In an unweighted graph, this measure equals to the number of neighbours of a vertex.

For a group of three vertices **(A, B, C)**, if they are connected by two edges **(ab, bc)**, they form a *connected triplet* **<(A, B, C), (ab, bc)>**. Further, if a third edge **bc** exists so that a triangle **<(A, B, C), (ab, bc, ca)>** is constructed, then a *closed triplet* **<(A, B, C), (ab, bc), (ca)>** is formed.

From another angle, a triangle **<(A, B, C), (ab, bc, ca)>** contains three *closed triplets*, which are:

< (A, B, C), (ab, bc), (ca) >

< (A, B, C), (ab, ca), (bc) >

< (A, B, C), (bc, ca), (ab) >

Based on the concept of triplets, two metrics to measure the degree to which vertices tend to cluster together are:

- The **transitivity** of a garph *G* (denoted as $T_{(G)}$) is the ratio of number of closed triplets to number of connected triplets.

$$T_{(G)} = \frac{N_{closed-triplets}}{N_{connected-triplets}} = \frac{3 \cdot N_{triangle}}{N_{connected-triplets}} \tag{4.3}$$

- The **clustering coefficient** of a vertex *v* is the ratio of number of existing triangles that contain the vertex *v* to number of all possible triangles which contain the vertex *v* and two of its neighbours. For a vertex *v*, defining NT_v as number of triangles containing *v* and D_v as the degree of v, the equation to calculate its clustering coefficient $CC_{(v)}$ is:

$$CC_{(v)} = \frac{2 \cdot NT_v}{D_v \cdot (D_v - 1)} \tag{4.4}$$

  The concept can be simplified as ratio of existing edges between neighbours of *v* to all potential edges between them.

To quantify the intuitive feeling that some vertices are more central than others in most graphs, the concept of *centrality* is introduced. It measures the extent of how important or influential a node is within a given graph (Martin, Zhang & Newman, 2014).

- Within an unweighted graph **G**, the **degree centrality** of a vertex is the ratio of its degree to its potential highest degree, which equals to the number of all other vertices in **G**.

- Within an unweighted, connected graph **G** which contains *N* vertices, the **closeness centrality** of a vertex *v* is the reciprocal of the sum of distances between *v* and all other vertices in **G**, multiplied by *N -1* as a measure of normalisation. The

equation is:

$$Closeness_{(v)} = \Big(\sum_{i=1}^{N-1} Distance_{(v,v_i)}\Big)^{-1} \cdot (N-1) \tag{4.5}$$

- Within an unweighted, connected graph **G** which contains *N* vertices, for every pair of vertices *(s, t)*, there exsits one or various shortest paths, of which some or all might pass through a specific vertex *v*. For the vertex *v*, defining $\sigma_{(s,t)}$ as the number of shortest paths between the pair *(s,t)*, and $\sigma_{(s,t|v)}$ as the number of these paths passing through *v*, the **betweenness centrality** is the sum of the ratios of $\sigma_{(s,t|v)}$ to $\sigma_{(s,t)}$ for every pair of vertices, multiplied by the reciprocal of the number of all potential pairs of the vertices except *v* as a measure of normalisation. The equation is:

$$Betweenness_{(v)} = \Big(\sum_{s,t \epsilon V} \frac{\sigma_{(s,t|v)}}{\sigma_{(s,t)}}\Big) \cdot \frac{2}{(N-1)\cdot(N-2)} \tag{4.6}$$

The degree centrality is a simple neighbourhood-based measure. In a given graph, the degree centrality of a specific vertex is only affected by direct neighbours. The closeness centrality is distance based. It could be influenced by the status of non-neighbour vertices. The betweenness centrality, which is shortest path-based, is more advanced. Even edges between non-neighbour vertices can contribute to it.

The *normalisation* procedures are essential to calculations of closeness and betweenness centralities, otherwise, the scale of the graph can be problematic when comparing these values of vertices from different graphs (Koschützki, Lehmann, Tenfelde-Podehl & Zlotowski, 2005).

Not every centrality metric is suitable to every application (Koschützki, Lehmann, Peeters et al., 2005). Thus, besides the above mentioned examples, many other sophisticated centrality metrics have been suggested, such as eigenvector centrality,

page-rank centrality, and various edge-level centralities. Nevertheless, these are not involved in this study.

4.3.2 Social Graph and Character Network

The social graph (also social network) is a special category of graph. The vertices of a social graph are people, and the edges correspond to some kind of relationship between people. For instance, in a social network service (also social network site, SNS), every account is a vertex, and the friend relation between two accounts is an edge. In an email system, every user is a vertex, and an edge could be established if email communication between a pair of users has occurred. Since many emails could be delivered between two users, a weight could be assigned to an edge.

In the current study, social networks are built up from television scripts. The characters of an episode serve as the vertices in the graph. Accordingly, the term *character network* is used as an alias in the current setting.

A co-participating of two characters in a dialogue is considered as a connection between them. Because a pair of characters could co-participate in several dialogues, multiple connections between them are possible. This fact is reflected by an edge between the two vertices with a weight value larger than one. Accordingly, a weighted graph is constructed.

By omitting the weight attribute of the edges, an unweighted graph could be drawn from the weighted graph.

4.3.3 Primary Characters

The characters are sorted (arranged in a certain order, from more to less) according to five criteria:

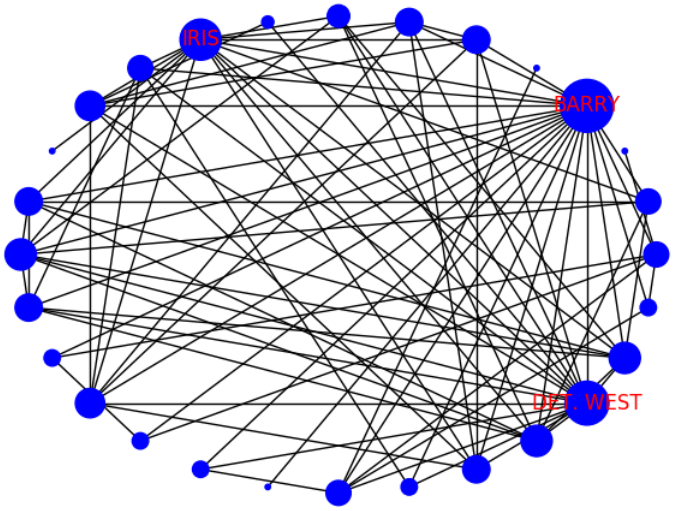


Figure 4.1: Character Network - Pilot of "The Flash"

- by how many scenes they present
- by how many times they talk
- by how many words they speak
- by their degrees in the weighted graph
- by their degrees in the unweighted graph

Then five lists are generated.

By selecting the top two from each list, a group of *primary characters* for every script is determined. A character could be selected from multiple lists. For example, a man who has the largest degrees might also speak the most words.

These primary characters are adopted in further processes of feature extraction.

4.3.4 Features

For each television script, character networks are established.

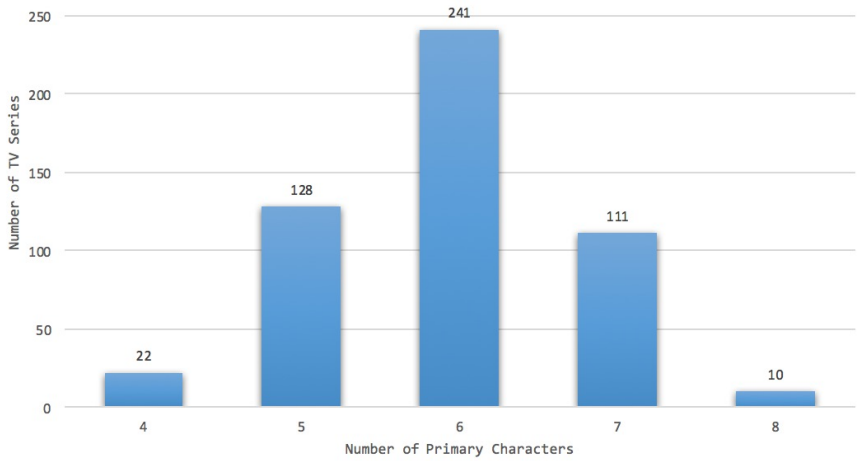


Figure 4.2: Numbers of Primary Characters

Those include the weighted graph (which could also be seen as a *multi-graph* introduced in 4.3.1) here named **MG**, and the unweighted graph (which is an *uni-graph* introduced also in 4.3.1) here named **UG**. The largest components within **MG** and **UG** are named **MC** and **UC** respectively.

By adopting the same method, but only using the first half of the scenes, character networks **HALF_MG** and **HALF_UG** are constructed. Similarly, **HALF_MC** and **HALF_UC** are defined.

For each of the graph-level metrics involved, the corresponding feature set comprises three valuse: (1) value of the character network constructed by using the first half of the scenes; (2) value of the character network constructed by using all of the scenes; (3) the ratio of the first two.

Accordingly, the features reflect not only the value of the metric, but also how it changes along with advancement of the story.

The **graph-level** (character network level) feature sets include:

- ***FS_cnet_num_components*** includes three values: the first value v_1 is the number of components of **HALF_MG** which equals to that of **HALF_UG**; the second

values v_2 is the number of components of **MG** which equals to that of **UG**; the third one v_3 is the ratio of v_2 to v_1.

- ***FS_cnet_density*** *density* of **HALF_UG**, of **UG**, and the ratio.
- ***FS_cnet_transitivity*** *transitivity* of **HALF_UG**, of **UG**, and the ratio.
- ***FS_cnet_diameter*** *diameter* of **HALF_UC**, of **UC**, and the ratio.
- ***FS_cent_radius*** *radius* of **HALF_UC**, of **UC**, and the ratio.
- ***FS_cent_num_nodes*** number of nodes of **HALF_UC**, of **UC**, and the ratio.
- ***FS_cent_num_edges_u*** number of nodes of **HALF_UC**, of **UC**, and the ratio.
- ***FS_cent_num_edges_m*** number of nodes of **HALF_MC**, of **MC**, and the ratio.
- ***FS_cent_ration_edge_m2u*** the ratio of number of edges of **UG** to that of **MG**

For every detected primary character, values of various metrices were collected. However, the collected values could not be used as features directly since numbers of detected primary characters vary from script to script. Instead, for every metric, the mean value and the SD of the values are adopted.

For example, the feature ***FS_cnet_role_degree_u*** is related to the metric of *degree* in **UC**. In other words, the feature is composed by mean value and SD of the values of the meric *degree* of the primary characters in **UC**.

The **vertex-level** (character level) feature sets include:

- ***FS_cnet_role_degree_u*** is related to *degree* in **UC**
- ***FS_cnet_role_degree_m*** is related to *degree* in **MC**
- ***FS_cnet_role_clustering*** is related to *clustering coefficient* in **UC**

- ***FS_cnet_role_eccentricity*** is related to *eccentricity* in **UC**
- ***FS_cnet_role_betweenness*** is related to *betweenness centrality* in **UC**
- ***FS_cnet_role_closeness*** is related to *closeness centrality* in **UC**

4.4 Summary

This chapter introduced:

1. features from general statistics;
2. features from character based statistics;
3. features from graph-level metrics of the character networks;
4. features from vertex-level (character level) metrics of the character networks.

It also introduces how to construct character networks and how to determine primary characters.

The collection of all the statistic-related features is denoted as the ***FS_stat***, and the collection of all the character network-related features is denoted as the ***FS_cnet***.

Chapter 5

Distributed Representation

This chapter introduces schemes to represent a word or a piece of text, for example, a paragraph or a document, as a real number vector, where every single value is commonly not easy to explain, but the entirety is expressive.

5.1 Introduction

5.1.1 Word Embedding

The artificial neural network (henceforth *ANN*) based, distributed word representations are also known as the *word embedding*s (Y. Li et al., 2015). The basic idea is that the contextual information of a word is feasible of constituting a significant representation of the word. Furthermore, in the projected multi-dimensional space, words would be near to each other provided they appear in similar contexts in the training corpus (Levy, Goldberg & Ramat-Gan, 2014).

Some characteristics of word embeddings are:

- A single word is mapped to a real number vector space.

- Dimension of the vector is relatively low, i.e. between 50 and 1000.
- A matrix, which is composed of *word embedding*s of a number of words, is dense, which means that the vast majority of the cell values are not zero, and any cell value is different to most of the others.
- The vector represents contextual and semantic information of a word. From another perspective, words that have similar semantic meanings will have similar vectors. Also, similarities of words are easy to calculate.
- The word embedding models are trained with ANN.

word2vec

The idea of representing words in vector space can be traced back to as early as 1986 [1]. This approach is capable of capturing similarities between words. Conversely, the intuitive approach, which treats words as atomic units, lacks such ability.

An early study on employing ANN to construct language models and generate distributed word representations was conducted by Bengio, Ducharme, Vincent and Jauvin (2003). The employed ANN is a feedforward neural network which embeds a linear projection layer and a nonlinear hidden layer. The proposed model became popular and the study was followed by various others (Mikolov, Chen et al., 2013).

Another genre is to adopt a recurrent neural network (Mikolov, Kopecky, Burget, Glembek et al., 2009; Mikolov, Karafiát, Burget, Cernockỳ & Khudanpur, 2010), which overcomes limitations of feedforward NN such as the necessity of declaring context length. In addition, comparing with the shallow neural networks, the recurrent NN is eligible of encoding more complex patterns efficiently (Mikolov et al., 2010; Bengio, LeCun et al., 2007). The recurrent NN has only one hidden layer besides the input layer and the output layer. What makes it superior is that the hidden layer is connected to

[1] That is by Hinton, Mcclelland and Rumelhart (1986), according to Mikolov, Chen et al. (2013)

itself. From another angle, the hidden layer takes input from both the input layer and previous state of itself. This fact reveals some kind of short term memory since former states of the hidden layer are capable of influencing latter states.

The nonlinear hidden layers in these models make the neural networks sophisticated. However, on the other hand, they are also the main reason for computational complexities.

To address the issue of computational expensiveness during the procedure of learning distributed representations of words, Mikolov, Chen et al. (2013) proposed two relatively simple model architectures, which might produce less precise results, but is capable of training much more data efficiently.

The first model architecture is named the Continuous Bag-of-Words Model (*CBOW*), whose architecture is similar to the feedforward NN but without the nonlinear hidden layer. During the training, a few words from the history and from the future of the word being examined is involved. The fact that the term bag-of-words is part of the name results from the fact that orders of the history words are not taken into the computation. On the other hand, while the CBOW yields continuous distributed representations of the context of a certain word as output, the standard bag-of-words model pursues a single value for a distinct word.

The second model architecture is the Continuous Skip-gram Model (*skip-gram*), which is similar to CBOW. The underlying thought of skip-gram is to learn vector representations of words that are good at predicting surrounding words. More formally, given a sequence of training words $(w_1, w_2, ..., w_T)$, the objective is to maximise the average log probability (Mikolov, Sutskever, Chen, Corrado & Dean, 2013):

$$\frac{\mathbf{1}}{\mathbf{T}} \cdot \sum_{t=1}^{T} \sum_{j \in nb(t)} \log p(w_j \mid w_t) \tag{5.1}$$

where $nb(t)$ is the collection of neighbours of the index t (exclude t itself). Conversely,

the *CBOW* tries to predict a word based on the context.

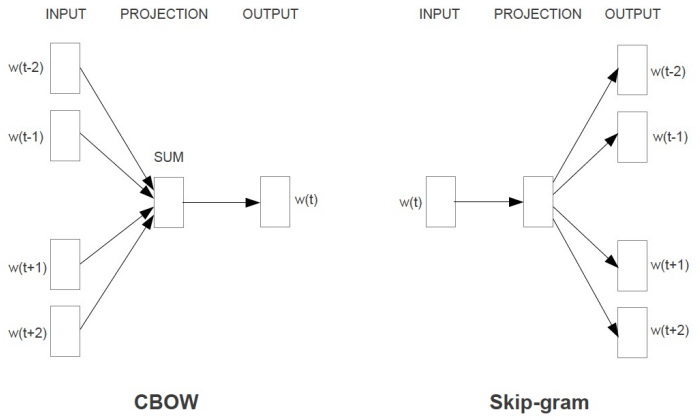


Figure 5.1: CBOW and skip-gram, from Mikolov, Chen et al. (2013)

Mikolov, Sutskever et al. (2013) proposed extensions for the *skip-gram* model. One extension, the *negative sampling* algorithm, is a simplified variant of Noise Contrastive Estimation. It was proposed to replace the hierarchical *softmax* algorithm which was originally adopted, in order to accelerate the calculation of $p(w_j \mid w_t)$ in the Equation 5.1 and yield better outcomes for frequent words. Another extension is *subsampling* of frequent words, which improveed the accuracy of vector representations of less frequent words.

The proposed models by Mikolov, Sutskever et al. (2013); Mikolov, Chen et al. (2013) are implemented in the ***word2vec*** project[2].

Details of the *CBOW* and *skip-gram* are explained by Rong (2014); Mikolov, Sutskever et al. (2013); Goldberg and Levy (2014).

bag of contexts

Another approach to mapping a word into vector space without employing a neural network, is the *bag of contexts*, also named distributional similarity representation or

[2]https://code.google.com/archive/p/word2vec/

explicit vector-space representations (Levy et al., 2014), which captures the contexts where the word occurs. According to Levy et al. (2014), the *bag of contexts* representations have been extensively studied.

The *context* of a certain word is its surrounding words within a specific window size. For instance, given a word sequence $(w_a\ w_b\ w_c\ w_d\ w_e\ w_f\ w_g)$, the context of the word w_d could be defined as $< w_b,\ w_c,\ w_e,\ w_f >$.

Given a vocabulary $\mathbf{V}$ and a collection of context $\mathbf{C}$, the scheme will generate a sparse matrix $\mathbf{S}$ of size $|\mathbf{V}| \times |\mathbf{C}|$. The value S_{ij} of each cell within the matrix expresses the measure of the association between a certain word $w \in \mathbf{V}$ and a certain context $c \in \mathbf{C}$. There are many methods to calculate the S_{ij}. A common method is the PPMI (positive pointwise mutual information) (Bullinaria & Levy, 2007). The equations to calculate the PPMI are:

$$S_{ij} = PPMI(w_i, c_j) \tag{5.2}$$

$$PPMI(w,c) = PMI(w,c)\ \ \textbf{if}\ \ PMI(w,c) > 0\ \ \textbf{else}\ \ 0 \tag{5.3}$$

$$PMI(w,c) = \log\frac{P(w,c)}{P(w)\cdot P(c)} = \log\frac{freq(w,c)\cdot |\ corpus\ |}{freq(w)\cdot freq(c)} \tag{5.4}$$

In the Equation 5.4, $|\ corpus\ |$ is the number of words in the corpus; $freq(w)$ and $freq(c)$ are the numbers of occurrences of word w and context c respectively; $freq(w,c)$ is the number of times that w appears in c.

For a brief summarisation, some facts of the *bag of contexts* scheme are: (i) each dimension of the vector directly corresponds to a particular *context*, thus each cell in the constructed value matrix could be explained; (ii) the dimension of the vector is very high; (iii) the matrix is sparse; (iv) ANN is not involved during establishment of the

models.

conclusion

Experiments conducted by Mikolov, Chen et al. (2013) and Mikolov, Sutskever et al. (2013) reveal that the continuous skip-gram model with negative-sampling algorithms outperforms other models included in analyses, in terms of both training speed and quality of produced word embeddings. A substantial drawback of the *bag of contexts* scheme is that the generated matrix is high-dimensional and sparse.

Therefore, the present study will adopt ***word2vec*** in *skip-gram* mode to generate word embeddings in later work.

5.1.2 Document Embedding

In many NLP tasks, such as text classification and text clustering, fixed-length vector representations of the input texts are required. The term $document\ embedding$ is used by Lau and Baldwin (2016) to refer to the embedding of a word sequence, ignoring the granularity concern.

A renowned scheme with a long history[3] that meets the requirement is the bag-of-words (BOW) model. The procedure of building a BOW model is briefly introduced in section §6.1. During the early step, a term could be created in forms of unigram (one single word), bigram (two adjacent words), or n-gram (several continuous words), or further skip-gram (several words that might not be consecutive in the text).

The BOW model has some drawbacks. First, the word order information is less reflected. For instance, while using unigram terms, the two sentences $Tom\ chased\ Jerry$ and $Jerry\ chased\ Tom$ would have the exact same vector representation. Second, the

[3]Could be traced back to Harris (1954) according to Le and Mikolov (2014)

representations are high-dimensional and sparse. Moreover, it has very little sense concerning word semantics (Le & Mikolov, 2014).

The ***word2vec*** proposed by Mikolov, Sutskever et al. (2013); Mikolov, Chen et al. (2013) sparked the boom of many studies adopting word embeddings. Amongst these studies, Sultan, Bethard and Sumner (2015); Wieting, Bansal, Gimpel and Livescu (2015); Lau and Baldwin (2016) exploit the simple averaging of the word embeddings of the constituents of a document as its vector representation.

This scheme overcomes the drawbacks of the BOW model, which are the sparsity and high-dimension of generated data matrix. However, the order of words is still not taken into consideration. For instance, the two sentences $Tom\ chased\ Jerry$ and $Jerry\ chased\ Tom$ would have identical dense vector representations.

Le and Mikolov (2014) proposed the *Paragraph Vector*, also known as *paragraph2vec* and ***doc2vec***[4], as an extension to word2vec to extend the learning of embedding from words to pieces of texts, ranging from sentences, paragraphs, to documents. Two modes of *doc2vec* are introduced.

The first mode, the *dbow* (distributed bag of words), works in a similar way as the skip-gram mode of *word2vec* (Le & Mikolov, 2014), except that "the input is replaced by a special token representing the document" (Lau & Baldwin, 2016). In the *dbow* mode, the order of words in the text is ignored. However, it is conceptually simple and requires to only store the "softmax weights" (Le & Mikolov, 2014).

The other mode is named *dmpv* (distributed memory of paragraph vector). For the input, a document token is introduced, and is concatenated, rather than summed, with representations of words from a certain sliding window over the text (Le & Mikolov, 2014). This mode takes the order of words into consideration.

[4]See https://radimrehurek.com/gensim/models/doc2vec.html

Other proposed schemes for representing texts into vector space include skip-thought (Kiros et al., 2015) and paragram-phrase (Wieting et al., 2015).

Results of experiments by Lau and Baldwin (2016) showed that: (1) *doc2vec* outperforms *word2vec* and *bag-of-words* model (with n-gram), especially in tasks with long documents; (2) *dbow* is better than *dmpv*; (3) the *skip-thought* model performs poorly; (4) comparing with *doc2vec*, the *paragram-phrase* model performs better on shorter texts, but worse on longer texts.

5.2 Models

The present work utilised two pre-trained models [5]. These two models were trained with the English Wikipedia Corpus [6] by Lau and Baldwin (2016). One model was trained by the *word2vec* scheme in *skip-gram* mode and was renamed from *wiki_sg* to *model_wiki_w2v*. The other model was trained by the *doc2vec* scheme in *dbow* mode and was renamed from *enwiki_dbow* to *model_wiki_d2v*. The size of a vector generated by any of these two models is 300.

Two extra models were trained by exploiting the English part of the *OpenSubstitle2016* corpus (see §3.1.2) . One was trained by *word2vec* scheme in *skip-gram* mode and is referred to as *model_subt_w2v*. The other model was trained by *doc2vec* scheme in *dbow* mode and is referred to as *model_subt_d2v*. The size of a vector generated by any of these two models is 500.

[5] https://github.com/jhlau/doc2vec
[6] See https://github.com/attardi/wikiextractor

5.3 Results

Four matrices representing document embeddings of the pilot script were generated:

- ***FS_dr_wiki_w2v*** learned with *word2vec* and the model *model_wiki_w2v*, and is averaging of the word embeddings.
- ***FS_dr_wiki_d2v*** learned with *doc2vec* and the model *model_wiki_d2v*.
- ***FS_dr_subt_w2v*** learned with *word2vec* and the model *model_subt_w2v*, and is averaging of the word embeddings.
- ***FS_dr_subt_d2v*** learned with *doc2vec* and the model *model_subt_d2v*.

Each of these matrices was generated via two steps. Take ***FS_dr_wiki_w2v*** as an example:

- Assemble the *word2vec* representations, learnt with *model_wiki_w2v*, of all the pilot scripts, and denote the resulting data matrix as DM.
- Apply the *min-max-scale* procedure to each dimension of DM so as to transform the value range of the column to $[0, 1]$. Given a column X, the equation to transform a cell value x is:

$$x_{scaled} = \frac{x - min_X}{max_X - min_X} \tag{5.5}$$

These matrices will be exploited in the establishing and comparison of predictive models in later works.

5.4 Summary

This chapter introduced the notions of word embedding, which is the low-dimensional dense vector representation of words, and document embedding, which is the low-dimensional dense vector representation of sequences of words.

This chapter also introduced the two schemes to generate word embeddings and document embeddings respectively: *word2vec* and *doc2vec*.

Two pre-trained models were downloaded: *model_wiki_w2v* and *model_wiki_d2v*. By exploiting the English part of OpenSubtitle2016, two new models were trained: *model_subt_w2v* and *model_subt_d2v*.

Four datasets of document embeddings of the pilot scripts, corresponding to the four models, were generated.

Chapter 6

Features from NLP

This chapter introduces the features extracted by exploiting *Natural Language Processing (NLP)* techniques.

6.1 Venues of Scenes

In television series scripts, the venues of the scenes are declared in the slug lines. A slug line always begins with "INT.", or "EXT.", or both. Then follows the venue phrase, which comprises one word or several consecutive words. An optional third part, which is normally separated from the venue phrase by a dash sign, indicates the time.

The venue phrases were collected and processed via a series of steps.

Step I is to remove worthless words. Firstly, a list of *stopwords* was determined by combining *sklearn*[1] stopwords and *nltk*[2] English stopwords. Any word in this list is considered to be extremely common and was therefore removed from venue phrases. Secondly, the name of a character might appear in the venue phrases, for example, "Barry's apartment". Therefore, a procedure of filtering out characters' names was

[1] http://scikit-learn.org/
[2] http://www.nltk.org/

involved. Lastly, since the slice indicating time is not always explicitly separated, time words might incorrectly be selected, such as the "day" and "night". These words were also eliminated.

In **Step II**, terms were determined from the venue phrases by collecting single words as *unigram*s and pairs of adjacent words as *bigram*s.

Both unigram and bigram are worthy to choose. Compared with unigram, the bigram model reserves sequential information of the words to some extent. Hence, the venue "meeting room" could be discriminated from "living room". On the other hand, the unigram model helps to pick out "room" from both "meeting room" and "living room", which are distinct from the venue "street".

There are more than 14 thousand venue terms determined in total in this step. Among them, the number of venues whose appearance frequency is larger than 50 is less than 200. Considering that the working collection contains more than 500 scripts, it could be deduced that the vast majority of the venue terms appear in only a few scripts.

Figure 6.1 shows the top 20 most frequent terms. Figure 6.2 shows the distribution of frequencies of the top 100 most frequent terms. Figure 6.3 shows top frequent terms in the word cloud style.

In **Step III**, a matrix of term quantity was established. A term vector was firstly created by selecting the existing venue terms. As soon as a venue term appears in one television script, it will be selected, and every unique term will be selected no more than once. Subsequently, for each script, a value vector was created by counting the numbers of occurrences of every venue term. Lastly, the matrix was composed by combining value vectors of all the scripts.

It is not surprising that the created matrix was sparse since most of the venue terms only appear in a few scripts.

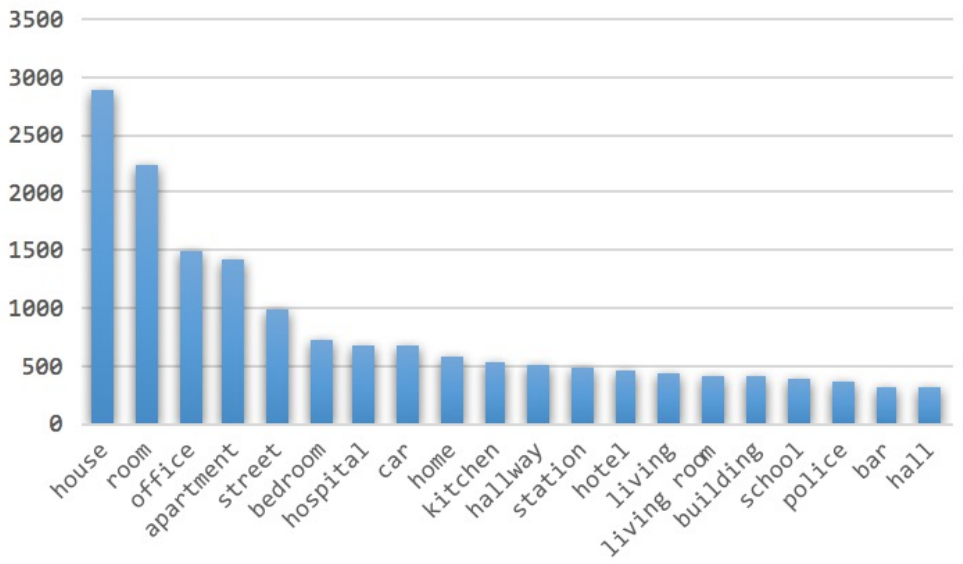


Figure 6.1: Venue Terms - Top 20

In **Step IV**, a matrix of **TF-IDF** values was established.

The *term quantity* (also *raw term frequency*) model supposes that, a script A would be more relevant to a specific venue term *t* than another script B if the occurrence of *t* in A is slightly more frequent than in B. However, if the number of all occurrences of all terms in A is much larger than in B (more generally, if the length of document A is much longer then another document B), then this supposition is clearly doubtable. To address this issue, an adjustment was introduced, where for a document *d*, the number of occurrences of a term *t* ($\mathbf{N}_{(t,d)}$) was divided by the number of all occurrences of all terms ($\mathbf{N}_{(T,d)}$), then the **Term Frequency (TF)** value (denoted as $TF_{(t,d)}$) is obtained. The equation is:

$$TF_{(t,d)} = \frac{N_{(t,d)}}{N_{(T,d)}} \tag{6.1}$$

Another effect of this adjustment is the range of values were scaled down to $[0, 1]$. Hence very large values were avoided.

The *term frequency* model supposes that, for a specific document *d*, if the **TF** values of two terms are equal, then they are of the same significance to the document. However,

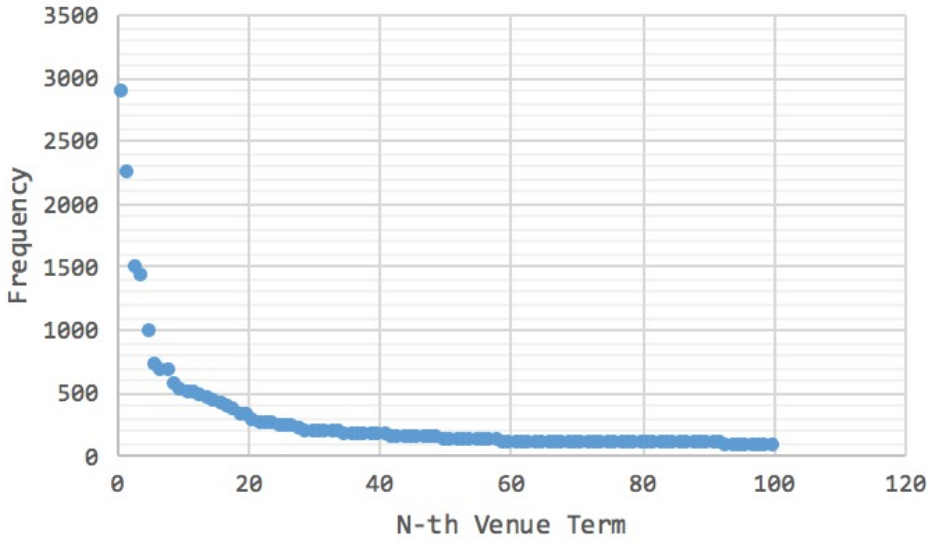


Figure 6.2: Venue Terms - Frequency of Top100

in cases where one term appears only in *d* whilst the other one appears everywhere, then their importance is clearly distinctive from each other. Accordingly, the concept of **Document Frequency (DF)** is introduced. For a term *t* in a collection of documents *D*, its document frequency value $DF_{(t,D)}$ is the number of documents that contain t. Furthermore, the **Inverse Document Frequency (IDF)** value of the term *t* (denoted as $IDF_{(t,D)}$) in the collection of document *D*, which is constituted by N_D documents, could be calculated by the equation:

$$IDF_{(t,D)} = \log \frac{N_D + 1}{DF_{(t,D)} + 1} + 1 \tag{6.2}$$

This equation ensures that the **IDF** value will never be zero ($\log 1$), and a term that never appeared, whose *DF* value is zero, would not cause failure.

Lastly, the **TF-IDF** value (denoted as $TFIDF_{(t,d)}$) of a term *t* in document *d* can be computed by:

$$TFIDF_{(t,d)} = TF_{(t,d)} \cdot IDF_{(t,D)} \tag{6.3}$$

Figure 6.3: Cloud of Frequent Venue Terms

In **Step V**, the **TF-IDF** matrix is normalised. For each column $x = [[x_1, x_2, ..., x_n]]$ in the matrix, firstly its l^2-norm $|x|_2$ is calculated:

$$|x|_2 = \sqrt{\sum_{k=1}^{n} |x_k|^2} \tag{6.4}$$

Next, every single value in the column was divided by the $|x|_2$. As a result, the scale of the values in each column was adjusted to $[0, 1]$.

Through these steps, a data matrix was generated for the feature set:

- ***FS_venues***

6.2 Named Entities

6.2.1 Introduction

A *named entity (NE)* is a physically existing object or an abstract existence, which is referred to via a *proper name*. In a text, a *named entity* is commonly denoted by a proper noun, which could be a single word or a combination of several adjacent words.

Some examples include Sheldon Cooper, Auckland, and AUT.

A named entity could be classified into a certain category. The most studied categories include PERSON, LOCATION, and ORGANIZATION. Some other examples include DATE, TIME, and GPE (geo-political entity).

The process of *Named Entity Recognition (NER)* aims to identify textual mentions of the named entities from the surrounding texts, and to further classify them into predefined categories.

There are many approaches to performing the NER task. One group of approaches is based on gazetteers and rules, compared to another, which is based on statistical models.

6.2.2 Approaches based on Gazetteers and Rules

An intuitive method to perform *NER* is to firstly for each category compile a list of entity names, which is also called a *gazetteer* that commonly refers to a domain specific lexicon, and to then check whether words (and phrases) within a text ever appear in these gazetteers. However, the compilation of such gazetteers is reportedly the bottleneck in the design of NER systems (Mikheev, Moens & Grover, 1999).

In addition, *rule*s to determine named entities could be involved. For instance, the system introduced by Riloff and Phillips (2004) has the ability to exploit manually defined pattern rules, each of which can check for lexical, syntactic, and semantic properties associated with words or constituents. Table 6.1 presents two examples of these rules.

The gazetteers and rules could be completely handcrafted. They could also bootstrap from some initial handcrafted instances, and then get iterative extensions semi-automatically. Nevertheless, both of these two strategies require expensive manual efforts. Additionally,

Table 6.1: Examples of Pattern Rules exploited by Riloff and Phillips (2004)

Pattern	<NP-HEAD:word=Corp.> –>NP-COMPANY
Example	*International Business Machines Corp.*
Pattern	<NP-ALLWORDS:case=Capitalized> <PUNC:word=COMMA> <NP-HEAD:sem=LOCATION> –> NP-LOCATION
Example	*Salt Lake City, Utah*

these gazetteer-rule-based approaches have low coverage and only work well on narrow domains (Mohit, 2014).

6.2.3 Statistical Supervised-Learning Approaches

By adopting statistical approaches, models to detect named entities are established by supervised learning.

The fundamental requirement for a statistical approach is an annotated corpus, wherein information is embedded to describe whether words in sentences belong to a specific category of NE or not. One scheme to represent the NE information is to mark them as chunks with SGML tags. For example:

> *As introduced by <PER> Sheldon Cooper </PER>, there are significant educational institutions located in the <LOC> Auckland Central </LOC>, notably the <ORG> University of Auckland </ORG> and the <ORG> AUT </ORG>.*

The more popular *sequence labelling* schemes assign a label to each word. For instance, the BIO representation uses labels to indicate whether a word is at the **B**eginning, **I**nside, or **O**utside an NE.

> *As/O introduced/O by/O Sheldon/B-PER Cooper/I-PER, there/O are/O significant/O educational/O institutions/O located/O in/O the/O Auckland/B-LOC Central/I-LOC, notably/O the/O University/B-ORG of/I-ORG Auckland/I-ORG and/O the/O AUT/B-ORG.*

A variant of BIO, the BILOU representation which significantly outperforms the former

one (Ratinov & Roth, 2009), additionally uses the label **L** to point out the last word of an NE, and uses **U** to denote that a word is a unit-length NE.

> *As/O introduced/O by/O Sheldon/B-PER Cooper/L-PER, there/O are/O significant/O educational/O institutions/O located/O in/O the/O Auckland/B-LOC Central/L-LOC, notably/O the/O University/B-ORG of/I-ORG Auckland/L-ORG and/O the/O AUT/U-ORG.*

The creation of an annotated corpus requires intensive manual work. However, as long as it accomplished, the corpus can be used by an unlimited number of statistical and machine learning approaches.

Subsequently, a probabilistic representation of the annotated corpus needs to be established. This representation is a *probabilistic model* (also *statistical model*). From the mathematical perspective, a statistical model encompasses a collection of possible observations (also *event*s) and the probability distributions of these observations. The collection of events is called the *sample space*. In the setting of NER, an example *event* is that the word "Central" appears after "Auckland" and is part of a LOCATION entity.

More specifically, the models involved in statistical approaches for NER are generally belong to the *probabilistic graphical model (PGM)*, which uses a graph to express the structure of conditional dependencies between the observations. For instance, in a naive PGM, a node "Auckland" could have two outgoing connections to the node "Central" and the node "and", while for each of the connections a probabilistic value is correlated.

Some well known PGMs for the NER task include the *Hidden Markov Model (HMM)* (Mohit, 2014; Freitag & McCallum, 1999), the *Max Entropy Markov Model* (*MEMM*, also named *Conditional Markov Model, CMM*) (Borthwick, 1999), and the *Conditional Random Field (CRF)* (Finkel, Grenager & Manning, 2005). These PGM approaches take the *sequence labelling* view over the NER task which considers it as a *structured prediction* problem for a sequence of variables (Mohit, 2014). That is to

take sequences of tokens (sentences which are constituted by words in the setting of NER, as well as *Part-Of-Speech (POS)* tagging) as inputs and deliver a sequence of labels for each sequence of tokens as output. Named entities can then be recognised by understanding these labels. This is different from the approaches based on gazetteers and rules, which directly pick out entities from sentences. This is also distinct from a classification process, which produces a class label for each input entry.

More specifically, for a given sequence of tokens $(t_1...t_n)$, the task is to determine the most possible sequence S of labels $(y_1...y_n)$:

$$S = \underset{y_1...y_n}{\operatorname{argmax}} \; P(y_1...y_n \mid t_1...t_n) \tag{6.5}$$

According to the Bayes' theorem:

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)} \tag{6.6}$$

for a certain token sequence $(t_1...t_n)$, the problem could be transferred and simplified to:

$$S = \underset{y_1...y_n}{\operatorname{argmax}} \; P(t_1...t_n \mid y_1...y_n) \cdot P(y_1...y_n) \tag{6.7}$$

More Details of HMM

The HMM assumes that the token sequence is generated from the learnt statistical representation of the training data. During the generation, there is a *Markov process*, which means that the probability of assigning a label to a token depends only on a limited number of recent tokens and their labels. If an HMM only examines one previous token and its label during the generation, it is called the *first order HMM*. The

assumption of the first order HMM further allows to shorten the context for computing $P(y_1...y_n)$ and simply use $P(y_i \mid y_{i-1})$. Thus for a given sequence of tokens $(t_1...t_n)$, the intended sequence S of labels $(y_1...y_n)$ should satisfy:

$$S = \underset{y_1...y_n}{\operatorname{argmax}} \prod_{i=1}^{n} P(t_i \mid y_i) \cdot P(y_i \mid y_{i-1}) \tag{6.8}$$

In the HMM approach, a *finite state machine (FSM)* is involved, wherein each state is corresponding to a label. The Figure 6.4 shows the FSM of a simplified HMM which could help to detect NE without category information.

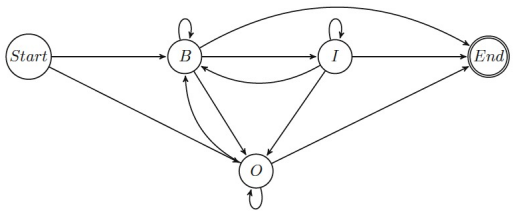


Figure 6.4: FSM of A Simplified HMM for NER (Mohit, 2014)

During the process of generating, each token position in the sequence is corresponding to a probabilistic state transition, where next label is decided, and after the transition, a new token is generated according to the new label.

During the training of the HMM, two sets of parameters are studied from the training data (annotated corpus in NER setting):

- $P(y_i \mid y_{i-1})$ - *state transition probability*, that is the conditional probability of the label of current token given the label of previous token
- $P(t_i \mid y_i)$ - the probability of generating a specific token given a certain status

In general, the procedure of determining a state sequence for a certain token

sequence is called *decoding*. For the HMM decoding, a commonly used algorithm is the Viterbi. It is well explained by Jurafsky and Martin (2014).

A Brief Comparison

The HMM, MEMM, and CRF are all *statistical hidden state sequence models* (Finkel et al., 2005). They need statistics of annotated corpora for training. They work with sequence of tokens whose states are not explicitly known.

The HMM assumes that tokens are independent of each other. By contrast, the MEMM and CRF do not have such an assumption. This allows the two to benefit from various further features.

While labelling a token, the HMM and CRF both consider future observations, which however, is not taken into account by MEMM.

To conclude, the CRF is the best among the three introduced models.

6.2.4 Other Approaches

Unsupervised techniques are reportedly used to augment, but not replace, handcrafted features and domain-specialised knowledge resources such as gazetteers (Lample, Ballesteros, Subramanian, Kawakami & Dyer, 2016).

The SVM technique has been employed as a supervised learning approach for NER tasks (Asahara & Matsumoto, 2003; Ju, Wang & Zhu, 2011).

Neural architectures have also been used. Collobert et al. (2011) adopted CNN together with CRF. Lample et al. (2016) adopted LSTM, a variant of the Recurrent Neural Network, also together with CRF, to construct their NER system.

In practice, many (if not most) of the NER systems combine two or more of the

previously mentioned approaches. These systems have shown improvement over their non-hybrid counterparts (Mohit, 2014).

6.2.5 Features Employed by NER Systems

Nadeau and Sekine (2007) summarised several features of words, which could be exploited by NER systems. The word-level features include:

- **case** - whether a word is uppercased, lowercased, capitalised, or mixed-cased
- **punctuation** - whether a word contains special punctuations such as period, apostrophe, hyphen, or ampersand
- **digit** - whether a word matches special digit patterns, for example, the forms of dates, percentages, etc.
- **morphology**:
  - prefix, suffix of a word
  - stem, or lemmatised form of a word
  - whether a word has a special ending
- **features by special functions**:
  - n-grams of characters
  - alphabetical or non-alphabetical
  - lowercased or uppercased transforms
- **POS-tag**:

  Standing for the **p**art-**o**f-**s**peech tag, a POS-tag is a label correlated to a word which denotes its category, from the perspective of the role the word

plays within the grammatical structure of the surrounding context, which is commonly a sentence.

Interestingly, the task of automatic POS-tagging is similar to NER. The techniques used in NER tasks for feature extracting (introduced in this subsection except for POS-tag itself), model establishing and label predicting (such as HMM, CRF, and the ANNs) for NER are also suitable for the POS-tag task.

While some NER implementations require the POS-tag as a necessity, e.g. NLTK [3], some others do not, for instance, the Stanford NER [4], which declares that it use similar features for POS-tag and NER tasks [5].

The context level features include:

- number of occurrences of a word (or phrase) and its transforms such as uppercased, lowercased, and capitalised, in the document
- other entities in the context
- cooccurrences, which means other words frequently appear alongside a certain word
- coreferences, which are other expressions of the same entity

In a recent study, Lample et al. (2016) combined *character-based representation* (Ling et al., 2015) and *distributed representation* (Mikolov, Sutskever et al., 2013) as the features of the to be examined words.

[3] http://www.nltk.org/book/ch07.html
[4] https://nlp.stanford.edu/software/CRF-NER.shtml
[5] https://nlp.stanford.edu/software/crf-faq.html#pos

6.2.6 Results

The present study employed the Stanford NER to recognise named entities from desc-blocks and talk-blocs of the prepared television series scripts. The Stanford NER expolits the CRF as the underlying statistical model. Meanwhile, it adopts the Gibbs sampling method to exploit features from long distance structures (Finkel et al., 2005).

Three categories of NEs were collected, including LOCATION, ORGANIZATION, and PERSON.

A PERSON entity who is not a character in the television series might be a famous figure in the real world or in an fictional setting. The mentions of these famous figures might be influential to the ratings of the episodes, hence these entities are of interest. A series of processes were adopted to identify potential famous individuals.

- First, the PERSON entities whose names are identical to a character's name were filtered out. The collections of characters' names used here were compiled from the speakers of the talk-blocks as introduced in §4.2.1.
- Second, entities with some special names were abandoned. These include *John Doe* and *Jane Doe*, which are names used as placeholders in some circumstances[6]. Another instance is *DING DONG*, which is the sound of a doorbell nevertheless mistakenly classified.
- Third, the names which refer to identical PERSON entities were unified.
  - The names *JFK*, *JOHN F KENNEDY*, *KENNEDY* were all altered to the *JFK*.

[6]https://en.wikipedia.org/wiki/John_Doe

  - The names *SHERLOCK*, *HOLMES*, *S HOLMES* were all altered to *SHERLOCK HOLMES*.
  - The name *JESUS* was accounted as *JESUS CHRIST*.
  - The name *EINSTEIN* was accounted as *ALBERT EINSTEIN*.
  - *OBAMA* was accounted as *BARACK OBAMA*.
  - *J LO* was accounted as *JENNIFER LOPEZ*.

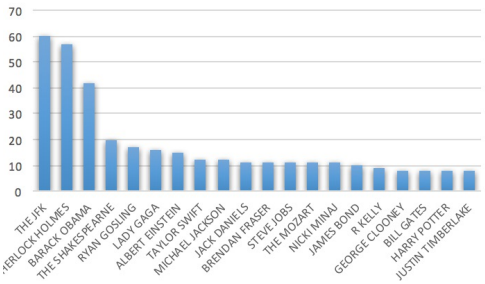
- Lastly, the names which appear in one script only, or comprise only one word whose length is less than eight, were discarded. Before this procedure, there were more than eight thousand items in the examining list, which contains many very common name such as John, Danny, Emily, etc. Remarkably, after this procedure, only some 330 were kept and these are much more expressive.

  Among them, *JESUS CHRIST* is the most frequently mentioned. The name appears 483 times in 216 pilot scripts. Both numbers are much larger than the following examples. To be precise, according to the criteria of script frequency, the second and third most popular names, *BARACK OBAMA* and *JFK*, appear in 28 and 23 scripts respectively; according to the criteria of frequency, the second and third most popular names, *JFK* and *SHERLOCK HOLMES*, appear for 60 and 57 times respectively. Moreover, in the vast majority of cases, people use other names to express person entities. By contrast, people say *JESUS* or *JESUS CHRIST* mostly in order to represent their emotions.

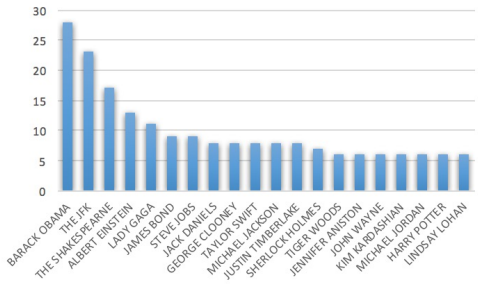
  Therefore, the name JESUS CHRIST is not included in the list of potential famous figures.

Figure 6.5(a) shows names and frequencies of the 20 most popular PERSON entities. Figure 6.5(b) shows names and frequencies of the 20 most popular names by script frequency. An interesting observation is that *JFK* and *OBAMA* are within the top

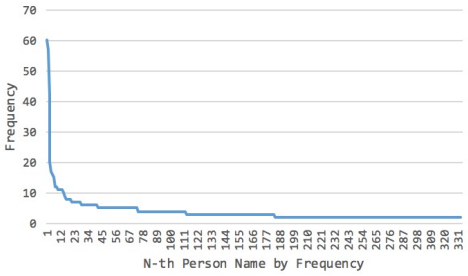
three in both of the two lists. In contrast, the popularities of other politicians, such as *GEORGE BUSH*, *BILL CLINTON* and *HILLARY CLINTON*, are much lower. The scientist *EINSTEIN* and writer *SHAKESPEARE* are also admired by these scripts, whilst the most liked fictional characters are *SHERLOCK HOLMES* and *HARRY POTTER*.



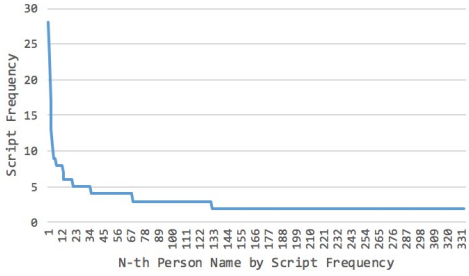
(a) Person - Top 20 by Frequence



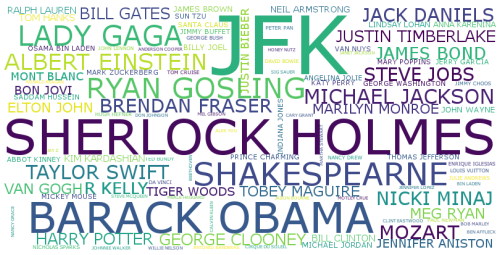
(b) Person - Top 20 by Script Frequency



(c) Person - All by Frequency



(d) Person - All by Script Frequency



(e) Person - Cloud by Frequency

(f) Person - Cloud by Script Frequency

Figure 6.5: Named Entity - Person

The collected LOCATION and ORGANIZATION entities were also processed by handcrafted rules. Some of them were eliminated, while some of them were renamed.

Firstly, in many scripts, names of characters are incorrectly recognised as locations

and organisations. Thus, a procedure of eliminating the locations and organisations whose name is also the name of a character in the same script, was involved.

Secondly, some certain locations are referred to by multiple names. These were unified. For instances, *US* and *USA* were reformed to *UNITED STATES*, while *LA* was changed to *LOS ANGELES*.

Thirdly, many brands were selected as organisations. For example, *PORSCHE*, *CADILLAC*, *NIKE*, and *CHANEL*. A list of brands was manually composed, and these brands were removed from the collection of identified ORGANIZATION entities.

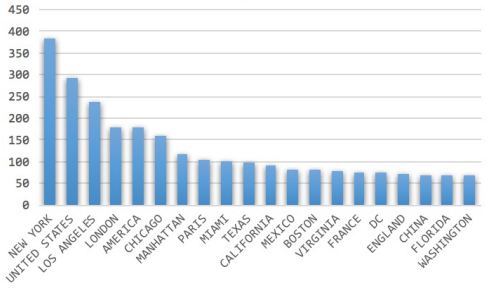
Figure 6.6(a) shows the frequencies and names of the top 20 most popular locations, while Figure 6.6(g) shows these of the top 20 most popular location by script frequency. The Figure 6.6(c) shows frequencies of the top 100 most popular locations.

For the organisations, the three counterpart figures are Figure 6.6(b), Figure 6.6(h), and Figure 6.6(d).

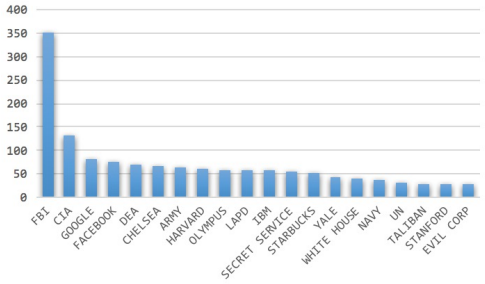
Three categories of named entities for each pilot episodes were determined. Then, names of these entities were directly used as terms instead of drawing n-grams in the process of creating matrices of TF-IDF values as introduced in §6.1.

After these steps, three data matrices corresponding to the three NE categories were generated for these three features sets:

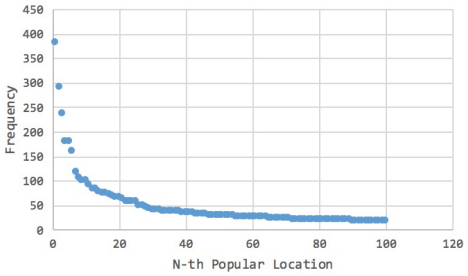
- ***FS_NE_PERSON***
- ***FS_NE_LOCATION***
- ***FS_NE_ORGANISATION***



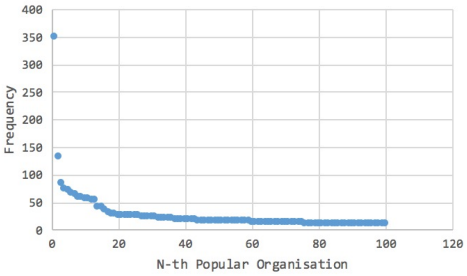
(a) Location - Top 20



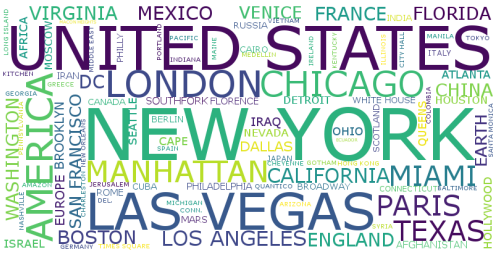
(b) Organisation - Top 20



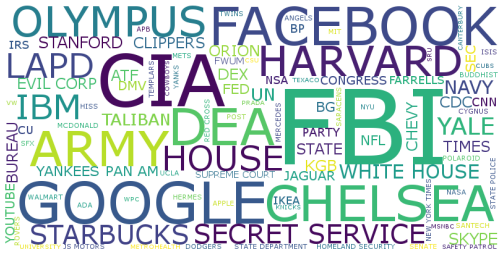
(c) Location - Frequency of Top 100



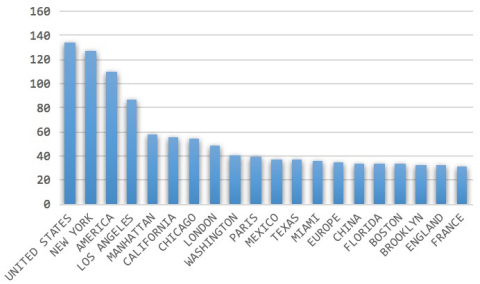
(d) Organisation - Frequency of Top 100



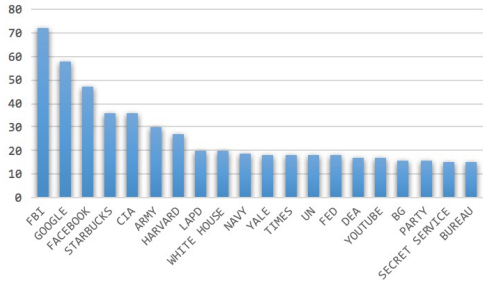
(e) Location - Cloud



(f) Organisation - Cloud



(g) Location - Top 20 by Script Frequency



(h) Organisation - Top 20 by Script Frequency

Figure 6.6: Named Entity - Location and Organisation

6.3 Keywords of Primary Characters

6.3.1 Keywords and Dependencies

In pilot scripts, descriptions of the primary characters give valuable information concerning their appearance, age, profession, and other characteristics. Additionally, some sentences in the scripts provide temporary statuses of the primary characters. Moreover, certain phrases could reflect the nature of a character indirectly.

The following list shows some example sentences selected from the scripts [7]. In sentences 1. and 2., ages (early30s; late 30s) and appearances (attractive, shoulder-length dark hair; athletic build, rugged good looks) are given. In sentences 1. and 4., professions (doctor and detective) are given. It can also be learnt from sentence 3. that the character is a time traveller. Sentences 5. and 6. represent the primary characters' statuses, which are frustrated and confused. The phrase *holding court* in sentence 7. is a reflection of the social skill or social status of the character.

1. *DR.* CASSIE REYNOLDS, *early 30s, attractive, shoulder-length dark hair*, checks the time.

2. He steps into a beam of light coming from a broken window, revealing: COLE, *late 30s, athletic build, rugged good looks*.

3. "A *time traveler* named Cole" disappears before your eyes.

4. *Detective* Olshansky, NYPD.

5. And as she turns to get in line for a dog, and a *frustrated* Sherlock follows.

6. Joan, *confused*, looks to her hands.

[7] Sentences 1., 2., 3., 7. are selected from the pilot script of *12 MONKEYS*; and 4., 5., 6. are selected from *ELEMENTORY*

7. The hand belongs to: Mason Frost, *holding court* with a group of attractive socialites.

These basic settings of the primary characters are also basic settings of the pilot episodes, and the temporary status of a character can be seen as a reflection of an event in the storyline. Therefore, they might be influential to the ratings of these episodes. In the present study, the words and phrases which offer such information, either directly or indirectly, are named *keywords* of the primary characters.

By understanding the relations between the words within the sentences which contain one or more names of the primary characters, the keywords can be detected.

According to Nivre (2006), the syntactic structure of a sentence is constituted of lexical elements, i.e. the words, which are linked by *binary asymmetrical relations*; and these relations are named the *dependencies*. Within each of the *dependency* relations, one word is named the *head* (also *governor, regent*), while the other is the *dependent* (also *modifier*). Moreover, the dependences of dependents to governors can be of various types. For example, a dependence could be established between a verb and its subject or object. Also, it could be established between a noun and an adjective which further specifies the noun. Figure 6.7 shows the dependencies within a sample sentence. In the figure, each dependency is illustrated by a *directed arc*, which points from the *governor* to the *dependent*, with the type of the dependency being presented too. Summarily, a dependency relation could be described by a triple: the two involved words, and the type of the relation.

Many NLP tasks such as machine translation and question answering also benefit from dependency representations since it allows easy access to information concerning predicate-arguments structure and modifiers (De Marneffe, MacCartney, Manning et al., 2006). Moreover, the triple form maps straightforwardly onto semantic web

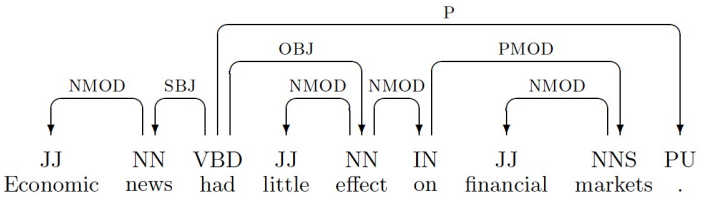


Figure 6.7: Dependencies within a sentence (Nivre, 2006)

representations (De Marneffe & Manning, 2008b).

A list of criteria for constructing dependency relations and for distinguishing the governor and the dependent is compiled by Nivre (2006) based on previous proposals. Denoting the governor (head) as H, the dependent as D, and the grammaric construction (which contains H and D) as C, these criteria are:

1. H determines the syntactic category of C and can often replace C.
2. H determines the semantic category of C; D gives semantic specification.
3. H is obligatory; D may be optional.
4. H selects D and determines whether D is obligatory or optional.
5. The form of D depends on H (agreement or government).
6. The linear position of D is specified with reference to H.

Nevertheless, it is not necessary for a dependency to comply with all of these rules. According to Nivre (2006), Hudson (1991) suggested that typical dependency instances could satisfy all or most of these criteria, while more peripheral instances could satisfy fewer of them. From another perspective, according to Nivre (2006), Mel'čuk (1988) advocated that dependencies can be divided into three categories, which are morphological, syntactic, and semantic, while the syntactic ones are the focus of most

of the works in the field of dependencies. For each category, only a certain subset of the rules are applicable.

Further, the syntactic dependencies in *endocentric* and *exocentric* constructions are worthy to be distinguished. A grammatical construction is endocentric if it fulfils the same linguistic function as one of its parts; otherwise, the construction is exocentric. According to the definition, an endocentric construction naturally satisfies the first criterion, while the exocentric construction apparently fails on that. For instance, in the sentence in Figure 6.7, "*financial* ← *markets*" is endocentric thus it could be replaced by the word *markets*, while in comparison "*on* → *market*" is exocentric so the word *on* cannot serve as a substitute. Another endocentric example is the phrase "*frustrated* ← *Sherlock*" in the 5^{th} sentence selected from scripts. Two typical exocentric instances are the relations between a verb and its object and subject, though these two kinds of dependencies are semantic rather than syntactic.

From another perspective, dependency relations can be classified into categories of *head-complement, head-modifier*, and *head-specifier* (Nivre, 2006).

The distinction between head-complement and head-modifier connects to the conception of *valency*, which is usually related to the semantic predicate-argument structure. As explained by Nivre (2006), "the idea is that the verb imposes requirements on its syntactic dependents that reflect its interpretation as a semantic predicate". A dependent in a *head-complement* relation corresponds to an argument of the semantic structure. The argument could be obligatory, or optional, in which case only one occurrence for each structure instance is permitted. By contrast, a dependent in a *head-modifier* relation does not correspond to any particular argument. It could appear for multiple times. Meanwhile, it tends to be optional.

The head-complement relations are exocentric, and head-modifier relations are

endocentric. As for the head-specifier relation, which is typically exemplified by the determiner-noun relation, its instances are also exocentric (Nivre, 2006).

The collection of all the dependencies extracted from a sentence is eligible to represent the sentence, and this collection is called the *dependency structure* of the sentence. A directed acyclic graph *(DAG)* could be established by using words as vertices and dependencies as directed edges, and ordinarily, this DAG has an explicit root node. For each sentence, a *root* dependency is also usually created, which points from a dummy *"ROOT"* word to the actual root word of the dependency graph, which is typically the main verb of the predicate. It is helpful to the downstream applications.

Commonly, the underlying undirected graph of the dependency graph (the DAG) is a *tree*. That means, by replacing the directed edges with undirected edges, the generated graph is also acyclic. However, this is not always true. For example, from the sentence:

- *Tom and Jerry are running.*

the Enhanced Universal Dependencies Representation proposed by Schuster and Manning (2016) will draw out these dependencies: *nsubj(running → Tom)*; *conj:and(Tom → Jerry)*; *nsubj(running → Jerry)*. By removing the directness, a cycle which involves *(running, Tom, Jerry)* will emerge.

A competing schema for understanding the sentence structure is the *phrase structure*, also known as the *constituency structure*, which views sentences in terms of the constituency relation. From the viewpoint of constituency relation, a sentence is initially divided into two parts, a subject noun phrase (NP) and a predicate verb phrase (VP). Then each phrase will be recursive split until only one word left within a phrase. Accordingly, the phrase structure described is the nesting of multiword constituents, whereas for dependency structure, it is the relations between individual

words (De Marneffe et al., 2006).

In a tree that represents the phrase structure of a sentence, the root node corresponds to the entire sentence, the leaves correspond to the words, and other nodes correspond to different levels of phrases. Meanwhile, the content of each non-leaf node is the combination of the contents of its children. For instance, for a simple two-word sentence, *"Cole disappeared"*, construction of the phrase tree demands for three nodes, which are the root for the entirety, and two nodes for the subject and predicate respectively.

In a dependency graph, which is not always a tree, each node exactly corresponds to an atomic element, which is a word or punctuation. Therefore, if ignoring the punctuation nodes, the number of its nodes would be consistently less than the number of nodes in a phrase tree that represents the same sentence. Figure 6.8 [8] shows the phrase tree and dependency tree of the same sentence.

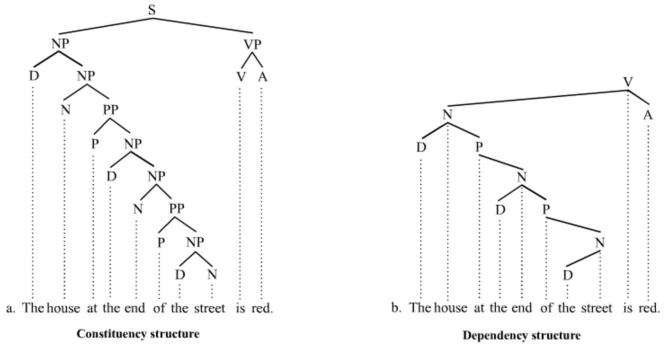


Figure 6.8: Phrase Tree and Dependency Tree of a sentence

A *treebank* is a corpus where the sentence structures are parsed and annotated. While original treebanks only provide *phrase structure* trees, several later ones adopt *dependency structure* representation as the primary annotation format (De Marneffe et al., 2006). A treebank could be created completely manually or semi-automatically wherein the premiere result produced by a parser is later verified and corrected if necessary. Either method requires intensive work by trained linguists. De Marneffe et

[8] https://en.wikipedia.org/wiki/Treebank

al. (2006) described a system that is capable of taking in phrase structure parses and then generating typed dependency relations for English sentences.

6.3.2 Dependency Representation

There are various dependency representation standards, which employ different methods to extract and categorise dependency relations. Between a particular pair of words in a sentence, according to a certain standard, there might be no dependency relation; while according to another standard, there might be a relation of a certain type; further, according to a third standard, the relation between the two words might be of a distinct type.

The system introduced by De Marneffe et al. (2006) presents word relations in the *Stanford Typed Dependencies Representations* (the *SD*). According to De Marneffe and Manning (2008b), the development of SD is based on three early works, which are the Lexical-Functional Grammar (recently explained by Bresnan, Asudeh, Toivonen and Wechsler (2015)), GR (Carroll, Minnen & Briscoe, 1999), and PARC (King, Crouch, Riezler, Dalrymple & Kaplan, 2003), especially in terms of the grammatical relations and type naming. What makes SD different is that it is designed to be an empirical model for sentence representation, particularly in the contexts of relation extraction and information extraction (De Marneffe & Manning, 2008b). Besides distinguishing the argument and adjunct relation, SD pays more attention on NP-internal relations and introduces new types such as the appositional modifier and adjectival modifier. Opposite to the PARC representation, which provides information about individual words, such as verb tense, adjective degree, type of named entity, etc., the SD does not.

The *SD* defines more than 50 grammatical relations, and these could be organised in a hierarchy, rooted by the *dependent* relation which is the most generic (De Marneffe

& Manning, 2008b, 2008a). The first level children include *arg* (argument), *mod* (modifier), *cc* (coordination), *conj* (conjunct), etc. Some descendants of the *arg* node are *nsubj* (nominal subject) and *dobj* (direct object). Some descendants of the *mod* node are *amod* adjectival modifier, *advmod* (adverbival modifier), and *appos* (appositional modifier). When categorising a dependency relation, if it could be identified more precisely, then a type tag further down in the hierarchy is assigned to it. Otherwise, a more generic type tag is used.

There are various styles of SD representations (De Marneffe & Manning, 2008a). Some are briefly introduced in the following discussion. The *basic* typed dependencies of a sentence construct a tree structure. Each word in the sentence is a dependent in exact one relation. In the *collapsed* style, dependencies related to prepositions, conjuncts, etc. are collapsed to get direct dependencies, i.e. two dependency entries in *basic* style could merge into one in this style. Dependencies, which could break the tree structure, are taken into consideration. The *CC-processed* style (collapsed dependencies with propagation of conjunct dependencies) makes more efforts towards sentences that contain conjunctions. It does not guarantee the tree structure either.

The *SD* has been adopted by researchers in different domains, including for evaluating newly proposed parsers (De Marneffe & Manning, 2008b).

A promotion for the original *SD* (De Marneffe et al., 2006; De Marneffe & Manning, 2008a, 2008b) was proposed by De Marneffe et al. (2013) to handle two limitations. First, the original SD was incapable of understanding many unusual but difficult constructions such as tough adjectives, free relatives, comparative constructions, and small clauses. Second, the original SD was developed against newswire data. Hence constructions which appear less, including questions, imperatives, discourse particles, sentence fragments, and ellipsis have not been adequately considered. (De Marneffe et al., 2013)

Among many dependency categories introduced by the promotion, the *vocative*, is used to label the dialogue participant recognised in texts.

The *SD* was originally designed only for, and eventually emerged as the de facto standard of, the dependency analysis of the English language. (Nivre et al., 2016)

A representation named *Universal Stanford Dependencies* was proposed by De Marneffe et al. (2014) which attempts to reconstruct the underlying typology of *SD* so as to make it more applicable cross-linguistically while keeping faithful to the design principles of SD.

The Google Universal Dependency Treebank project introduced by McDonald et al. (2013) uses SD as the starting point and adds modifications to capture the variety of grammatical structures in diverse languages.

The *Universal Dependency* (the *UD*) is another proposal on multi-language competent representation (Nivre et al., 2016). It is a single coherent standard by merging the *Universal Stanford Dependencies* and *Google Universal Dependicies*, as well as some other previous works [9], and it intended to replace these predecessors. The *UD* provides a universal inventory of categories to facilitate consistent annotations of similar constructions. It allows language-specific extensions to handle the idiosyncrasies among the typological differences between the languages.

Schuster and Manning (2016) proposed two extensions of *UD*: the *enhanced UD* and *enhanced++ UD* representations. The original UD (Nivre et al., 2016) work focuses on the *basic* UD representation, which corresponds to the *basic SD* representation, and mainly aims to meet the requirements of treebank annotation tasks. Meanwhile, the *enhanced UD* and *enhanced++ UD* correspond to the *collapsed* and *CC-processed SD* representations. The graph of a sentence by one of the later two might not be a tree

[9]These include SD, UDT (Google Universal Dependency Treebank), Google universal P.O.S tags, HamleDT, etc., according to Nivre et al. (2016).

(Schuster & Manning, 2016). Compared with the *basic UD*, they are more suitable for tasks which are concerned with the relations between content words, such as relation extraction.

The Stanford Dependency Parser [10], which is employed in the present work to extract word dependencies from pilot scripts, has once adopted the *SD* as the representation of its output. Since the release of version 3.5.2 of the tool in 2015, the default output representation switched to the *UD* [11] [12].

6.3.3 Dependency Parsing

Dependency parsing is the process of extracting dependency relations between words (and possible punctuations) from sentences. Nivre (2006) divided the dependency parsing approaches into two genres: the *grammar-driven* and the *data-driven*, which are not mutually exclusive.

Three trends of the *grammar-driven* approaches are distinguished. First, some grammar-driven approaches are tied to a formalisation of dependency grammar which is closely related to context-free grammar (Nivre, 2006). Second, some of them consider the task as a constraint satisfaction problem and adopt eliminative parsing, which means possible representations of a sentence are successively eliminated by checking whether they violate certain constraints. The third trend does not essentially demand complex grammars and parsing algorithms as the first two do. Instead, it is based on a deterministic parsing strategy, which, according to Nivre (2006), basically "accept words one by one starting at the beginning of the sentence and try linking each word

[10] https://nlp.stanford.edu/software/stanford-dependencies.shtml
[11] See the *READEME* in https://nlp.stanford.edu/software/stanford-parser-full-2016-10-31.zip
[12] For the newest version of UD: http://universaldependencies.org/

as head or dependent of every previous word" (Covington, 2001). Its requirement to the grammar is to provide a function to determine whether a word could be *governor* (head) of another.

The hallmark of a *data-driven* approach is the exploiting of a *probabilistic model*, while for a *grammar-driven* approach, the hallmark is the employment of a formal *dependency grammar*.

According to Nivre (2006), Eisner's works is influential in the field of dependency parsing. These include the introductions of several probabilistic models (J. M. Eisner, 1996; J. Eisner, 1997) and a further proposal of the notions *bilexical grammar* and *weighted bilexical grammar* (J. Eisner, 2000). These works showed that *generative probabilistic modelling* and supervised learning could be used in dependency parsing missions and produce relatively good accuracy.

The approaches suggested by Kudoh and Matsumoto (2000); Yamada and Knight (2001) use *SVM* to train classifiers to predict the next action of a shift-reduce procedure, which repeatedly takes in words of a sentence from left to right along with producing dependencies. During the shift-reduce procedure, two adjacent words are focused and referred to as target words. The two target words are denoted as w_i and w_{i+1} respectively. Possible actions, which are output of the classifier, include:

- *Shift* moves the focus window to right: $(w_i, w_{i+1}) \Leftarrow (w_{i+1}, w_{i+2})$.
- *Right* constructs a dependency relation between the two target words; add w_i as child of w_{i+1}; and: $(w_i, w_{i+1}) \Leftarrow (w_{i-1}, w_{i+1})$.
- *Left* constructs a dependency relation between the two target words; add w_{i+1} as child of w_i; and: $(w_i, w_{i+1}) \Leftarrow (w_{i-1}, w_i)$.

Chen and Manning (2014) described the Stanford Neural Network Dependency Parser (the *Stanford nndep*), which is based on neural networks and transitions.

The transition system of *Stanford nndep* adopts the *arc-standard* algorithm. In the arc-standard system, there are a buffer **b** which stores the to-be-processed part of a sentence, a stack **s** which contains the being focused elements (words and maybe punctuation), and the collection of determined dependency arcs named **A**. At any certain moment, the collection of the states of **b**, **s** and **A** is called the *configuration* **c**.

During the progress of processing a sentence, the configuration changes. The initial configuration is: **s** = $[ROOT_{dummy}]$; **b** = $[w_1, ..., w_n]$; **A** = $\varnothing$. The final configuration is: **s** = $[]$; **b** = $[]$; **A** = $\{d \mid all\_extracted\_dependencies\}$.

The action of the configuration changes from one to another is named a transition. There are three types of transitions defined in the arc-standard algorithms. Denoting s_i as the i^{th} element on the stack, the three transactions are:

- *SHIFT*: fetch next element from **b** and push into **s**.
- *LEFT-ARC(l)*: create arc $s_2 \leftarrow s_1$ with label l and add it to **A**; remove s_2 from **s**.
- *RIGHT-ARC(l)*: create arc $s_2 \rightarrow s_1$ with label l and add it to **A**; remove s_1 from **s**.

The label of an arc is the category of the dependency relation. For example, it could be $amod$ or $nsubj$.

The arc-standard algorithm was employed early by Nivre et al. (2007), followed by various other dependency parsing studies, such as Rasooli and Faili (2012); Zhou, Zhang, Huang and Chen (2015).

The neural network is employed by Stanford nndep to select transactions for the configurations. For each configuration, features are extracted via several steps:

- Compile a set of words S^W, whose size is 18. These include:

  - top three words on the stack and the buffer
  - for the top two words on the stack, select their first and second leftmost / rightmost children
  - for the top two words on the stack, select their leftmost of leftmost and rightmost of rightmost children.

- Compile a set of POS tags S^T by selecting POS tag of each word in S^W.
- Compile a set of arc label S^L, whose size is 12. For each work in S^W, if it is not one of the top three on the stack or the buffer, select its corresponding arc label.
- Subsequently, every word in S^W is represented as a multi-dimensional vector value by adopting the *word embedding* technique. Similarly, every single POS tag in S^T and arc label in S^L is mapped to multi-dimensional vector space.

The feature matrix constituted by these vector values is dense.

Both the POS tag set and the arc label set (either SD or UD) are small discrete sets. Nevertheless, they contain semantical similarities (Chen & Manning, 2014). For instance, the tag NN (singular noun) is closer to NNS (plural noun) than JJ (ajective), and $amod$ (adjective modifier) is closer to $nmod$ (nominal modifier) comparing with obj (object of verb). Thus, the word embedding representation of tags and arc labels are adopted with the expectation of helping the classifier to capture semantic meanings.

Experiments suggested that the nndep can provide accurate dependency parsing results at a fast processing speed. (Chen & Manning, 2014)

6.3.4 Keywords Extraction

Prepareration

For each primary character in each pilot script, sentences that contain the name of the character are collected, from both the descriptive content (*desc-block*) and spoken content (*talk-block*). Subsequently, the *Stanford Neural Network Dependency Parser* and the pre-trained model *english_UD* [13] are employed to extract Universal Dependencies from these sentences.

In addition, a collection named ***person_names*** is compiled, which contains all the names of the recognised Person entities (§6.2) and identified characters (§4.2.1). To be more precise, each element is one word, instead of a full name. For example, the person name "Sheldon Cooper" will generate two elements for the collection: "Sheldon" and "Cooper".

compound

The *compound* dependency is used for noun compounds. Within a noun phrase, the rightmost word is the core, and every other word has a compound dependency on the core. For instance, in the sentence:

- *The author of the paper is Dr. Cassie Reynolds.*

two compound dependencies will be extracted, which are (Reynolds → DR.) and (Reynolds → Cassie).

The majority of the identified character names comprise only one word, which is either the forename or the surname. In case it is the surname, by selecting the compound dependencies whose first word is the character name, the compound dependents can be gathered. If the forename, a little more effort is required for the gathering.

[13] https://nlp.stanford.edu/software/nndep.shtml

Subsequently, by excluding the dependents who appear in ***person_names*** , a group of *keyword*s (though normally contains zero or only one for each noun phrase) is determined, which commonly indicates the profession of the character, such as a doctor, detective, or professor.

apposition

An apposition (or appositional modifier) of a noun phrase *NP*, is another noun phrase immediately to the right of *NP*, which serves to modify or further define *NP*. This relation is represented by the *appos* dependency. For instance, within the sentence:

- *The author of the paper is Dr. Cassie Reynolds, the professor.*

there is an appos dependency: (Reynolds → professor).

For a noun phrase, which is a primary character in the current setting, multiple appositional modifiers may coexist. For example, from the sentence:

- *The author of the paper is the professor, Dr. Reynolds, also a musician.*

two appos dependencies could be extracted, which are (professor → Reynolds) and (professor → musician).

In any of the two circumstances, all the *appositional words*, which include the appositional modifiers and the core noun itself, refer to a same entity. In a group of appositional words, the name of the character might be the core word, which means it is the governor of all other *appos* dependents; it could also be a dependent, in which case its governor, also the core word, is another noun. The case where the name of the primary character serves as both *appos* governor in some dependencies and *appos* dependent in some other dependencies in the same sentence has not been observed.

It is vital to find out all the *appositional words* which refer to the primary character from the sentence being examined. The adopted algorithm is briefly explained below.

Step I, collect all the words any of which is involved in at least one *appos* dependency, with no word is permitted to appear for more than once. It needs to be realised that this collection could not be simply used as the group of appositional words which all refer to the primary character being examined because more than one entity might exist in the sentence referred by multiple appositional words.

Step II, for each selected word, create a *group* that contains it; then gather all the groups to create a *set* named **GS**. Consequently, the size of **GS** is equal to the size of word collection established in Step I. Each element of **GS** is a group that comprises one word, and any pair of groups contain different words.

Step III, for each *appos* dependency, denoting the two involved words as *w_a* and *w_b* respectively; then from **GS**, find out the group *g_a* which contains *w_a* and the group *g_b* which contains *w_b*; finally, if *g_a* does not equal to *g_b*, do merge them into a new group, which will cause the size of GS be reduced by one.

Step IV, repeat the process in Step III until all the *appos* dependencies have been passed through. At last, every group in **GS** will comprise at least two words.

Step V, select the group which contains the name of the character being examined.

In case the identified character name is a forename, more effort is demanded to determine the appositional word group.

It is possible that no group will be eligible. In this case, the name of the character or its compound governor must never have been selected in Step I. An optimisation to handle the case has been adopted.

Each word in the selected appositional word group, except the name of the character itself, is determined as a *keyword*, unless it is in ***person_names*** .

adjective modifier

The *amod* dependency represents the relation between an adjective modifier and the core noun. All the *amod* dependents whose governor is either in the appositional word group determined in the previous subsection or is the name of the primary character being examined are collected and correlated to its governor.

Each selected *amod* dependent is used as a *keyword*. Furthermore, if its governor is not the character's name and not in ***person_names*** , then an additional *keyword* is generated by combining the adjective and the governor.

From the sentence:

- *There comes the famous high-rank agent, the energetic Captain James Bond.*

these dependencies can be extracted:

- *amod* *(agent → famous)*
- *amod* *(agent → high-rank)*
- *amod* *(Bond → energetic)*
- *compound* *(Bond → Captain)*
- *compound* *(Bond → James)*
- *appos* *(agent → Bond)*

Provided the character name is Bond or is James, the *compound* list will be *[Captain, James, Bond]*, the appositional word group will be *{Bond, agent}*, and the selected *amod* dependents will be: *{Bond: [energetic], agent: [famous, high-rank]}*. Eventually, the determined *keyword*s will be *{ captain, agent, energetic, famous, famous-agent, high-rank, high-rank-agent }*.

"named" or "called"

Given the character name is *Goodfellow*, then from a sentence:

- *She is the assistant named Goodfellow.*

the word *assistant* should be selected as a *keyword*. Dependencies extracted from the sentence contain:

- *acl (assistant → named)*
- *xcomp (named → Goodfellow)*

According to De Marneffe and Manning (2008a), the *acl* dependency "is used for finite and non-finite clauses that modify a noun", and "an open clausal complement (*xcomp*) of a verb or an adjective is a predicative or clausal complement without its own subject". A similar dependency of *xcomp* is the *ccomp*, which differently has an internal subject (De Marneffe & Manning, 2008a).

By analysing the *acl* and *xcomp* dependencies, together with checking the words "named" and "called", the objective of extracting *keyword*s from such patterns can be achieved. Further, the *amod* dependencies are also inspected. Therefore, from the sentence:

- *A time traveller named Cole disappears before your eyes.* [14]

for the character Cole, determined *keyword*s include *{ time, traveller, time-traveller }*.

6.3.5 Results

The *Stanford Neural Networks Dependency Parser* achieved accuracies approximate to 0.92 in experiments where models were trained and tested on the English Penn Treebank (Chen & Manning, 2014). The dataset with which the *english_UD* model was trained

[14] selected from *12 MONKEYS*

was not introduced. Nevertheless, since *english_UD* is published for general use, it could be assumed that the style of the language of the pilot scripts is, to a particular extent, distinct from the style of the language of the dataset on which *english_UD* is trained. Subsequently, it could be speculated that accuracy of the extracted dependencies is lower than 0.92.

For instance, from the sentence:

- *The hand belongs to: Mason Frost, holding court with a group of attractive socialites.* [15]

two extracted dependencies are *amod(court → holding)* and *appos(Frost → count)*, which means that *court* is recognised as an apposition of the person Frost and *holding* is an adjective modifier of *court*. This is apparently wrong. However, based on the incorrect dependencies, the *keyword* "holding-court" is determined, which is also reasonable.

Another instance, from the sentence:

- *DR. CASSIE REYNOLDS, early 30s, attractive, shoulder-length dark hair, checks the time.* [15]

extracted dependencies include: *amod(30s → early)*; *appos(REYNOLDS → 30s)*; *amod(30s → attractive)*; *amod(hair → shoulder-length)*; *amod(hair → dark)*; *appos(30s → hair)*; *appos(30s → checks)*. It is inappropriate to recognise "30s", "hair", and the verb "check" as appositional modifiers of the character. In addition, "attractive" should be an adjective modifier of the person instead of her age.

Based on these dependencies, for the character REYNOLDS, the determined *keyword*s are: *{30s, early, early-30s, attractive, attractive-30s, hair, shoulder-length,*

[15] selected from *12 MONKEYS*

shoulder-length-hair, dark, dark-hair}. Despite the inappropriateness of the dependencies, fortunately again, most of these *keyword*s are plausible, except the selection of "check".

However, similar to the word "check", many other selected words by the procedure introduced in the previous subsection seem unsuitable to be a keyword. Some of these words include "sit", "notice", "exchange", "study", "drive", "day", "thing", "thanks", "pace", "watch", and so forth.

Accordingly, a word blacklist was compiled by manually choosing implausible ones from the raw determination of keywords. Then all the selected words were filtered out from the raw selection. This procedure could be further repeated, which will lead to an increase of the blacklist and refinement of the determination of the keywords.

Table 6.2 shows keywords of several primary characters.

Table 6.2: Examples Keywords of Primary Characters

TV Series	**Character**	**Keywords**
12 MONKEYS	Cole	*30, frustrated, late-30s, light, restrained, straightens, team, time, time-traveler, traveler*
ELEMENARY	Sherlock	*cop, eye, frustrated, hesitation, imperceptible, imperceptible-hesitation, racing, smiling, uneasy, unhappy, white-knuckles*
HELIX	Hataki	*accent, american, american-accent, dr., muffled, muffled-accent, pharmaceutical, prepared, team*

Overall, nearly six thousand keywords associated with the primary characters were determined, then a matrix of TFIDF values for the pilot scripts was generated, which is named the feature set ***FS_keywords***.

6.4 Activities

The present work employed the *Stanford Open Information Extraction* [16] (*Stanford Open IE*) to extract relation triples from pilot scripts. Terms that describe activities were then created based on these triples.

6.4.1 Stanford Open IE

The approach of *Stanford Open IE* consists of two stages: split a sentence into independent clauses, and shorten the clauses by removing unnecessary components.

splitting

An example of splitting a sentence into clauses is from the sentence:

- *Born in a small town, she took the midnight train.*

the produced clauses include:

- ▷ *she took the midnight train*
- ▷ *she born in a small town*

The objective of this stage is to yield clauses, any of which is independent both syntactically and semantically, and is entailed by an original sentence.

To perform the split, firstly the establishment of the dependency tree of a sentence is required by the *Stanford Open IE*. Subsequently, the dependency tree is traversed recursively. At every node, for each of the outgoing edge, which is an edge that points to any of its children, the action required needs to be determined. This action could be decomposed into two parts that include the action to be applied on the dependency edge, and the action to be applied on the being focused node.

[16] https://nlp.stanford.edu/software/openie.html

There are three categories of actions that could be performed on the dependency edge:

- **Yield** yields a new clause on the dependency edge and then recurses on the edge. A typical example of this action is that the dependency *ccomp(said → like)* in the sentence "$Poseidon\ said\ that\ you\ like\ to\ swim$", which would yield the clause "$you\ like\ to\ swim$".

- **Recurse** recurses on the edge without yielding new clause. A circumstance suitable for this action is that in the sentence:

  ▸ *The incident that Mr. Careless lost his gun was known to the public.*

  the intermediate constituent "$the\ incident\ that\ Mr.\ Careless\ lost\ his\ gun$" should be further examined to identify the clause "$Careless\ lost\ gun$".

- **Stop** instructs not to recurse on this edge since it is supposed that no clause is entailed in the subtree. This is the appropriate action for most (if not all) of the edges which connect leaf nodes. It might also be selected for other edges in which case the algorithm could be made more efficient.

There are also a number of categories of actions that could be performed on the node being examined:

- **Subject Controller** - This action is designed for the case that the subject of the node being examined needs to serve as the subject of the child node. For example, in the sentence:

  ▸ *Born in a small town, she took the midnight train.*

  there exists the dependency *advcl(took → born)*, and the subject of $took$, which is she, should be used as the subject of $born$ in the extracted clause.

- **Object Controller** - Analogous to the above action, this is designed for the case that the object of the node being examined needs to serve as the subject of the child node. For example, in the sentence:

  - *I persuaded Fred to leave the room.*

  there exists the dependency *xcomp(persuaded → leave)*, and the object of $persuaded$, which is $Fred$, should be used as the subject of $leave$ in the extracted clause.

- **Parent Subject** - As Angeli, Premkumar and Manning (2015) explained, this action is prepared for the case that the node being examined has only one outgoing edge and should be taken "as the (passive) subject of the child".

Although more actions for the focus node could easily be imagined, Angeli et al. (2015) reported that these three already supply good coverage in practice.

A classifier was trained to determine actions for traversing the dependency tree (Angeli et al., 2015). The adopted features to train the classifier include the label of the edge (category of the dependency), the label of the incoming edge of the node being examined, POS tags of the involved nodes, etc.

For a brief conclusion, the classifier determines actions for an edge and the governor of the dependency, and a Yield action will generate a clause.

shortening

The objective of this stage is to, (1) by removing unnecessary components, further compact the clauses while retaining their core semantics, and eventually, (2) based on the condensed clauses, create triples each of which contains a subject, a verb phrase, and an object.

In the logic for shortening clauses, many factors are considered:

- whether there is one of the *natural logic operators*, which include all, $some$, no, $many$, etc.
- whether removing the dependent of an arc will make the governing component more general, more specific, or neither. For example, removing red from *amod(car → red)* will make it more general, while removing $approximately$ form *advmod(100 → approximately)* will make it more specific. Therefore, removing a nonsubsective adjective usually results in a significant change of the meaning of a constituent.
- whether the dependent is a *nonsubsective* adjective.

  The *nonsubsective adjective* is a special type of adjective. If a phrase consists of a nonsubsective adjective and a noun, then this phrase is not an instance of that noun (Pavlick & Callison-Burch, 2016). For example, a *fake policeman* is not a *policeman*.

  Therefore, removing a nonsubsective adjective usually results in a significant change of the meaning of a constituent. The list of nonsubsective adjectives compiled by Nayak, Kowarsky, Angeli and Manning (2014) is exploited by *Stanford Open IE*.
- whether removing of the prepositional attachment would break the integrity of a meaning.

Details of the compact logic is explained by Angeli et al. (2015).

Once compact clauses are produced, triples are built by matching them with several straightforward patterns.

6.4.2 Results

A triplet comprises a subject, a verb phrase, and an object phrase. Activity terms for each generated triplet are built. The steps are:

- For any word in the verb phrase and the object phrase, if it is equal to a recognised person name, then it is replaced with the word $someone$.
- All the words in the verb phrase and the object phrase are lemmatised.
- Combine all words in the verb phrase and determine it as an activity term, unless the combination is included by a manually created blacklist, which contains be_in, be_on, be_at, etc.
- Combine all words in the verb phrase and the object phrase and determine this as an activity term, unless the combination contains more than five words.

Thus, usually two activity terms are created from one triplet.

Eventually, nearly 294 thousand activity terms were determined. Among them, approximately 19 thousand appear in more than one pilot scripts. These circa 19 thousand terms were selected as features, and a data matrix of TFIDF values of these features for the pilot scripts was generated, which is named the features set ***FS_activies***.

Figure 6.9 shows the top 20 activity terms by frequency. Figure 6.10 shows the top 20 activity terms by script frequency. [17]

[17] The lemmatisation result of $stares$ is $star$. The word $gonna$ is transformed into gon in triplets.

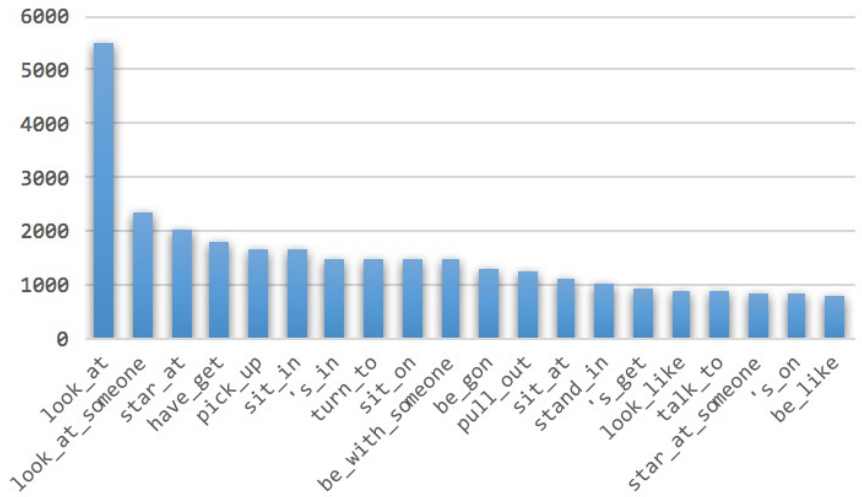


Figure 6.9: Activity Terms - Top 20

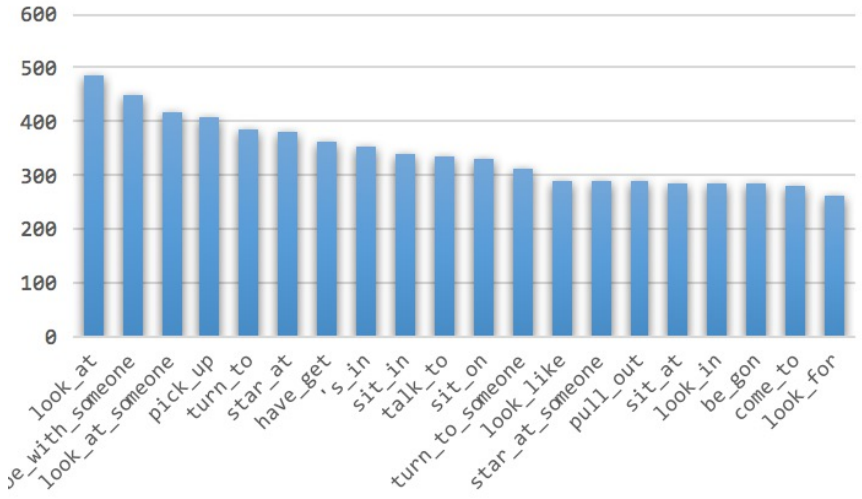


Figure 6.10: Actitity Terms - Top 20 by Script Frequency

Chapter 7

Predictive Models and Results

7.1 Genres

The genre information of the TV series drawn out from the IMDB data source is used as a part of the predictive features.

Different genres are not mutually exclusive. In practice, a television series commonly belongs to multiple genres. For instances, the series *12 Monkeys* belongs to *drama, mystery, adventure, thriller*, and *sci-fi*; and another series, *The Chicago Code*, belongs to *action, crime*, and *drama*.

The 512 television series involved in the present work belong to 20 genres. The Figure 7.1 shows these genres and their frequencies. The most popular genre is *drama*, with 333 TV series belonging to it. Meanwhile, some genres rarely appear, such as biography, musical, and western.

After the establishment of the intended predictive model, the genre information of an input pilot script (which apparently is not one of the 512) might not be provided. Hence, eleven genres, whose frequency is larger than 20, were selected and classifiers will be trained for each of them. These eleven genres are: *action, adventure, comedy,*

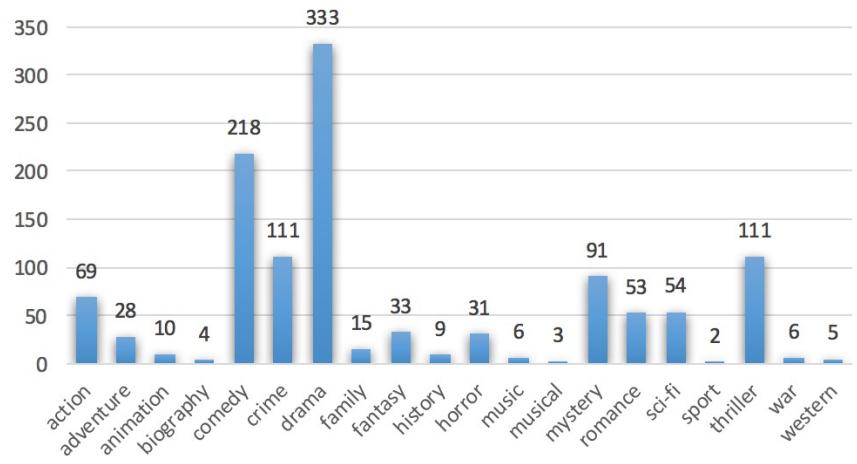


Figure 7.1: Frequencies of Genres

crime, drama, fantasy, horror, mystery, romance, sci-fi, and *thriller*.

The present work employs the *scikit-learn*[1] library to train genre classifiers.

7.1.1 Genre Classifiers

The present study employed two classification algorithms, which are the Naive Bayes and Multilayer Perceptron, to construct genre classifiers. Naive Bayes is fast and easy to implement, thus is often adopted as a baseline in text classification tasks (Rennie, Shih, Teevan & Karger, 2003). The Multilayer Perceptron approach has the advantage of yielding good performances (C. H. Li & Park, 2006).

For each genre, three groups of classifiers were trained.

Group I: BOW-NB

The datasets exploited in this group are matrices of **TF-IDF**[2] values extracted via the bag-of-words language models of the pilot scripts.

During the process of constructing the bag-of-words models, various values for the

[1] http://scikit-learn.org
[2] see **Step IV** in §6.1

parameters were explored. The five parameter options for the step of creating n-gram terms are: $(1,1), (1,2), (1,3), (2,2), (2,3)$. Here, the pair (min, max) indicates that a term is permitted to contain at least min words, but no more than max. Thus, $(1,1)$ allows only the unigram, while $(1,3)$ allows unigram, bigram, and trigram. The five options for the parameter concerning how many top frequent terms to remain are $1000, 2000, 5000, 10000$, and 100000. Consequently, there are 25 possible parameter combinations, which lead to 25 data matrices of **TF-IDF** values.

The classifier employed in this group is the *Naive Bayes* [3].

Group II: DR-NB

The datasets exploited in this group are the four data matrices which comprise distributed representations of the pilot scripts (§5.3).

The classifier employed in this group is also the *Naive Bayes*.

Group III: DR-MLP

The datasets exploited in this group are identical to the datasets in Group II.

The classifier employed in this group uses a *multilayer perceptron* (*MLP*, which is a type of *feedforward neural network*) algorithm that trains via Backpropagation [4]. Several options are provided concerning the structure of the hidden layers: $(5,)$, $(10,)$, $(15,)$, $(15,10)$, $(20,)$, $(15,15,15)$. Here, the length of a tuple explains the number of hidden layers, and a figure expresses how many neurones are in its corresponding layer. For instances, $(5,)$ indicates one hidden layer containing five neurones, and $(15,10)$ indicates two hidden layers containing fifteen and ten neurones respectively. Two options for the parameter concerning the maximum number of iterations are 100 and 200. Options for the $L2$ penalty parameter are 0.1, 1, 5, 10, 15, and 20. There are 72 different combinations of the parameters in total. Thus 72 instances of the MLP classifier were trained.

[3] http://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes
[4] http://scikit-learn.org/stable/modules/neural_networks_supervised.html#classification

For each genre, there are two possible predicted classes, *pos* and neg, indicating whether a TV series belongs to the genre or not. An issuer for any of the genres is that the classes are unbalanced. Accordingly, the *SMOTE (Synthetic Minority Over-sampling TEchnique)* (Chawla, Bowyer, Hall & Kegelmeyer, 2002) procedure was applied to all the four datasets for all of the eleven genres to oversample the minority class to achieve balances.

The SMOTE procedure first determines a number of (five in the current work) nearest neighbours of an instance (the seed instance) of the minority class, and then creates synthetic samples based on the neighbours and the seed instance itself as new fake instances of the minority class.

A synthetic instance is generated by selecting a random point at the line segment which connects the seed instance and one of its neighbours in the multi-dimensional space defined by the features.

Some advanced variants of the original SMOTE include Borderline-SMOTE (Han, Wang & Mao, 2005) and SVM-SMOTE (Nguyen, Cooper & Kamei, 2011).

7.1.2 Results

The accuracies of the classifiers were measured by the *F-score* of the *pos* class. Denoting the *F-score* of the *pos* class as F_{pos}, the equation is:

$$F_{pos} = 2 \cdot \frac{Precision_{pos} \cdot Recall_{pos}}{Precision_{pos} + Recall_{pos}} \tag{7.1}$$

wherein, $Precision_{pos}$ is the rate of the number of correctly identified positive items to the number of all identified positive items, $Recall_{pos}$ is the rate of the number of correctly identified positive items to the number of all actual positive items, and the *F-score* is the harmonic mean of the two values.

The classifier instances are assessed via the three-fold cross validation.

The accuracies of the best classifiers in each of the three groups, for all the eleven genres, are in Table 7.1. Some observations are:

Table 7.1: Results of Genre Classifiers

GENRE		BOW-NB	DR-NB	DR-MLP
action	69	.4031	.4318	.5003
adventure	28	.2732	.3448	.4002
comedy	218	.7790	.8195	.8209
crime	111	.5856	.5694	.6511
drama	333	.8521	.8627	.8721
fantasy	33	.2508	.3710	.4261
horror	31	.1862	.2885	.4215
mystery	91	.4698	.4785	.4836
romance	53	.2090	.2597	.3477
sci-fi	54	.3790	.4392	.5638
thriller	111	.5304	.5303	.5655

- The best classifier, whose accuracy is 0.8721, was trained for the genre *drama* with the *MLP* algorithm, exploiting distributed representations associated with the feature set ***FS_dr_subt_d2v***.
- Only two genres, the *drama* and *comedy*, have trained classifiers that can produce accuracies better than 0.80.
- With the number of television series that belong to a certain genre becoming larger, the accuracy of the best classifier for this genre becomes higher. The genre *drama* possesses the most (333) TV series and its best classifier gives the best accuracy (0.8721). The genre *comedy* possesses the second most (218) instances, and its best classifier gives the second best accuracy (0.8209). Other genres follow the same trend, except for *mystery* and *romance*.

- While comparing classifiers of a single genre, the best DR-MLP classifier always outperforms the best DR-NB classifier, which normally outperforms the best BOW-NB classifier.

  An exception is the genre *drama*, whose best BOW-NB classifier is better than its best DR-NB classifier. Another exception is the genre *thriller*, whose best BOW-NB classifier and best DR-NB classifier have very similar performances.

  Moreover, the best DR-MLP classifiers of *horror* and *sci-fi* promote the classification accuracies of these two genres for more than 10 percent.

More details about the genre classifiers are presented in Appendix C.

The feature set ***FS_genre*** is designed to represent whether a TV series belong to the discussed genres.

7.2 Model

7.2.1 Data Preprocessing

Ratings

The ratings of pilot scripts extracted from the IMDB data source are used as the target feature. The IMDB ratings are on a scale from 0 to 10, rounded to one decimal place. For instance, the rating of the pilot of *12 Monkeys* is 8.2.

Amongst the ratings of the 512 pilots, the highest value is 9.4, and the lowest value is 3.7. The most frequent rating value is 7.5. The Figure 7.2 depicts the distribution of the pilot ratings.

According to their ratings, the 512 pilots were then classified into three bins.

- ***RB_30*** ranges from 0.0 to 7.3 and contains 162 pilots.

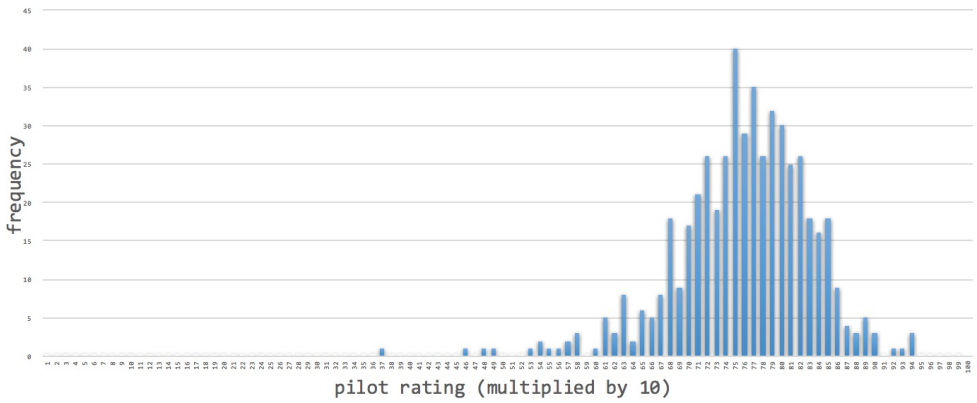


Figure 7.2: Distribution of Pilot Ratings

- ***RB_31*** ranges from 7.4 to 7.8 and contains 156 pilots.
- ***RB_32*** ranges from 7.9 to 9.9 and contains 194 pilots.

An imbalance is observed amongst the three bins; that is the number of pilot scripts in RB_32 is nearly 1.25 times that of RB_31. Though the difference is relatively insignificant, the imbalance could lead to incorrectness of predictions of the to be built classifiers, since many data mining algorithms tend to ignore or misclassify minority samples (Longadge & Dongre, 2013).

An undersampling procedure was applied by removing pilots from RB_30 and RB_32. Afterwards, each of the three bins contained the same number, i.e. 156, of pilots. Eventually, in total **468** pilot scripts participated in the process of establishing predictive models.

Terms of keywords and activities

Terms of keywords and activities have been determined in early works (§6.3 and §6.4). Some examples of keyword terms are *athletic, working mother, small owner* and *teammate* [5]. Some examples of activity terms are *be sorry, be very angry, see person*

[5]Selected from keyword terms of *Outnumbered*.

and *wear sexy dress* [6].

Four language models have been prepared for generating distributed representations (§5.2). Two of them, ***model_wiki_w2v*** and ***model_wiki_d2v***, are pre-trained by other research work and were fetched from the internet. The other two, ***model_subt_w2v*** and ***model_subt_d2v***, were trained in the present work, by exploiting the English part of the *OpenSubtitle2016* corpus, which contains circa 1.8 billion words.

For the collection of keyword terms of a particular pilot script, two distributed representations were generated: one using the two ***wiki*** models (the ***kw-wiki-rep***), the other using the two ***subt*** models (the ***kw-subt-rep***).

The steps to generate the ***kw-wiki-rep*** are described below:

- **Step I**: for a specific keyword term, compute its representation by averaging the *word2vec* representations of all of its constituent words, using the ***model_wiki_w2v*** language model.
- **Step II**: average the representations of all keyword terms computed in Step I; denote this average as $w2v\_rep$.
- **Step III**: for a specific keyword term, if it comprises more than one word, then get it *doc2vec* representation by using the ***model_wiki_d2v*** language model.
- **Step IV**: average the representations of keyword terms computed in Step III; denote it as $d2v\_rep$.
- **Step V**: average $w2v\_rep$ and $d2v\_rep$, then the ***kw-wiki-rep*** is generated.

Similarly, the ***kw-subt-rep*** is generated, with using ***model_subt_w2v*** instead of *model_wiki_w2v* and using ***model_subt_d2v*** instead of *model_wiki_d2v*.

[6] Selected from actitivy terms of *Outnumbered*

Correspondingly, for the collection of activity terms of a particular pilot script, two distributed representations are generated: the ***ac-wiki-rep*** and the ***ac-subt-rep***.

For keywords, by assembling all the ***kw-wiki-rep*** and ***kw-subt-rep*** of all the pilot scripts, and subsequently performing the *min-max-scale* (introduced in §5.3) procedure on each dimension, data matrices for these two feature sets were generated:

- ***FS_dr_kw_wiki*** (contains 300 dimensions [7])
- ***FS_dr_kw_subt*** (contains 500 dimensions)

For activities, by assembling all the ***ac-wiki-rep*** and ***ac-subt-rep*** of all the pilot scripts, and subsequently performing the *min-max-scale* procedure on each dimension, data matrices for these two feature sets were generated:

- ***FS_dr_ac_wiki*** (contains 300 dimensions)
- ***FS_dr_ac_subt*** (contains 500 dimensions)

Matrices by the bag-of-words model

Matrices for the four feature sets: ***FS_venues***, ***FS_ne_person***, ***FS_ne_location***, and ***FS_ne_organisation***, were constructed with the *bag-of-words* model. Widths of these matrices were more than thirteen thousand, eight thousand, three thousand, and three thousand, respectively.

A procedure of *dimension reduction* was applied by selecting K-best columns from each matrix by measuring the value of *mutual information* between a column and the target feature, i.e. the rating. After the procedure, only 500 dimensions remained for each of the four data matrices.

The feature set ***FS_genre*** was originally low dimensional, thus the dimension reduction action was not applied to it.

[7] See §5.2.

Normalisation

All the prepared data matrices were normalised.

The data matrices containing distributed representations of the pilot scripts were normalised via the *min-max-scale* approach (§5.2). These include data for the feature sets of ***FS_dr_kw_wiki***, ***FS_dr_kw_subt***, ***FS_dr_ac_wiki***, ***FS_dr_ac_subt***, as well as ***FS_dr_wiki_w2v***, ***FS_dr_wiki_d2v***, ***FS_dr_subt_w2v***, ***FS_dr_subt_d2v***.

The data matrices containing TFIDF representations of the pilot scripts were normalised via the $l^2 - norm$ approach (§6.1). These include data for the feature sets of ***FS_venues***, ***FS_ne_person***, ***FS_ne_location***, ***FS_ne_organisation***.

The data matrices for ***FS_stat*** and ***FS_cnet*** were normalised via the *min-max-scale* approach.

Datasets

As the last step of the preparation of datasets for training predictive models for pilot rating, some of the data matrices were concatenated.

- The data matrices for feature sets of ***FS_stat***, ***FS_cnet***, ***FS_venues***, ***FS_ne_person***, ***FS_ne_location***, ***FS_ne_organisation***, and ***FS_genre***, were concatenated. The resulting dataset was denoted as **DS_basic**.
- The data matrics for ***FS_dr_kw_wiki*** and ***FS_dr_ac_wiki*** were concatenated, and the resulting dataset was denoted as **DS_kw_ac_wiki**.
- The data matrics for ***FS_dr_kw_subt*** and ***FS_dr_ac_subt*** were concatenated, and the resulting dataset was denoted as **DS_kw_ac_subt**.
- The four data matrices that contain distributed representations of the pilot scripts are named as below:

- **DS_script_wiki_w2v** for ***FS_dr_wiki_w2v***
- **DS_script_wiki_d2v** for ***FS_dr_wiki_d2v***
- **DS_script_subt_w2v** for ***FS_dr_subt_w2v***
- **DS_script_subt_d2v** for ***FS_dr_subt_d2v***

Table 7.2: Final Datasets

DS_all_wiki_w2v	=	**DS_basic**	+ **DS_kw_ac_wiki**	+ **DS_script_wiki_w2v**
DS_all_wiki_d2v	=	**DS_basic**	+ **DS_kw_ac_wiki**	+ **DS_script_wiki_d2v**
DS_all_subt_w2v	=	**DS_basic**	+ **DS_kw_ac_subt**	+ **DS_script_subt_w2v**
DS_all_subt_d2v	=	**DS_basic**	+ **DS_kw_ac_subt**	+ **DS_script_subt_d2v**

Subsequently, four final datasets were generated by concatenating these intermediate datasets, as showed in Table 7.2. These four will be exploited to train the predictive models.

Moreover, a dataset named **DS_script_bow** was prepared which contains TFIDF values extracted from the unigram bag-of-words model of the pilot scripts. The measure employed for dimension reduction is the *mutual information*, and 3000 features are kept. Classifiers that trained with DS_script_bow will be used as baseline classifiers.

7.2.2 Classifiers

The *scikit-learn*[8] (Pedregosa et al., 2011; Buitinck et al., 2013) library implements various classifier algorithms.

The five datasets introduced in the previous section, namely **DS_all_wiki_w2v**, **DS_all_wiki_d2v**, **DS_all_subt_w2v**, **DS_all_subt_d2v**, and **DS_script_bow**, are fitted into several of the classifiers in *scikit-learn*, with different parameter options.

[8]http://scikit-learn.org/

Random

A Random Classifier predicts class labels randomly, and the input dataset is completely ignored. It is suitable to be used as a simple baseline.

The scikit-learn class `DummyClassifier` could be used as a random classifier.

Naive Bayes

Given a predictive feature vector $(x_1, x_2, ..., x_n)$ and denoting the target class as y, then based on the Bayes theorem:

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)} \tag{7.2}$$

and the assumption that predictive features are independent:

$$P(x_i \mid y, x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = P(x_i \mid y) \tag{7.3}$$

the *Naive Bayes* algorithm considers that:

$$P(y \mid x_i, ..., x_n) = \frac{P(y) \cdot \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, ..., x_n)} \tag{7.4}$$

Therefore, the predicted class y needs to satisfy:

$$\underset{y}{\operatorname{argmax}}\, P(y) \cdot \prod_{i=1}^{n} P(x_i \mid y) \tag{7.5}$$

The scikit-learn class `MultinominalNB` implements the Naive Bayes algorithm and is suitable for text classification tasks [9].

[9] http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

KNN

The *KNN* stands for *K-Nearest Neighbours*. The most popular class label of the `K` number of nearest neighbours, measured by the Euclidean distance, in the multi-dimensional space defined by the feature vector, is selected as the prediction for a to be classified entry.

The option of the value `K` is limited to the range $[2, 5]$. Thus based on a certain dataset, four KNN classifier instances were trained and tested.

The KNN implementation in scikit-learn is the `KNeighborsClassifiers`.

Nearest Centroid

A centroid for each class is firstly calculated. Then the label of the class whose centroid is the nearest to a test entry is assigned to it.

In scikit-learn, the implementation is `NearestCentroid`.

Random Forest

By adopting the random forest approach, a number of decision trees are trained by exploiting several subsamples of the training dataset, and these trees then vote to decide a class label for a to be classified entry.

In the present work, a subsample was generated by randomly selecting entries *with replacement* from the training dataset. This means that an entry could be repeatedly selected and thus could appear multiple times in the generated subsample.

Comparing with stand along decision trees, the random forest improves the predictive accuracy and controls the issue of overfitting (Breiman, 2001).

In scikit-learn, the implementation is `RandomForestClassifier`.

Linear Models

Several scikit-learn classifiers based on linear models are adopted.

In scikit-learn, the class `SGDClassifier` implements the procedure of using the *Stochastic Gradient Descent (SGD)* approach to learn linear classifiers. The SGD is easy to implement and is an efficient method (Zhang, 2004), even on large scale dataset (Bottou, 2010).

The class `Perceptron` is another simple linear algorithm for large scale learning. It is a basic neural network architecture, and could be considered as a modified version of SGD (Aggarwal, 2015).

The class `LinearSVC` is an "implementation of *Support Vector* Classification for the case of a linear kernel". [10]

MLP

The multilayer perceptron is a kind of feedforward neural network. It has at least one hidden layer besides the input layer and output layer. It is capable of learning non-linear models. (Aggarwal, 2015)

The class `MLPClassifier` implements the MLP algorithm, and uses *backpropagation*[11] to train.

Several groups of the parameters were exploited for the MLP model training in the present work (Appendix B).

7.2.3 Results

The accuracies of classifiers were measured by the F-score, which is the weighted average of the F-score of each class. The three-fold cross validation was employed to

[10] http://scikit-learn.org/stable/modules/svm.html#svm-classification

[11] Briefly introduced by Sathyanarayana (2014).

assess the classifiers.

In each of the five datasets, entries are averagely distributed into three classes. Therefore, the random classifier will always provide an accuracy of approximate $1/3$.

The best results of all the trained classifier instances on all five datasets are presented in Table 7.3.

It could be observed that, the highest accuracy of all the classifiers is only 0.4850, which is unexpectedly poor.

The MLP classifier outperformed all others on all of the five datasets. The `SGDClassifier` and the `Perceptron` performed worse even than the naive random classifier in most of the circumstances. The Naive Bayes approach also underperformed the random classifier on the baseline dataset.

Five classifiers out of the eight (exclude random classifier) reached their best score when fitted with the dataset **DS_all_subt_d2v**. Two of the other three gave the highest accuracies on the dataset **DS_all_wiki_d2v**. Different from all others, the KNN classifier, performed best on the baseline dataset, which distinctively does not include a sector of distributed representations or any particularly extracted feature.

The tables from 7.4 to 7.8 show more details of the best classifier instances on the five datasets separately.

Table 7.3: Resuls of Classifiers on Five Datasets

algorithm	**baseline**	**wiki-w2v**	**wiki-d2v**	**subt-w2v**	**subt-d2v**
Random	.3333	.3333	.3333	.3333	.3333
Naive Bayes	.2774	.3890	.3839	.3970	.4189 ⋆
Nearest Centroid	.3684	.4043	.4087	.4094	.4154 ⋆
KNN	.4052 ⋆	.3792	.3501	.3715	.3724
Perceptron	.1771	.3116	.3837 ⋆	.3391	.2684
Random Forest	.4186	.4188	.3849	.4045	.4407 ⋆
SGD	.2488	.2722	.3065 ⋆	.2507	.2227
Linear SVC	.3589	.3576	.4036	.4311	.4314 ⋆
MLP	**.4636**	**.4615**	**.4786**	**.4722**	**.4850** ⋆

*Table 7.4: Models atop **DS_script_bow***

algorithm	**accuracy**	**parameters**
Naive Bayes	.2774	
Nearest Centroid	.3684	
KNN	.4052	n_neighbors: 5
Perceptron	.1771	alpha: 0.0001
Random Forest	.4186	criterion: entropy
SGD	.2488	alpha: 0.001
Linear SVC	.3589	tol: 0.0001
MLP	**.4636**	hidden-layers: (15,) max-iter: 100 alpha: 0.1

*Table 7.5: Models atop **DS_all_wiki_w2v***

algorithm	**accuracy**	**parameters**
Naive Bayes	.3890	
Nearest Centroid	.4043	
KNN	.3792	n_neighbors: 3
Perceptron	.3116	alpha: 0.0001
Random Forest	.4188	criterion: entropy
SGD	.2722	alpha: 0.001
Linear SVC	.3576	tol: 1e-05
MLP	**.4615**	hidden-layers: (15, 10, 10) max-iter: 200 alpha: 8

*Table 7.6: Models atop **DS_all_wiki_d2v***

algorithm	**accuracy**	**parameters**
Naive Bayes	.3839	
Nearest Centroid	.4087	
KNN	.3501	n_neighbors: 5
Perceptron	.3837	alpha: 0.0001
Random Forest	.3849	criterion: entropy
SGD	.3065	alpha: 0.0001
Linear SVC	.4036	tol: 0.0001
MLP	**.4786**	hidden-layers: (20,) solver: sgd learning-rate: adaptive max-iter: 500 momentum: 0.9

*Table 7.7: Models atop **DS_all_subt_w2v***

algorithm	**accuracy**	**parameters**
Naive Bayes	.3970	
Nearest Centroid	.4094	
KNN	.3715	n_neighbors: 5
Perceptron	.3391	alpha: 0.0001
Random Forest	.4045	criterion: entropy
SGD	.2507	alpha: 0.0001
Linear SVC	.4311	tol: 0.0001
MLP	**.4722**	hidden-layers: (15, 10, 10) max-iter: 200 alpha: 3

*Table 7.8: Models atop **DS_all_subt_d2v***

algorithm	**accuracy**	**parameters**
Naive Bayes	.4189	
Nearest Centroid	.4154	
KNN	.3724	n_neighbors: 3
Perceptron	.2684	alpha: 0.0001
Random Forest	.4407	criterion: entropy
SGD	.2227	alpha: 0.0001
Linear SVC	.4314	tol: 0.0001
MLP	**.4850**	hidden-layers: (50,) max-iter: 100 alpha: 1

Chapter 8

Discusson

8.1 Predictive Models

The present study prepared five datasets. These include the **DS_script_bow**, which comprises TFIDF values of terms calculated via the unigram bag-of-words language model and is used for learning baseline classifiers. The other four datasets, namely **DS_all_wiki_w2v**, **DS_all_wiki_d2v**, **DS_all_subt_w2v**, and **DS_all_subt_d2v**, were constructed with data corresponding to the extracted features (§7.2.1).

The IMDB ratings of the pilot episodes, which range from 0 to 10 rounded to one decimal place, were used as the target feature after being sorted into three bins. In addition, some entries of the datasets were discarded to achieve a balance amongst the three classes.

Several classifier implementations from the scikit-learn library were employed. These include the Naive Bayes classifier `MultinominalNB`, the `KNeighborsClassifiers`, the nearest centroid classifier `NearestCentroid`, the `RandomForestClassifier`, three linear models (`SGDClassifier`, `Perceptron`, `LinearSVC`), and the multilayer perceptron `MLPClassifier`. In addition, the `DummyClassifier`, which always ignores input data and gives random class labels as predictions, was also adopted as a baseline

classifier.

The datasets, together with the rating labels, were fitted into the classification algorithms, and consequently, classifier instances were created and tested. Multiple options were prepared for some parameters of the classification algorithms, especially the `MLPClassifier`, thus on each dataset, a specific classifier could be instantiated for numerous times. Among them, the champion instance was then selected for further comparison with (1) instances trained on the same dataset with different classification algorithms, or (2) instances trained on different datasets with the identical classification algorithm with probable different parameter settings. The accuracies of these champion classifier instances are presented in Table 7.3.

The highest accuracy, achieved by the MLPClassifier on the **DS_all_subt_d2v** dataset, is only 0.4850. This is a poor result that is even lower than 0.5. Compared with the accuracy $1/3$ given by the baseline dummy random classifier, the value 0.4850 shows an undeniable improvement, however, the advantage against the accuracy 0.4636 provided by the `MLPClassifier` on the baseline dataset, is not very compelling.

To the best of the knowledge of the present work, the only line of research on predicting the success of new television series based on pre-greenlighting information was conducted by Starling Hunter and his colleagues.

In the works of Hunter, Chinta et al. (2016); Hunter, Smith and Chinta (2016), three previously untested predictive factors were employed to forecast the audience sizes of television series.

The three factors are (1) originality of the concept of the television series, (2) the track record of success of the creators, and (3) the size of the conceptual network generated from the pilot script. All three can be gathered before the decision on whether to launch the new TV series or not. However, collection of the first two predictive factors requires extra resources other than scripts, and demands heavy manual work. The

approach to constructing the conceptual network, which is also named as text network, was introduced by Hunter (2014); Hunter, Chinta et al. (2016). The third factor is also referred to as the *cognitive complexity* by Hunter and Breen (2017b, 2017a).

These two works adopted the *audience size*s, measured in millions, as the target feature. The data was collected mainly from the "Episodes" subsection of the Wikipedia[1] pages concerning the television series, and also from *TV Series Finale*[2] and TV.com[3].

Hunter, Chinta et al. (2016) concluded that "the size of the largest component of the text network was a positive and significant predictor of the total audience". The statement is supported by the regression models constructed in the works, including GLS (generalized least squares) (Hunter, Chinta et al., 2016) and OLS (ordinary least squares) (Hunter, Chinta et al., 2016; Hunter, Smith & Chinta, 2016). Conversely, though the features extracted in the current work are relatively diverse, their significance has not been sufficiently proved.

The major differences between the present work and the two studies discussed above (Hunter, Chinta et al., 2016; Hunter, Smith & Chinta, 2016) include:

- The two studies only selected TV series which were cast by one of the four major US television networks, namely ABC, NBC, CBS and Fox, while the present research collected all accessible English pilot scripts.
- The two studies exploited the audience sizes as the target feature, while the present work exploited IMDB ratings.
- The two studies adopted regression algorithms, while the present work adopted classification algorithms.

[1] https://en.wikipedia.org
[2] http://tvseriesfinale.com
[3] http://www.tv.com/

- The demands for the manual work between the two previous studies (Hunter, Chinta et al., 2016; Hunter, Smith & Chinta, 2016) and the current research for feature extraction are different.

  The *cognitive complexity* was generated semi-automatically. The collection of the other two factors (i.e. the originality of the concept and the track record of the creators) also requires intensive manual works.

  By contrast, only a small amount of manual labour is required for feature extraction in the present work. The current study does not have sufficient human resources available for intensive handwork.

When the current study was about to end, another two works in Hunter's study line were declared. Hunter and Breen (2017b) adopted the three same predictive factors to forecast the numbers of episodes in the first season of new television series. Hunter and Breen (2017a) similarly exploited the three factors, together with some extras (including whether a series belongs to the crime genre, the calendar year when the series is debuted, etc.), to predict whether a television series would be cancelled in the first two seasons, or be renewed for more seasons. Moreover, the second study (Hunter & Breen, 2017a) focuses on 30 cable networks and streaming services, instead of the four major networks.

The two works report a positive and significant impact of the cognitive complexity on the performance of series.

According to Hunter and Breen (2017a), Bielby and Bielby (1994) have expressed the argument that "all hits are flukes" concerning prediction of success of television series, which means that "there are no formulas for success, no single factor or combination of factors that predict commercial success of new television series". Hunter

and Breen (2017a) recommended that their results, together with related prior studies, have challenged and overturned the argument.

However, the current study would argue differently. Though the studies conducted by Hunter and his colleagues are admirable, their results are not solid enough to overbalance the "all hits are flukes" viewpoint. The reasons for this argument are listed below.

- To survive two seasons is not a good sign of the success of a television series bacause people's opinions towards multiple-seasonal series vary over time.
- The fact that the number of episodes in the first season is large is not a good sign of success, as there are other factors that could be influential.
- The substantial relevance between cognitive complexity and audience size was reported only on television series selected from the four major network.

Nevertheless, for further work, it is worthy to adopt the cognitive complexity as a predictive feature to forecast the IMDB rating exploited in the current work.

8.2 Distributed Representation

Table 7.1 shows that, with the same Naive Bayes algorithm, nine genres amongst the eleven achieved higher accuracy on the data matrix of distributed representations compared with on the data matrix established based on the bag-of-words language model.

In addition, it could be observed from Table 7.3 that amongst the eight involved classification algorithms, only one, namely the KNN, achieved the best score on the bag-of-words based dataset, compared with others which mainly consist of distributed representations of texts.

These two facts suggest that the distributed representations, which are low-dimensional and of high-density, better characterise the texts compared with the bag-of-words models.

Five of the eight classifiers performed better on the dataset **DS_all_wiki_d2v** than on the dataset **DS_all_wiki_w2v**. As shown in Table 7.2, the difference between the two datasets is their third constituents, which are respectively the **DS_scirpt_wiki_d2v** and the **DS_script_wiki_w2v**. The dataset **DS_all_wiki_d2v** was generated with the model *model_wiki_d2v*; meanwhile, the dataset **DS_all_wiki_w2v** was generated with the model *model_wiki_w2v*. These two models were similarly trained with the English Wikipedia corpus, but by different algorithms (schemes), which were *doc2vec* and *word2vec* respectively.

Similarly, six of the eight classifiers performed better on the dataset **DS_all_subt_d2v** than on the dataset **DS_all_subt_w2v**. The difference between these two datasets is that their third constituent datasets, which are the **DS_script_subt_d2v** and the **DS_script_subt_w2v** respectively, were generated with the models *model_subt_d2v* and *model_subt_w2v* respectively. These two models were similarly trained with the English part of the OpenSubtitle2016 corpus, but by different schemes, which were *doc2vec* and *word2vec* respectively.

These two facts suggest that the *doc2vec* is a better approach than *word2vec* to learning distributed vector representations of texts. This confirms the results reported by Lau and Baldwin (2016).

It also could be observed from Table 7.3 that among the eight classifiers, five performed better on **DS_all_subt_w2v** than on **DS_all_wiki_w2v**, and six performed better on **DS_all_subt_d2v** than on **DS_all_wiki_d2v**. These four datasets corresponded to the four models, i.e. *model_subt_w2v*, *model_wiki_w2v*, *model_subt_d2v*, and

model_wiki_d2v, respectively.

A possible cause of this observation is the higher dimension, i.e. 500, of the generated vectors by model_subt_*, while the number of dimension of the generated vectors by model_wiki_* is 300. Another possible cause is that model_subt_* were trained with domain corpus, considering that the corpus OpenSubtitles2016 is constituted of a huge number of subtitles of television series and movies.

For future work, it could be helpful to train distributed language models with a corpus that combines the English Wikipedia, the OpenSubtitle2016, and scripts of television series and movies. It also could be beneficial for the trained model to be capable of generating relatively higher dimensional (e.g. 800) vector representations.

A number of other schemes to train distributed language models have been proposed. These include the GloVe (Pennington, Socher & Manning, 2014), the FastText (Joulin, Grave, Bojanowski & Mikolov, 2016), and the LexVec (Salle, Idiart & Villavicencio, 2016a, 2016b). It might be helpful to employ these schemes to learn distributed models for further research.

8.3 Character Network

The current study adopted the characters' co-participating in dialogues to construct character networks. Furthermore, values of several metrics of the networks were computed and later used as predictive features to build classifiers. However, these values were combined with many others to perform as data matrices, hence have not been evaluated separately.

The innovation here is that the current study not only constructs character networks based on the information of full pilot scripts but also constructs networks based on the

information extracted from the first halves of the scripts. Accordingly, the evolutions of the networks, together with their final state, are expressed to some extent.

8.4 Emotions

The words which express emotions could be useful for characterising the pilot scripts.

A naive method to detect emotional words is to check whether the lexical name, provided by WordNet (Miller, 1995; Fellbaum, 2010), of a word, equals to *noun.feeling* or not. However, this method might select irrelevant words, such as *thing, distance*, or *weight*.

A better way is to adopt a prepared list of emotional words. A list proposed by Elliott (1992) contains 198 direct emotional words, while another list, collected by Salway and Graham (2003), contains 627 direct emotional words. The disadvantage of this method is that words containing hidden emotions would be omitted, e.g. the word *nightmare*.

A more advanced method is to adopt the technique of *Lexical Affinity*, which could associate a word with a probabilistic affinity for a particular emotion. According to Jung et al. (2013), the lexical affinity value denoting a word to an emotion can be calculated by *conceptual distance*, which can be measured by the number of nodes in a path connected by a hypernym or synonym (Richardson, Smeaton & Murphy, 1994; Park, Yoo, Kim & Jo, 2011).

By leveraging the emotion senses, together with the character network structure, Jung et al. (2013) divided characters into three categories: *protagonist* (main characters), *tritagonist* (supporters of main characters), and *antagonist* (opponent of main characters). The tritagonist and antagonist are both minor characters. If a minor character has a similar emotion vector to the main character, this minor is categorised as tritagonist.

Otherwise, the minor is an antagonist.

Makris and Vikatos (2016) has successfully further detected communities within the character networks of the movie scripts, by exploiting the linguistic and emotional features extracted with the LIWC (Pennebaker, Francis & Booth, 2001) tool. The community detecting algorithm employed is described in Blondel, Guillaume, Lambiotte and Lefebvre (2008); Fortunato (2010).

Though having been considered, nevertheless due to time limitation, the factor of emotion has not been engaged in the present work. For future work, it is recommended to adopt the emotion factor to build vector representations and analyse character networks.

8.5 Others

The *keywords* of the main characters are extracted by understanding the dependency parsing tree and recognising special patterns or dependency structures. However, there could be some patterns or dependency structures that contain useful information but have not been recognised. Thus for future work, a suggestion is to detect more dependency structures and patterns that support the extraction of keywords.

In the present work, some character-based information has been collected, including statistic data, keyword terms, activity terms, etc. Moreover, character-based data concerning emotion could also be gathered. Therefore, character-based analysis of the pilot scripts could be a viable research direction.

The movie scripts have a similar structure to television series scripts. In addition, a movie script commonly describes a complete story and its scenario is normally richer than a pilot script.

Therefore, the methods adopted and proposed in the present research could easily be ported to movie scripts, and the outputs might be more satisfactory.

Chapter 9

Conclusion and Future Work

The aim of the present work was to extract features from the scripts of television series, and subsequently construct predictive models to forecast the performance of the series.

Raw materials were collected from three data sources, i.e. the IMDB database, the OpentSubtitles2016 corpus, and a collection of television series scripts fetched from multiple websites. The genre and rating information were extracted from the IMDB database. The scripts of pilots were selected and parsed, then structures of these pilot scripts were obtained, including scenes, blocks of descriptions, and blocks of spoken content.

Several feature sets were determined and data matrices corresponding to these feature sets were created. These include:

- Features were firstly extracted from the basic statistics of the constituents of the pilot scripts.
- Based on the characters' co-participating in dialogues, a character network for each pilot script was established. Several metrics of the network were then

adopted as features.

- Two approaches to generating distributed representations of texts were adopted, which are *word2vec* and *doc2vec*. Two models, pre-trained with the English Wikipedia corpus, were downloaded. Two additional models were trained using the English part of the OpenSubtitles2016 corpus. By adopting these four models, distributed representations of the pilot scripts were generated.
- The TFIDF matrix of the venues of scenes was generated.
- Three types of named entities were recognised. Then TFIDF matrices of these named entities were created.
- By understanding the parsed dependencies between the words of sentences, terms of keywords for characters were determined. Subsequently, distributed representations of these keyword terms were created.
- Terms of activities were determined from the triplets, which represent relations between characters and other entities. Then similarly distributed representations of these activity terms were created.

By concatenating the created data matrices, four datasets were generated, which corresponded to the four distributed representation models. Each entry in these datasets corresponds to a pilot script.

The rating information extracted from the IMDB database was adopted as the target feature. The value of the IMDB rating ranges from 0 to 10, rounded to one decimal place. The values were binned into three classes. Furthermore, for keeping balance among the three classes, some entries in the datasets were discarded.

Several classification algorithms implemented in the scikit-learn library were employed to build the models. These algorithms include the Naive Bayes, the K-Nearest Neighbours, the Nearest Centroid, the Random Forest, three linear models, and the multilayer perceptron.

The four datasets, together with the rating class labels, were fitted into these classifier algorithms to create classifier instances. Multiple options for many parameters of these classification algorithms were prepared, especially for parameters of the multilayer perceptron. Therefore, a large number of classifier instances were built, tested, and compared.

However, having not lived up to the expectation, even the result delivered by the best classifier instance was poor.

Although the result is not as compelling as expected, the significance of the present research includes:

- Character networks were constructed by using both (1) all scenes of each script and (2) first half scenes of each script.
- The keyword terms of characters were determined by understanding the dependency parses of the sentences in the pilot scripts.
- The distributed representations of determined keyword terms, of determined activity terms, and of the entireties of pilot scripts, were exploited as predictive features.

The approach proposed in the present research to extract features from scripts of television series will be adopted by the company Parrot Analytics. The extracted

features, together with many other features which the company has already prepared, will be exploited to establish predictive models for forecasting performance of new television series.

Some suggestions for future work are:

- Try to employ cognitive complexity as a predictive factor.
- Try other metrics as the target feature.
- Develop criteria for further selection of television scripts.
- Try more options for the parameters of the classifier algorithms.
- Try other techniques that generate distributed representations. These include *GloVe, FastTest*, and *LexVec*.
- Develop more rules to detect keyword terms from parsed dependencies between words.
- Extract features associated with emotions.
- Try other sophisticated architectures to build the predictive models, for instance, deep neural networks.

References

Agarwal, A., Balasubramanian, S., Zheng, J. & Dash, S. (2014). Parsing screenplays for extracting social networks from movies. *EACL 2014*, 50–58.

Aggarwal, C. C. (2015). *Data mining: the textbook*. Springer.

Alberich, R., Miro-Julia, J. & Rosselló, F. (2002). Marvel universe looks almost like a real social network. *arXiv preprint cond-mat/0202174*.

Angeli, G., Premkumar, M. J. & Manning, C. D. (2015). Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd annual meeting of the association for computational linguistics (acl 2015)*.

Asahara, M. & Matsumoto, Y. (2003). Japanese named entity extraction with redundant morphological analysis. In *Proceedings of the 2003 conference of the north american chapter of the association for computational linguistics on human language technology - volume 1* (pp. 8–15). Stroudsburg, PA, USA: Association for Computational Linguistics. Retrieved from `http://dx.doi.org.ezproxy.aut.ac.nz/10.3115/1073445.1073447` doi: 10.3115/1073445.1073447

Asur, S. & Huberman, B. A. (2010). Predicting the future with social media. In *Web intelligence and intelligent agent technology (wi-iat), 2010 ieee/wic/acm international conference on* (Vol. 1, pp. 492–499).

Bae, S. M., Lee, S. C. & Park, J. H. (2014). Utilization of demographic analysis with imdb user ratings on the recommendation of movies. *Journal of Society for e-Business Studies*, *19*(3).

Bengio, Y., Ducharme, R., Vincent, P. & Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, *3*(Feb), 1137–1155.

Bengio, Y., LeCun, Y. et al. (2007). Scaling learning algorithms towards ai. *Large-scale kernel machines*, *34*(5), 1–41.

Bielby, W. T. & Bielby, D. D. (1994). " all hits are flukes": Institutionalized decision making and the rhetoric of network prime-time program development. *American Journal of Sociology*, *99*(5), 1287–1313.

Blondel, V., Guillaume, J., Lambiotte, R. & Lefebvre, E. (2008). Fast unfolding of community hierarchies in large network, 2008. *J. Stat. Mech. P*, *1008*.

Borthwick, A. (1999). *A maximum entropy approach to named entity recognition* (Unpublished doctoral dissertation). Citeseer.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier & G. Saporta (Eds.), *Proceedings of compstat'2010: 19th international conference on computational statisticsparis france, august 22-27, 2010 keynote, invited and contributed papers* (pp. 177–186). Heidelberg: Physica-Verlag HD. Retrieved from `http://dx.doi.org/10.1007/978-3-7908-2604-3_16` doi: 10.1007/978-3-7908-2604-3_16

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Bresnan, J., Asudeh, A., Toivonen, I. & Wechsler, S. (2015). *Lexical-functional syntax* (Vol. 16). John Wiley & Sons.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., ... others (2013). Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*.

Bullinaria, J. A. & Levy, J. P. (2007). Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior research methods*, *39*(3), 510–526.

Canet, F., Valero, M. Á. & Codina, L. (2016). Quantitative approaches for evaluating the influence of films using the imdb database. *Comunicación y Sociedad*, *29*(2), 151.

Carroll, J., Minnen, G. & Briscoe, T. (1999). Corpus annotation for parser evaluation. *arXiv preprint cs/9907013*.

Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16*, 321–357.

Chen, D. & Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. In *Emnlp* (pp. 740–750).

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, *12*(Aug), 2493–2537.

Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings of the 39th annual acm southeast conference* (pp. 95–102).

De Marneffe, M.-C., Connor, M., Silveira, N., Bowman, S. R., Dozat, T. & Manning, C. D. (2013). More constructions, more genres: Extending stanford dependencies. *Proceedings of DepLing*.

De Marneffe, M.-C., Dozat, T., Silveira, N., Haverinen, K., Ginter, F., Nivre, J. & Manning, C. D. (2014). Universal stanford dependencies: A cross-linguistic typology. In *Lrec* (Vol. 14, pp. 4585–92).

De Marneffe, M.-C., MacCartney, B., Manning, C. D. et al. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of lrec* (Vol. 6, pp. 449–454).

De Marneffe, M.-C. & Manning, C. D. (2008a). *Stanford typed dependencies manual* (Tech. Rep.). Technical report, Stanford University.

De Marneffe, M.-C. & Manning, C. D. (2008b). The stanford typed dependencies representation. In *Coling 2008: proceedings of the workshop on cross-framework and cross-domain parser evaluation* (pp. 1–8).

Dose, S. (2013). Flipping the script: A corpus of american television series (cats) for corpus-based language learning and teaching. *Corpus Linguistics and Variation in English: Focus on Non-native Englishes*.

Eisner, J. (1997). An empirical comparison of probability models for dependency grammar. *arXiv preprint cmp-lg/9706004*.

Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In *Advances in probabilistic and other parsing technologies* (pp. 29–61). Springer.

Eisner, J. M. (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on computational linguistics-volume 1* (pp. 340–345).

Eliashberg, J., Hui, S. K. & Zhang, Z. J. (2007). From story line to box office: A new approach for green-lighting movie scripts. *Management Science*, *53*(6), 881–893.

Eliashberg, J., Hui, S. K. & Zhang, Z. J. (2014). Assessing box office performance

using movie scripts: A kernel-based approach. *IEEE Transactions on Knowledge and Data Engineering*, *26*(11), 2639–2648.

Elliott, C. D. (1992). *The affective reasoner: A process model of emotions in a multi-agent system* (Doctoral dissertation, Evanston, IL, USA). Retrieved from https://condor.depaul.edu/elliott/ar/ftp/elliott-thesis.pdf (UMI Order No. GAX92-29901)

Elson, D. K., Dames, N. & McKeown, K. R. (2010). Extracting social networks from literary fiction. In *Proceedings of the 48th annual meeting of the association for computational linguistics* (pp. 138–147).

Fellbaum, C. (2010). Wordnet. In R. Poli, M. Healy & A. Kameas (Eds.), *Theory and applications of ontology: Computer applications* (pp. 231–243). Dordrecht: Springer Netherlands. Retrieved from https://doi.org/10.1007/978-90-481-8847-5_10 doi: 10.1007/978-90-481-8847-5_10

Finkel, J. R., Grenager, T. & Manning, C. (2005). Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics* (pp. 363–370). Stroudsburg, PA, USA: Association for Computational Linguistics. Retrieved from https://doi-org.ezproxy.aut.ac.nz/10.3115/1219840.1219885 doi: 10.3115/1219840.1219885

Fortunato, S. (2010). Community detection in graphs. *Physics reports*, *486*(3), 75–174.

Fraile, F. & Guerri, J. C. (2014). Simple models of the content duration and the popularity of television content. *Journal of Network and Computer Applications*, *40*, 12 - 20. Retrieved from http://www.sciencedirect.com/science/article/pii/S1084804513001859 doi: http://doi.org/10.1016/j.jnca.2013.08.010

Freitag, D. & McCallum, A. (1999). Information extraction with hmms and shrinkage. In *Proceedings of the aaai-99 workshop on machine learning for information extraction* (pp. 31–36).

Gil, S., Kuenzel, L. & Caroline, S. (2011). Extraction and analysis of character interaction networks from plays and movies. *Retrieved June*, *15*, 2016.

Goldberg, Y. & Levy, O. (2014). word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

Han, H., Wang, W.-Y. & Mao, B.-H. (2005). Borderline-smote: a new over-sampling method in imbalanced data sets learning. *Advances in intelligent computing*,

878–887.

Harris, Z. S. (1954). Distributional structure. *Word*, *10*(2-3), 146–162.

Hinton, G. E., Mcclelland, J. L. & Rumelhart, D. E. (1986). *Distributed representations, parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations.* MIT Press, Cambridge, MA.

Hsu, P. Y., Shen, Y. H. & Xie, X. A. (2014). Predicting movies user ratings with imdb attributes. In D. Miao, W. Pedrycz, D. ŚlE(C)zak, G. Peters, Q. Hu & R. Wang (Eds.), *Rough sets and knowledge technology: 9th international conference, rskt 2014, shanghai, china, october 24-26, 2014, proceedings* (pp. 444–453). Cham: Springer International Publishing. Retrieved from `http://dx.doi.org/10.1007/978-3-319-11740-9_41` doi: 10.1007/978-3-319-11740-9_41

Hudson, R. A. (1991). *English word grammar*. B. Blackwell.

Hunter, S. (2014). A novel method of network text analysis. *Open Journal of Modern Linguisitics*, *4*(2), 350.

Hunter, S. & Breen, Y. (2017a). Predicting the success of new cable series from their pilot episode scripts: An empirical approach. *Business and Management Studies*, *3*(3), 1–9.

Hunter, S. & Breen, Y. (2017b). W(h)ither the full season: An empirical model for predicting the duration of new television series? first season. *Advances in Journalism and Communication*, *5*(3), 38–97. doi: 10.4236/ajc.2017.52005

Hunter, S., Chinta, R., Smith, S., Shamim, A. & Bawazir, A. (2016). Moneyball for tv: A model for forecasting the audience of new dramatic television series. *Studies in Media and Communication*, *4*(2), 13–22.

Hunter, S., Smith, S. & Chinta, R. (2016). Predicting new tv series ratings from their pilot episode scripts. *International Journal of English Linguistics*, *6*(5), 1.

Hunter, S., Smith, S. & Singh, S. (2016). Predicting box office from the screenplay: A text analytical approach. *Journal of Screenwriting*, *7*(2), 135–154.

Joulin, A., Grave, E., Bojanowski, P. & Mikolov, T. (2016). Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Ju, Z., Wang, J. & Zhu, F. (2011, May). Named entity recognition from biomedical text using svm. In *2011 5th international conference on bioinformatics and biomedical engineering* (p. 1-4). doi: 10.1109/icbbe.2011.5779984

Jung, J. J., You, E. & Park, S.-B. (2013). Emotion-based character clustering for managing story-based contents: a cinemetric analysis. *Multimedia tools and applications*, *65*(1), 29–45.

Jurafsky, D. & Martin, J. H. (2014). *Speech and language processing* (Vol. 3). Pearson.

Kabinsingha, S., Chindasorn, S. & Chantrapornchai, C. (2012). Movie rating approach and application based on data mining. *International Journal of Engineering and Innovative Technology (IJEIT) Volume*, *2*.

King, T. H., Crouch, R., Riezler, S., Dalrymple, M. & Kaplan, R. M. (2003). The parc 700 dependency bank. In *Proceedings of the eacl03: 4th international workshop on linguistically interpreted corpora (linc-03)* (pp. 1–8).

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A. & Fidler, S. (2015). Skip-thought vectors. In *Advances in neural information processing systems* (pp. 3294–3302).

Koschützki, D., Lehmann, K. A., Peeters, L., Richter, S., Tenfelde-Podehl, D. & Zlotowski, O. (2005). Centrality indices. In U. Brandes & T. Erlebach (Eds.), *Network analysis: Methodological foundations* (pp. 16–61). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-540-31955-9_3 doi: 10.1007/978-3-540-31955-9_3

Koschützki, D., Lehmann, K. A., Tenfelde-Podehl, D. & Zlotowski, O. (2005). Advanced centrality concepts. In U. Brandes & T. Erlebach (Eds.), *Network analysis: Methodological foundations* (pp. 83–111). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-540-31955-9_5 doi: 10.1007/978-3-540-31955-9_5

Kudoh, T. & Matsumoto, Y. (2000). Use of support vector learning for chunk identification. In *Proceedings of the 2nd workshop on learning language in logic and the 4th conference on computational natural language learning-volume 7* (pp. 142–144).

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K. & Dyer, C. (2016). Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Lau, J. H. & Baldwin, T. (2016). An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368*.

Le, Q. & Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proceedings of the 31st international conference on machine learning (icml-14)* (pp. 1188–1196).

Levy, O., Goldberg, Y. & Ramat-Gan, I. (2014). Linguistic regularities in sparse and explicit word representations. In *Conll* (pp. 171–180).

Li, C. H. & Park, S. C. (2006). Text categorization based on artificial neural networks. In *International conference on neural information processing* (pp. 302–311).

Li, Y., Xu, L., Tian, F., Jiang, L., Zhong, X. & Chen, E. (2015). Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In *Ijcai* (pp. 3650–3656).

Ling, W., Luís, T., Marujo, L., Astudillo, R. F., Amir, S., Dyer, C., ... Trancoso, I. (2015). Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*.

Lison, P. & Meena, R. (2016). Automatic turn segmentation for movie & tv subtitles. In *2016 ieee workshop on spoken language technology :.* (QC 20161014)

Lison, P. & Tiedemann, J. (2016). Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles. In *Proceedings of the 10th international conference on language resources and evaluation.*

Longadge, R. & Dongre, S. (2013). Class imbalance problem in data mining review. *CoRR*, *abs/1305.1707*. Retrieved from `http://arxiv.org/abs/1305.1707`

Makris, C. & Vikatos, P. (2016). Community detection of screenplay characters. In *Ifip international conference on artificial intelligence applications and innovations* (pp. 463–470).

Martin, T., Zhang, X. & Newman, M. E. J. (2014, Nov). Localization and centrality in networks. *Phys. Rev. E*, *90*, 052808. Retrieved from `https://link.aps.org/doi/10.1103/PhysRevE.90.052808` doi: 10.1103/PhysRevE.90.052808

McDonald, R. T., Nivre, J., Quirmbach-Brundage, Y., Goldberg, Y., Das, D., Ganchev, K., ... others (2013). Universal dependency annotation for multilingual parsing. In *Acl (2)* (pp. 92–97).

Mel'čuk, I. A. (1988). *Dependency syntax: theory and practice*. SUNY press.

Mesnil, G., Mikolov, T., Ranzato, M. & Bengio, Y. (2014). Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews. *CoRR*, *abs/1412.5335*. Retrieved from `http://arxiv.org/abs/1412.5335`

Mestyán, M., Yasseri, T. & Kertész, J. (2013). Early prediction of movie box office success based on wikipedia activity big data. *PloS one*, *8*(8), e71226.

Mikheev, A., Moens, M. & Grover, C. (1999). Named entity recognition without gazetteers. In *Proceedings of the ninth conference on european chapter of the association for computational linguistics* (pp. 1–8).

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J. & Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech* (Vol. 2, p. 3).

Mikolov, T., Kopecky, J., Burget, L., Glembek, O. et al. (2009). Neural network based language models for highly inflective languages. In *Acoustics, speech and signal processing, 2009. icassp 2009. ieee international conference on* (pp. 4725–4728).

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 26* (pp. 3111–3119). Curran Associates, Inc. Retrieved from `http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf`

Miller, G. A. (1995, November). Wordnet: A lexical database for english. *Commun. ACM*, *38*(11), 39–41. Retrieved from `http://doi.acm.org.ezproxy.aut.ac.nz/10.1145/219717.219748` doi: 10.1145/219717.219748

Mohit, B. (2014). Named entity recognition. In *Natural language processing of semitic languages* (pp. 221–245). Springer.

Moretti, F. (2011). Network theory, plot analysis. *New Left Review*. Retrieved from `https://litlab.stanford.edu/LiteraryLabPamphlet2.pdf`

Nadeau, D. & Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, *30*(1), 3–26.

Nayak, N., Kowarsky, M., Angeli, G. & Manning, C. D. (2014). *A dictionary of nonsubsective adjectives* (Tech. Rep.). Technical Report CSTR 2014-04, Department of Computer Science, Stanford University, October.

Nguyen, H. M., Cooper, E. W. & Kamei, K. (2011). Borderline over-sampling for imbalanced data classification. *International Journal of Knowledge Engineering*

and Soft Data Paradigms, *3*(1), 4–21.

Nivre, J. (2006). Dependency parsing. In *Inductive dependency parsing* (pp. 45–86). Dordrecht: Springer Netherlands. Retrieved from `http://dx.doi.org/10.1007/1-4020-4889-0_3` doi: 10.1007/1-4020-4889-0_3

Nivre, J., De Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., ... others (2016). Universal dependencies v1: A multilingual treebank collection. In *Lrec*.

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., ... Marsi, E. (2007). Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, *13*(02), 95–135.

Park, S.-B., Oh, K.-J. & Jo, G.-S. (2012). Social network analysis in a movie using character-net. *Multimedia Tools and Applications*, *59*(2), 601–627.

Park, S.-B., Yoo, E., Kim, H. & Jo, G.-S. (2011). Automatic emotion annotation of movie dialogue using wordnet. In *Asian conference on intelligent information and database systems* (pp. 130–139).

Park, S.-B. & You, E. (2012). Story modeling for green light decision making. In *Proceedings of the international conference on it convergence and security 2011* (pp. 517–521).

Pavlick, E. & Callison-Burch, C. (2016). So-called non-subsective adjectives. *The SEM 2016 Organizing Committee.*, 114.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... others (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830.

Pennebaker, J. W., Francis, M. E. & Booth, R. J. (2001). Linguistic inquiry and word count: Liwc 2001. *Mahway: Lawrence Erlbaum Associates*, *71*(2001), 2001.

Pennington, J., Socher, R. & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Emnlp* (Vol. 14, pp. 1532–1543).

Rasooli, M. S. & Faili, H. (2012). Fast unsupervised dependency parsing with arc-standard transitions. In *Proceedings of the joint workshop on unsupervised and semi-supervised learning in nlp* (pp. 1–9).

Ratinov, L. & Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proceedings of the thirteenth conference on computational natural*

language learning (pp. 147–155).

Rennie, J. D., Shih, L., Teevan, J. & Karger, D. R. (2003). Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of the 20th international conference on machine learning (icml-03)* (pp. 616–623).

Richardson, R., Smeaton, A. F. & Murphy, J. (1994). Using wordnet as a knowledge base for measuring semantic similarity between words. In *Proceedings of aics conference* (pp. 1–15).

Riloff, E. & Phillips, W. (2004). *An introduction to the sundance and autoslog systems* (Tech. Rep.). Technical Report UUCS-04-015, School of Computing, University of Utah.

Ronfard, R. & Thuong, T. T. (2003). A framework for aligning and indexing movies with their script. In *Multimedia and expo, 2003. icme'03. proceedings. 2003 international conference on* (Vol. 1, pp. I–21).

Rong, X. (2014). word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Salle, A., Idiart, M. & Villavicencio, A. (2016a). Enhancing the lexvec distributed word representation model using positional contexts and external memory. *arXiv preprint arXiv:1606.01283*.

Salle, A., Idiart, M. & Villavicencio, A. (2016b). Matrix factorization using window sampling and negative sampling for improved word representations. *arXiv preprint arXiv:1606.00819*.

Salway, A. & Graham, M. (2003). Extracting information about emotions in films. In *Proceedings of the eleventh acm international conference on multimedia* (pp. 299–302).

Sathyanarayana, S. (2014). *A gentle introduction to backpropagation.* July.

Sawhney, M. S. & Eliashberg, J. (1996). A parsimonious model for forecasting gross box-office revenues of motion pictures. *Marketing Science*, *15*(2), 113–131.

Schuster, S. & Manning, C. D. (2016). Enhanced english universal dependencies: An improved representation for natural language understanding tasks. In *Proceedings of the tenth international conference on language resources and evaluation (lrec 2016).*

Sultan, M. A., Bethard, S. & Sumner, T. (2015). Dls @ cu: Sentence similarity from

word alignment and semantic vector composition. In *Semeval@ naacl-hlt* (pp. 148–153).

Tiedemann, J. (2016). Finding alternative translations in a large corpus of movie subtitles. In *Proceedings of the 10th international conference on language resources and evaluation (lrec-2016), portoroz, slovenia.*

Turetsky, R. & Dimitrova, N. (2004). Screenplay alignment for closed-system speaker identification and analysis of feature films. In *Multimedia and expo, 2004. icme'04. 2004 ieee international conference on* (Vol. 3, pp. 1659–1662).

Weng, C.-Y., Chu, W.-T. & Wu, J.-L. (2009). Rolenet: Movie analysis from the perspective of social networks. *IEEE Transactions on Multimedia*, *11*(2), 256–271.

Wieting, J., Bansal, M., Gimpel, K. & Livescu, K. (2015). Towards universal paraphrastic sentence embeddings. *arXiv preprint arXiv:1511.08198*.

Yamada, K. & Knight, K. (2001). A syntax-based statistical translation model. In *Proceedings of the 39th annual meeting on association for computational linguistics* (pp. 523–530).

Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on machine learning* (p. 116).

Zhou, H., Zhang, Y., Huang, S. & Chen, J. (2015). A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Acl (1)* (pp. 1213–1222).

Appendix A

List of 512 Series

100-questions	*10-things-i-hate-about-*	*11-22-63*	*12-monkeys*
1600-penn	*32-brinkburn-street*	*666-park-avenue*	*a-gifted-man*
a-to-z	*about-a-boy*	*absolutely-fabulous*	*accidentally-on-purpos-*
across-the-river-to-mo-	*agent-x*	*alice*	*alien-nation*
allegiance	*almost-human*	*alpha-house*	*alphas*
american-crime	*american-family*	*american-gothic*	*american-horror-story*
angel-from-hell	*angie-tribeca*	*animal-kingdom*	*apple-tree-yard*
aquarius	*archer*	*are-you-there-chelsea*	*arrow*
ashes-to-ashes	*astronaut-wives-club*	*awake*	*awkward*
back-in-the-game	*backstrom*	*bad-girls*	*bad-judge*
bad-teacher	*ballers*	*banshee*	*bates-motel*
battle-creek	*beast*	*beautiful-people*	*being-human*
being-mary-jane	*believe*	*benched*	*bent*
best-friends-forever	*betas*	*betrayal*	*better-with-you*
big-shots	*big-time-in-hollywood–*	*billions*	*bionic-woman*
black-ish	*black-mirror*	*black-sails*	*blindspot*
bloodline	*blue-bloods*	*blunt-talk*	*body-of-proof*
bonekickers	*bored-to-death*	*bosch*	*boss*
boy-meets-girl	*braindead*	*breaking-bad*	*breaking-in*
breakout-kings	*broad-city*	*brooklyn-nine-nine*	*bunheads*

californication
camp
cane
canterbury-s-law
carpoolers
casanova
cashmere-mafia
castle
casual
chaos
charlie-s-angels
chase
chicago-fire
chozen
code-black
colony
combat-hospital
common-law
community
complications
containment
cougar-town
crazy-ex-girlfriend
criminal-minds-suspect-
crisis
cristela
crowded
da-vinci-s-demons
dallas
danni-lowinski
dark-skies
dead-boss
dead-of-summer
death-valley
defying-gravity
detectorists
devious-maids
dexter
dickensian
difficult-people
dig
dirty-sexy-money
do-no-harm
dominion
dr-ken
dracula
drop-dead-diva
eden
elementary
eleventh-hour
empire
endgame
enlisted
episodes
extant
eye-candy
fargo
fear-the-walking-dead
finding-carter
flaked
flashpoint
flesh-and-bone
forever
frankenstein
frankie
franklin-bash
free-agents
friday-night-dinner
friends-with-better-li-
from-darkness
galavant
game-of-silence
gang-related
gcb
gilmore-girls
girlfriends-guide-to-d-
girls
glory-daze
go-on
golden-boy
gossip-girl
gotham
grace-and-frankie
graceland
gracepoint
grandfathered
gravity
grimm
ground-floor
growing-up-fisher
guilt
guys-with-kids
hallelujah
halt-catch-fire
hand-of-god
hannibal
happy-town
happy-valley
happyish
happyland
harry-s-law
hart-of-dixie
hawaii-five-0
helix
hell-on-wheels
hellcats
hidden
highston
hindsight
hit-the-floor
hostages
house-of-lies
house-of-saddam
how-i-met-your-dad
how-to-be-a-gentleman
how-to-get-away-with-m-
huge
human-target
humans
hustle
hysteria
i-just-want-my-pants-b-
i-live-with-models
identity
impastor
in-the-club
in-the-flesh
inside-no-9

intelligence *into-the-badlands* *ironside* *isabel*
izombie *jamaica-inn* *jane-by-design* *jane-the-virgin*
jennifer-falls *justified* *kevin-from-work* *killer-women*
killjoys *king-and-maxwell* *kingdom* *kirstie*
last-man-standing *last-resort* *last-tango-in-halifax* *legends*
life-in-pieces *life-on-mars* *lights-out* *line-of-duty*
london-spy *lone-star* *longmire* *lost*
love *love-bites* *low-winter-sun* *lucifer*
lucky-7 *luther* *mad-dogs* *mad-love*
mad-men *madam-secretary* *magic-city* *man-seeking-woman*
man-up *manhattan* *marco-polo* *married*
marry-me *masters-of-sex* *me-and-mrs-jones* *men-at-work*
mercy-street *merlin* *miami-medical* *mind-games*
minority-report *missing* *mistresses* *mixology*
mob-city *modern-family* *monday-mornings* *motive*
mozart-in-the-jungle *mr-robinson* *mr-robot* *mr-sunshine*
mystery-girls *narcos* *nashville* *necessary-roughness*
new-girl *nikita* *no-heroics* *no-ordinary-family*
noir *ny-lon* *nyc-22* *of-kings-and-prophets*
off-the-map *olympus* *once-upon-a-time* *one-big-happy*
one-mississippi *only-fools-and-horses* *ordinary-lies* *orphan-black*
our-house *outcast* *outlander* *outlaw*
outnumbered *outsiders* *outsourced* *pan-am*
parenthood *partners* *patito-feo* *peaky-blinders*
perception *perfect-couples* *person-of-interest* *playing-house*
political-animals *power* *preacher* *pretty-little-liars*
prime-suspect *proof* *public-morals* *pulling*
pushing-daisies *quantico* *queen-of-the-south* *raising-hope*
ray-donovan *reckless* *recovery-road* *rectify*
red-band-society *red-oaks* *red-widow* *reign*
remember-me *resident-advisors* *resurrection* *revolution*
rex-is-not-your-lawyer *ringer* *ripper-street* *rita*

rita-rocks
roadies
rob
rogue
roommates
rosewood
royal-pains
rubicon
running-wilde
rush
rush-hour
salem
samantha-who
satisfaction
scandal
scorpion
sean-saves-the-world
secrets-and-lies
selfie
sense8
sex-drugs-rock-roll
shadowhunters
signed-sealed-delivere-
silicon-valley
single-ladies
sirens
sit-down-shut-up
sneaky-pete
sons-of-tucson
southland
spy
stalker
star-crossed
state-of-affairs
state-of-georgia
stranger-things
suburgatory
suits
super-fun-night
superstore
surviving-jack
survivors
switched-at-birth
teen-wolf
telenovela
terra-nova
terriers
the-100
the-affair
the-americans
the-big-bang-theory
the-blacklist
the-bridge
the-cape
the-carrie-diaries
the-catch
the-chicago-code
the-client-list
the-comedians
the-crazy-ones
the-dead-zone
the-detour
the-divide
the-event
the-exes
the-expanse
the-fades
the-family
the-firm
the-flash
the-following
the-fosters
the-game
the-gates
the-girlfriend-experie-
the-glades
the-goldbergs
the-good-guys
the-good-wife
the-goodwin-games
the-grinder
the-honourable-woman
the-killing
the-knick
the-last-man-on-earth
the-librarians
the-line
the-lottery
the-magicians
the-mccarthys
the-messengers
the-michael-j-fox-show
the-middle
the-mindy-project
the-missing
the-mob-doctor
the-musketeers
the-mysteries-of-laura
the-night-manager
the-night-shift
the-odd-couple
the-passing-bells
the-philanthropist
the-playboy-club
the-politician-s-husba-
the-red-road
the-replacement
the-river
the-royals
the-secret-of-crickley-
the-slap
the-strain
the-strip
the-syndicate
the-thundermans
the-tomorrow-people
the-twilight-zone
the-unusuals
the-walshes
the-whispers
those-who-can-t
those-who-kill
tin-man
togetherness
touch
traffic-light
transparent
trauma

trophy-wife	*true-detective*	*turn*	*twin-peaks*
twisted	*tyrant*	*unbreakable-kimmy-schm-*	*uncle-buck*
undateable	*undeclared*	*under-the-dome*	*underemployed*
underground	*unforgettable*	*unreal*	*up-all-night*
utopia	*v*	*vegas*	*veronica-mars*
waterfront	*wayward-pines*	*we-are-men*	*wedding-band*
weird-loners	*welcome-to-the-family*	*westside*	*what-remains*
whitney	*wicked-city*	*wild-card*	*witches-of-east-end*
wizards-vs-aliens	*wonderland*	*work-it*	*workaholics*
working-class	*working-the-engels*	*worst-week*	*wrecked*
wynonna-earp	*you-are-the-worst*	*you-me-and-the-apocaly-*	*you-re-the-worst*
young-hungry	*your-family-or-mine*	*zero-hour*	*zoo*

Appendix B

Parameters for MLP

The parameters to train the **sklearn.neural_network.MLPClassifier** are presented by Table B.1.

0	*hidden-layers*	[(5,), (10,), (15,), (15, 10), (20,), (15, 15, 15)]
	max-iter	[200, 500]
	alpha	[0.1, 1, 5, 10, 15, 20]
0k	*hidden-layers*	[(5,), (10,), (15,), (15, 10), (20,), (30,), (100,)]
	max-iter	[100, 200, 300]
	alpha	[0.1, 1, 5, 10]
1	*hidden-layers*	[(5,), (10,), (15,), (20,)]
	max-iter	[100, 200, 300, 500, 800, 1000]
	alpha	[1, 10, 20]
1x	*hidden-layers*	[(50,), (100,)]
	max-iter	[100]
	alpha	[1, 10, 20]
2a	*hidden-layers*	[(15, 10)]
	max-iter	[50, 100, 200, 300, 500]
	alpha	[0.1, 1, 3, 5, 8, 10]

2b	*hidden-layers*	[(30,)]
	max-iter	[50, 100, 200, 300, 500]
	alpha	[0.1, 1, 3, 5, 8, 10]
2c	*hidden-layers*	[(30, 20)]
	max-iter	[50, 100, 200, 300, 500]
	alpha	[0.1, 1, 3, 5, 8, 10]
2d	*hidden-layers*	[(15, 15), (15, 10, 10)]
	max-iter	[50, 100, 200, 300, 500, 800]
	alpha	[0.1, 1, 3, 5, 8, 10]
2e	*hidden-layers*	[(15,), (20,), (30,)]
	max-iter	[50, 100, 200, 300, 500, 800]
	alpha	[0.1, 1, 3, 5, 8, 10]
2f	*hidden-layers*	[(30, 20), (30, 15), (30, 10)]
	max-iter	[50, 100, 200, 300, 500, 800]
	alpha	[0.1, 1, 3, 5, 8, 10]
2g	*hidden-layers*	[(20, 15), (20, 10)]
	max-iter	[50, 100, 200, 300, 500, 800]
	alpha	[0.1, 1, 3, 5, 8, 10]
2ha	*hidden-layers*	[(10,), (15,), (20,)]
	max-iter	[10, 50, 100, 200, 500]
	alpha	[1, 5, 8]
	learning-rate	['adaptive']
2hc	*hidden-layers*	[(10,), (15,), (20,)]
	max-iter	[10, 50, 100, 200, 500]
	alpha	[1, 5, 8]
	learning-rate	['constant']
2hi	*hidden-layers*	[(10,), (15,), (20,)]
	max-iter	[10, 50, 100, 200, 500]
	alpha	[1, 5, 8]

	learning-rate	['invscaling']
3aa	*hidden-layers*	[(10,), (15,), (20,), (100,), (200,), (300,), (500,)]
	max-iter	[200, 500, 1000]
	alpha	
	learning-rate	['adaptive']
	momentum	[0.9, 0.2]
	solver	['sgd']
3ac	*hidden-layers*	[(10,), (15,), (20,), (100,), (200,), (300,), (500,)]
	max-iter	[200, 500, 1000]
	alpha	
	momentum	[0.9, 0.2]
	learning-rate	['constant']
	learning-rate-init	[0.3]
	solver	['sgd']
3ai	*hidden-layers*	[(10,), (15,), (20,), (100,), (200,), (300,), (500,)]
	max-iter	[200, 500, 1000]
	alpha	
	momentum	[0.9, 0.2]
	learning-rate	['invscaling']
	learning-rate-init	[0.3]
	solver	['sgd']

Appendix C

More Details of Genre Classifiers

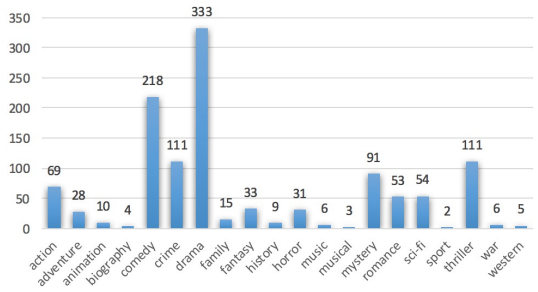


Figure C.1: Frequencies of Genres

Table C.1: Genre Classifier - Naive Bayes

GENRE	accuracy	ngram-range	num-features
action	.4031	(2, 2)	10k
adventure	.2732	(2, 3)	2k
comedy	.7789	(2, 2)	2k
crime	.5856	(2, 2)	10k
drama	.8520	(2, 2)	100k
fantasy	.2507	(2, 2)	10k
horror	.1861	(2, 3)	100k
mystery	.4698	(2, 3)	100k
rank	.4726	(2, 3)	10k
romance	.2090	(2, 3)	5k
sci-fi	.3790	(2, 2)	10k
thriller	.5304	(2, 3)	1k

Table C.2: Genre Classifier - DR

GENRE	**NB**				**MLP**			
	w2v		**d2v**		**w2v**		**d2v**	
	wiki	**subt**	**wiki**	**subt**	**wiki**	**subt**	**wiki**	**subt**
action	.3902	.4042	**.4318**	.3837	.4779	.4555	.4565	**.5003**
adventure	.2433	.2727	**.3448**	.2479	.3393	.3688	.3693	**.4002**
comedy	.7722	.7507	**.8195**	.7891	.7954	.8106	**.8209**	.8197
crime	.4917	.5286	**.5694**	.5219	.6273	**.6511**	.5888	.5951
drama	.7931	.7895	.8328	**.8627**	.8432	.8414	.8620	**.8721**
fantasy	.2328	.2355	**.3710**	.2467	.3510	.3948	**.4261**	.3971
horror	.2125	.2513	**.2885**	.2790	.2804	**.4215**	.3501	.3536
mystery	.4358	**.4785**	.4341	.4443	.4696	**.4836**	.4350	.4558
romance	.2223	.2288	.2166	**.2597**	.2996	.2876	.2104	**.3477**
sci-fi	.3919	.3877	**.4392**	.3849	.4890	**.5638**	.4880	.5145
thriller	.4797	.5115	**.5303**	.4923	.5580	**.5655**	.5237	.5043

Table C.3: Genre Classifier - MLP parameters

GENRE	**accuracy**	**dataset**	**layers**	**max-iter**	**alpha**
action	.5003	d2v-subt	(10,)	100	1
adventure	.4002	d2v-subt	(15,)	100	5
comedy	.8209	d2v-wiki	(15,)	200	20
crime	.6511	w2v-subt	(15, 10)	200	1
drama	.8721	d2v-subt	(15,)	100	15
fantasy	.4261	d2v-wiki	(15,)	100	1
horror	.4215	w2v-subt	(15, 15, 15)	200	10
mystery	.4836	w2v-subt	(15, 15, 15)	200	5
romance	.3477	d2v-subt	(10,)	100	0
sci-fi	.5638	w2v-subt	(15, 15, 15)	100	0
thriller	.5655	w2v-subt	(20,)	200	1

Appendix D

Non-Print Material

Some non-print material is collected and compressed.

- The file **vault.tar.xz** contains a fold for each of 512 television series.Intermediate results are included.
- The folder **w2v** contains the trained *word embedding* model of the English part of the OpenSubtitles2016 corpus with *word2vec*.
- The folder **d2v** contains the trained *document embedding* model of the English part of the OpenSubtitles2016 corpus with *doc2vec*.
- The file **code-n-data.tar.xz** mainly contains the code.
- The file **treasure.tar.xz** contains data matrices and test records of the classifiers.

These could be downloaded from:

https://drive.google.com/open?id=0B6iYEYKt0aqFRWgtRE9COF8wX0E

Appendix E

Example Classifier Test Record


```
[
  ["best_score",  0.48504273504273504],
  [
    "best_param",
    "{'clf__hidden_layer_sizes': (50,)," \
    "'clf__max_iter': 100, 'clf__alpha': 1}"
  ],
  [
    "mean_test_score",
    [
      0.48504273504273504,
      0.4722222222222222,
      0.452991452991453,
      0.4465811965811966,
      0.4465811965811966,
      0.42948717948717946
```

```
      ]
    ],
    [
      "params",
      {
        "clf__hidden_layer_sizes": [[50], [100]],
        "clf__max_iter": [100],
        "clf__alpha": [1, 10, 20]
      }
    ],
    ["xsource",  "combined.d2v.subt.data"],
    ["ysource", "rank.data"],
    ["param_key", "1x"]
]
```

Appendix F

Example Code Snippet

Parsing of PDF Files

```
from pdfminer.pdfdevice import PDFDevice

class _Extracter(PDFDevice):
  def begin_page(self, page, ctm):
    info = {
      'page-id': self._pageno,
      'box': page.mediabox,
    }
    output = '## PAGE-BEGIN ## %s\n' % json.dumps(info)
    self._w(output)

  def end_page(self, page):
    self._pageno += 1
    self._w('## PAGE-END\n')

  def render_string(self, textstate, seq):
    info = {'matrix': textstate.matrix}
    output = '## LINE ## %s\n' % json.dumps(info)
    self._w(output)
```

```
    self._w(self._seq_to_text(seq, textstate.font))
    self._w('\n')
```

Build Multi-Graph

```
import networkx as nx

def build_multigraph(scene_list):
  model = nx.MultiGraph()
  for scene in scene_list:
    roles = []
    for block in scene['block']:
      if block['type'] == 'TALK':
        roles.append(block['role'])
    role_set = set(roles)
    if len(role_set) == 1:
      mg.add_node(roles[0])
    else:
      mg.add_weighted_edges_from(build_edges(role_set))
```

Dependency Parsing

```
def parse_dep():
  ctx ={
    'nndep-path': os.environ['STANFORD_PARSER_PATH'],
    'pwd': os.environ['PWD'],
  }

  nndep_cmd =
      'cd␣%(nndep-path)s␣&&␣java␣edu.stanford.nlp.parser.nndep.DependencyParser'
      '␣-model␣edu/stanford/nlp/models/parser/nndep/english_UD.gz' \
      '␣-textFile␣%(pwd)s/db.sentences' \
      '␣-outFile␣␣%(pwd)s/db.sentences.nndeps' % ctx
```

```
  os.system(nndep_cmd)
```

Multi-Layer Perceptron

```
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

def search_mlp(xsource, ysource, params):
  clf = MLPClassifier(random_state=42)
  pipeline = Pipeline([('clf', clf)])
  gs_clf = GridSearchCV(pipeline, params, n_jobs=1,
      scoring=None, cv=None).fit(xsoiurce, ysource)

  info(gs_clf.best_score_)
  info(gs_clf.best_params_)
  info(gs_clf.best_index_)
  info(gs_clf.best_estimator_)
  info(gs_clf.cv_results_)
```