



ASSIGNMENT OF MASTER'S THESIS

Title: Analyzing Supreme Court Oral Arguments with Natural Language Processing
Student: Bc. Lukáš Polák
Supervisor: Nicholas Webb, Ph.D.
Study Programme: Informatics
Study Branch: Knowledge Engineering
Department: Department of Theoretical Computer Science
Validity: Until the end of winter semester 2017/18

Instructions

The goal here is to examine topic chains in oral argument transcripts of the Supreme Court of the United States. Often, justices agree on outcome of votes, but they disagree on justification of their vote. The student will try to find patterns in their use of language, with two possible goals. First, to try to predict outcome of their voting. Second, to detect groupings of justices based on overlapping arguments.

For this purpose, the student will parse transcripts freely available on the US Supreme Court website and explore current NLP approaches and techniques, such as dialogue act tagging, polarity analysis, topic introduction metrics, and others. Then, the student should select metrics and approaches based on results of their findings, in order to create a model that can predict outcome of the voting. The model can be evaluated against publicly available results of the rulings.

References

Will be provided by the supervisor.

L.S.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrđík, CSc.
Dean

Prague February 23, 2016

CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF THEORETICAL COMPUTER SCIENCE



Master's thesis

Analyzing Supreme Court Oral Arguments with Natural Language Processing

Bc. Lukáš Polák

Supervisor: Nicolas Webb, Ph.D.

10th January 2017

Acknowledgements

I would like to express my acknowledgements to my supervisor, who provided me with valuable advice during our cooperation.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 10th January 2017

.....

Czech Technical University in Prague
Faculty of Information Technology

© 2017 Lukáš Polák. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Polák, Lukáš. *Analyzing Supreme Court Oral Arguments with Natural Language Processing*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Táto práca používa jednoduché techniky NLP pre natrénovanie modelu schopného predpovedať rozhodnutie prípadu vypočutého Najvyšším Americkým Súdom na základe analýzy ústneho pojednávania súdu. Pokúša sa ukázať, že jednoduché techniky dokážu poskytnúť relevantné výsledky na komplexných problémoch. Preto oba modely vyvinuté v tejto práci ich preferujú.

Prvá časť práce sa zameriava na natrénovanie jednoduchého doménovo nezávislého modelu detekcie sentimentu na úrovni jednotlivých viet. Model je natrénovaný na korpuse MPQA, pretože obsahuje dokumenty z rozličných domén. Vety boli označované hodnotiacim algoritmom vyvinutým v tejto práci.

Druhá časť práce sa zameriava na exploratívnu analýzu pojednávania súdu a následný výber atribútov modelu. Na základe jej výsledkov sú tréňované modely využívajúce jeden z troch odlišných klasifikátorov a niektorú z podmnožín možných atribútov modelu. Následne sú navzájom porovnávané, aby sa skúmal dopad jednoduchých NLP techník na ich výsledky.

Kľúčová slova NLP, Najvyšší Americký Súd, Analýza sentimentu, Synonymá

Abstract

This thesis uses simple NLP techniques to train a model that is capable to predict outcome of a case heard in the US Supreme Court based on analysis of its Oral Argument. It attempts to demonstrate that simple NLP techniques perform comparatively well when used on complex tasks. Therefore, models in both parts of the thesis prefer them.

The first part of the thesis focuses on training a simple domain-independent sentence-level sentiment model. This model is trained on MPQA corpus, because it contains manually annotated documents from multiple domains. The dataset is ranked by custom ranking algorithm developed in the thesis.

Second part of the thesis focuses on exploration analysis of the Oral Arguments and feature selection. Based on its results, models using one of the three different classifiers and several feature sets are trained. They are compared against each other to assess effect of the simple NLP features on their performance.

Keywords NLP, US Supreme Court, Sentiment Analysis, Synonyms

Contents

Introduction	1
Motivation	2
Goal	3
1 US Supreme Court	5
1.1 Overview	5
1.2 How US Supreme Court works	6
1.3 Available sources of data	9
2 Theoretical Background	17
2.1 NLP techniques	17
2.2 Sentiment Analysis	33
2.3 Optical Character Recognition	46
2.4 Technologies	48
2.5 Datasets and Corpora	51
3 Sentiment Analysis	57
3.1 MPQA Dataset and Ranking Algorithm	57
3.2 Feature extraction	64
3.3 Training and results	66
4 Oral Argument Analysis	75
4.1 Related Works	75
4.2 Oral Arguments Dataset	76
4.3 Feature Extraction and Selection	79
4.4 Training and Results	89
Conclusion	101
Applications and Future Work	103

Bibliography	105
A Acronyms	111
B Ranking constants MPQA	113
C MPQA features list	115
D Command-line script	121
D.1 Controller - Inquirer	121
D.2 Controller - Largereviews	121
D.3 Controller - Mpqa	122
D.4 Controller - Pdf	122
D.5 Controller - Pipeline	123
D.6 Controller - Reports	123
D.7 Controller - Sentiment	124
E Contents of enclosed CD	127

List of Figures

1.1	US Court system	7
1.2	A snippet of an opinion written on a US Supreme Court case . . .	8
1.3	Demonstrations in front of the US Supreme Court supporting same-sex marriages	10
1.4	Snippet from one Oral Argument; each turn is visibly separated for demonstration	11
1.5	Example of interruption of a speech during the Oral Argument . .	13
1.6	Example of Supreme Court records from Washington Law University organized by docket	15
2.1	Eight basic categories of words used in English grammar	20
2.2	Example of HMM implementation of POS tagger for sentence "Fed raises interest rates."	23
2.3	Example of syntax trees for sentence " <i>I shot an elephant in my pyjamas</i> "	26
2.4	Measure m calculated on word <i>crepuscular</i>	27
2.5	Example of dependency tree for path-based similarity with $pathlen(nickel, coin) = 2$, $pathlen(nickel, money) = 6$ and $pathlen(nickel, Richterscale) = 8$ [1]	30
2.6	Example of WordNet hierarchy augmented with probability (pictured is only a part of the tree) $P(c)$ [1]	31
2.7	Example of Facebook post on <i>Arrow</i> page captured on Dec 17, 2016 at 10:20PM CET	39
2.8	Example of one possible vector space representation of animals . .	43
2.9	Comparison of monospaced and proportional font	47
2.10	Overview of Tesseract OCR processing pipeline	49
2.11	Example of Jupyter Notebook used to clean up data for this thesis	50
3.1	Final version of ranking algorithm for MPQA corpus	60
3.2	Distribution of counts of annotations in sentence in MPQA corpus	61

3.3	Frequency of word counts in sentences with at least 31 words . . .	62
3.4	Frequency sentiment intensity in dataset for class <i>positive</i> and <i>negative</i>	64
4.1	Example of one page from oral argument in "top-right" mode with offsets in pixels	79
4.2	Example of one page from oral argument in "bottom-middle" mode with offsets in pixels and undetected word <i>the</i> outside of reading area	79
4.3	Number of dockets each justice speaks up at least once during the Oral Argument	80
4.4	Number of turns each justice has in all dockets together and their average number per docket	81
4.5	Number of words each justice utters in all dockets together and their average number per docket	82
4.6	Frequency of asked questions per different phase and type of dialog; 'res-other-pet' means that winning party is <i>Respondent</i> , not justice is asking the questions and it is <i>Petitioner's</i> part of the hearing . .	82
4.7	Frequency of asked questions per different phase and type of dialog; 'res-other-pet' means that winning party is <i>Respondent</i> , not justice is asking the questions and it is <i>Petitioner's</i> part of the hearing . .	83
4.8	Distribution of lengths of topic chains in dataset split into 10 equal-width bins	86
4.9	Comparison of different settings for algorithm that groups similar words together	87
4.10	Average numbers of turns per docket split into sentiment classes (label 'res-pet' represent numbers for dialog phase when Respondent wins the case and the turn is in Petitioner's part)	88
4.11	Ratio between numbers of dockets in two classes created by k-means algorithm split into two groups of dockets by winning party	91
4.12	Confusion matrix for Gaussian Naive Bayes classifier	93
4.13	Confusion matrix for k-Nearest Neighbor classifier	94
4.14	Best F_1 score for different settings of parameter <i>k-neighbors</i> for k-Nearest Neighbor classifier	95
4.15	Best achieved F_1 score for different settings of <i>n_estimators</i> for Random Forest Tree classifier in round 1	95
4.16	Best F_1 score for different settings of parameter <i>max-depth</i> for Random Forest Tree classifier	97
4.17	Confusion matrix for Random Decision Forest classifier	97

List of Tables

2.1	Mapping important Universal tags to Penn Treebank tags	19
2.2	Example of rules for rule-based POS tagger	21
2.3	Example of rules for syntax tree context free grammar	25
2.4	Example of rules for a stemmer	27
2.5	List of interesting corpora included in NLTK	52
2.6	Statistics for WordNet	53
3.1	Sources of documents in MPQA 2.0 corpus	58
3.2	Distribution of sentences into sentiment classes by ranking algorithm	64
3.3	Thresholds for feature selection for classes in sentiment model . . .	69
3.4	Confusion matrix for simple classifier on Large movie reviews dataset	71
3.5	Confusion matrix for trained classifier on MPQA dataset	71
3.6	Confusion matrix for trained classifier on Large movie reviews dataset	72
3.7	Distribution of feature categories in feature set of the trained sen- timent model	73
4.1	Distribution of interruptions by justices in dataset	83
4.2	Distribution of <i>Most Followed by</i> in dockets per every justice . . .	84
4.3	A few examples for most occurring phrase lengths in dataset . . .	85
4.4	Number of appearances of attorneys in front of the Court in dataset	87
4.5	Distribution of Difference in Sentiment Changes in dockets split by winning party	88
4.6	Size of train and test set with different stratified n-folds in cross- validation with ratio of Petitioner:Respondent samples in set; format is <i>petitioner(respondent)</i> size of the fold	92
4.7	Results (F_1 score and accuracy) for Gaussian Naive Bayes clas- sifier with respect to BASE results	93

LIST OF TABLES

4.8	Results (F_1 score and accuracy) for k-Nearest Neighbor classifier with respect to BASE results	94
4.9	Results (F_1 score and accuracy) for Random Decision Forest classifier with respect to BASE results	97
4.10	Comparison of performance (F_1 score and accuracy) of best models for each classifier	99
B.1	"Intensity" attribute	113
B.2	"Expression Intensity" attribute	113
B.3	"Polarity" attribute	114
B.4	"Attitude type" attribute	114
B.5	"Attitude uncertain" attribute	114
C.1	List of extracted features - Comparison features	115
C.2	List of extracted features - Category Counts	116
C.3	List of extracted features - Negation-related features	117
C.4	List of extracted features - N-gram	118
C.5	List of extracted features - POS n-grams	119
C.6	List of extracted features - Other features	119
D.1	Parameters for task " <i>read-corpus-raw</i> " in controller	121
D.2	Parameters for task " <i>read-pdf</i> " in controller <i>Pdf</i>	123
D.3	Parameters for tasks in controller <i>Pdf</i>	123
D.4	Parameters for task <i>all-tasks</i> in controller <i>Pipeline</i>	123
D.5	Parameters for task <i>all-tasks-dir-no-pdf</i> in controller <i>Pipeline</i>	124
D.6	Parameters for task <i>all-tasks-dir</i> in controller <i>Pipeline</i>	124
D.7	Parameters for tasks <i>nlp-reports</i> and <i>basic-reports</i> in controller <i>Reports</i>	124
D.8	Parameters for tasks in controller <i>Sentiment</i>	125

Introduction

NLP (Natural Language Processing) can be used to harvest the vast knowledge in written materials available on the Internet. To cope with this situation, countless number of techniques designed to analyze a structured text were devised. However, there is also an indisputable number of other forms of text having different structures which are waiting to be analyzed. Among the datasets with exotic structures, there is a group of datasets - discussions forums, chat bots, messengers or dialogues - that share common traits. All of them require a different type of analysis - **Discourse Analysis**.

In comparison to an ordinary written text like article or novel, its rules are more relaxed. The used language may be more informal, abstract and may not be grammatically correct. Even body language of participants matters. Thus, when a verbal dialog is transcribed to a text form, it is a writer's job to capture all the circumstances relevant to the discourse too.

A great example of discourse are *Oral Arguments* at US Supreme Court. The court publishes the transcripts of every *Oral Argument*. They contain one hour long dialog between Justices, petitioner and/or attorney. Their analysis can provide everybody closer look on court's decision making process.

The goal of this work is to provide perspective on US Supreme Court through behavioral analysis of all actors in the discourse leveraging simple NLP techniques and their validity for such a complex task. We examine to extent a model that can predict ruling of the court can be created. The rest of this thesis is separated into the following chapters.

The later sections of this chapter emphasis author's motivation to work on the topic. It also highlights the pitfalls of this topic which have to be dealt with in detail in other chapters. Lastly, it states the hypothesis which this thesis is trying to study.

Chapter 1 explains how US Supreme Court works and what documents were available and used during the analysis. The reader is informed about consequences which the court's ruling present to public. Also, the reader learns about examples of rulings which affected a major group of people or

their outcome was unexpected or controversial to some degree. Lastly, all data sources available for a case are mentioned and materials important for the thesis are described in detail.

Chapter 2 lists all the NLP techniques proposed in the work. Plus, it explains models used for sentiment analysis and ruling prediction. This overview is enriched with their technical explanation. Also, the author reasons why some of the popular techniques were not used.

Chapter 3 introduces the sentiment model created in this work. The chapter opens up with a brief explanation of different types of sentiment analysis. This is followed with overview of current state of the art approaches to the problem. Then, it presents datasets used to train, test and validate the sentiment model along with necessary data preparation steps. Lastly, it provides a closer look at data itself and how the final sentiment model is created.

Chapter 4 focuses on applying techniques introduced in Chapter 2 and sentiment model developed in Chapter 3 on analysis of transcriptions of the Oral Arguments. It begins with data extraction and exploration of the text. This process is used to identify relevant features for the ruling prediction model. Next, these features are tested on potential models to find out which model makes the most reliable predictions. All relevant results are presented in the chapter to provide overall view on behavior of the models.

Finally, the last Chapter draws a conclusion based on the findings from the previous chapter. Then, the author points out several possible improvements which can be considered for the future work on this topic.

Motivation

Studying a discourse between multiple individuals can provide insight into numerous aspects of social interaction in groups. For example, a degree of disagreement between the parties or their attempts to influence each other can unveil inner-social dynamics inside the group. Or, basically in any online interaction between people (for example in discussion forums). So, the results may be applied to enhance user experience when talking to chat bots on social media (e. g. automatic detection when customer is hostile or evaluation of customer's interest in the offer) or to implement automatic advanced governing policies on discussion forums (like preventing bullying members or stopping flame wars at their beginnings).

In real world, there is one particular set of individuals who bear a very important role in the US society as their decisions affect the lives of every US citizen, **Justices of the US Supreme Court**. It probably sparks everybody's curiosity to see, if their decisions can be predicted to any extent.

2016 is the right time to study and question how the US Supreme Court makes its decisions. A newly elected US president is in a unique position.

Having only 8 Justices in Supreme Court after Justice Scalia passed away in February 2016, there is a vacancy that needs to be filled by the US President [2]. Both political views being in equilibrium at the time, the choice of Scalia's successor poses a great deal of controversy. Plus, there is an expectation that another 3 Justices may be replaced in the next 4 years. This assumption only highlights the importance to understand and uncover any bias among the Justices with a certain political view.

There is several distinctive groups who are eager to gain any edge they can on Justices. The first group are petitioners who are about to submit their case to the Court. Having 7,000-8,000 petitions to go through each Term (8 months) and selecting only approximately 80 cases for hearing fuels petitioner's effort to learn as much about the Court as they can beforehand [3]. Another group are law students who want to get insight on Court's ruling process. Many of them aspire to be a judge at point in their carer, so understanding the dynamics between Justices is highly relevant for them. Then, there are attorneys who are about to argue a case in front of the Court. Acquiring tips on how to manipulate the Court or read Justice's tells during the Oral Argument is invaluable for them.

By successfully using simple tools to explore Court's arguments in detail, we will try to take a peak at minds behind US Supreme Court.

Goal

The goal of this thesis is twofold.

The main goal of this thesis is to evaluate behavior of simple NLP techniques when applied on a complex problem in discourse analysis. They are used to analyze the Oral Arguments in US Supreme Court and to create a novice approach to predict outcome of cases from period of years 2013-2015.

The secondly goal is to create a simple domain-independent sentiment model to support the prediction model created as the main goal. The sentiment model is capable of detecting positive, negative and neutral sentiment on sentence level. In order for the model to work, it depends upon a specific training set created by customized ranking algorithm.

US Supreme Court

This chapter summarizes relevant information about the US Supreme Court. It gives a glimpse of the Court's history and how it works nowadays. Plus, there is an emphasis on the impact of the Court's ruling with examples of peculiar cases which had wide-spread consequences on US society. Then, it describes the format and source of datasets used in this thesis, namely transcripts of the Oral Arguments¹ and meta data on cases collected by *University of Washington*.

1.1 Overview

The US Supreme Court is the highest law institution in the USA. According to the Constitution, there can be only one Supreme Court [4]. Even though there are no formal requirements to become a Supreme Justice, its members are carefully selected by the US Senate. The potential nominees are picked by the US President. While there is no official term limit for a Supreme Justice, they can be impeached, if convicted in a Senate trial. This occurred only once in history of the US Supreme Court in 1805 [5] and was stopped by the Senate in the end.

Historically, the US Supreme Court first convened on February 2 1790 [6] in New York City. Nowadays, they meet in Washington DC. Initially, there was 6 Justices – one Chief Justice and five Associate Justices. Later, their number was increased to 10 and, recently, the US Congress decreased their number down to 9 [4]. This odd count allows them avoid any potential ties in their ruling. Each term starts in October and ends in July. Typically, decisions in all the argued cases are announced by end of June [7].

The US Supreme Court procedure for a case is following: [8]

1. Firstly, Justices select a case using *writ of certiorari*

¹Oral Arguments are publicly available for free at https://www.supremecourt.gov/oral_arguments/argument_transcript.aspx

1. US SUPREME COURT

2. Then, relevant parties (petitioner and respondent, if there is any) are asked to write their briefs
3. They are basis for the Oral Argument which occurs between the petitioner, respondent and/or their attorneys and Justices (**this part is the focus of this thesis**)
4. Afterwards, the Court votes on the case during *Conference*
5. Lastly, opinions on Justices ruling are written

1.2 How US Supreme Court works

The US Court system is split into two parts – State/Local Courts and Federal Courts. Figure 1.1 shows this split. Courts on the left side represent Federal Courts. Consequently, courts on the right side represent State/Local Courts. Their areas of interest and competence are fundamentally different from each other. For the most part, US Supreme Court as a Federal Court, handles only a few types of cases such as [9]:

- Disputes between two or more states
- Cases involving violation of federal laws or the US Constitution
- Cases in which the United States is a party
- Bankruptcy, copyright or patent infringement

Being the highest legal body in the country, the Court has power to overrule any decision made by any other court. In general, its goal is to take cases with legal issues which will affect as much US population as possible. The Court's ruling usually impacts the Constitution or its outcome affects the whole nation. [4] Nevertheless, there is no way for the Court to enforce their decision upon the nation except for declaring or changing the law. Therefore, it is not in its power to physically see to it that its ruling is implemented all across the nation wherever it is relevant.

There are many ways a case is picked up by the Court. Around 90 % of the cases are hand-picked by Justices themselves exercising *writ of certiorari* [10]. This prerogative allows them to review cases which otherwise would not be entitled to review. Once selected for review by one Justice, others typically read it as well and discuss it. If only four Justices vote to hear the case in front of the Court, *Rule of Four*² applies. The Rule of Four prevents a majority of the Court to control Court's docket.

²Rule of Four means that if four Justices vote to hear a case then the case is taken to the Court

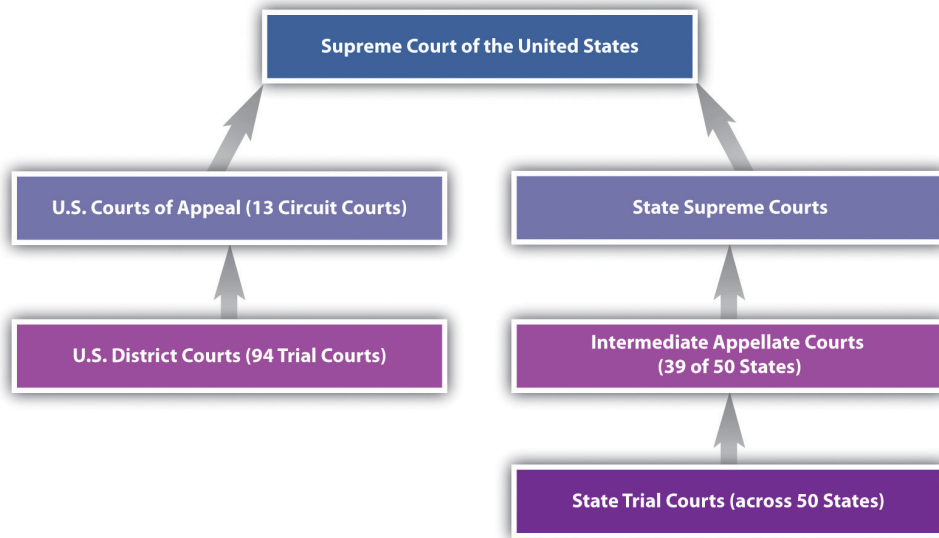


Figure 1.1: US Court system

Before the Oral Argument starts, the petitioner writes a brief stating his grounds for the case. Then, the opposing party (respondent) writes a brief from their point of view. If there are any other parties interested in the case, they can write *amicus curiae*³ briefs offering their opinions on the matter. These briefs are evaluated by Justices and their clerks before the Oral Arguments starts.

The date of the Oral Argument is set in advance when the case is picked by the Court. Studies shows that importance of the Oral Argument should not be underestimated. [11]. The hearing usually takes place on Monday, Tuesday or Wednesday every week of the Term (unless the Court says otherwise). Mostly, they last 60 minutes and are divided into several sections. Each party is given approximately the same amount of time to present their case. But, each party can be represented by multiple speakers. [11]

Generally speaking, Justices vote to affirm or reverse the lower court's decision). The voting takes place during *Conference* either on Wednesday or on Friday, of the given week. Only Justices of the Court are allowed to attend it. The first order of business at the Conference is to decide what cases to accept next. Then, they move on to discuss the cases of the given week starting with the most senior Justice and continues in descent order of their seniority⁴. No Justice gets interrupted. Their voting follows the same ordering pattern and starts with the Chief Justice as well. In case of a tie, decision of the lower court remains. [8].

³"amicus curiae" is Latin for "friend of the court"

⁴Seniority of a Justice is given by number of years in the US Supreme Court

Cite as: 578 U. S. ____ (2016)

1

THOMAS, J., dissenting

SUPREME COURT OF THE UNITED STATES

UNITED STUDENT AID FUNDS, INC. *v.*
BRYANA BIBLE

ON PETITION FOR WRIT OF CERTIORARI TO THE UNITED
STATES COURT OF APPEALS FOR THE SEVENTH CIRCUIT

No. 15–861. Decided May 16, 2016

The petition for a writ of certiorari is denied.

JUSTICE THOMAS, dissenting from the denial of
certiorari.

This petition asks the Court to overrule *Auer v. Robbins*,
519 U. S. 452 (1997), and *Bowles v. Seminole Rock & Sand
Co.*, 325 U. S. 410 (1945). For the reasons set forth in my
opinion concurring in the judgment in *Perez v. Mortgage
Bankers Assn.*, 575 U. S. ____, ____ (2015), that question is
worthy of review.

Figure 1.2: A snippet of an opinion written on a US Supreme Court case

After the decision is made, several types of opinions may be written – concurring, dissenting and plurality opinion. The senior Justice in the majority group assigns the job to write the majority (concurring) opinion on the case (Figure 1.2). This opinion explains rationale behind the court’s decision. Even though this task can be carried out by a Justice, studies show that in 30 % cases it gets assigned to a Supreme Court clerk. These clerks are usually graduates from Ivy League schools in their mid-to-late 20s. [12] Therefore, the views in them might not 100 % match Justice’s reasons for the voting. In some cases Justice in the majority group may have different reasons to vote in the same way so multiple concurring opinions may be written. In this case, the concurring opinion joined by the greatest number of judges is referred to as plurality opinion. On the other hand, dissenting opinions may be written by judges in the minority group. There is no rule which dictates that the opinion has to be written. In its nature, a dissenting opinion does not have any binding nature for the Court. It only written by one or more Justices to express their disagreement with the majority opinion.

1.2.1 Impact of rulings

Recent history of the US Supreme Court ruling demonstrates the broad spectrum of different topics to deal with. [13] [14] Most of them had a controversial tone.

For example, in 2016 Abigail Fisher ⁵ sued University of Texas for rejecting her admission application. She claimed the rejection was race motivated – less-qualified minority students were accepted instead of her. The university’s defense is based on its effort to keep its student body as diverse as possible. This case initially appeared on the Court’s radar in 2013. The lower court’s decision sided with the university. Back then, they sent the case back to the lower court to review the case. When revisited in 2016, the US Supreme Court ruled in favor of the university again. Implication of their decision is that taking race into consideration during the admission process is constitutional.

Another example is case *”Elonis vs. United States”*. In short, Anthony Elonis posted explicit posts on Facebook publicly threatening his wife. Initially, Anthony was sentenced to 44 months behind the bars. The lower court’s ruling was based on a federal law that it is a crime to communicate *”any threat to injure the person of another”*. Later, the US Supreme Court can reverse this decision. In this case, Justices claimed the prosecutors did not do enough to prove Anthony’s intent. Because of their decision, it is now harder to prosecute people for making threats on social media.

The next controversial case at the Court was *”Obergefell v. Hodges”* in 2015. Same-sex marriages were legal in more than 30 US states already. However, this was not true for Ohio. James Obergefell had a life-partner who was terminally ill. Since they wished to get married, they decided to travel to Maryland to get wedded. Unfortunately, the ceremony had to take place in the airport because of the illness. Despite being properly married, state Ohio refused to list James as his surviving spouse. This case provoked demonstrations in front of the Court house (Figure 1.3) and encouraged couples in many other states to challenge their bans on gay marriages. Thus, Justices decided to look at all these cases and consider them as part of the Obergefell’s case. In the end, they ruled in favor of same-sex marriages. The result of their ruling made same-sex marriages a national wide right backed by the Constitution.

This is not, by all means, a complete list of all important decisions made by the Court in a couple of last years. However, it clearly demonstrates the diversity of topics that Justices handle and their indisputable importance.

1.3 Available sources of data

As mentioned above, there is numerous data sources available for every case: [15]

- **Briefs** All briefs written for cases since October 2003 (with some exceptions since October 1990) can be found on sites linked from the US Supreme Court site⁶. There are also amicus curiae briefs, if written for a

⁵Case ”Fisher vs University of Texas”

⁶https://www.supremecourt.gov/oral_arguments/briefsource.aspx

1. US SUPREME COURT



Figure 1.3: Demonstrations in front of the US Supreme Court supporting same-sex marriages

case. These text are highly opinionated and are written by the interested parties, not Justices. Therefore, they do not provide any additional clues about Justice's impressions of the case.

- **Docket Information** For every case, there is several other documents filed in the case and information about status of the case. They can be accessed via *Court's automated docket system*⁷. Only docket information since January 2000 are currently available.
- **Oral Argument** Almost every case argued at the US Supreme Court gets to be heard by Justices during 60-minute long Oral Argument. Transcript of the argument along with its audio recording are publicly released⁸ since October 2006. They are not videotaped.
- **Opinions** After the decision is made, Justices can write their opinions to rational their votes (more information in section 1.2). They are available in limited manner in printed version in libraries nationwide. Or, they are released in PDF format on US Supreme Court site⁹.
- **Speeches** Justices may hold a speech on a subject related to the case afterwards. Their transcripts are also available online in PDF format¹⁰.

⁷<https://www.supremecourt.gov/docket/docket.aspx>

⁸https://www.supremecourt.gov/oral_arguments/availabilityoforalargumenttranscripts.aspx

⁹<https://www.supremecourt.gov/opinions/opinions.aspx>

¹⁰<https://www.supremecourt.gov/publicinfo/speeches/speeches.aspx>

7	JUSTICE SCALIA: Mr. Goldstein, if I understand your argument, you're saying that there is an obligation to conciliate, but that obligation does not have to be pursued in good faith.
11	MR. GOLDSTEIN: No, Justice Scalia. We think --
13	JUSTICE SCALIA: So that the -- even if the Commission enters upon a conciliation process, you think that the outcome of that is reviewable?

Figure 1.4: Snippet from one Oral Argument; each turn is visibly separated for demonstration

These speeches can be considered extended versions of their opinions and are not in any way regular.

Since this thesis is focused on analyzing only materials directly related to Justices and their decision-making process, it puts its focus only on the Oral Argument as its primary source (more information in section 1.3.1). There is also several materials concerning each Justice's biography which are available online, but provide no relevant insight on their decision-making chain of thoughts.

Also, there are other publicly accessible data which provide concentrated information about every case. One of the most popular sources is *The Supreme Court Database* accessible on *University of Washington* sites (more information in section 1.3.2). It describes background of the cases. All this data can be collected before the decision on a case is made and it directly relates to the Oral Argument and Justices decision process. Thus, it is used as a secondary data source in this thesis.

1.3.1 Oral Arguments

Procedure during the Oral Arguments is strict. Always, Chief Justice introduces the case by its number and its name. Only the Chief Justice is called "*Chief Justice*". Other Justices are called by their names, for example "*Justice Scalia*" or "*Justice Breyer*". Then, petitioner or its attorney have the word and may give a short opening statement. This statement is followed by a dialog with Justices (see Figure 1.4 for an example of the dialog during the Oral Argument). After this part is over (typically, after 30 minutes),

Then, the Chief Justice thanks the petitioner and their attorney and allows respondent to present their opening statement. As with the petitioner, the opening statement is followed by a dialog with justices. Again, this part takes approximately 30 minutes.

There is several essential aspects related to the Oral Argument. Here is how they are captured in PDF format of the transcript (more information about the actual format is in Section 4.2):

- **Actors of speech** Whenever somebody in Court says something, it is transcribed word by word. Whatever they say is prefixed with actor's name. In case the actor is a Justice, the prefix says "*JUSTICE *NAME**". Otherwise, it simply says "*MR./MRS. *NAME**".
- **Turns in discourse** Even though the transcript of the Oral Argument evokes assumption that participants take turns when talking, it is not true. For sake of simplicity, this thesis considers every remark a separate turn. Therefore, when somebody is interrupted, its speech is represented by two turns. This matches the real world observation of the case in the court room.
- **Interruptions** When somebody is interrupted, the transcript tries to capture this event using two consequent hyphens at the end of their turn. If the actor tries to finish their thought after the interruption, their next turn starts with two hyphens. (see Figure 1.5). Needless to say, respondent and petitioner have respect for Justices so they rarely interrupt them. In contrast, Justices do not hesitate to interrupt attorneys or petitioner/respondent, if they deem it necessary.
- **Pauses in speech** Whenever somebody makes a pause in speech during their turn, it is captured by two consequent hyphens. This pause can mean, either hesitation in their speech, or they need a bit of time to collect their thoughts.

1.3.2 Meta data

This dataset was created based on suggestion from professor Harold Spaeth from Washington's Law University in St. Louis about three decades ago. In its first version, it contained record for every vote by Supreme Court justice covering all cases over period of five years. During the initial research phase, he performed reliability checks to maintain integrity of the final dataset and then made it available for public. [16]

Since its first version, more contributors joined the effort. Together, they created the most extensive database of all votes made by Supreme Court justices since establishing the institution of Supreme Court. At the same

MR. GOLDSTEIN: Yes.

JUSTICE GINSBURG: The EEOC did send a
letter saying --

MR. GOLDSTEIN: Give us a call.

JUSTICE GINSBURG: -- give us a call. And

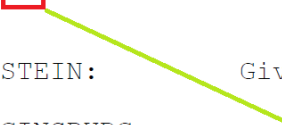


Figure 1.5: Example of interruption of a speech during the Oral Argument

time, they managed to keep the final data consistent by performing extensive integrity checks before adding records. Their joint effort helped to advance the whole field around studying decision process of the Court. In case researchers do not have IT background and want to perform simple analysis on the dataset, the website provides rudimentary interface to carry out simple analysis operations.¹¹

Nowadays, there is 247 pieces of information for each case which can be split into six higher-level categories:

1. **identification variables** - citation information, case number and ID, docket number and vote ID, ...
2. **background variables** - information about petitioner, reason why the Court took jurisdiction, where the case originated, ...
3. **chronological variables** - when the case was issued and when the decision was released, term of Court, ...
4. **substantive variables** - legal provision, direction of decisions, ...
5. **outcome variables** - winning party, disposition of the case, information about precedent, ...
6. **voting and opinion variables** - how individual justices votes and their opinions

These records are currently available in two separate versions of the publicly available databases:

- **MODERN** List of cases handled by the Court during terms 1946-now¹². Using data from recent past, it is able to provide multiple different perspectives on the same case.

¹¹Analytics interface is available on <http://scdb.wustl.edu/analysis.php>

¹²At the moment of writing this thesis, it covered cases until July 2016

1. US SUPREME COURT

- **LEGACY** Collection of historical data about cases from terms 1791-1946. The data is not well-covered due to its nature. Thus, only list of votes of each Justice for every recorded case is available. More granular data might be added later by the university.

Both databases offer two main views on the data – *Case Centered Data* or *Justice Centered Data* – with multiple perspectives. Case Centered Data view provides case level information. For each case, there is one row with data. On the contrary, Justice Centered Data view contains Justice level information. This means that there is information about how each Justice voted in every case.

Each main view in MODERN database offers four different perspectives on the data: ¹³

- **Cases Organized by Docket**¹⁴ One row represents one justice vote in a docket. Several cases may appear multiple times (for being in multiple dockets).
- **Cases Organized by Supreme Court Citation** One row represents a single dispute. Cases with multiple issues and consolidated cases¹⁵ are included only once.
- **Cases Organized by Issue/Legal Provision** There is a row for each issue and legal provision dealt with by the Court for each docket.
- **Cases Organized by Issue/Legal Provision Including Split Votes** This perspective is an expansion on *Cases Organized by Issue/Legal Provision*. It adds rows with rare instances when the Court has multiple vote coalitions on a single issue or legal provision.

Since this thesis analyses cases in years 2014-2016 and watches behavior of Justices, it uses MODERN database with Justice Centered Data. For simplicity *Cases Organized by Docket* perspective is used. Detailed description of attributes in the dataset and their selection process is explained in Chapter 4. Although, the dataset is available in number of different formats, CSV¹⁶ format is used in scripts in this thesis for convenience.

¹³Source: <http://scdb.wustl.edu/data.php?s=1>

¹⁴*Docket* is a list of cases pending the Oral Argument during one day

¹⁵Cases with multiple dockets

¹⁶CSV - Comma-Separated Values

1.3. Available sources of data

caseId	docketId	decisionType	term	naturalCo	chief	docket	petitioner	petitionerState	respondent	respondentState	jurisdiction	lcDisagreement
2000-001	2000-001-01	4	2000	1607	Rehnquist	8 ORIG	28	4	28	6	9	0
2000-001	2000-001-01	4	2000	1607	Rehnquist	8 ORIG	28	4	28	6	9	0
2000-001	2000-001-01	4	2000	1607	Rehnquist	8 ORIG	28	4	28	6	9	0
2000-001	2000-001-01	4	2000	1607	Rehnquist	8 ORIG	28	4	28	6	9	0
2000-001	2000-001-01	4	2000	1607	Rehnquist	8 ORIG	28	4	28	6	9	0
2000-001	2000-001-01	4	2000	1607	Rehnquist	8 ORIG	28	4	28	6	9	0
2000-001	2000-001-01	4	2000	1607	Rehnquist	8 ORIG	28	4	28	6	9	0
2000-001	2000-001-01	4	2000	1607	Rehnquist	8 ORIG	28	4	28	6	9	0
2000-001	2000-001-01	4	2000	1607	Rehnquist	8 ORIG	28	4	28	6	9	0
2000-001	2000-001-01	4	2000	1607	Rehnquist	8 ORIG	28	4	28	6	9	0
2000-002	2000-002-01	1	2000	1607	Rehnquist	99-1238	28	37	137		1	0
2000-002	2000-002-01	1	2000	1607	Rehnquist	99-1238	28	37	137		1	0
2000-002	2000-002-01	1	2000	1607	Rehnquist	99-1238	28	37	137		1	0
2000-002	2000-002-01	1	2000	1607	Rehnquist	99-1238	28	37	137		1	0
2000-002	2000-002-01	1	2000	1607	Rehnquist	99-1238	28	37	137		1	0
2000-002	2000-002-01	1	2000	1607	Rehnquist	99-1238	28	37	137		1	0

Figure 1.6: Example of Supreme Court records from Washington Law University organized by docket

Theoretical Background

This work consists of a great number of basic NLP techniques providing only partial information. Yet, their strength lies in their combination. Together, they create a powerful pipeline for analytic model. Having several simpler features working together means more control over the process. In case any of them does not perform well enough or there is a significant improvement in its area, it can be easily replaced. Moreover, it is easier to fine-tune them to work together. Also, during the analysis phase, it is easy to take them out of the pipeline, in case they do not provide enough additional information.

This chapter is divided into five sections. The first section introduces relevant NLP techniques. The second part of the text explains how they are used in context of this thesis. Then, the third section describes sentiment analysis, current state of the art approaches in this area and its pitfalls. The fourth section describes complexity of task of reading text from images and when it is used in the thesis. The fifth section mentions technologies used in the process of making this thesis for data preparation, data exploration and predictions. Lastly, there is a section describing all corpora used in the experiments in the thesis.

2.1 NLP techniques

All techniques used in this thesis aim to be simple, yet powerful. Two key elements supporting them are *Part of Speech Tagger* and *Synonyms*. Due to reasons mentioned in the section 2.1.3, *Syntax Tree* was not used to enhance analysis. Then, there are two seemingly similar approaches to token normalization in NLP solving the same problem - called *lemmatization* and *stemming*. Lastly, there is several other discourse-specific techniques discussed at the end of this section.

2.1.1 Basic NLP terms

Here is a list of basic NLP terms used in this thesis:

- **Token** A single word. For example "nice", "ah", "right" and other.
- **N-gram** It is a contiguous subsequence of length n from a given sequence (sentence). 1-gram is token. 2-gram (also called *bi-gram*) is the most used one. Their main purpose is to help to create small, yet informative state space for other algorithms to use. When $n = 2$ or $n = 3$ there is usually enough data for algorithms to make an educated guess. Yet, at the same time, it is not too much data so the state space would be too sparse. After a certain n , the bigger n -grams are, the less additional information provide and make the state space more sparse.
For example all bi-grams from sentence "A dog ran back home." are ('A', 'dog'), ('dog', 'ran'), ('ran', 'back') and ('back', 'home')
- **Stop word** Words which are filtered out during the preprocessing process. There is no single list of these words, however, they are commonly words which occur in a sentence the most often
For example, "the", "a", "an", "that" and other.
- **Noun Phrase** It is a phrase in a sentence which plays a role of a noun. The head word¹⁷ is usually either noun, or pronoun.
For example (head words are highlighted), "I like to make burgers at home." or "My pets finally arrived,"
- **Verb Phrase** It is a part of a sentence which contains, both a verb, an object on which the verb directly (or indirectly) depends.
For example "Sam is going to the movie." or "Do you you think it is really your shirt?".
- **Prepositional phrase** It is a phrase that starts with a preposition and ends with a noun (or a pronoun). The words after preposition are called *object of a preposition*.
For example: "From what she said, this is her new job".

2.1.2 Part of Speech Tagger

In a nutshell, *Part of Speech* tagging is a clustering problem. The tagger assigns a word to a category (POS tag) based on its grammar-based similarity to words in the category, its usage and function in the sentence. However,

¹⁷Head word is a word in a phrase that all other words in the phrase tie their meaning to it

Table 2.1: Mapping important Universal tags to Penn Treebank tags

Category	Universal tag	Penn Treebank tags
Adjective	ADJ	JJ, JJR, JJS
Preposition	ADP	IN
Adverb	ADV	RB, RBR, RBS
Conjunction	CONJ	CC
Determiner	DET	DET
Interjection	INTJ	IN
Noun	NOUN	NN, NNS, NNP, NNPS
Pronoun	PRON	PP, PP\$
Verb	VERB	VB, VBD, VBG, VBN, VBP, VBZ, MD

there is usually more than one possible categories for each word. So the tagger handles this disambiguation by using additional information about local context around it.

Application of such a tagger are countless. Knowing the correct POS tag tells a purpose that each word has in a sentence. Thus, the tags can be used in speech recognition software to better understand sentences. Or chatbots can use it for the same reason. Another application is in *Speech-to-text*¹⁸ software. By knowing previous words and their POS tags determines list of possible POS tags. Based on the current sentence structure, it restricts list of possible POS tags for the next word. In combination with approximate spelling of the word, it makes task of finding the correct word easier and more accurate.

Basic principles of every POS tagger are rooted in grammar of a specific language. Every one is language-specific. In English, there is eight distinctive grammar categories (see Figure 2.1). Each category has several subcategories which better specify the type of information about the sentence. There is a universal set of POS tags. However, POS tagger used in this thesis works with *Penn Treebank tags* which can be easily mapped to universal tags (see Table 2.1).

Commonly, the most interesting categories from context point of view are **adjective**, **nouns** and **verbs**. In terms of sentence, verbs tend to express action. The verb itself can bear positive (e.g. help), negative (e.g. accuse) or neutral (e.g. go) meaning. To certain extend, their meaning is adjusted by adjectives in the sentence. Lastly, actors of this action is usually determined by nouns. From word ordering point of view, each language has its specific rules, thus no generic assumptions can be make (for more information on English language word ordering see section 3).

¹⁸*Speech-to-text* software is capable of transcription of spoken text into its written form.

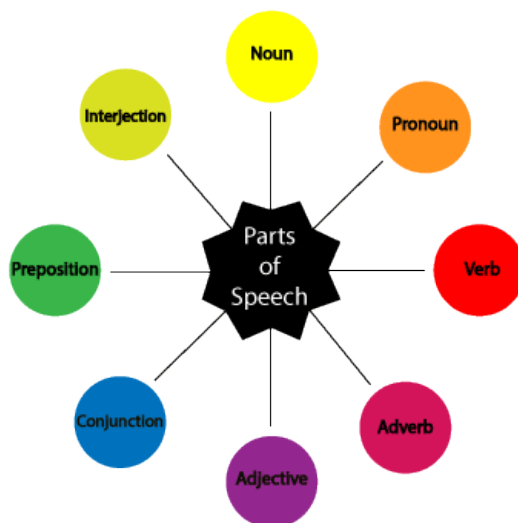


Figure 2.1: Eight basic categories of words used in English grammar

Based on the accuracy achieved by state of the art models (see section 2.1.2.4), it may appear that this task is solved from academic point of view. However, it is not the case according to [17]. Currently available models tend to be domain-specific and perform well only on text with similar writing style as the training set. Also, there is another important metric to be considered for POS taggers - *Sentence accuracy*. It states what portion of sentences is correctly tagged out of all sentences. In order for a sentence to be tagged correctly, every word in the sentence has to have the right POS tag. The available models at the moment achieve accuracy around 57 % in this metric while having overall accuracy up to 97 %. These results say that in almost every other sentence the tagger assigns at least one POS tag incorrectly. Achieving such a high per-word accuracy suggests that it mostly assigns exactly one tag incorrectly. Importance of this discrepancy in results depends on task the POS tagger helps out in the application.

In terms of implementation, there are three common ways of implementing a POS tagger. Each of them solves the ambiguity problem differently and offers specific trade-offs. In terms of benchmarks of POS taggers of English language, on one hand, the baseline is 90 %. On the other hand, current state of the art stochastic models perform at 97 %. To put it into perspective, it is estimated that a human can reach accuracy 97 % (in cooperation with somebody else, they can reach accuracy 100 % in rare cases). [18] [19] In terms of training set, Penn Treebank is popular among researchers, however, it contains incorrectly assigned tags (see Figure). The errors were not fixed in the dataset, but maintainers kept them there arguing that the current state simulates human behavior better.

Table 2.2: Example of rules for rule-based POS tagger

Rule	Description	Example
DET ADJ NN	S Determiner is followed by adjective and noun	a good job
NN VB	Noun is followed by a verb	Peter won
ADP DET NN	Preposition is followed by determiner and a noun	on the shelf
NN(<i>river</i>) -DET NN	Noun <i>river</i> is followed by another noun, not a determiner	river bank

2.1.2.1 Rule-based algorithm

The oldest approach is a **rule-based algorithm**. It uses a set of explicit rules defined by a user (see Table 2.2 for example of rules). This set is manually created based on grammar of the specified language. Ideally, they are created by a linguistic specialist. Rules itself can be either content-pattern rules or regular expressions. All of them together form a finite-state automata. Disambiguation is resolved by creating more robust rules. Due to their nature, they require significant amount of work for maintenance. On the other hand, they can be used as a cornerstone to build a syntax tree (more in section 2.1.3).

Being the simplest approach, its accuracy is not high. If the rules are descriptive enough, they can cover a great number of cases. Implementation of *Taggit* tagger by Green and Rubin in 1971 claim to achieve 77 % accuracy over the Brown corpus (see Section 2.5.1 for more information about Brown corpus). [20]

2.1.2.2 Transformation-based learning

More advanced approach is using *Transformation-based learning*. It is an extension of *rule-based algorithms* with following description:

- \mathcal{X} denotes space of all spaces. Dependent on type of POS tagger, a sample can be, either a single word, or a group of words (in language)
- \mathcal{C} denotes list of all possible POS tags (ADP, NN, DET, ...)
- $\mathcal{S} = \mathcal{X} \times \mathcal{C}$ represents state space
- \mathcal{T} is set of all samples from the used set
- π is a predicate defined on the space \mathcal{S}^+ representing logical part of the rule (i. e. $\pi(n) = w_{n-1} \wedge (w_n^1 \vee w_n^2)$)
- r is a pair of (π, c) , where $c \in \mathcal{C}$; A rule r means that if predicate π is true for a statement s , then the statement gets assigned tag c

2. THEORETICAL BACKGROUND

- s is current state which is defined by (x, c) , where x is current sample and c is currently assigned tag

Given a rule $r = (\pi, c)$ and state $s = (x, c')$, result of applying rule to the state is:

$$r(s) = \begin{cases} s & , if \pi(s) = true \\ (x, c') & , if \pi(s) = false \end{cases}$$

After each iteration, score for every rule is calculated using formula:

$$Score(r) = \sum_{s \in \mathcal{T}} score(r(s)) - \sum_{s \in \mathcal{T}} score(s)$$

where

$$score((x, c)) = \begin{cases} 1 & , if c = truth(x) \\ 0 & , if c \neq truth(x) \end{cases}$$

The goal of the algorithm is to minimize $Score(r)$ for each rule. As long as $Score(r) > \theta$, rule should be changed and new rules are derived from it. θ is an arbitrary small constant. One of the biggest advantages of this approach is that its results are dependent on its initial set of rules which is curated.

There are relevant works showing a relatively simple transformation-based algorithms perform comparatively well [21]. This algorithm starts off by calculating occurrences of each tag in train set. As input, it receives manually created set of rules. Then, it goes in rounds and creates *patches*. Each patch is a rule that specifies under what conditions one tag should be replaced by another one. Basically, it automatically generates rules for rule-based tagger. During measurements, the corpus was split accordingly - training set is 90 % and test set is 5 % of the corpus. They use stratified sampling from all genres to diminish overfitting problems. Stated results claim that after about 70 rounds the algorithm achieves only about 96 % accuracy on the Brown corpus.

2.1.2.3 Stochastic approach

The most advanced approach is using stochastic models for the task. Principally, it is an extension of *transformation-based learning*. The initial set of rules is modified using statistical models. In comparison to the previous approach, the initial set may not be a manually created set of rules. Instead, it may be a random set of rules which is gradually improved. Very popular approaches for the task were using Maximum Entropy model and Hidden Markov model (HMM) - both of them falling into the category of *supervised learning*.

The concept of Maximum Entropy Model tries to maximize entropy of a distribution within certain constraints. It works with a list of possible orders

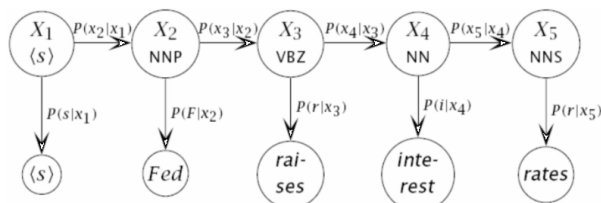


Figure 2.2: Example of HMM implementation of POS tagger for sentence "Fed raises interest rates."

of words h_i and their POS tags t_i . Thus, probability that the list of words h_i have POS tags t_i is:

$$p(h_i, t_i) = \pi \mu \prod_{j=1}^k \alpha_j^{f_j(h_i, t_i)}$$

In the formula, π is a positive normalization parameter which is set manually. On the contrary, α_j are positive parameters set by model itself to maximize entropy of the model. Then, $f_j(h_i, t_i)$ (known as features) represents the generated rule and its value is either 0 (when the rule is not true) or 1 (otherwise). Rules are generated using multiple templates like detection of suffixes (e.g. "-ing" or "-ed") or basic grammar rules. The output of the model for each word is tag with the highest probability. Often, the model returns the probability as well. Implementation of this model usually use *Beam search* to keep n most likely tags for each word to speed up the tagging process. [22]

In comparison, Hidden Markov model is based off Markov model with hidden states (see Figure 2.2). It is a tool for representing probability distribution over sequence of observations. In Markov model, all states are visible to the observer. In contrary, HMM contains several hidden states forming a Markov chain which directly influence output of the model. Consequently, each output state is visible. In MM, transition from x_n 'th state to state x_{n+1} 'th state is affected only by x_n 'th state. Whereas, in HMM the transition can be also affected by a latent state. In a simplified view, it can be understood as having Markov chain inside any state of main Markov chain. Detailed description of HMM is beyond scope of this work.

2.1.2.4 State of the art

At the moment of writing this thesis, *Stanford POS Tagger*¹⁹[23]. Software we will use in this thesis provides implementation of this model via *nltk.tag.stanford*. According to the paper, it uses *Cyclic Dependency Network* with surrounding words as features. In short, it generates directional graph during training phase. Edge between two nodes in one direction are denoted

¹⁹Source of Stanford POS tagger: <http://nlp.stanford.edu/software/tagger.shtml>

by probability of making a jump from node to another. The model uses maximum likelihood estimation to find the correct tag. Since the model uses a huge number of features (more than 460,000), it uses quadratic penalization to prevent overtraining. In this setup, the model achieves 55.31 % sentence accuracy.

However, its implementation is rather slow in comparison to *TextBlob*'s POS tagger. The second tagger uses *Perceptron* model, a type of neural network, with averaging weights which claims to provide even higher accuracy than NLTK's base implementation of POS tagger (98.8 % vs 94 % [24]) and about the same accuracy as Stanford's POS tagger. On top of that, it is significantly faster which is the key requirement when analyzing hundreds of pages of text.

2.1.3 Syntax tree

Building a syntax tree for a sentence is a popular way to get inside on relationship between words in the text. The algorithms split sentence into noun phrases (NP), verb phrases (VP) and prepositional phrases (PP). As a result of the analysis, the algorithm has to handle and solve a great number of ambiguity. Subsequently, the analysis can detect *subject* or *object* in the sentence. Just as POS tagger, it employs a knowledge of language grammar. Application of syntax tree analysis are for example chatbots or automatic answering tools.

There is several strategies to tackle this problem usually using context free grammar (see Table 2.3). The most fundamental approaches are *top-down* and *bottom-up* strategy to analyze a sentence (see Figure 2.3). Top-down strategy is expectation-based. It starts from root node S and works its way down. At every step, it picks a rule that can be added to the current syntax tree until there are no more rules to use on the tree and each word is in a leaf of the syntax tree (see Figure 2.3b). In contrary, bottom-up strategy starts applying rules to words in sentences to create nodes. These nodes can be then connected together using the same set of grammar rules. This process is repeated until nodes meet in root node S or no further rules can be applied (see Figure 2.3a).

Both strategies lead to a valid syntax tree. However, their results may not be equal (see Figure 2.3). Neither of the strategies usually find the correct result on their first try. Instead, both of them require a certain degree of backtracking and may end up checking the whole state space defined by their grammar. This process tends to be rather performance-demanding. Top-down strategy has a valid syntax tree at all times. However, it wastes a lot of time on checking trees which cannot match the input sentence. On the other hand,

This kind of analysis has several big drawbacks. Not only its results are highly ambiguous, it is also rather time demanding. Then, finding a correct syntax tree is highly dependent on set of grammar rules. There is a huge difference in used language between public speech, poetry, ordinary discourse

Table 2.3: Example of rules for syntax tree context free grammar

Rule	Description
$S \rightarrow NP VP$	S (sentence) starts with NP (noun phrase) which is followed by VP (verb phrase)
$NP \rightarrow DET N$	Noun phrase (NP) consists of a determiner (DET) and a noun (N).
$NP VP \rightarrow DET ADV$ $NN VB.$	Noun phrase (NP) followed by a verb phrase (VP) consists of a determiner (DET), an adjective (ADJ), a noun (NN) and a verb (VB).

among a small group of people or any other literature piece. Moreover, individuals with different background, social status and age tend to use distinct set of grammar rules. Supposedly, there is countless unknown factors influencing one's use of language. Therefore, the task of creating one universal grammar capable of accommodate all the above-mentioned factors is an extremely difficult. On top of that, there can be multiple plausible interpretations of the sentence so it is hard to pick the correct one (even for a human being).

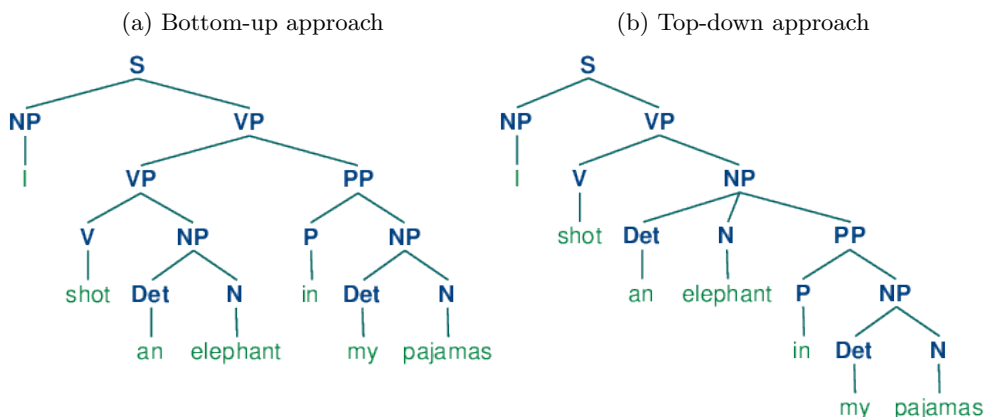
At the moment of writing this thesis, the state of the art model created in Stanford [25] uses combination of transition-based approach and neural networks. The parse occurs in one linear scan over the words of a sentence. At every step, it maintains partial parse and other relevant information about its progress. Transition between states is controlled by a neural network. The model is trained on manually annotated dataset provided by Stanford University which contains more than 250,000 words. The model excels in accuracy on Penn Treebank corpus with accuracy of 92 %. At the same time, it excels in performance as it is able to parse approximately 1000 sentences per second while other solutions with comparable accuracy can parse only up to 600 sentences/second.

However, the state of the art model is trained on samples from weblogs, newsgroups, emails, reviews and question-answers. Due to reasons mentioned above, using it on analysis of discourse at the Court could provide misleading results. Since there is no syntax tree parser available for law-oriented English language and creating a specific model for this purpose is outside of scope of this thesis, syntax tree analysis is not used.

2.1.4 Lemmatization and Stemming

These popular concepts try to solve the same issue in seemingly the same way. However, their approach to tackle the issue of *token normalization* is fundamentally different. On top of that, their results usually are rather distinct. Process of *token normalization* solves another huge issue in data analysis -

Figure 2.3: Example of syntax trees for sentence "I shot an elephant in my pajamas"



data sparsity. One particular word tends to have several different inflected forms (e.g. "wish/wishes" or "good/better/best") in particular languages. This property negatively influences performance of some NLP techniques by making the state space for them rather sparse. Therefore, these metrics prefer to work with normalized tokens. In theory, a result of token normalization of any two inflected forms of the same word should return the same normalized token. Unfortunately, this is not the case. Special cases for the token normalization are words with irregular forms like verbs with irregular past tense. Each strategy addresses these challenges differently and has their own pros and cons. Thus, this thesis uses both approaches. More information on their usage is available in appropriate chapters below.

2.1.4.1 Stemming

In linguistics, stemming is a process of reducing a word to its *stem*[26]. A stem may not be a morphological root of the word or a word at all. At the moment of writing this thesis, there are two major stemming algorithms - *Porter* and *Lancaster* - both of them are rule-based (see Table 2.4). The later mentioned is faster in terms of computational speed. However, it often produces very obfuscated stems. Thus, Porter stemmer is considered the state of art algorithm.

The newest iteration of **Porter stemming** algorithm is written in *Snowball* programming language²⁰ and is ported to other programming languages²¹. The first version of algorithm was introduced in the 1980's. Nowadays, the

²⁰Snowball is a small string processing language designed for creating stemming algorithms

²¹In NLTK, it is available in package *nltk.stem.porter*

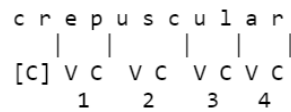
Figure 2.4: Measure m calculated on word *crepuscular*

Table 2.4: Example of rules for a stemmer

Rule	Example
$(m > 0)SSES \rightarrow SS$	caresses \rightarrow caress
$(m > 0)IES \rightarrow I$	ponies \rightarrow poni ties \rightarrow ti
$(\text{VERB})ED \rightarrow$	morphed \rightarrow morph
$(m > 0)IED \rightarrow y$	bullied \rightarrow bully

algorithm itself is divided into 5 steps which are further broken down into smaller ones. [27] In each pass of the algorithm, a word is processed from right to left. The first step handles plurality and past participles by detecting specific suffixes in three substeps. The later steps provide more normalization rules based on suffixes and measure m from Snowball. The measure m is defined as (see Figure 2.4 to see how the measure $m = 4$ is calculated for word "crepuscular"):

$$[C](VC)^m[V]$$

where

- **C** represents a sequence of one or more *consonants*. A consonant is defined as a letter other than *A, E, I, O* or *U* and other than *Y* preceded by a consonant.
- **V** is a sequence of one or more letters which are not consonants (such a letter is called a vowel).

The rules in later steps of the algorithm contain many types of rules combined together into larger conditions. The rules can for example check with which letter the stem ends or starts, length of measure m or whether the stem contains a vowel.

2.1.4.2 Lemmatization

In computational linguistics, lemmatization is a process of finding a *lemma* for a specific word. [26] In comparison to a stem, lemma is a base form of a word. Inflecting words (like saw, see, seen) share common lemma (see).

Also, lemmatizers tend to require more information about the word in order to detect its proper lemma. On top of the word itself, they may take advantage of several preceding words and/or their POS tags. Together with a large vocabulary, lemmatization produces more relevant results. However, this approach cannot handle unknown words, since it is highly dependent on words in its vocabulary. Also, it is slower in terms of computational speed than conventional stemmers.

Lemmatizer implemented in NLTK uses WordNet vocabulary (introduced in Section 2.5.1). Its implementation uses only the word itself without any additional information. In case the word is not found in its vocabulary, the algorithm returns the word itself. Otherwise, it returns its lemma.

2.1.5 Synonyms

A **synonym** is a word or phrase which means exactly the same or has nearly the same meaning as another word or phrase in the same language. Their usage can notably help to reduce data sparsity during analysis by replacing synonyms with a common token. There are is another term closely connected to synonyms - *related words*. Any two words may not share the same meaning, however, they do are related (e.g. "government" - "politicians" or "art" - "paintings"). In general, there are two fundamentally different approaches to find a suitable synonym described in the following subsections. [1]

2.1.5.1 Thesaurus-based algorithms

The more straight-forward approach uses large vocabularies with defined relationships between words in order to find the correct synonyms. Sadly, there is several apparent drawback connected to this approach. To name a few, the biggest issue is the size of vocabulary needed for this analysis. Since the algorithms can only work with words available in their vocabularies, unknown unexpected words tend to lead to unwanted and, potentially useless, results. Another disadvantage is tied to adjectives and verbs. In dictionaries, they use to have less structure information about additional relationship relevant to them. Therefore, the algorithms do find less synonyms for other words than nouns. [1]. On top of it, the vocabularies do not usually contain words from all grammar categories (especially for conjunction). This makes them, by definition, useless for them. Lastly, there is not a thesaurus for every language rendering the algorithms useless for those.

When working with synonyms, **WordNet 3.0**[28] is the go-to dataset among researchers (see section 2.5.1). For simplistic analysis, NLTK offers access to WordNet's list of synonyms with their POS tags. Here is a sample output for word "good" as an adjective (for sake of simplicity, duplicities were removed in output)²²:

²²Format of output records is '{synonym}.{grammar category}.{order of meaning}'

```
>> from nltk.corpus import wordnet as wn
>> wn.synsets('good', pos='s')

[Synset('good.a.01'), Synset('full.s.06'),
Synset('estimable.s.02'), Synset('beneficial.s.01'),
Synset('adept.s.01'), Synset('dear.s.02'),
Synset('dependable.s.04'), Synset('effective.s.04')]
```

Unfortunately, this simple interface does not provide any information denoting to what extent meaning of words in result match the meaning of input word. This can lead to misleading results. Namely, when looking for synonyms to word "pet" one of the provided words is "positron_emission_tomography" which is clearly wrong. The other suggested synonym is 'darling' which also is a relevant option. However, there is a missing word which can be considered (to some extent) a synonym as well - 'animal'. Clearly, while synsets provide an easy and fast access to synonyms, their results are very rudimentary

Fortunately, there are two general extensions to thesaurus-based approach that helped to produce a measure of similarity between words. Being able to control this measure enables necessary filtration of distant words²³). However, they are computationally more demanding.

The first extension, *path-based similarity*, builds a dependency tree (an oriented graph) describing relations between terms. An edge represents a relationship between two terms and has the same length (1). This implies that the higher a word in this hierarchy is, the more generic meaning it has. On this structure, we can define following metrics based on path length ($pathlen(n_1, n_2)$ denotes the number of edges in the shortest path between nodes n_1 and n_2 in graph plus 1):

$$simpath(c_1, c_2) = \frac{1}{pathlen(c_1, c_2)}$$

$$wordsim(w_1, w_2) = max(simpath(w_1, w_2))$$

Apparently, these metrics by themselves are not sufficient or reliable enough. For example, $wordsim(budget, currency) = \frac{1}{5}$ and $wordsim(budget, standard) = \frac{1}{5}$. The same irregularity occurs with term *medium of exchange* for which terms *coinage* and *scale* have the same similarity measure. This issue is caused by the uniform length of every edge. Because of it, their similarity is the same despite being in different branches of the tree.

The second extension, *information content similarity*, solves the issue with uniform lengths of edges. At first, let's define probability $P(c)$ and information content $IC(c)$:

²³Words having a rather big difference between their meaning

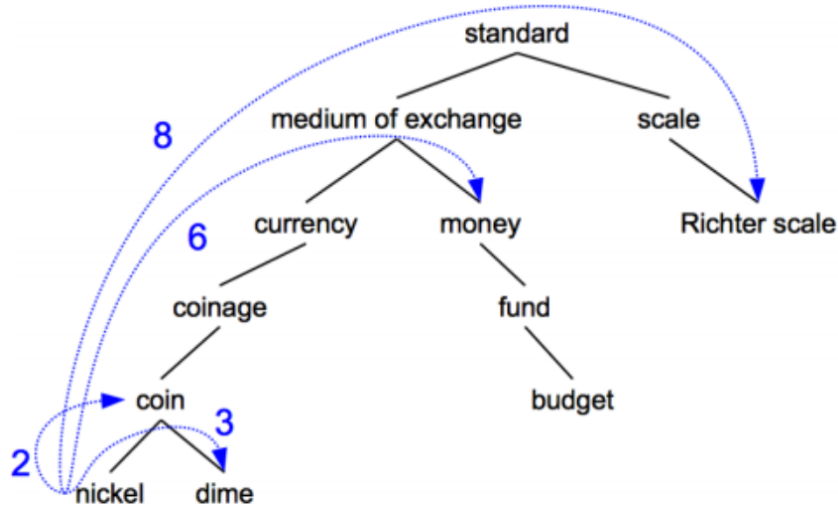


Figure 2.5: Example of dependency tree for path-based similarity with $pathlen(nickel, coin) = 2$, $pathlen(nickel, money) = 6$ and $pathlen(nickel, Richterscale) = 8$ [1]

$$P(c) = \frac{\sum_{w \in words(c)} count(w)}{N}$$

$$IC(c) = -\ln P(c)$$

Random variable $P(c)$ denotes probability that a randomly selected word in a corpus is an instance of concept c . Concept c is a node in the dependency tree and a word is an instance of a concept when it is its child node (for example "grotto" is instance of concept "cave" 2.6). It is defined that every word is member of root node $P(root) = 1$ (thus, creating a tree). Based on its definition, the lower a word is in hierarchy, the lower probability $P(c)$ it has. Then, information content $IC(c)$ defines the amount of information contained in a word. The lower parts of the hierarchy have higher IC and, thus, bear more concrete information. Lastly, there is another metric, *Lowest common subsummer* (LCS), that is known in theory of graph as *lowest common ancestor*. For nodes w_1 and w_2 , LCS is w_3 which is the lowest node in hierarchy (the furthest from the root node), yet its descendants still are both w_1 and w_2 (for example, $LCS(natural - elevation, coast) = geological - formation$ in 2.6).

These extensions are used in *Dekang Lin* algorithm[29] to address the issues with simple thesaurus algorithms. The intuition says that similarity between words w_1 and w_2 is defined by not only their common features. So,

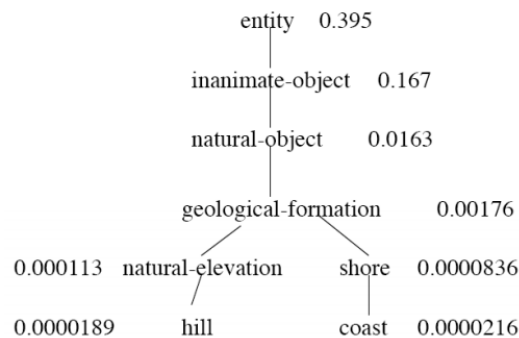


Figure 2.6: Example of WordNet hierarchy augmented with probability (pictured is only a part of the tree) $P(c)$ [1]

the similarity between two tokens according to Dekang’s proposition is:

$$\text{sim}_{Lin}(w_1, w_2) \propto \frac{IC(\text{common}(w_1, w_2))}{IC(\text{description}(w_1, w_2))}$$

The commonality is expressed by formula:

$$IC(\text{common}(w_1, w_2)) = 2\ln P(LCS(w_1, w_2))$$

And description of words w_1 and w_2 is defined as:

$$IC(\text{description}(w_1, w_2)) = \ln P(w_1) + \ln P(w_2)$$

Dekang Lin algorithm is implemented in NLTK in a dedicated package `nlk.corpus.len_thesaurus`:

```
>> from nltk.corpus import lin_thesaurus as lin
>> print(sorted(lin.scored_synonyms('home',
                                   fileid='simN.lsp'),
               key=lambda row: row[1],
               reverse=True)[:5])

[('house', 0.254551), ('apartment', 0.214802),
 ('building', 0.177342), ('hotel', 0.172639),
 ('residence', 0.161917)]
```

In comparison to previously mentioned thesaurus-based algorithms, these results are clearly more relevant and richer in their content. The original paper was written and validated on Wordnet 1.5 so its results are not relevant at the moment of writing this thesis any more. However, empirical testing was carried out with the purpose of proving its validity for this thesis. In the end, the provided results appeared to be more relevant than output provided by

other algorithms. Unfortunately, its implementation in NLTK contains only dictionaries for adjectives, nouns and verbs. Still, other implemented solutions do not provide dictionaries for other grammar categories either. Plus, even though $sim_{Lin}(w_1, w_2) \in \langle 0, 1 \rangle$, it is not guaranteed that every word has a synonym with $sim_{Lin}(w_1, w_2) = 1$. So it is harder to determine how many words from result are relevant enough. This problem is tackled in Chapter 3.

2.1.5.2 Distributional-based algorithms

These algorithms use distributional context around word to find synonyms. Instead using the word itself, a word is defined by a vector of features. These features usually describe frequency of occurrence of various words around the specified word. The algorithm then searches its database to find the closest match using a similarity measure (e.g. cosine distance or Jaccard distance). In comparison to thesarus-based approaches, they can handle unknown words and work well with adjective and verbs. A more detailed explanation of these algorithms is out of scope of this work.

2.1.6 Discourse analysis techniques

As mentioned previously, discourse analysis require special treatment from a NLP point of view. There are several metrics which apply to discourse only or have special meaning in this type of analysis. They focus on unveiling tendencies of participants in the discourse.

A very basic metric is **number of turns** (turns). One turn is a consistent portion of discourse said by one person without being interrupted by anybody else. There is no time restriction on its length. However, the number of tokens in one turn is also a relevant information. When a person is interrupted by anybody else and later finishes their thought, this situation is often considered as two separate turns of that person. The number of turns can reveal several interesting information about the speaker. Having a high number of turns with medium to higher length during a discourse can indicate higher initiative on the speaker's and vice versa. On the other hand, several frequent, consequent and short turns may suggest that the participant is submissive and easy to interrupt.

Another relevant metric describing the inner dynamics of the group during a discourse is **follow ratio**. The metric calculates how often is who followed using formula:

$$follow(A, B) = \frac{\# \text{ of times person } A \text{ takes a turn after } B}{\# \text{ of times person } A \text{ takes a turn after anybody}}$$

The value is in range $\langle 0, 1 \rangle$. Higher values may indicate a one-sided relationship between two participants in the dialog. Due to nature of the definition, this relationship is oriented and not bidirectional. Thus, it may unveil

signs of one-sided personal disputes or preferences between certain actors of the dialog.

Other advance metrics rely on **noun phrases** and their extraction. This metric is not discourse-specific, however, it is incredibly relevant to this area. Because it sheds light on the topic of turns in the dialog. However, detecting the whole noun phrase properly is rather complex task similar to building a syntax tree (see Section 2.1.3). Also, there may be adjectives modifying its meaning in the noun phrase. Thus, considering only nouns themselves as noun phrases can be considered a good-enough approximation of the problem. Its additional modification by adjective is supplied by additional sentiment analysis.

Building on this, there is a metric called **topic-chain index** (TCI) which uses extracted noun phrases. The metric itself was devised by Broadwell et al, [30]. It calculates the difference between the first and the last occurrence of the same noun phrase in the discourse. The noun phrase might be introduced by one person and its last mention may be uttered by a different one. The length of the chain represents the time-frame when object (or subject) of the noun phrase is most-likely relevant to the discussion. Study of the length and content of the chains provides clues on direction of the discourse.

Lastly, there is one specific metric which - number of **asked questions**. In nutshell, it counts number of question marks (consequent question marks are counted only once) in all turns per actor. Their number and placement describe actor's effort to either gain additional information or to state a question to suggest their idea to others. Asking only very few or no question at all during the discourse may a be rather strong signal suggestive of specific client's behavior.

2.2 Sentiment Analysis

Sentiment analysis (some literature refers to it as *opinion mining*) refers to a process of identification and extraction of subjective information in source text. Generally speaking, this process attempts to interpret a sample of text and detect the subject's attitude. This analysis is performed by a *sentiment model* that takes text as its input and returns sentiment class the text fits the best. Some models also know the level of certainty for their provided answer. The basic sentiment classes that most models can detect are positive, negative and neutral.

Due to nature of this task, sentiment classification is a subjective task with somewhat vaguely defined correct answers. Hence, the accuracy of models is measured on datasets with subjectively ranked samples. It is an open question whether it is possible to create a sentiment classification model having 100 % accuracy. Given that it is a tough task to get many people to agree on sentiment of larger number of sentences, it is doubtful that any sentiment

model will be able to. [31]. Therefore, it is more favorable to have reliable model giving stable results for all classes, than having model with higher average accuracy skewed toward one (or two) class.

Once created, sentiment models have wide range of applications. Each application puts different expectations on speed, accuracy of presented results and source of data. To name only a few interesting use cases for them:

- **Opinion extraction on social media** This is probably the biggest use case and driving force for sentiment analysis at the moment. Usually, text on social media is rather short which makes the machine analysis harder (see section 2.2.1.2). Typically, sentiment detection is performed on social interactions containing certain hash tags or belonging to a specific piece of content (e. g. posts under shared photo, video or status). Each social interaction may include additional contextual elements specific to the platform (e. g. number of likes or replies on Facebook, or number of retweets on Twitter). Advanced sentiment models utilize all these information to provide more accurate classification of text.

Results of this type of analysis are interesting for several groups of users, for example:

- **Marketers** see a huge opportunity to understand the feedback about their product. Extracting opinions manually is time-consuming or even impossible task. Modern technology allows them to collect customer’s interaction from any relevant social network (Facebook, Twitter, Instagram or other) and analyze them almost in real time. Therefore, they prefer estimated results with certain degree of error provided in real time to accurate results available after significant delay.
 - **Public opinion analysts** study market trends in general and try to predict public opinion on any matter. Instead of focusing on one particular brand or product, they are watching wide range of sources of public opinions. While their mission is to provide relevant information about public opinion on a matter, they sacrifice utmost accuracy to analysis of more data sources.
 - **Political analysts** are *public opinion analysts* who specialize in studying society opinion on political parties. In their effort, they do not use only data from social media. There are other interesting sources like blogs or dedicated written materials which provide useful context to on-line discussions. Due to delicate nature of their analysis, they require more reliable models .
- **Analysis of opinionated materials** Another big source of opinions are specialized texts which express author’s opinion on the discussed matter. Such an article contains author’s opinion backed with an extensive

explanation. This makes their analysis more straightforward once their domain is known (see section 2.2.1.1). Typically, sentiment model classifies the text as whole. In special cases, texts can be split into smaller parts to provide detailed view on author's perspective.

Currently, there are two popular types of opinionated materials that researchers analyze:

- **Movie reviews** are rather popular for sentiment analysis. Researchers tend to use them as training sets for their models (either *IMDB movie reviews* or *Rotten Tomatoes movie reviews*). One of the reason for the huge popularity is their availability. Movie sites allow their users to write their own reviews for movies, so there is no shortage of ranked reviews available on-line. The range of final estimation depends on complexity of model. In most cases, the basic models estimate only sentiment class of a whole review. Advanced models can also estimate the user-assigned score. Overall accuracy of those models reaches up to 99 % [32].
- **Political materials** Another interesting area for researchers are political materials written by authors with a certain political view. This area is particularly hard to study, because of its nature. Even though there are many data sources (e. g. *Amici curiae briefs*²⁴), it is hard to work with a legal language (see section 2.2.1.1). Also, while author's political view is known, there are other interesting questions about these materials - for example to what degree two texts written with the same political view in mind share the same arguments.
- **Discourse analysis** This field of linguistic analysis has been recently popularized and is gaining traction. Technically speaking, it merges two different fields together - computational linguistics and psychology. In short, it uses elements of computational linguistics and apply them to psychology concepts. Result of the analysis attempts to infer participants position toward the discussed topic (for more information on discourse analysis see section 2.1.6).

There is a couple of different types and forms of discourse. There can be a verbal conversation taking place at one physical place. Another form of discourse can be an interaction on social media which occurs in reference to the same piece of content (e. g. conversation under Facebook post or tweet replies to one tweet). Each form influences the content and rendering of the discourse.

²⁴For example available at <http://www.psci.unt.edu/~pmcollins/data.htm>

Furthermore, there is a number of discourse types. Each type having a distinct set of rules defined by society or law. Here is a list of interesting types from research perspective:

- **Ordinary dialog** between several participants poses a great example of the most common analysis (e. g. dialog between friend on street). Any number of participants engages in an open conversation on a certain topic. The used language may be informal and may contain a lot of sarcasm (see section 2.2.1.3 on how to deal with it). All in all, performing sentiment analysis on this type of discourse is a hard, but expectations on its accuracy tend not to be high. From researchers point of view, their analysis is not very attractive, because of very little relevant information extracted from it in form of participant’s opinions.
- **Operator-customer dialog** that occurs between operator (in support center or selling merchandise) and a client. In this type of discourse, analysis studies customer’s behavior toward operator. The dialog is mostly semi-formal with almost no sarcasm and usually revolves around predefined topic. Plus, the dialog occurs over the phone so there is almost no body language involved in the discourse. These features make it easier to examine it with relatively great accuracy. Since companies always try to improve their customer service and upsell methods.
- **Legal dialog** that occurs between multiple participants (usually justices, lawyers or government officials) discussing a legal issue (or a topic related to law) in official manner. Given the strict form of the dialog defined implicitly by its circumstances, its analysis is more streamlined. However, participants tend to have long monologue speeches with very long sentences. This feature is given by nature of the law language and its effort to be comprehensive at all costs. In terms of accuracy, there is a little bit of pressure on researchers that any results should be reliable. Therefore, it is more desirable to know limits of respective technologies and adjust the expectations accordingly, than have overtrained model with high number of false positives and negatives.

Since this thesis focuses on discourse analysis of legal dialogs at the courthouse, it is hard find a suitable sentiment model. Therefore, a significant amount of effort is exercised to create a fitting sentiment model (described in Chapter 3) that can be used in analysis of the Oral Argument (described in Chapter 4).

2.2.1 Challenges

Clearly, creating a sentiment model is a complex task with countless of pitfalls. As mentioned in previous section, each type of sentiment analysis resembles different set of expectations on it. One of the biggest problems of sentiment models are ambiguous words and domain dependent. Another problem arises from the level of detail expected from the model. Examples of other challenges are given below.

2.2.1.1 Domain and ambiguity

While both problems represent a slightly different issue, they share a common ground in terms of usage words. Both problems influence participant's vocabulary depending on the present situation in a way. An **ambiguous** word can have multiple meanings in the sentence and it is not possible to pick the correct one at the first glance. In contract, when talking about certain **domain**, participants use specific vocabulary that may contain words not used in other context. x Every language contains ambiguous words. There is no specific grammar category they belong to. In fact, many words are in multiple grammar categories and the correct one is determined based on number of external factors (e. g. *check* or *run*). Or a word may have multiple different meanings within the same grammar category. A simple example of ambiguity in a sentence is:

"Let's meet next to the bank".

From this one sentence, it is not possible to decide, whether the bank is an institution or a river bank. This kind of ambiguity is impossible resolve without any additional information. Useful context clues may be presented in surrounding sentences in form of other words giving the original term specific meaning:

"Let's meet next to the bank. And do not forget to bring your boat!"

While the following sentence provides clues aiming toward *bank* being a river bank, it may as well be a meeting point for an upcoming trip to a river. There is no general rule saying how far around the sentence to look for useful clues. Likewise there are no general rules that help to identify the clues either. On top of that, there may be various idioms or specific phrases established by the circumstances of the discourse (e. g. talking in movie quotes when discussing a quality of movie). All this make their detection and handling clues to disambiguate words a complex problem.

Every topic has its own jargon, domain-dependent words, with very specific expressions which may have different meaning depending on several aspects. Here is a great example of domain-dependent vocabulary:

” You’re killing it!”

Clearly, the expression can have, both positive, and negative connotation. When talking about somebody’s achievements in any field, it has very positive meaning. In contrast, when discussing somebody’s life or in being in situation when one’s life is in danger, it has very negative meaning. Apparently, there is no middle ground so getting the correct domain in this sentence matters a lot. To make the task even harder, the domain of the conversation can abruptly change.

2.2.1.2 Level of detail

Opinion mining can be understood as a complex process that is performed on text as a whole, or on its parts. Also, there is a difference between sentiment analysis carried out on a large article, its individual sentences or on a tweet²⁵. In particular, they differ in access to additional information about the examined text. Given the extreme variance in expectations on each type of analysis, it is an elaborate task to create a versatile model.

In particular, there are three dominant types of sentiment analysis, each requiring a different approach to the classification (see section 2.2.2 for relevant information on state of the art accuracy):

- **Document-level sentiment analysis** is considered to be the easiest one to perform. A document in this representation is a lengthy text written by one or a group of authors in structured way on one topic (not as a transcript of discourse). Depending on its length, commonly, there is enough data to make reasonable accurate estimation of its sentiment. Occasionally, documents are split into smaller sections (paragraphs) and then classify them separately in context of the document. Given fixed domain of analyzes texts, their accuracy is rather decent.
- **Sentence-level sentiment analysis** is more granular version of *document-level* analysis that requires substantially better work with every word. Due to small number of words, it is harder to extract features necessary to detect any sentiment. The assumption is that each sentence contains only one opinion.
- **Social-post-level sentiment analysis** utilizes techniques for *sentence-level analysis* and add additional information from social media (e. g. likes for each comment on Facebook). Also, models have to deal with more informal language full of *emoji* and links to other sites (see Figure 2.7). All things considered, their sentiment classification is a significantly different process. In addition to the pure text, it also has to consider

²⁵Tweet is a short (maximum length is 140 characters) message posted on social network Twitter.



Figure 2.7: Example of Facebook post on *Arrow* page captured on Dec 17, 2016 at 10:20PM CET

multiple additional data to estimate the sentiment properly. Also, there is a difference between extracting people’s overall opinion on the post, or their individual opinions. For the overall opinion analysis may use additional information about *likes* and *shares* (on top of all comments with their *likes*).

2.2.1.3 Other challenges

There is several seemingly smaller challenges which should be addressed (or at least considered) in sentiment models. In spite of them taking up only a small portion of the actual discourse, they significantly affect outcome of the analysis.

Namely, **sarcasm detection** influences polarity of sentence, even though sarcasm is rarely present in them. *Sarcasm* is a bitter remark that while appearing to have one polarity, its true meaning is the exact opposite. Due to its nature, sarcasm detection adds to subjectiveness of sentiment classification task. In some texts, sarcasm is signaled by putting the sarcastic part between double quotes.

Next example is **sentence length** that complicates its parsing process. Longer sentences may contain several distinct subsentences which affect each other. Therefore, it is harder to identify true meaning of sentence. At the

same time, short sentences do not contain enough information for any further analysis. This is in particular relevant to analysis of posts on social media.

Another great example is **negation detection** which requires deeper knowledge of relationship between words in sentence. When negation is detected in sentence, it is important to know which sentiment-bearing words it influences. One of the popular approaches is to consider all following words "affected" until another negation is detected. However, this approximation of problem fails if the sentence contains complex subsentences. Another popular approach is to predefine range of influence and mark only following n words after negation as negated. Either way, their detection is particularly subjective task, so it is hard to compare results of both approaches reliable.

Last but not least, **language vocabulary** presents a problem from linguistic perspective. It is a fluid ever-changing body that is hard to freeze in time. Straightaway, languages contain countless of idioms (e. g. *hang out* vs *hang on*) which can be detected using their dictionary. Furthermore, languages allow and even support making up new words spontaneously (e. g. "friend me" - connect with me on Facebook). Guessing true meaning of such words is a problematic task with no concrete solution. The one apparent fix for this situation is keeping the model's vocabularies periodically updated.

2.2.2 Approaches and State of the art

Because it is a rapidly changing field, it is difficult to capture state of the art approaches. Although, as explained in section 2.2.1.1, each level of sentiment analysis takes slightly different approach, they share various bits and pieces in process. Sentiment classification, as any other data analysis procedure, can be divided into two big parts - *feature extraction* and *modeling*. *Data preparation* phase differs for every dataset so much that it is not usually covered by any standardized approach.

While feature extraction is to certain degree common for all models (each model usually uses a subset of them), their underlying stochastic model is different. It has been an universally accepted approach by researchers that only stochastic methods are useful in this field.

2.2.2.1 Feature extraction

Extracted features compose of three main elements (types) - word tokens (or their n-grams), various language dictionaries and POS tags. Although, there are other features ready to be extracted (like punctuation), the main elements convey the most sentiment clues. From a sentence structure point of view, there are two main approaches to feature extraction - *bag of words* or *syntax tree*²⁶. The selected extraction approach influences the amount of information

²⁶There can be any other interpretation of advanced NLP knowledge about structure of the sentence

available to the model for each feature. Additionally, extracted features may be represented in multiple formats, like pure word tokens or points in vector space.

Bag of words is a very popular model mainly because of its simplicity. Instead of focusing too much on structure of the sentence, it only provides word tokens in arbitrary order. Unfortunately, this approach in its pure form presents several issues. The most obvious is its inability to distinguish between following two sentences because of no knowledge of word ordering.

Bus is better than train.

vs

Train is better than bus.

As opposed to bag of words, **syntax tree** approach (see section 2.1.3) considers relationship between every word in the sentence. Under the hood, this approach tries to understand dynamics in the sentence. As explained in section above, this task is difficult to carry out with reasonable accuracy.

When comparing both approaches [31], syntax tree approach outperforms bag of words, if carried out properly. For bag of words, any token is taken as-is without any additional contextual information. Therefore, models using it require significantly larger training set to learn necessary patterns. On the contrary, in syntax tree approach tokens already bear basic information about their meaning in context of the sentence. Hence, significantly smaller training set is required to learn approximately same patterns. Considering the complexity of creating a computational model that understands a sentence, the most favorable approach at the moment is a mixture of both.

With this in mind, **n-grams** (see section 2.1.1) offer a great compromise. Since this type of feature has information about words in its near surroundings. Some of those words may provide useful information to disambiguate meaning of the analyzed word (or whole phrase). However, size of the n-gram matters a lot. Short n-grams (bi-grams or tri-grams) provide only rudimentary information. These n-grams may be useful when analyzing texts with simple English. Any complex word structures are going to slip away. But simple and powerful phrases like *"very good performance"* or *"nicely cleaned table"* are still addressed. On the other hand, longer n-grams (e. g. four-gram or longer) capture a decent chunk of sentence for further evidence. For example a phrase

"fast, yet relatively hated, car"

can be correctly interpreted due to longer n-grams. Unfortunately, this snapshot might carry over a great amount of noise to the analysis too. For example a phrase

"great singer who had terrible performance"

contains two separate opinions which influence the overall sentiment of the sentence. Also, longer n-grams introduce a great deal of sparsity to feature spaces. Hence, the n-gram has to be additionally parsed by advanced NLP technique to understand its meaning.

Another set of features takes advantage of **various dictionaries** (e. g. *General Inquirer* or *Subjectivity Lexicon* mentioned in Section 2.5). Each dictionary provides additional facts about every word. Even though a word is not included in a dictionary, this information is almost as important as if it was there. For example, word "painful" is definitely not in list of positive adjectives. Yet, knowing this indicates that the word is either negative or not an adjective at all. This ambiguity can be removed by POS tags. However, there are several serious pitfalls to using dictionaries. Most importantly, there are not relevant dictionaries available for every language. Obviously their availability is dependent on amount of research being done on the language. Another problem is that not all languages share the same grammatical devices. Therefore, using vocabulary to look for them would fail (e. g. no gender forms in the Turkish language[33]). On top of these issues, there is a problem with *word morphology*. Of course the obvious way to solve it is performing word normalization (see section 2.1.2), but there are caveats as well. The implemented normalization should match dictionary's normalization. However, this makes merging multiple dictionaries together a complex task.

Further, *Part-of-speech tags* for words in the sentence provide clues to disambiguate their true meaning. Also, their position helps to create a simple structure of the sentence. Instead of applying advanced analytics, looking at their order in sentence may provide enough information to make an educated decision. For more information how POS tagging works see section 2.1.2.

Lastly, feature representation, *word embedding*, influences manipulation with them in stochastic models. Currently, there are two basic forms of representation:

- **Vector space model** representation transform every word to a vector (see Figure 2.8²⁷). The transformation process is customized to the current needs of model and preserves internal logic and relationship between words. Also, there is no restriction on range of values. Due to mathematical properties, vectors then may be added together, subtracted from each other or compared to each other.
- **Token** is the word (or phrase) itself in its written form (e. g. "sun", "home" or "go out"). Admittedly, tokens can be looked at as a special case of *vector space model* with vectors of length N , where N is number of unique words in dataset. Then, every word is represented by sequence

²⁷Image source: http://www.marekrei.com/blog/wp-content/uploads/2014/10/vector_space_model_multilingual.png

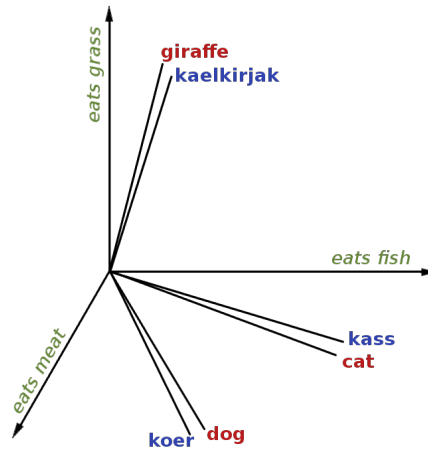


Figure 2.8: Example of one possible vector space representation of animals

of zeros and a single one uniquely denoting the word in question. However, this use-case do not allows any mathematical vectors performed on them.

Considering their merits, it is not possible to say which representation is better. Definition of *vector space model* is a complex model requiring domain knowledge of the dataset and used words. In some cases, it is impossible to create a valid transformation for every word in dictionary to uphold internal validity of mathematical operations. On the other hand, vector space model allows usage of algorithms like *Word2vec* created by Tomas Mikolov at Google in 2013[34]. This algorithm aims to automate generate vector representation of any word using neural networks. In order to achieve high accuracy, the algorithm should be train on hundreds of millions of words. In return, the algorithm preserves with similar content and sentiment closely together. According to [34] it achieves accuracy of predicting correct vector representation of an unknown word as high as 59 % trained on 100 billion words.

2.2.2.2 Modeling

As presented at the beginning of this section, there is a variety of specialized use-cases for sentiment analysis with different specific requirements. So it is hard to find state of the art approach for each use-case. Instead, this section focuses on a brief overview how sentiment models are typically created and their most accurate implementations.

The basic stochastic model uses **Bayesian Learning** with various hypo-

2. THEORETICAL BACKGROUND

thesis. It is based on *Bayes' Theorem*:

$$p(C_k|x) = \frac{p(C_k) * p(x|C_k)}{p(x)}$$

where:

- x is a sample
- C_k is a classification class k
- $p(C_k|x)$ is **posterior** probability representing probability that classification class should be C_k given the sample is x
- $p(C_k)$ is **prior** probability representing probability that classification class C_k occurs
- $p(x|C_k)$ is probability (or **likelihood**) that sample x occurs given that classification class is C_k
- $p(x)$ is probability that a sample x occurs

The oversimplified approach, *Naive Bayes Classifier*, assumes that all samples (or events) occur independently (with the same probability). Given hypothesis that the process resembles *Maximum A Posteriori* (MAP) estimation for $P(y)$, the class y is determined by formula:[35]

$$\hat{y} = \operatorname{argmax}_y P(y) \prod_{i=1}^n P(x_i|y)$$

According to recent measurements [36], this model trained on movie reviews dataset can achieve accuracy up to 81 % using *bag of words*. Their work focused on classification entire reviews into three classes. The model uses individual word tokens (unigrams) as features. To remove useless features, it only uses tokens that occur in at least five documents in training set. Results of this model suggest that even simple models can achieve relevant results, when the right features are extracted.

Another work tried to mine opinions from Twitter tweets [37] and compare accuracy of SVM²⁸ and Naive Bayes Model in two-class and three-class classification. In their situation, SVM performed worse, but additional explanation as to why is not provided. In two-class classification, they claim to achieve 75 % accuracy. But, in three-class classification they accomplish only 60 %. Nevertheless, again, simpler model outperformed more robust stochastic model in this task.

²⁸Support-Vector machine

Newer approaches experiment with HMM or deep learning using RNN²⁹ or LSTM³⁰ networks. Recent breakthrough in deep learning field suggests that tide is turning. After popularization of *TensorFlow*TM³¹ and *Theano* Python library³², researchers started digging around. One Stanford paper [38] published comparison between deep RNN and LSTM networks on movie reviews dataset. While both models reached great accuracy on training set (around 94 %), their accuracy on test set is about 84 % (performed on document-level). These results indicate that they may not be universal answers to all the questions. Or, at least, there is a long way ahead for researchers to fully understand estimation of hyper parameters for these models.

Be that as it may, HMM offers improvement over Naive Bayes model based on results in [39]. In two-class classification of customer product reviews (document-level), the model scored 93 % accuracy with 4:1 train-test split ratio of dataset. However, it is important to realize that this model was specifically crafted for one only type of product reviews. Its features were extracted by *OPINE*³³ that utilized POS tags provided by Amazon with the dataset. The model considers only opinion-bearing words (mostly adjectives) for the analysis. Plus, no further sentence analysis is performed to build better understanding about it. Though, in this case it is not needed.

When looking closely at *sentence-level* sentiment analysis, available models do not perform that well. Famous paper [40] from University of Pittsburgh attempts to perform *phrase-level* analysis. This level of text classification tries to interpret only sentiment-bearing phrases which may contain only a few words within context of sentence (e. g. "*good and evil*" or "*grave human rights violators*"). Authors split the classification process into two steps - *Neutral-Polar Classification* and *Polarity Classification*. Each step focuses on different set of features. Number of them is highly dependent on dependency tree that is built for each sentence. Notably, one of the features determines domain of the analyzed document which makes the model highly domain-dependent. In terms of model representation, instead of using a simple stochastic model, authors decided to combine *AdaBoost*³⁴ with another stochastic model to improve its learning properties. Specifically, AdaBoost combines a great number of weak classifiers and tweaks them to avoid overfitting. In terms of accuracy, the first step (*Neutral-polar classification*) and the second step (*Polar Classification*) achieve 76 % and 66 %, respectively. However, the presented work has several issues. Firstly, it is problematic achieve presented accuracy when trying to reproduce their process. And secondly, annotations in MPQA corpus provide

²⁹Recurrent Neural Network

³⁰Long Short Term Memory

³¹TensorFlow is available at <https://www.tensorflow.org/>

³²Theano is available at <http://deeplearning.net/software/theano/>

³³OPINE is a unsupervised information extraction system developed in 2005 by a team from University of Washington

³⁴"AdaBoost" is shortcut for "Adaptive Boosting"

more than only three classes for their phrases annotations and the work does not mention how they prepared the dataset in this fashion.

In terms of domain-independent sentiment models, there is not much research done. Work of Ohana and his team[41] tried to offer a solution to this problem. Their two-class (positive and negative class) classifier implies that using multiple different lexicons improve overall accuracy of the model. As features, their model uses explained scoring system and set of n-grams with POS tags. Probably in effort to maintain control over the model, they decided to not use a stochastic model. Instead, they sum up scores of all positive and negative terms in document individually and then pick the class with higher total sum. This approach achieves accuracy around 66 % depending on the domain. However, recall of those results tend to be imbalanced when comparing classes leaving space for improvements.

2.3 Optical Character Recognition

Optical Character Recognition task is tries to read any form of text from an image and transform it into computer-editable form. Its ultimate goal is being able to reliably read any typed or handwritten text. The concept itself appears very intriguing since it offers a vast variety of applications. Namely, it can be used for text-to-speech for blind people or to replace humans repeating tasks (e.g. sorting letters in post office by deliver address or solving text CAPTCHA's).[42]

Even though the notion of OCR systems has been around for a long time, the first working OCR solution was created in 1951 by David Shepard. The first attempts in the area tried to normalize font and paper size and characters look in order to simplify the algorithms. Current state of the art solutions are capable of reading almost any kind of typed text. They abandoned the effort to create an extensive list of rules to detect each character. Rather, they take advantage of stochastic models trained on a large dataset. *MNIST Database*³⁵ is widely accepted as a gold standard dataset for these models and contains 60,000 training and 10,000 test samples. The best model achieves error rate only approximately 0.23 % on this dataset.

Based on this data, it may seem intriguing to create a customized model. But, it includes dealing with multiple problems at once. Probably the biggest issue is recognizing individual characters in the image. The dataset contains only images of individual characters so it does not address this problem. In real-world text, there are fonts of different types (see Figure 2.9). Hence, it is recommended to use more sophisticated strategy than fixed-box approach to find characters. Some of the OCR algorithms are language-dependent and use vocabularies to improve their accuracy. This adds another layer of complexity, because the text can contain typos.

³⁵ Available at <http://yann.lecun.com/exdb/mnist/>



Figure 2.9: Comparison of monospaced and proportional font

To draw a conclusion, the best approach is to use a third-party solution which handles majority of the complex preprocessing tasks. **Tesseract OCR** fits these requirements and allows users to update its trained model to fit their unique demands.

2.3.1 Reading PDF files

Despite the fact that OCR algorithms are designed to work with images, they are currently being used to extract text from PDF files as well. In its essence, PDF format's main goal is to look exactly the same across different platforms and the users should be able to print them out. There is no strictly defined structure. Instead, there is a set of elements which can be freely used in the file to position text or other components wherever on the page. At the end of the day, this philosophy makes the format very popular among users. In fact, many governments picked it as one of the official file formats for their forms and documents. [43]

However, this flexible structure with very few rules presents a problem for text extraction algorithms. In many cases, the final position of partial text blocks is defined in complicated nested tree of components. Therefore, it is almost impossible to read the text as-is since the text itself is not continuous. Instead, the PDF file is rendered into a set of images and these are then transcribed by an OCR algorithm to computer-editable form.

Unfortunately, this process does not guarantee 100 % accuracy of text extraction. Every component can be significantly distorted. Or, any number of components can be placed over each other. Also, the file can contain exotic fonts which are rather hard to read, unless the OCR algorithm is trained for it. Therefore, a direct parser of PDF format may be a desirable solution for the problem. There is several open source attempts to create such an algorithm like *PDFMiner*³⁶ or *PyPDF*³⁷. But, none of them gives any satisfactory precision (mostly due to the problems with structure of PDF files mentioned above).

Currently, there is several available popular solutions which read PDF files. The most accurate OCR algorithm offers *ABBYY FineReader*. When tested on Oral Argument dataset, it showed almost 100 % accuracy even in the most

³⁶PDFMiner can be found at <http://www.unixuser.org/~euske/python/pdfminer/index.html>

³⁷PyPDF can be found at <https://github.com/mstamy2/PyPDF2>

complex parts of the documents. The only problematic part was *Register* (Register contains list of used terms in the hearing with their positions in the file in format "page:line" - e. g. "alive 25:18") which is not important for the analysis in Chapter 4. But, this program could not have been used for number of reasons. The biggest problem is that the program is a commercial solution so the final analysis would not be accessible to everybody. Another major issue is a lack of any form of public API to control the conversion. Hence, the analysis process would require extensive manual work on user part. On the opposite front, there are open source technologies like **Tesseract OCR**. They achieve great accuracy, are for free and have accessible API so they can be integrated into analysis process.

2.3.2 Tesseract Open Source OCR Engine

Initially, the project was created at Hewitt-Packard Laboratories Co in 1990's. It only became open source project in 2005 and is being developed by Google since 2006.³⁸ Its current version provides support for several other languages. Under the hood, it uses neural network to classify detected characters. For character detection, it uses combination of ordinary spacing method and fuzzy spaces (overview of architecture is outlined in Figure 2.10). [44].

From programmer point of view, it is a C++ command-line utility. There is several wrappers that allow Python use the library. This thesis uses **Tesseract OCR 3.1** and Python library **PyOCR 0.3.1**.

2.4 Technologies

The thesis utilizes only open-source libraries. *Anaconda 4.2.0*³⁹ was used to set up the development environment. In its default installation, it contains more than 100 packages designated to speed up data analysis process in Python in three ways:

1. making extraction of the raw data and their subsequent preprocessing easier
2. handling the data exploration
3. providing tools to prepare stochastic or other models and their simple evaluation

2.4.1 Programming Language

Source codes of the thesis are written in *Python 2.7.12* due to backward compatibility concerns. No significant benefit would come from using the most up

³⁸Tesseract is available at <https://github.com/tesseract-ocr/tesseract>

³⁹<https://www.continuum.io/downloads>

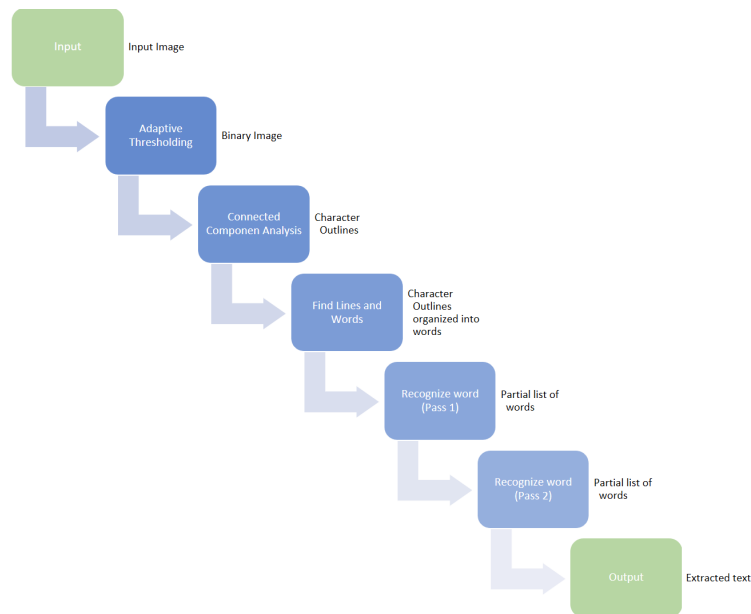


Figure 2.10: Overview of Tesseract OCR processing pipeline

to date version of Python. This language was selected as main programming language for the project because of its philosophy, flexibility and strong support from data analysis community. For Python relies heavily on indentation of the code, its scripts tend to be easier on the eyes. At the same time, it offers basic OOP⁴⁰ capabilities which help to greatly improve code structure.

As Python is often used by data analysts to explore datasets, the community created a specialized tool for this purpose – *Jupyter*⁴¹ (see Figure 2.11). The tool extends functionality of basic interactive command-line Python interpret. Three most useful features in Jupyter are:

- **Code Separation** Code is divided into cells which may be run in arbitrary order and changed at will. This encourages rapid prototyping of the cleaning process. However, overusing this approach can lead into losing track of execution order of cells.
- **Table-view** Support for Pandas (see section 2.4.2) provides better displaying of data from DataFrames in foldable tables.
- **Inline Graphs** Support for *matplotlib* enables inline graphs. They give the analysis a cleaner look. Considering *matplotlib*'s direct integration with Pandas, it makes plotting graphs easy to take advantage of.

⁴⁰Object-Oriented Programming

⁴¹Formerly known as *IPython*

2. THEORETICAL BACKGROUND

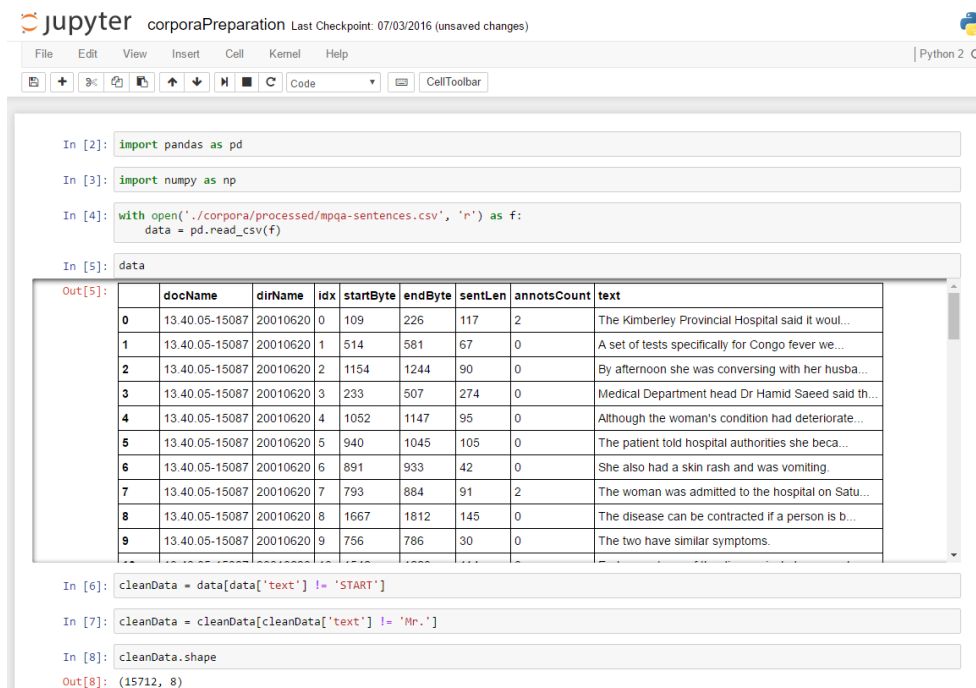


Figure 2.11: Example of Jupyter Notebook used to clean up data for this thesis

2.4.2 Python Libraries

While there is a number of very useful libraries for analysis, this thesis focuses on working with the ones which are easy to set up. Therefore, no library using advanced parallel distributed systems is used. Also, no cutting-edge databases are used either for the same reasons. In general, these components tend to be harder to install on some systems. Thus, instead of spending valuable time figuring out how to configure them properly, the thesis focuses on leveraging speed and wit of the simple libraries. Also, there is no apparent aspect of the thesis which requires their features.

Backbone of the technological stack is **NLTK 3.2.1**[45]. What started as a smaller project in 2001 slowly became an industry standard when performing NLP analysis. The library is used to support basic NLP operations to work with text and access to WordNet. Namely, *Tokenizer* provides an easy way to tokenize sentences into words or create n-grams. Also, it includes providers for Stanford POS tagger (see section 2.1.2) and implementation of *Dekang Lin algorithm to find synonyms* (see section 2.1.5). The community around this library constantly keeps the library updated. Moreover, it provides several corpora (see Table 2.5⁴² [46][47][48][49][28]) reachable directly from the lib-

⁴²*Project Gutenberg* includes more than 53,000 free books.

rary to benchmark created models and NLP techniques. For several of these corpora, users provided access to manually assigned POS tags in standardized form (see section 2.1.2).

Majority of the analysis is done utilizing **Pandas 0.18.1** library⁴³. It leverages *NumPy* library in the background to speed it up. Thanks to its easy R-like⁴⁴ interface for working with DataFrames, it is simple to carry out the usual analytics operations like drill-down, creating aggregations or calculating statistical information about the data inside. A *DataFrame* is an immutable structure emulating R-like data structure. Methods *apply* and *map* provide an easy and fast way to iterate over the dataset and calculate additional information about it.

Scikit-learn 0.17⁴⁵ is built on *SciPy*⁴⁶. The project was started in 2007 by a student as part of a Google Summer of Code. Since that moment on, countless of volunteered helped to keep the project updated and added new features. Currently, it provides tools for data mining and data analysis. As part of the tool set, there are algorithms for:

- **Classification** For example SVM, nearest neighbor or random forest
- **Regression** For example logical or linear regression.
- **Clustering** For example k-Means or spectral clustering.
- **Dimensionality reduction** For example PCA or non-negative matrix factorization.
- **Model selection** For example cross validation.

Tools in each section are implemented so they use identical interface. This approach makes switching them during analysis process rather easy and stream-lined. To speed up the training process, models support multiprocessing via simple flag *n_flags*.

2.5 Datasets and Corpora

This thesis uses several distinct corpora and datasets beside the Oral Arguments (see section 4.2). Majority of them are in some way connected to the created sentiment model (see chapter 3). The exceptions are *WordNet* and *Supreme Court Metadata*. Only *WordNet* is used in both experiments to determine synonyms in text and assign POS tags to words. *Supreme Court Metadata* is describe in detail in section 1.3.2. The rest of datasets is leveraged to support the sentiment model.

⁴³Source Pandas: <http://pandas.pydata.org/>

⁴⁴R is an open-source Statistical Computing software

⁴⁵Source Scikit-learn: <http://scikit-learn.org/stable/>

⁴⁶Source *SciPy*: <https://www.scipy.org/>

Table 2.5: List of interesting corpora included in NLTK

Corpora	POS Tags	Description
Brown Corpus	Yes	The corpus was created in first published in 1964 and undergone several revisions. Currently, it contains 500 samples of text (374 informative and 126 imaginary prose samples) containing more than 1 million words. Each text has at least 2000 words. Texts are written by native English speakers.
Gutenberg Corpus	No	The library contains only subset of text from <i>Project Gutenberg</i> . There is only 18 books. However, they are full books which were proofread in order to catch any typos in the text.
Penn Treebank Corpus	Yes	This is only a small subset of the whole <i>Penn Treebank corpus</i> created in University of Pennsylvania. It contains approximately 40K words. Original corpus includes Brown corpus, Wall Street Journal article's and articles from several other sources. The variety of articles makes this corpus desirable for researchers. That is why its full version is under a commercial license.
IMDB Movie Reviews	No	The library includes older version of this corpora - version 2.0. There is 2,500 positive and the same number of negative reviews from users. Except for removing rating from the review, no additional preprocessing was performed on the dataset. Thus, each review consists of its content and decision whether it was positive, or negative. This thesis uses newer version of this corpus (see section 3).
WordNet	Yes	This corpora provides access to database of English synonyms. There is also a metric expressing to what extend two words are similar to each other. More information on how this corpora is used in this thesis can be found in section 2.1.5.

Table 2.6: Statistics for WordNet

Category	# of Unique Strings	# of Synsets
Noun	117,798	82,115
Adjective	22,479	18,156
Verb	11,529	13,767
Adverb	4,481	3,621

2.5.1 WordNet

WordNet[28] contains a list of words with additional connection between them called *synsets*. In English version⁴⁷, it contains only nouns, verbs, adjectives and adverbs (see Table 2.6⁴⁸). They were manually collected by a group of researchers at Princeton University. Even though it is currently not in development, it still proves to be an invaluable data source for researchers. The additional attributes for each word help to understand connections between them and find more appropriate synonyms. These connection are called *synsets*. Under the hood, synsets are linked by means of contextual-semantic and lexical relations. The current version contains about 117,000 synsets. Understandably, the majority of links is between words having the same grammar category (POS tag). This way, it lowers the possibility of finding the wrong synonym for an input word. Plus, NLTK has an API that provides access to it.[28]

2.5.2 General Inquirer

This dataset provides additional meaning to words created in mid 1990's. It basically is an output of a mapping tool that maps input files from researchers to categories. The dataset was developed for social-science content-analysis research applications. An impulse to create it came from realization that it is hard to reach consensus on categorization words in this manner. So, a complex algorithm that can in somewhat unbiased manner assign words to selected categories was desired. The analysis is carried out on a large set of documents focusing on word frequencies. During the analysis, before a word is assigned to particular category, it removes regular suffixes and performs other disambiguation operations. Output of the algorithm is controlled to a certain extent by user input (weights of each category or contrast ratio between word frequencies).[50]

Currently, it combines 5 different sources to assign words to any number of 182 categories. The category with the most words in it is "*negative*" which

⁴⁷There is several language mutations of WordNet available online - Arabic, Chinese, German and others

⁴⁸WordNet statistics are available at <http://wordnet.princeton.edu/wordnet/man/wnstats.7WN.html>

has 2,291 entries (out of 4,206 words). Here is the list of several interesting categories which are used in this thesis (more categories was used for sentiment analysis in section 3):

- **positiv** A set of words having positive outlook (for example adore, angel or assist). It contains 1,915 words and "yes" words are excluded from it. This category is extended version of "Pstv" category with 1,045 entries.
- **negativ** A set of words having negative outlook or connotation (for example accuse, darken or desert). It contains 2,291 words and "yes" words are excluded from it. This category is extended version of "Ngtv" category with 1,160 entries.
- **active** There is 2,045 entities expressing an active attitude or orientation (for example act, break or control).
- **passive** There is 911 entities reflecting a passive orientation (for example inherent, permit or stand).
- **strong** It includes 1,902 entities implying any form of strength (for example act, enrich or endure).
- **hostile** A subset of words which indicate hostility or aggressiveness toward something or somebody (for example anger, betray or combat). It contains 833 entities.
- **yes** A group of words indicating agreement (for example agree, right or sure). There are 20 entries.
- **no** A group of words expressing "no" (for example no, nope or wrong). There are only 7 entries.

2.5.3 Large movie reviews corpora

This dataset[51] is specifically designated for binary sentiment classification. In its current version, it includes 50,000 IMDB reviews with equal split between positive (score ≥ 7) and negative (score ≤ 4) reviews (classes). Both classes are further split in half to a train set and a test set. However, there is no explicit note claiming that each review is for a different movie in the same set. On the other hand, it is guaranteed that reviews from train set are for different movies than the ones in test set. [51]

Each review is relatively short - containing about 20 sentences. Also, numbers or other numeric reference that can clearly suggest the final score have been removed in preprocessing phase. Yet, HTML tags are present.

This dataset is similar to commonly used *Movie Reviews* corpus available in NLTK. Since it is used as a validation set in this thesis for created sentiment model, and due to the nature of the sentiment model, its purpose is to be, as

different from commonly used dataset, as possible. For a validation set, it is unnecessarily too large. Considering all reviews, model would have to classify approximately 1,000,000 sentences ever time. This classification would take a very long time and provided only marginally better results. Because of this performance issue, only a small, yet carefully selected, portion of the dataset is sampled for benchmarking (more information in Chapter 3).

Sentiment Analysis

One of the goals of this thesis is to attempt to create a domain-independent sentiment model. The model is later applied in Chapter 4 to examine the behavior of justices. Since they cover many different topics during their hearings, the model needs to be versatile enough to accommodate discourse in multiple domains. As argued in section 2.2.2, simpler models may offer comparable accuracy to the complex state of the art algorithms. Indisputably, their advantage lies with being easy to tweak. Whereas, complex stochastic model leaves much of the decision behind elaborate hidden structure controlled by multiple parameters.

In the last section of this chapter, a developed model is trained and tested on *MPQA* corpus that contains combination of documents covering distinct domains (more information in section 3.1). Furthermore, the section explains what feature are extracted from dataset. Then, it argues how they are selected for the final model. Lastly, parameters of the selected statistical model are tweaked to achieve the best trade off between high average accuracy and balanced per-class accuracy of the model.

In order to verify domain independence of the trained model, it is later verified on *Large movie reviews* corpus (see section 2.5.3).

3.1 MPQA Dataset and Ranking Algorithm

This dataset was created in 2002 as part of workshop on ”*Multi-Perspective Question Answering*”. The most up-to-date version as of writing this thesis was 3.0, but the thesis works with version 2.0 because it provides more documents, yet includes all relevant annotations. In total, it contains 692 manually annotated documents from five main sources (see Table 3.1). [52] Clearly, the annotated documents are selected to cover a variety of distinct topic. Granted the number of documents per topic is not big, it still gives researchers a decent set of documents to train their models on.

3. SENTIMENT ANALYSIS

Table 3.1: Sources of documents in MPQA 2.0 corpus

Source	Description
MPQA Original subset	This subset contains original documents (articles from various foreign and U.S. newspapers). Majority of the documents is tied to one of the 10 main topics such as economy, space or politics. A smaller subset was randomly selected from 270,000 documents.
OpQA (Opinion Question Answering) subset	It contains 98 documents which were annotated for the research on opinion answering models. Only documents that contributed to one of the 30 defined question are annotated. Out of those questions, exactly one half is opinion-based and the other part is fact-based.
XBank	This medium-size subset (85 documents) contains articles from Wall Street Journal included in Penn TreeBank corpus.
ULA (Unified Linguistic Annotations)	This small subset contains only 48 documents covering one category (out of six categories which include travel guides, 9/11 reports or transcriptions of spoken conversation). All documents are annotated.
ULA-LU (Language Understanding)	Another small subset that contains only 24 documents which fall into one of the five distinct categories (spoken language transcriptions is one of them)

As a byproduct of [40], the work also created a *Subjectivity Lexicon* that contains 8,219 entries. One word may have multiple entries for different POS tags. Each entry bears four information:

- **Type** of polarity. It defines strength of polarity determined by the word. Possible values are either *weaksubj* or *strongsubj*.
- **POS** tag it represents. Some words can have multiple entries for every POS tag they can be assigned. Instead of using universal set of POS tags, they stick to their own naming convention which is easy to understand. One of the possible values is *anypos*.
- **Stemmed** denotes, if the entry contains a word that is already stemmed. Unfortunately, the material does not specified which stemming algorithm they used in process.

- **Priorpolarity** represents the polarity that the entry word evokes. On top of ordinary values *positive* and *negative*, there are two additional classes - *neutral* and *both* - that help to better understand the word.

3.1.1 Ranking Algorithm

The proposed ranking algorithm uses only subset of annotations provided in MQPA corpus, because many of them have descriptive nature. When developing the ranking algorithm, only a subset of annotations was used. There is a simple reason backing this decision. Other annotations provide additional information about their target or just identify a phrase that annotators feel is relevant to the sentence. However, they were created manually and it would be hard to create yet another model that is capable of detecting similar phrases in documents. Therefore, only these annotation types which bear sentiment or opinion are used:

- **GATE_expressive-subjectivity** marks words or phrases that indirectly express an opinion about a subject.
- **GATE_direct-subjective** provides information about target's opinion (sentiment) on a specific subject in sentence.
- **GATE_attitude** represents attitude of a subject toward a target in sentence.

Here is an example of one annotation from document "20010627/23.46.20-17835"⁴⁹. The phrase is "is believed". Every annotation has an identifier that is unique within the document (first value). In their original form in corpora, only indexes to beginning and end of the phrase are provided (second pair of values):

```
48 1785,1796 string GATE_direct-subjective
expression-intensity="low" attitude-link="a2" intensity="low"
nested-source="w, implicit" polarity="neutral"
```

The corpus contains 35,359 annotations. Not every sentence or even every have at least one annotation, though. In fact, out of 15,802 sentences, only 9,785 sentences contain at least one annotation (their frequency is shown in Table 3.2). When considering distribution of their frequencies, only 1,96 % of sentences (192 sentences) contains more than 10 annotations. An average number of words in those sentences is 48 (minimum is 24 and maximum is 161). However, the third quantile of word count in those sentences is 58 which is rather close to the mean. Therefore, there is only a few sentences (less than

⁴⁹All documents from MPQA in this section are references following the same naming convention - *set/document name*

3. SENTIMENT ANALYSIS

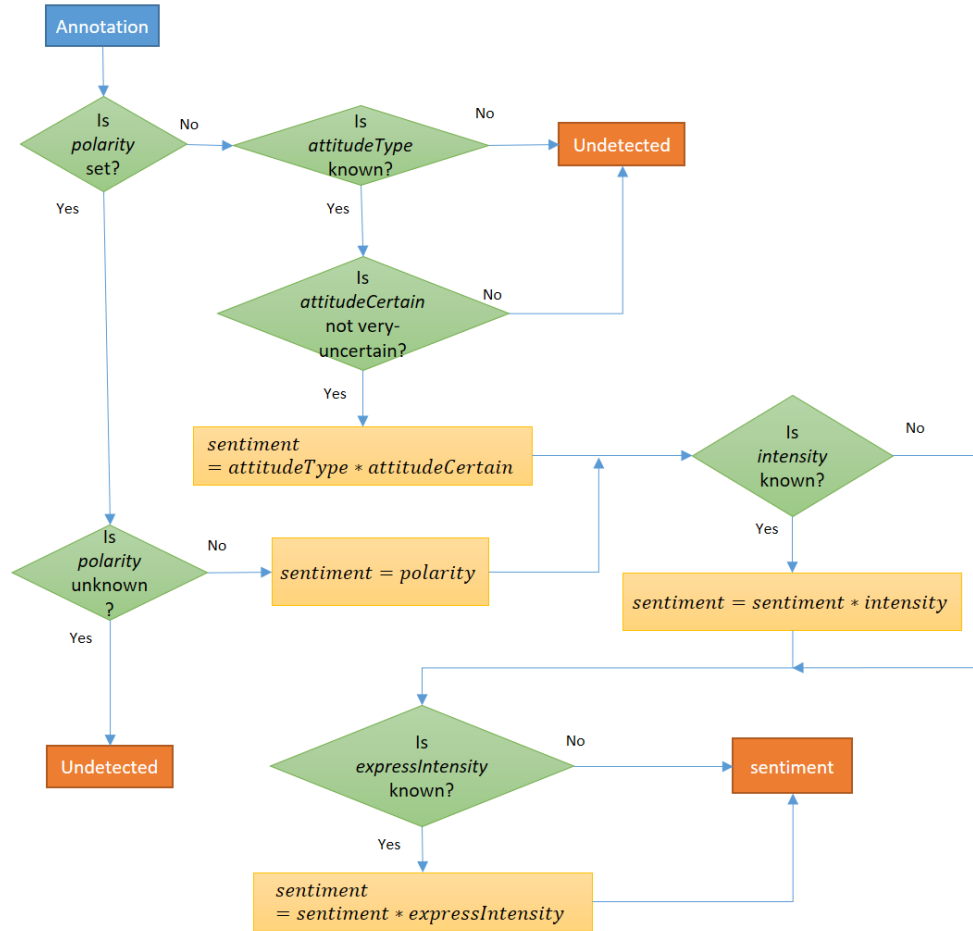


Figure 3.1: Final version of ranking algorithm for MPQA corpus

48) having significantly larger number of annotations. This suggest that there might not be as much information in every sentence from in set, as in other sentences. Due to their low count in comparison to the rest of dataset, there is not enough information for any model to reliably learn learn their traits. Instead, these outliers could cause overfitting of the model for these specific cases. Therefore, they are excluded from further analysis.

In contrast, an average number of words in sentences with less than 11 is 23 (minimum is 1 and maximum is 136) and its third quantile is 31 words. When looking at sentences with more than 30 words (2,464 of them), 66 % of them have up to 40 words (see Figure 3.3). Sentences in this subset have 4 annotations on average and 90-th percentile of number of annotations is only 8. Considering the ratio, these sentences contain . This suggests that there may not present enough information to make a precise decision about their sentiment. The rest of the sentences have wide range of word

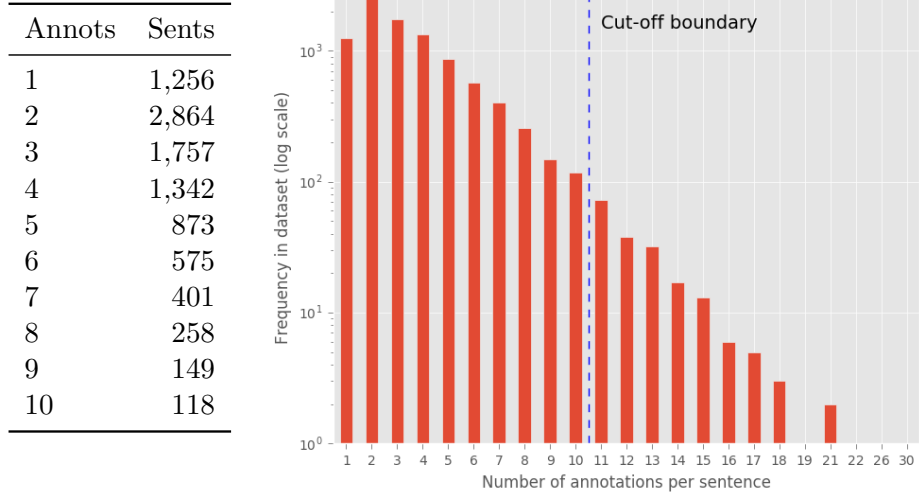


Figure 3.2: Distribution of counts of annotations in sentence in MPQA corpus

counts and annotation counts which makes data sparse and not useful for other preprocessing as well.

To sum it up, only sentences with at least one annotation are selected for sentiment analysis. Out of them, only sentences with maximum of 30 words are used in the dataset for training and testing of the created sentiment model. So, **7,113 sentences** can be used to train and test the model.

Before the model can be train and tested on MPQA corpus, every relevant sentence has to be ranked. Initially, the effort was to create a three-step process with five possible classes, in which each step would provide only partial answer. Workflow of the initially proposed classification process was following:

- **Step 1** determined, whether sentiment can be detected, or not (class "undetected" or advancing to *Step 2*). This was meant to be a crucial step as it was meant to lower number of false positives in results.
- **Step 2** decided general class of the sentence. Either, it has polarized sentiment (positive or negative further specified in *Step 3a*) or mixed sentiment (neutral or mixed further detected in *Step 3b*).
- **Step 3a** classified a sentence into one of the classes - *positive* or *negative* - and assigned it intensity as a number in range $< 0, 1 >$.
- **Step 3b** classified a sentence into one of the classes - *neutral* or *mixed* - and assigned it intensity as a number in range $< 0, 1 >$. Sentences with mixed sentiment contained dominant traits of both polarized classes.

3. SENTIMENT ANALYSIS

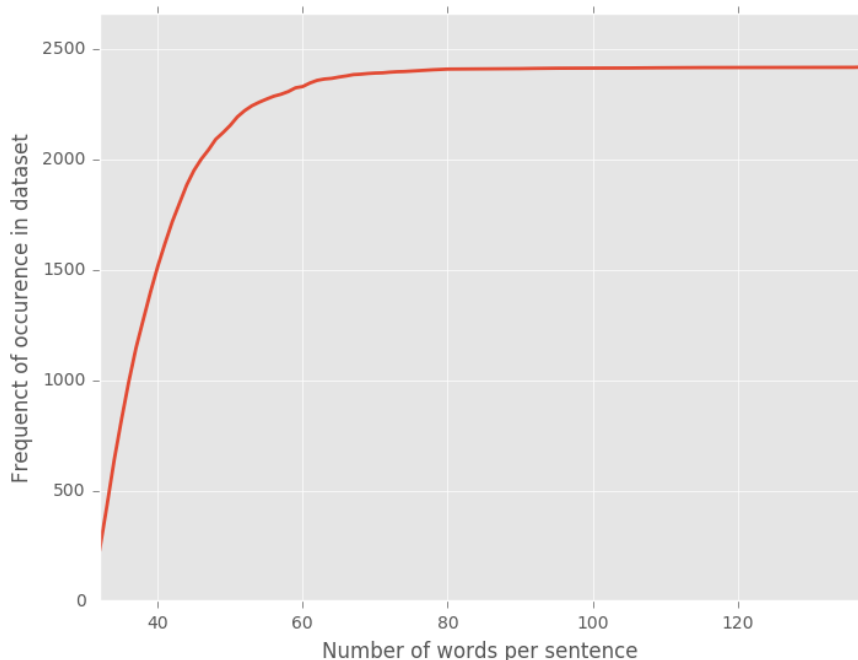


Figure 3.3: Frequency of word counts in sentences with at least 31 words

To get the final sentiment classification, all three parts would be evaluated together. Additionally, the model was meant to estimate intensity of the detected sentiment as well. This approach followed mindset proposed in [40] so each step would be implemented as a separate, simple, regression classifier.

However, after extensive work on the ranking algorithm and fine-tuning its parameters, this path proven to be a blind alley for several reasons. The biggest problem is the lack of data. Not all annotated documents are written in English language and in Latin alphabet. So, these sentences are excluded from the set.⁵⁰ Thus, the whole set contains **7,040 sentences**. Lets consider a case, when for each class, there is 10 different values. There are 4 classes and sentences in each of them should have from 1 to 30 words. In ideal case, 4 or 5 different examples should be sufficient to learn the pattern (preferably even more, since the effort is to create domain-independent model). Putting it all together, the model would require 4,800-6,000 sentences.

In theory, the dataset has enough data. Unfortunately, the data is rather sparse. Figure 3.4 displays distribution of sentiment intensities in dataset after normalization. The ranking is performed by algorithm described later in this

⁵⁰Names of documents written in non-Latin alphabet with annotations are "im_401b_e73i32c22_031705-2", "IZ-060316-01-Trans-1", "20000815_AFP_ARB.0084.IBM-HA-NEW", "NapierDianne"

section. The graphs clearly demonstrates that even after considerable effort to separate different levels of sentiment, there is not enough clues. In class *positive*, there are two dominant spikes around values 0.7 and 0.8 which cover majority of positive sentences. Therefore the trained sentiment model would be biased to estimate a great portion of inputs around these values.

Another problem is in distribution of sentences in classes in dataset. There is only 364 annotations (out of 35,359) having *polarity* equal to, either *both*, or *uncertain-both*. Clearly, working with only 1 % of annotations does not provide enough clues to reliably work with the class. Other combinations of annotations (e. g. equal number of positive and negative annotations with similar features) were tried to compensate for this lack with no significant results. Therefore, class *both* is not included in the final sentiment model.

Considering all above, the initial idea for sentiment model was abandoned, so the ranking algorithm had to reflect the change. As it turned out, estimation of the intensity would not be reliable. Thus, even though the final proposed ranking algorithm gives a score in range $< -1, 1 >$ to every sentence, the score has only informative character. Assigned constants to values of each possible value of annotation properties were initially given empirically. Then, after several iterations to achieve cleaner cut of data for each class, the constants stabilized. After the sentences are ranked, their score has to be normalized to range $< -1.1 >$ using:

$$sentiment_{pos} = \frac{value - min_{pos}}{max_{pos} - min_{pos}} \quad (3.1)$$

$$sentiment_{neg} = \frac{value - min_{neg}}{max_{neg} - min_{neg}} \quad (3.2)$$

In order for a sentence to classify as neutral, its sentiment intensity has to be within interval $< -0.1, 0.1 >$ after normalization. This interval was deduced empirically after several iteration followed by manual verification of results on a small representative sample (30 random sentences from each class, ideally, having various word counts).

The final distribution in classes is shown in Table 3.2. Counts of sentences are not roughly equal, but, according to distribution of annotations in dataset, it seems valid. There is only 22 % annotations with some degree of negative polarity. But, 15,44 % of them have *low* or *neutral* intensity. Also, number of negative annotations is absorbed by *neutral* class because of being in sentences with positive annotations. On top of it, negative sentences tend to have more negative annotations in them.

The complete diagram that shows the final ranking algorithm is pictured in Figure 3.1. Values for each attributes are listed in Appendix B.

3. SENTIMENT ANALYSIS

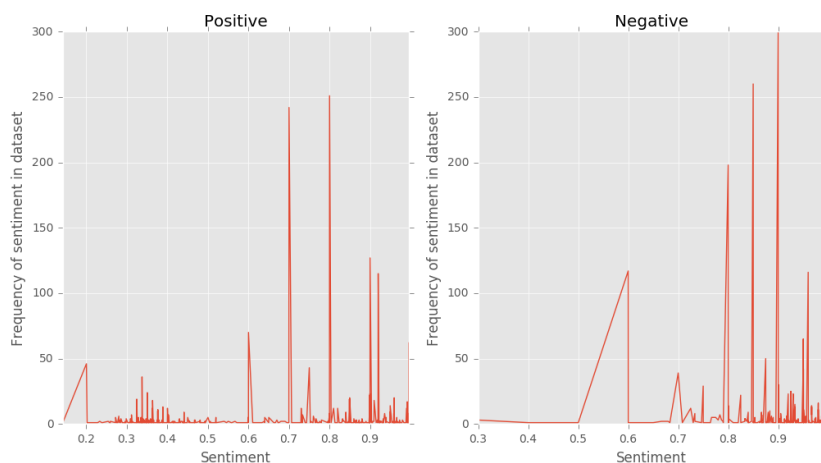


Figure 3.4: Frequency sentiment intensity in dataset for class *positive* and *negative*

Sentiment	# of Sentences
Negative	836
Neutral	2,071
Positive	2,692

Table 3.2: Distribution of sentences into sentiment classes by ranking algorithm

3.2 Feature extraction

This work proposes a different approach to modeling a sentiment model than many other works. They work with word tokens themselves or their n-grams. This work, instead, takes advantage of number of dictionaries. Thus, they can be later refreshed to update the model without having to retrain it. Extracted features can be split into several main categories (all words are stemmed with *Porter Stemmer*2.1.4.1 before they are extracted):

- **Category Counts** focuses on counting words which belong to specific categories defined by vocabularies. This features tells the model general context of the sentence. There are two special categories in dataset - *priorpolarity* and *priorpolarity-type* - that are set for every word in the dictionary (see below why) and can be also considered additional attributes. Strength and polarity of every word in the vocabulary is further defined by them.

- **Negation-related features** shed light on dynamics in the sentence. Its simpler version define, whether any word from specified category is in front of (or after) any negation word in the sentence. More complex version counts how many words from the category there are around negations.
- **N-gram features** are related to *Category Counts*, but are restricted to neighborhood of only a few words. Therefore, they do a decent job of detecting local clues for sentiment. Length of n-grams is an important factor which will be further discussed later..
- **POS n-grams** further enhance *n-gram features* by including respective POS tags to every word. They should help to disambiguate complex cases. While their length is also important, there are different reasons behind them being shorter described also later in the section.
- **Comparison features** indicate which word category dominates in the sentence over another one. These binary features provide a brief, yet consistent, view on context in sentences. On its own, they carry very little information. But, together with more descriptive features, it helps to tip the scales when necessary.
- **Other features** provide meta information about the sentence like punctuation.

Before feature extraction starts, all articles from sentences are deleted, because they do not convey any relevant information.

As one of first steps in process of feature extraction, *General Inquirer* and *Subjectivity Lexicon* from MPQA dataset are combined together to form one consistent vocabulary. Since there are several instances of a case when one word has multiple entries, two situations can happen. Either there is an entry with "anypos" POS tag, so that entry is used. Otherwise, the first entry from the subset is selected.

POS-related features help to approximate *syntax tree* analysis. In order to deduct required relationship between words, the n-grams need to be longer. Based on preliminary tests, n-grams with more than 6 words create very sparse space which made their usage in models unreliable. In contrast, n-grams with less than 4 words do not contain enough additional information to be useful in the models. In the end, tests shown that n-grams with length 4 or 5 offer a great compromise in this case. Example of one POS feature is:

count how many adjectives or adverbs with attribute *priorpolarity=positive* is there before a noun in any 4-gram

Together, **161 features** is extracted. For the complete list, see Tables C.2, C.3, C.4, C.5, C.1 and C.6 in Appendix C. Although, only a subset of

these features is used in the final sentiment model. Feature selection process is described in detail in Section 3.3.

3.3 Training and results

Final sentiment model is built on **Random Forest Tree** classifier⁵¹. In Scikit library, this classifier uses *Decision Trees* using *CART algorithm*. However, several other possible stochastic models were tested in the process and discharged for various reasons. The selected data samples provided a few challenges during the training process.

One of the first problems was imbalanced dataset. According to Table 3.2, clearly there is more *neutral* or *positive* samples than *negative* samples in the dataset. The difference is so significant that any stochastic models became quickly overtrained during the experiments. All of them tended to be biased toward the dominant class. This issue was remedied by making a stratified selection from each class. Thus, each class had the same amount of data samples in the each set (train or test). In order to keep as many samples as possible, all *negative* samples are used and the same amount of *positive* and *neutral* samples are randomly selected from the respective classes. Thus, **2,508 sentences** are used for training and testing the model. This approach also solves the problem how to determine baseline for the experiments. When each class is evenly distributed in the dataset, then the created model has **baseline accuracy 33 %**. In this section, when talking about accuracy, author always talks about the best achieved accuracy on the given test set.

Another problem was to determine, whether two-step binary classifier would have better overall accuracy than one-step multi-class classifier. Generally speaking, two-step binary classifier in this given situation is a variation to *One-to-Rest* approach. Early in the process, the idea was to have two separate classifiers:

- First one having classes *Neutral* and *Non-neutral*
- Second one having classes *Positive* and *Negative*

In this configuration, each class was classified by the first classifier. If its class was *Non-neutral*, second classifier determined its final class. Several different underlying stochastic models were tested with no significant differences in result on test set. In order to check reliability and stability of the tested model, 3 and 5 way cross-validation is performed each time with stratified parts. For both steps, the same statistic models were used at a given time.

⁵¹Description of *Random Forest Tree* implementation in Scikit can be found at <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

Since this approach was inspired by work in [40], *AdaBoost* boosting ensemble method was also included in experiments. This algorithm underlines the philosophy of this work - combines several weak components together to create more stable and more accurate model. Its components (another classifiers) are added sequentially and trained separately using weighted data. Training phase repeats until one of the pre-set conditions is met (e. g. no further improvements in accuracy were achieved or maximum number of classifiers defined by user has been reached). Its prediction is made by calculating weighted average of all the classifiers.[53] One of the drawback of this ensemble model is its tendency to being easily thrown off by outliers.

During the measurements, models using *AdaBoost* in combination with various other stochastic models (*SVM*, *Decision Tree* or *Naive-Bayes*) shown to perform rather comparably - poorly. For performance-issues, *Random Forest Tree* was not tested in combination with *AdaBoost*, because the training phase took rather long time. In general, any above-mentioned stochastic model alone without the *AdaBoost* performed about 3-4 % better. Unfortunately, the best results provided by *Random Forest Tree* were still around 70-72 % in 5-way cross-validation in step one and around 67 % in 5-way cross-validation. For 3-way cross-validation, their accuracy was even lower. Despite the mentioned individual accuracies being higher than the baseline, the overall accuracy was only marginally better than the overall baseline. Also, the first classifier was biased toward *Neutral* class which caused large number of false positives. This behavior suggest that the problem is two-fold. Firstly, selected dataset do not offer enough samples for the models to train on them using the mentioned features reliably. Secondly, selected features may not be descriptive enough for this model. Therefore, two-step model concept was abandoned and one multiclass classifier was developed instead.

The last outstanding problem was *feature selection*. Not all extracted features are eligible for the model and carry enough information to be important. Usually, this problem is solved by maximizing *Information gain*⁵² which expresses difference between two probability distributions. Its multiclass version for m classes and feature x is defined by formula:

$$\begin{aligned}
 G(x) = & - \sum_{i=1}^m Pr(c_i) * \log Pr(c_i) \\
 & + Pr(x) * \sum_{i=1}^m Pr(c_i | x) * \log Pr(c_i | x) \\
 & + Pr(\bar{x}) * \sum_{i=1}^m Pr(c_i | \bar{x}) * \log Pr(c_i | \bar{x})
 \end{aligned} \tag{3.3}$$

As discussed in [54], it is a great tool to condense a large number of

⁵²Also known as *Kullback-Leibler divergence*

features in dense feature space together. When used on thousands of features, it produces relatively well results ([54] claims to achieve 98 % precision). This metric selects features which have their IG above predetermined threshold. In text classification problems, this metric focuses on finding dominant terms in a vast set of words (tokens from all documents in training sets). Since this work approaches to text classification problems differently, IG does not provide reliable results to select only relevant features. Therefore, a new, simpler, combined metric called **Ratio Occurrence Index (ROI)** is proposed to select only important features. It combines two metrics together for the given class and feature to express their potential for the model. They are calculated using following formulas:

$$ROI(c_w, f_i)_{feature-ratio} = \frac{count_{present-value}(c_i, f_i)}{count_{different-values}(c_i, f_i)}$$

$$ROI(c_w, f_i)_{total-ratio} = \frac{count_{present-value}(c_i, f_i)}{count_{samples}}$$

where:

- $count_{present-value}(c_i, f_i)$ is number of samples having feature f_i set to non-default value and belonging to class c_i
- $count_{different-values}(c_i, f_i)$ is number of different values of feature f_i in samples belonging to class c_i
- $count_{samples}$ is number of samples in dataset

This metric tries to look at each class individually and limit its descriptive features by detecting very dominant features. In essence, they pose a problem for model as they tend to make it bias toward their class. $ROI_{feature-ratio}$ address this issue. It calculates how often, on average, the same value of feature f_i occurs in samples from class c_i . The bigger this number is, the more descriptive this feature is. In contrast, if the ratio is too big, then values are too sparse. Experiments provided clues that $ROI_{feature-ratio}$ value for samples \bar{c}_i ⁵³ holds informative value. In terms of class c_i and feature f_i , this value offers an insight on how descriptive f_i is for other classes as well. The last control part of this metric is $ROI_{total-ratio}$ which makes sure that the feature f_i covers representative portion of dataset. Its value is in range $< 0, 1 >$ and the bigger its value is, the bigger portion of dataset this feature in class c_i covers. This criteria helps to filter out weak features.

Experiments shown that each class in the sentiment model requires different thresholds for ROI metric (see Table 3.3). Selected features from each class are combined together using set union operation. Numbers of selected features are 86, 73 and 45 for classes *Positive*, *Negative* and *Neutral*, respectively. After union, model uses 87 features in total.

⁵³Samples in dataset that do not belong to class c_i

Sentiment	$ROI_{feature-ratio}(c_i)$	$ROI_{feature-ratio}(\bar{c}_i)$	$ROI_{total-ratio}(c_i)$
Positive	2.5	No restriction	No restriction
Negative	2.2	4.0	No restriction
Neutral	2.5	No restriction	0.05

Table 3.3: Thresholds for feature selection for classes in sentiment model

As mentioned above, created model is also validated against a completely different set - *Large movie reviews* (see section 2.5.3 for more information about the dataset). As opposed to the final sentiment model, this dataset contains only whole reviews and no reviews from *Neutral* class. There is no information about sentiment of every individual sentence in reviews. Therefore, following approximation of sentiment estimation is used for validation purposes:

- Each review is split into individual sentences
- Each sentence is assigned to one of the three sentiment classes (*Positive*, *Negative* or *Neutral*)
- Positive, negative and neutral sentences are counted and the class with the highest number of sentences is assigned to the review

Since this dataset contains only *positive* and *negative* reviews (both classes are equally represented in the dataset), baseline for any sentiment model is 50%. Just to make a sanity check on the dataset, another simple classifier is tested on it. This model extracts only two features (again, all words in dataset are stemmed by *Porter Stemmer* - see section 2.1.4.1):

- Number of words in sentence is also in *Sentiment Lexicon* and have *priorpolarity* equal to *positive*
- Number of words in sentence is also in *Sentiment Lexicon* and have *priorpolarity* equal to *negative*

The final class of sentence is determined by larger number of featured. If both features are equal, sentence is *Neutral*. Then, the same evaluation process is applied on each review to classify it. For performance reasons, only a subset of the dataset is evaluated - 1,000 positive and 1,000 negative reviews. Each rating is represented in the class by equal number of reviews, in order to ensure proper sampling of the dataset.

Measured average accuracy of the simple model is 52.5 %. Table 3.4 shows *confusion matrix*⁵⁴ after the classification. Even though average accuracy of the model performs above the baseline, its per-class accuracy clearly portraits

⁵⁴ *Confusion matrix* contains information about predicted and actual classifications.

the fact that this model is highly biased toward *Negative* class. Around 71 % reviews is classified as *Negative*. As a result of this imbalance in overall accuracy, this model is unreliable. However, it demonstrates that the dataset contains clues to create model which would surpass the baseline accuracy. Yet, the clues are not so trivial and descriptive that any simple classifier can use them to reliably classify reviews. Thus, it is possible to use it as validation set for this experiment.

Final sentiment model uses *Random Forest Tree* as its underlying statistical model. After experiments with other stochastic models (and simple *feedforward neural network with back-propagation*) or their combination with *AdaBoost*, it proven to provide the most reliable results. In the final model, higher accuracy is sacrificed to more balanced results in class accuracy. In real-life application, model that has statistically almost the same chances to classify a sample to any given class correctly gives more persuasive outcome. Also, a slight bias toward *Neutral* class is more desirable than toward any other class due its nature.

There are three parameters which need to be optimized for the selected stochastic model:

1. **max_features** determines strategy that figures out how many features is looked at when considering the best split
2. **max-depth** defines maximum depth for any estimator in the forest
3. **n_estimators** denotes maximum number of created estimators in the forest

Parameters 2 and 3 are closely connected so they cannot be tuned in separation. Only *max_features* parameter can be taken out and set up individually. There are three possible strategies - $|F|$, $\log_2(|F|)$ or $\sqrt{|F|}$ features are considered, where $|F|$ represents number of selected features. Dealing with too many of them when considering splitting decision tree makes it harder for the model to decide (which may lead to undertraining of the model). So only, either second, or third strategy can be used. When working with a huge number of estimators, it does not matter which strategy is used in this case. In the worst-case scenario when all features are used, $\log_2(161) = 7.33$ and $\sqrt{161} = 12.69$, so there is no significant difference between the numbers in comparison to 161. On the other hand, when dealing with smaller sets of features, strategy 2, $\log_2(|F|)$, is preferred.

Other two parameters are estimated based on series of measurements. When *max-depth* is bigger, then estimators tend to overtrain on individual samples. On the other hand, setting it too low allows the model only to learn general rules about the training set. These rules proven to not be sufficient enough to reliably detect sentiment. Not even with a significant number of estimators (more than 1,000). At the same time, considering large number

		Predicted		
		Positive	Neutral	Negative
Actual	Positive	0.247 (247)	0.143 (141)	0.610 (610)
	Neutral	0 (0)	0 (0)	0 (0)
	Negative	0.103 (103)	0.094 (94)	0.803 (803)
Total		0.175 (350)	0.119 (237)	0.707 (1,413)

Table 3.4: Confusion matrix for **simple** classifier on **Large movie reviews** dataset

		Predicted		
		Positive	Neutral	Negative
Actual	Positive	0.523 (49)	0.344 (32)	0.129 (12)
	Neutral	0.172 (16)	0.634 (59)	0.194 (18)
	Negative	0.118 (11)	0.387 (36)	0.495 (46)
Total		0.272 (76)	0.455 (127)	0.272 (76)

Table 3.5: Confusion matrix for **trained** classifier on **MPQA** dataset

of estimators with bigger *max-depth* is also not a good option. The model reaches a stable state during training phase⁵⁵ rather quickly when using estimators with bigger depths (overtrains), so excessive number of estimators is not an universal answer. Experiments shown that their number should be in range $< 200, 800 >$.

When tested on provided set, the final model performs rather well (see Table 3.5) with **average accuracy 54.8 %**. Even though the model is slightly biased toward *Neutral* class, it is fine, because neutral sentences do occur more often in ordinary language. No other class is showing signs dominance over the other class.

On validation set, the created model performance does not decrease. It manages to maintain above-baseline accuracy **54.5 %** with class accuracies **50.2 %** and **58.8 %** for *Positive* and *Negative* class, respectively (see Table 3.6). The result proves domain independence of the model, as train set does not contain movie reviews data. Also, in comparison to *simple classifier*, it is not significantly biased toward either class. This makes its output even more reliable.

The created model is stored as **model_8** - *Random Forest Tree* - trained having best accuracy with parameters *max-depth*=7, *max_features*=log2 and *estimators*=800 using *cross-validation* = 9.

⁵⁵When any additional training will not result in any significant improvement of the model

		Predicted		
		Positive	Neutral	Negative
Actual	Positive	0.588 (588)	0.141 (141)	0.271 (271)
	Neutral	0 (0)	0 (0)	0 (0)
	Negative	0.346 (346)	0.152 (152)	0.502 (502)
Total		0.467 (934)	0.1465 (293)	0.387 (773)

Table 3.6: Confusion matrix for **trained** classifier on **Large movie reviews** dataset

3.3.1 Final Evaluation

The final MPQA dataset (with balanced number of samples in each class) has **baseline for overall accuracy 33 %**. Even though it is lower than state of the art model developed in [41] with overall accuracy *66 %*, this model is biased toward one class and supports only two-class classification of sentiment. Therefore, the trained model outperforms it in both overall accuracy, and accuracy for individual classes. On top of it, validation on completely different dataset proved its stability on datasets with previously unseen domain. Consequently, it can be reliably used in Chapter 4 to detect sentiment of participant turns during Oral Argument.

Table 3.7 shows counts of features in the final model. As expected, majority of selected features in model (almost *61 %*) are *Category Counts* and *Negation-related features*. Thus, approximated structure of any sentence in the model is captured in almost *38 %* of the features (*1 %* is reserved for other feature). Ratio between these two groups of features demonstrates extensive emphasis of the model on content of sentences.

Observation shows that *Category Counts* features takes up the biggest portion of all features. Intuitively, this is correct approximation, as context of sentence is to most variable element in it and is related to its domain. Also, higher number of *Negation-related* features suggests that the model should be able to learn more complex negation patterns in sentences. They are supported by the rest of features which add informations about structure of the sentence to improve disambiguation. Given the number of features in both *N-gram* feature categories, the model is able to learn only slightly advanced sentence patterns. Considering size of the training set, it is more desirable to learn smaller number of patterns better. There is not enough data in the dataset to learn complex patterns reliably. Therefore, their number is corresponding to what their role in the model is.

Only sarcasm detection is not directly covered by any feature. Since it is very hard to detect and formal nature of Oral Arguments does not allow much room for using sarcasm, it is allowed.

Feature Category	Count	Portion
Category Counts	31	35.6 %
Negation-related	22	25.3 %
N-grams	15	17.2 %
POS N-grams	9	10.3 %
Comparisons	9	10.3 %
Other	1	1.2 %

Table 3.7: Distribution of feature categories in feature set of the trained sentiment model

Oral Argument Analysis

Second goal of this thesis is to analyze the Oral Arguments from the US Supreme Court. As a result of it, it should create a model capable of predicting outcome of a discussed case. This thesis takes a different approach to the problem than other related works in recent years (see section 4.1). Every step of the process from data preparation, feature extraction to training and testing the final prediction model is described in following chapter. First of all, transformation of PDF file to its text form is explained. Then, the extracted data are analyzed via myriad of customized reports to gain an insight on justice's behavior. This step uses a custom-made *sentiment model* from Chapter 3 to extract sentiment from every turn. Also, it uses other simple techniques described in detail in section 2.1. Lastly, it takes the extracted features and evaluates them to select only dominant ones to avoid problems with under/over training.

4.1 Related Works

There is only a few works which cover a similar topic. Probably the most recent one is *CourtCast* model developed by Chris Nasrallah[55] that also uses purely oral arguments features. The model use *linear SVM* statistical model to predict outcome of the case (whether the petitioner wins or loses). The author claims to achieve 70 % accuracy with its predictions. On input, it receives the text from justices and sentiment for words in turns of the dialog. Unfortunately, author never explains how to detect sentiment of the words. Under the hood, model extracts most used words during the dialog and compares their sentiment with *bag-of-words* approach. Then, it uses a few other features like number of times a justices interrupts a lawyer. Considering its results, the work provides relevant information on possible set of features which may help to predict outcome of the case. Also, it makes a valid point that prediction of individual votes of justices does not provide substantial

additional value to the outcome. However, development of the model stopped in May 2015 and is no longer maintained.

Another interesting work is thesis done by Timothy Hawes from Cornell University[56]. His work examines *turn-taking patterns* and tries to use them to predict outcome of the case. While it strongly argues that patterns can be reliably use in model, their informative value is questionable. For once, it is highly probable that their turn-taking is influenced by their seniority status in the group. Therefore, younger justices tend to interrupt senior justices a lot less and vice versa. This suspicious is partially confirmed by findings in [56]. Author claims to achieve accuracy around 65 %. During the experiments, author observed that *SVM* from *LIBSVM*⁵⁶ classifier outperforms *Decision Tree* classifier implemented in *WEKA 3.6.0*⁵⁷. This behavior is not observed during experiments in this thesis.

Last but not least, there is a rudimentary work from Feldman [57] carried out in 2016 that also uses *bag-of-words* approach. Instead of making predictions, it only studies behavior of justices during the oral argument of one specific case - *hole Woman's Health v. Hellerstedt (No. 15-274)*. The analysis also uses *sentiment analysis*. However, again, the sentiment model is rather basic - a small sentiment dictionary is used to detect sentiment on per-word basis. Despite this flaw, the work provides clues on relevant features which may bear enough information to predict the outcome. Also, as one of the few works mentioned in here, it splits the oral argument into two parts (petitioner's questioning and respondent's questioning) and analysis them separately. The author presents that sentiment plays a significant role when predicting outcome of the case. Another important feature seems to be *questions*. Their amount and set-out appear to hint justice's intentions.

4.2 Oral Arguments Dataset

Basic description and information about this dataset is in section 1.3.1. This section focuses on how to transform PDF files to clean text. Furthermore, it explains the process of preprocessing transformed text for later analysis.

There is no complete consistency in format of every Oral Argument PDF file in the dataset. At the beginning, there may or may not be a cover page introducing the case. As with the rest of the document, there is no strict structure of the cover page so it cannot be reliably parsed and used to gain more features either. Also, there is *Index* at the end of majority of Oral Arguments with list of words. Any open-source library had a great deal of difficulty to read it all correctly and reliably (without typos in words). Therefore, only actual dialog between petitioner, respondent and justices is read from the file.

⁵⁶LIBSVM is an open-source library that implements SVM classifier in multiple languages <https://www.csie.ntu.edu.tw/~cjlin/libsvm/>

⁵⁷WEKA is an open-source collection of machine learning algorithms written in Java

Every parsed document from years 2013-2015 is in one of the following formats:

- **bottom-middle** signals that page number is in the middle of footer of every page with dialog lines (see Figure 4.2)
- **top-right** represents pages in which page number is in top right corner of every page with dialog lines (see Figure 4.1)

After experiments with several open-source libraries, it appeared that it is rather difficult for them to recognize all text on the page as-is. Hence, only relevant region containing only dialog of text is extracted from each page. Unfortunately, some pages contain misplaced words outside of this region. In current approach, these words are intentionally ignored as they are hard to recognize them and consolidate them rest of the text accurately. Respective figures also mention offsets to crop only text part of the page.

During the transformation process, every page is converted to a black-and-white image with 300DPI⁵⁸ using *ImageMagick* library in Python. Even though it is computationally more demanding to run text recognition on image with 300DPI, smaller images do not provide enough information to recognize every character correctly.

Two other enhancements helped *Tesseract OCR* to achieve 100 % accuracy when recognizing text from Oral Arguments. Firstly, a *high-pass filter* is applied on every pixel to alter its color and remove noise. A simple *threshold filter* following this formula is sufficient for this task:

$$threshold(x, y) = \begin{cases} 1 & , if color(x, y) > \theta \\ 0 & , if color(x, y) \leq \theta \end{cases}$$

where:

- color is defined in range $\langle 0, 1 \rangle$ when 0 is complete white and 1 is complete black
- $color_{x, y}$ is a function that returns color of pixel at position x and y
- θ is the selected threshold

This filter sharpens edges of characters in order to make them easier to read. Based on several measurements, *Tesseract OCR* achieved the best accuracy with $\theta = 0.4508$.

Second improvement is training *Tesseract OCR* only to recognize font used in these files. In order to do so, whole *Tesseract OCR* model was deleted and

⁵⁸DPI stands for *Dots Per Inch* and symbolizes how many pixels is fit on a line with length 1 inch

trained on several manually annotated pages extracted from these files. Newly trained model is enclosed on attached CD.

Even though both of these enhancements greatly improved accuracy of recognized text, there two lasting problems. The first problem is caused by hyphens. Since there is several types of hyphens and they are not used consistently in PDFs, the model is not properly trained to recognize them and often recognizes them as strange hyphen-like unicode characters. Therefore, they need to be replaced with normal hyphen in post-processing phase of transformation. The other problem is varying line height. Unfortunately, this issue has not been solved for all documents. On a few occasions, the algorithm does not place recognized words in the correct order. As a result, a few turns in the dataset do not contain correct words. Since this problem occurs only in 0.14 % of turns in the dataset, it can be considered as negligible.

After the raw text representation of PDF file content is available, only dialog is further extracted from it. In order to find beginning of the hearing, the algorithm looks for line that goes:

P R O C E E D I N G S

This line is always followed by time stamp that declares at what time the hearing started. Also, it is important to notice that there is a blank line after every line with text.

Detection of the end of the hearing is slightly more complicated. There is several variations of phrases that can be uttered by *Chief Justice*. Experiments shown that there are two short sentences which reliably signal the end. Either a variation to:

The case is submitted. (Whereupon ...)

Or:

(Whereupon, at ...)

Afterwards, the dialog text is sent for further processing, such as each document is split into turns with following attributes using regular expressions:

- **Person** contains name of the person who speaks during the turn
- **Role** denotes role of the person in the case (*JUSTICE* or *OTHER*)
- **Text** contains full text spoken during the turn.
- **IsInterrupted** signalizes, whether the turn was interrupted by somebody else (ends with '–')
- **PositionDialog** remembers position of the turn in terms of the dialog and is in range $\langle 0, 1 \rangle$ (0 being the beginning of the dialog and 1 being the last turn of the dialog)

4.3. Feature Extraction and Selection

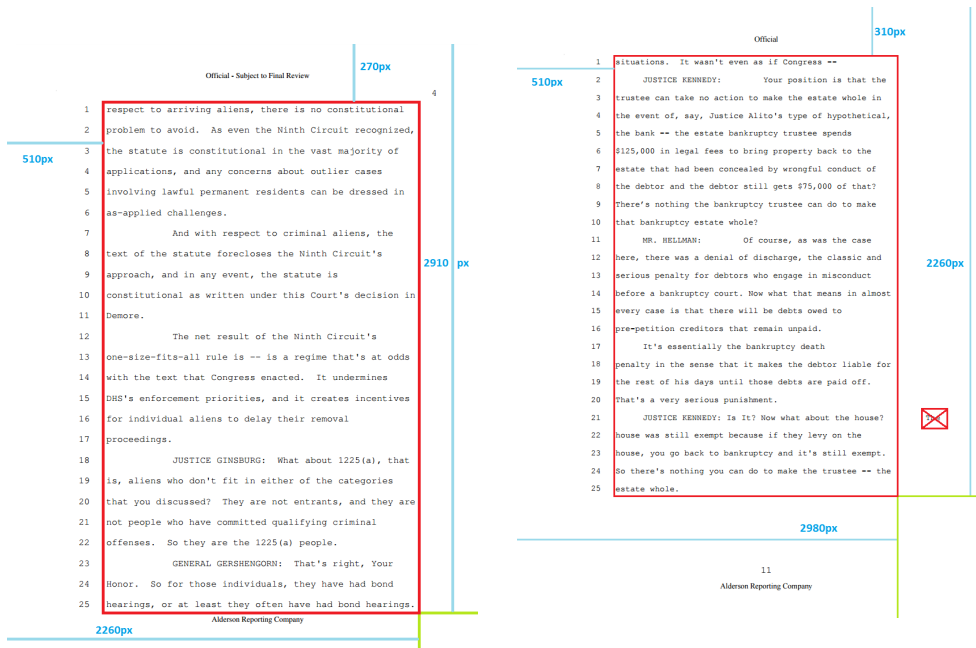


Figure 4.1: Example of one page from oral argument in "top-right" mode with offsets in pixels

Figure 4.2: Example of one page from oral argument in "bottom-middle" mode with offsets in pixels and undetected word *the* outside of reading area

These attributes are further aggregated and extended in the next section (Section 4.3).

4.3 Feature Extraction and Selection

This thesis works with **198 cases** from terms *2013-2015*. Each one of them has been split into turns. Further, every turn has been split into individual sentences. Unfortunately, original PDF files contain typos in words and names of participants. These typos are corrected in merged extracts from datasets (see Section 4.3)

There is several different ways how to look at this data. First of all, not every justice has to speak during the Oral Argument. A great example of "silence" justice is a justice *Clerance Thomas* who rarely speaks during them (see Figure 4.3). Since there is only one Thomas's turn in dataset, as an outlier, is not being analyzed. Others show approximately the same tendency to mostly engage with either petitioner or respondent during hearings. Only justice Scalia seems to be significantly less interested in cases than other justices. However, other two justices, Breyer and Alito, also seem to be noticeably less interested in higher number of cases than the rest of the justices.

4. ORAL ARGUMENT ANALYSIS

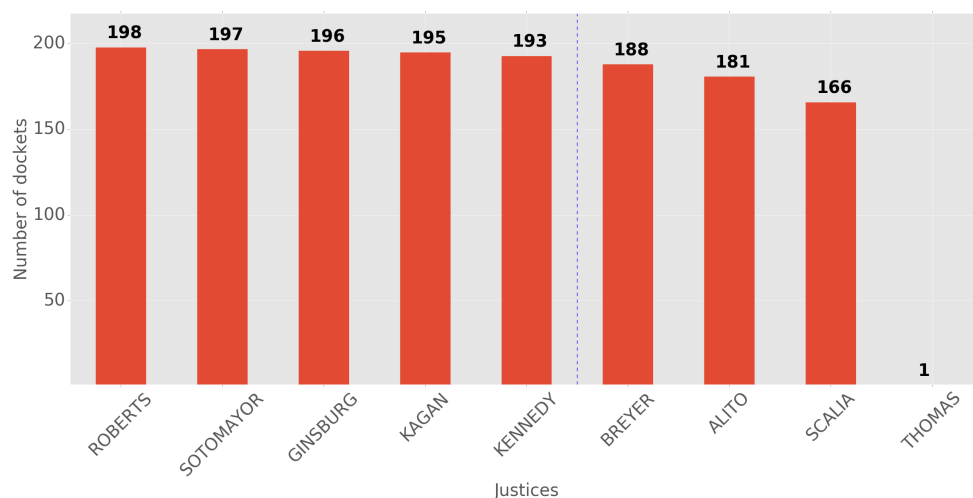


Figure 4.3: Number of dockets each justice speaks up at least once during the Oral Argument

Therefore, this when either one of those three justices shows interest in the case, it is most likely somehow different from other cases. So, this becomes a binary feature **Important Case**.

Another very interesting, yet rudimentary, view of dataset is number of turns each justice has in all dockets and their average number (see Figure 4.4). It confirms that seniority of justices affects their willingness to speak up during hearings. Chief justice Roberts having the highest seniority in the group speaks the most often out of all of them. Then, the group can be further split into dominant justices (Roberts, Sotomayor, Breyer, Scalia) and more submissive ones (Kagan, Kennedy, Ginsburg and Alito) based on total number of turns they have in dataset. This information is later use in combination with other metrics to compute Activity Index for the docket (see Section 4.4).

When looking at somebody’s willingness to speak, it is also important to consider how much time he spent talking. This information is represented in Figure 4.5 which shows number of words each justice uttered during hearings and their average number per docket. Surprisingly, Kagan is in the second place and Breyer is, by far, in the first place. These observations suggest that some justices may come to the hearings with clear decision in their heads (justices on the lower end of the Figure 4.5). Others need to extensively describe their point of view (higher end of the figure). In combination with their dominance status inferred from their frequency to speak up during hearings, it is a part of **Activity Index** for justices (see Section 4.4). There is not enough data to calculate the index for other participants in the hearings.

A great indicator of how much a justice may be influenced during the hearing is number of asked questions by individual justices. There are two phases

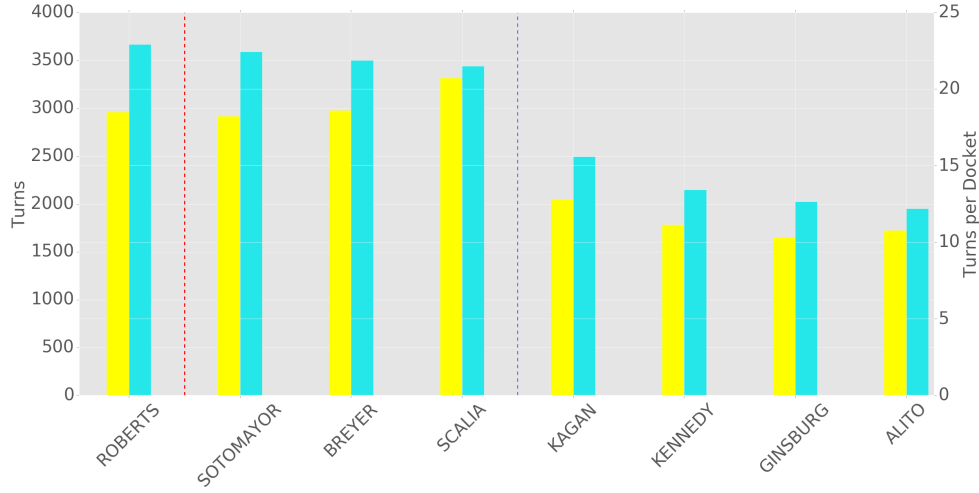


Figure 4.4: Number of turns each justice has in all dockets together and their average number per docket

of the hearing - *Petitioner's* and *Respondent's* part. Clearly, the amount of questions asked during either phase is consistent with respect to role of the participant (justice or other) and winning party. As expected, justices ask a lot more questions than attorneys. Figure 4.6 also shows that total number of questions asked by justices when *Petitioner* party wins the case is almost twice as big than when *Respondent* party wins. This is, however, caused by larger number of cases in which *Petitioner* party wins. When comparing their mean values, justices ask, on average, less question during the hearing, if *Respondent* party wins (combined for both phases). Even closer look (see Figure 4.7) shows that justice *Scalia* tends to ask more questions when *Petitioner* party wins. This is expected due to his habit to only speak when he is really hesitant. In the end, both of these attributes are quantitative discrete features **total number of asked questions by justices** and **number of questions asked by justice Scalia**.

Another indicator of unusual behavior of justices is number of speech interruptions. More interruptions of other participants suggests disagreement with what they are saying (or defending at the moment). Table 4.1 shows distribution of difference in number of interruption of both parties per docket in dataset using formula:

$$interruptions_{diff} = \sum_i^{interruptions} f(i)$$

where:

$$f(x) = \begin{cases} 1 & , if x = petitioner \\ -1 & , if x = respondent \end{cases}$$

4. ORAL ARGUMENT ANALYSIS

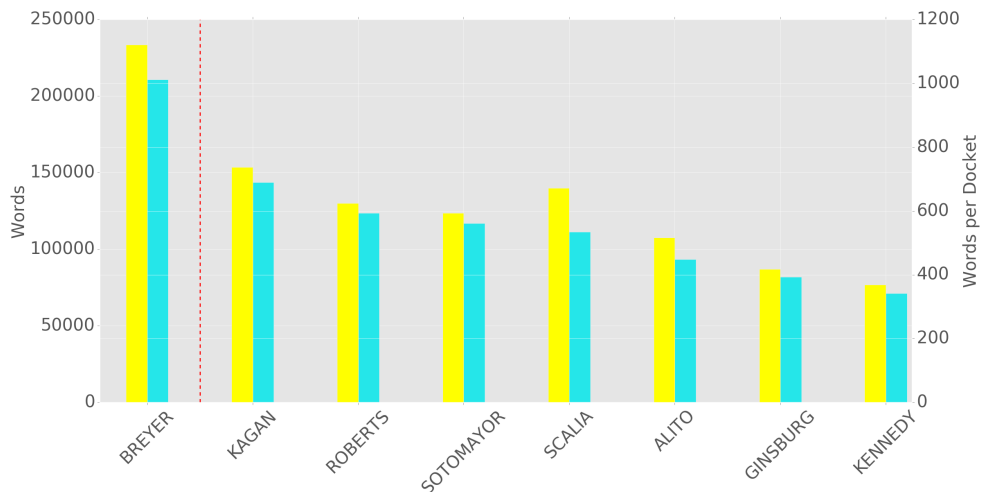


Figure 4.5: Number of words each justice utters in all dockets together and their average number per docket

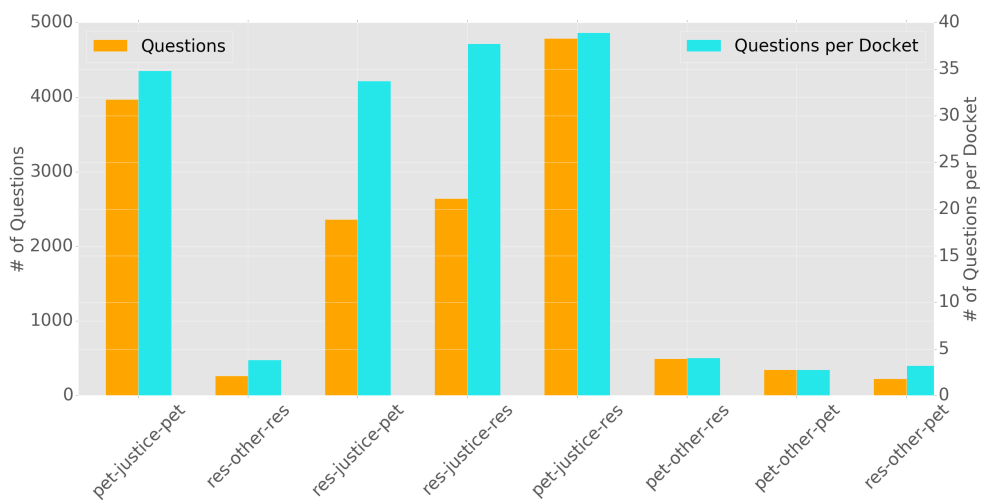


Figure 4.6: Frequency of asked questions per different phase and type of dialog; 'res-other-pet' means that winning party is *Respondent*, not justice is asking the questions and it is *Petitioner's* part of the hearing

4.3. Feature Extraction and Selection

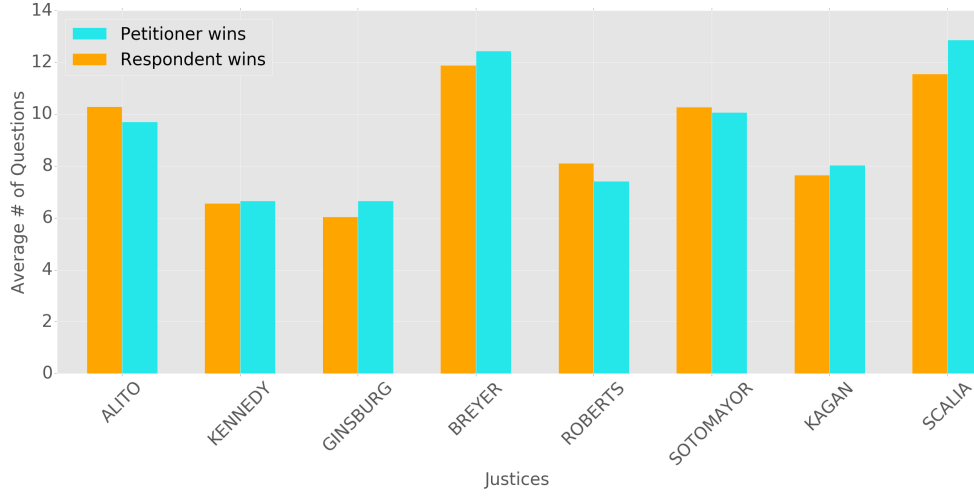


Figure 4.7: Frequency of asked questions per different phase and type of dialog; 'res-other-pet' means that winning party is *Respondent*, not justice is asking the questions and it is *Petitioner's* part of the hearing

Winning Party	$interruptions_{diff} < 0$	$interruptions_{diff} \geq 0$	Total
Petitioner	42 (33.6 %)	83 (66.4 %)	125
Respondent	35 (47.95 %)	38 (52.05 %)	73

Table 4.1: Distribution of interruptions by justices in dataset

The result suggests that positive $interruptions_{diff}$ signals that the winning party might be *Petitioner*. If this was a rule, only 38 cases would be incorrectly classified. As such, the direct link between $interruptions_{diff}$ and outcome of the case is not apparent at first. It comes from expectation that in order for petitioner to win, the discussion during hearing should be more vivid on their part (usually, they need to justify their case). Then, if justices have any doubts they want to clear up (possible misinterpretation of law), they tend to jump right into other's turns. Hence, **interruption difference** $interruptions_{diff}$ is used as a discrete quantitative feature. However, it is questionable, whether the correlation imply causation. That is why two different set of features should be tested - the second one without this feature - to see how it affects results.

Next studied property of dialog is *follow ratio* among justices. These metrics aim to identify any relationship between justice interaction between each other and outcome of the case. Justice A *follows* justice B, if their turns are separated by a single turn taken by non-justice participant of the hearing. If somebody tends to follow another participant often, it suggests that both of them share the same or have opposite opinion during the dialog. In other words, they have tendency to influence each other. A quick look at who

Justice	# dockets being Most Followed regardless of winning party						
	RO	SC	KE	BR	SO	KA	GI
Alito (AL)	(47)	(34)	(27)	(25)	(23)	(14)	(11)
Breyer (BR)	(54)	(36)	(29)	(28)	(19)	(13)	(9)
Ginsburg (GI)	(57)	(40)	(24)	(22)	(21)	(17)	(15)
Kagan (KA)	(35)	(33)	(28)	(28)	(26)	(25)	(20)
Kennedy (KE)	(58)	(32)	(30)	(28)	(20)	(13)	(12)
Roberts (RO)	(40)	(33)	(30)	(28)	(26)	(21)	(20)
Scalia (SC)	(55)	(31)	(21)	(17)	(17)	(14)	(11)
Sotomayor (SO)	(69)	(36)	(24)	(24)	(24)	(12)	(8)

Table 4.2: Distribution of *Most Followed by* in dockets per every justice

is followed the most by each justice (see Table 4.2) confirms the hypothesis that seniority plays significant role during the hearing. It turns out that almost every justice mostly follows justice *Roberts* during Oral Arguments. The only exceptions are justices *Breyer* and *Kagan*. The first mentioned most often follows justice *Scalia* regardless of winning party. On the contrary, justice *Kagan* likes to follow justice *Scalia* when *Petitioner* party wins and justice *Sotomayor*, if *Respondent* party wins. Both of them are respected justices. Also, *Roberts* tends to follow after *Scalia* the most in majority of cases. This findings confirm the seniority status and of *Roberts* and *Scalia*. Thus, justices can be divided into 3 groups (*influence circles*) with respective inf_{idx} (influence index) based on how often *Roberts* follows them the most in cases:

- $inf_{idx} = 1.25$ justices *Roberts* and *Scalia*
- $inf_{idx} = 1.0$ justices *Sotomayor*, *Alito* and *Breyer*
- $inf_{idx} = 0.75$ justices *Kennedy*, *Kagan* and *Ginsburg*

Influence index is part of the *Activity index* for justices (see Section 4.4). The index is not determined for other participants in hearings, because there is not enough data for them to be safely deduced.

More advanced metrics is **Topic Chain Index**. As explained in Section 2.1.6, it looks for first and last occurrence of a noun phrase. *TCI* index

says how many topic chains a docket contains. There are three important properties of each topic chain:

- *Intensity* TCI_{int} counts number of occurrences of the phrase in the chain
- *Length* TCI_{len} defines normalized distance between first and last occurrence of the phrase
- *Size* TCI_{size} counts number of characters in the phrase

TCI_{size} may seem like an odd metric, but it can be understood as a very simple measure of how interesting the word is. When comparing most occurring words in the dataset (see Table 4.3), phrases having at least 7 characters are clearly more interesting. However, there are smaller noun phrases which may be interesting. But, it is a good approximation of more complex metric to get rid of ordinary (not most occurring) noun phrases.

TCI_{size}	Count in Dataset	Examples
5	17.7 % (2768)	union, hours, wagers, peace
6	15.7 % (2462)	issue, safety, united, effect
7	14.9 % (2334)	respect, program, meeting, concern
8	13.2 % (2068)	medicaid, laughter, exercise, fairness

Table 4.3: A few examples for most occurring phrase lengths in dataset

Another important property of TCI is its length TCI_{len} . Rather short chains represent only a brief exchange of opinions between participants. Unless their Intensity TCI_{int} is significantly bigger, they are not relevant (their dynamics is covered by other, sentiment-based, metric in the model). Distribution of lengths of chains in dataset (see Figure 4.8). The bins show that 34.7 % of chains are rather long and first 6 bins (bin 0-5) make up only 9.93 % of chains. Therefore, the dataset contains many longer topic chains with $TCI_{size} > 6$. So only chains from bins 6-9 are further analyzed.

Experiments did not reveal any apparent relationship between number of topic chains in individual dockets and winning party. It does not necessarily imply that there is none, though. Therefore, the feature is tested in modified form in model to see, if it improves its prediction performance. Since the range of values is too large ($< 21, 87 >$), its value is put into one of the 4 equal-width bins created in the range (starting with 21, ending with 87 and having step 16). This creates a quantitative discrete feature with range $< 0, 4 >$.

Text-based feature **Topic Chain Index** groups similar tokens together before the chains are calculated. The groups are formed by *Dekang-Lin* synonyms detection algorithm (see Section 2.1.5). As mentioned above, it is a non-trivial problem to set its parameter correctly. Figure 4.9 shows different settings of the algorithm and their respective grouping ratio (what portion of all words is absorbed by their synonyms). Each setting is tested in

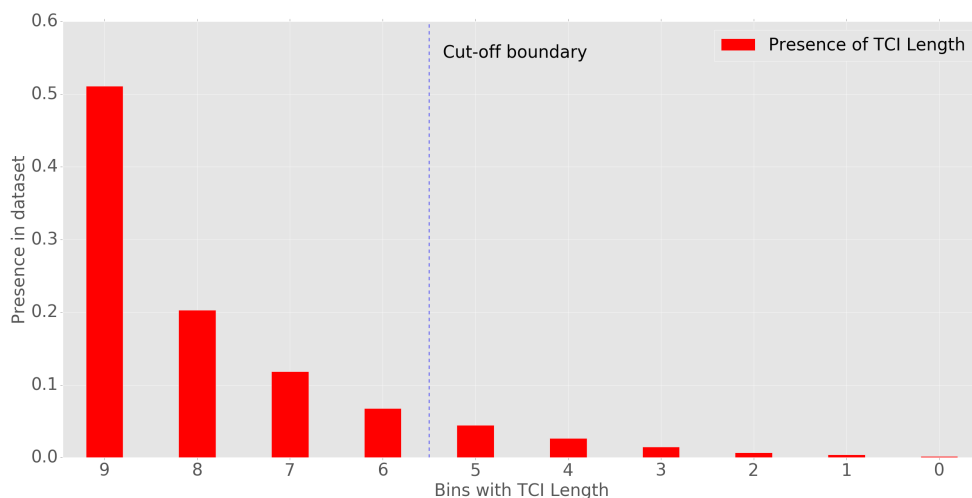


Figure 4.8: Distribution of lengths of topic chains in dataset split into 10 equal-width bins

two versions - with and without using lemmatizer (see Section 2.1.4.1). The test was performed on whole available dataset with 2,114,109 words (27,836 unique words). But, the algorithm is applied only on adverbs, nouns and verbs (23,230 words). Clearly, lemmatization significantly improves performance of the algorithm. Naturally, the wider similarity rate for similar words is allowed, the more words is covered. *Similarity rate* determines what is the maximum allowed difference between the most similar word found and another similar word. As mentioned above, not all provided similar words are true synonyms, so higher rate provide higher rate of false positives in results. The difference between coverage rate of $sim_{ratio} = 0.05$ and $sim_{ratio} = 0.1$ is only 4 % which is not big enough to compensate for the false positives. Thus, $sim_{ratio} = 0.05$ is used to group words together and calculate all TCI.

Another interesting piece of information about attorney is whether he has any previous experience with arguing in front the Court. Outcomes of their previous encounters are not that relevant. It is the experience that matters the most in this case. Given they are more experienced, they tend to focus only on relevant elements in their turns. So, attorneys like *Clement* (14 cases) or *Dreeben* (11 cases) have better chance to persuade justices about their intentions. This are two binary features **Has Experience** for petitioner and respondent side. Both values are set based on their attorneys and appearances in cases in dataset so far (see Table 4.4).

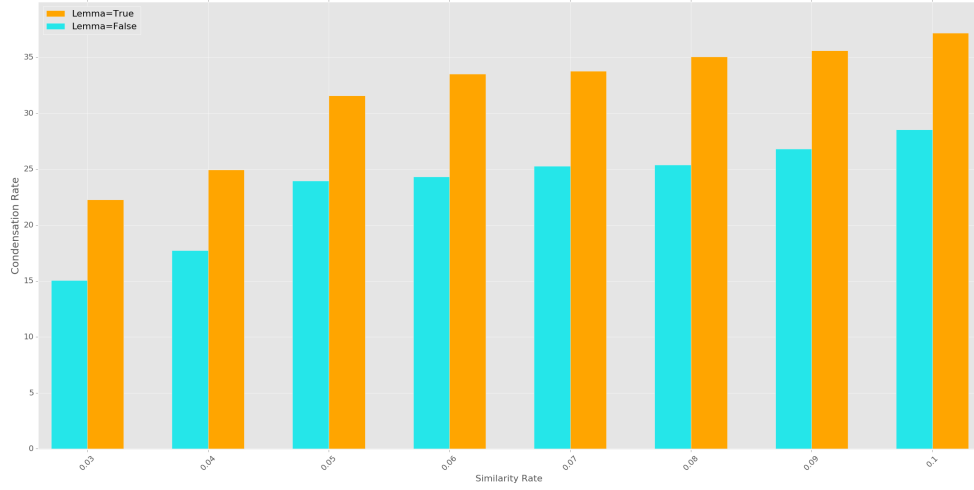


Figure 4.9: Comparison of different settings for algorithm that groups similar words together

# of Appearances	# of Attorneys
3+	48
2	29
1	171

Table 4.4: Number of appearances of attorneys in front of the Court in dataset

The next, more advanced, step is sentiment analysis of the hearing. It is done in a similar way that it is mentioned in Section 3.3. Every sentence in each turn is analyzed separately. Then, sentiment of the turn is defined by number of positive, negative and neutral sentences in the turn. The most occurring class in the turn defines class of the turn. Otherwise, sentiment of the turn is *Neutral*. Simple counts of turns belonging to each sentiment class (see Figure 4.10) shows no extraordinary relationship between their number and the winning party. Albeit, closer look at ordering of sentiment distribution in dockets provides insight on how often the participants changed tone of their verbal speech. So, a different metric called **Difference in Sentiment Changes** $sentDocketDiff_{sent}$ is proposed. It calculates how many times sentiment changed from one class to another in the dialog in a specific phase ($sentPhaseDiff_{pet}$ and $sentPhaseDiff_{res}$ for Petitioner and Respondent phases, respectively). Then, the metric expressed by formula:

$$sentDocketDiff_{sent} = sentPhaseDiff_{pet} - sentPhaseDiff_{res}$$

Distribution of $sentDocketDiff_{sent}$ (see Table 4.5) shows bigger tendency for swings of sentiment during Respondent phase in cases, when *Petitioner* party

4. ORAL ARGUMENT ANALYSIS

		<i>sentDocketDiff_{sent}</i>	
		Negative	Neutral and Positive
Winner	Petitioner	63.2 % (79)	36.8 % (46)
	Respondent	43.84 % (32)	56.16 % (41)

Table 4.5: Distribution of **Difference in Sentiment Changes** in dockets split by winning party

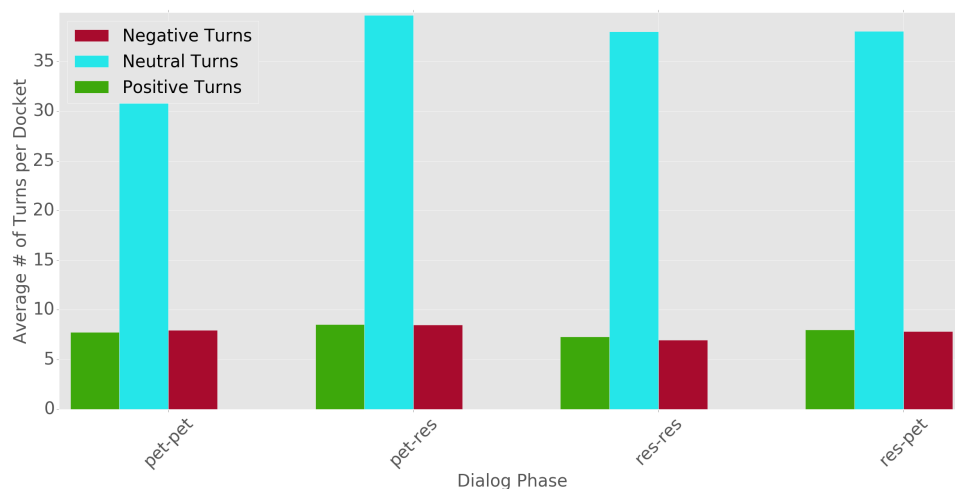


Figure 4.10: Average numbers of turns per docket split into sentiment classes (label 'res-pet' represent numbers for dialog phase when Respondent wins the case and the turn is in Petitioner's part)

wins (or in Petitioner phase when *Respondent* party wins). This indication can be interpreted as increased disagreement in opinions in phase of the winning party. Thus, this is a discrete quantitative feature in the model.

Lastly, metadata about cases provide context to the case (see section 1.3.2 for information about the dataset). Not all attributes for the case bring valuable information. One of the first interesting attributes is *precedentAlteration* which signals that the ruling changes a precedent from past. While this is very rare, it may help to better predict such cases. However, the subset of Oral Arguments contains only 1 such case. Thus, this attribute is not reliable enough.

On the contrary, dataset contains attribute representing direction of ruling from Lower Court (**lcDispositionDirection**) which appears to be more helpful when predicting the final outcome. There is no apparent link between directions of Lower Court's decision and the final decision of the Court. In case of both *conservative* and *liberal* decisions, *Petitioner* party has the same probability of winning (about 63 %) in both of them. However, the ruling of

the Supreme Court relatively often have the same direction (in 41 % cases). Now, considering this situation in combination with another attribute, *Lower Court Disagreement* (called **lcDisagreement**) provides interesting patterns in voting outcomes. If there was a disagreement in lower court and the Supreme Court's final ruling has different direction than lower court's decision, then there is 91 % probability that *Petitioner* party wins. In case the rulings have the same direction, there is 85 % probability that *Respondent* party wins. It is hard to explain the hidden rationale behind this voting pattern. Nonetheless, both attributes are used in model as binary features. They are based on prerequisite that outcomes from both courts are the same to certain extent and known. Even though the decision of the Court is not known during the hearing (it is estimated after the final ruling by experts manually), other features in this model can help to detect its direction.

4.4 Training and Results

This section describes how *Activity index* for justices is determined by clustering using *k-means* in order to reduce dimensionality. A few different settings are compared to achieve the best results. Then, all features are tested in several combinations to compare their performance in three classifiers - *Naive Bayes Classifier*, *k-Nearest Neighbor* and *Random Decision Forest*. Tested classifiers represent three fundamentally different approaches to approximation of the problem.

At first, *Activity index* is created by running *k-means* algorithm on vector space defined by features mentioned in Section 4.3:

- Influence index inf_{idx} of justice
- Number of turns $turns_{justice}$ the justice has during the hearing
- Number of words $words_{justice}$ the justice says during the hearing

Definition of the vector space is greatly influenced by the number of samples in it and their distribution in each docket (winning party) class $docket_{winner}$. The goal is to create two clusters which mimic split of dockets in the dataset. This way, the clustering may detect patterns that correlate and then cause the distribution of winning votes in dataset. Number of clusters should be small because of number of samples in dataset and number of already used features. More clusters create sparser features space which require more samples to train on. Furthermore, too many dimensions in the vector space make it harder for clustering algorithm to find the best decision boundary. So instead of using 16 dimensional space (2 dimensions per 8 justices), algorithm uses

only 2 dimensions constructed in following way:

$$AI_{coords}(docket) = \begin{pmatrix} \sum_{justice}^{justices} inf_{idx}^{justice} * \frac{turns_{justice}}{avgTurns_{justice}} \\ \sum_{justice}^{justices} inf_{idx}^{justice} * \frac{words_{justice}}{avgWords_{justice}} \end{pmatrix}$$

Criterion for the best clustering is ratio between number of samples in clusters and how similar it is to ratio between dockets won by Petitioner and Respondent Given that clusters cannot be implicitly assign to any *docket_winner* class, the best clustering has ratio between number of members in both classes close to one of those ratios in dataset. Ratio of dockets in dataset won by Petitioner to Respondent party is 1.712 (Respondent:Petitioner is 0.584).

Figure 4.11 shows behavior of k-means algorithm with respect to number of iterations⁵⁹. Clearly, the ratio between classes in the dataset or considering only dockets that belong to certain *docket_winner* behave similarly. It oscillates around value 1.038 between local minimum and maximum (and, occasionally, between global minimum and maximum too), so the algorithm has issue with converging to a clear split. Also, larger number of samples may be switching from one cluster to another back and forth while trying to converge. In this case, the ratio is not oscillating, but improving. Several runs showed that global maximum is 1.357 and global minimum is 0.737. Obviously, global minimum is closer to one of the above-mentioned ratios, so the last clustering with this ratio is used to determine the index for prediction model. Number of cluster is coded into k binary features. For a given value, all features are 0 except for the one at position of cluster number.

Baseline for the prediction model on selected subset of Oral Arguments is **63.13** % (125 cases out of 198 won Petitioner party). Apparently, the dataset is, again, rather imbalanced. Unfortunately, in this case, this problem cannot be solved by using equal numbers of samples from each class, because of the size of the dataset. Every sample is valuable. Therefore, bias toward *Petitioner* class should be handled by features inside the model. We test different variants of features to verify their performance. Two metrics are used to evaluate it:

- **Accuracy** measures how many true positives and negatives the model predicts out of total population using formula:

$$Accuracy = \frac{\sum t_p + \sum t_n}{\sum samples}$$

- **F_1 -score** which represents harmonic mean of *precision* and *recall* expressed by formula

⁵⁹Shown ratio is Petitioner:Respondent

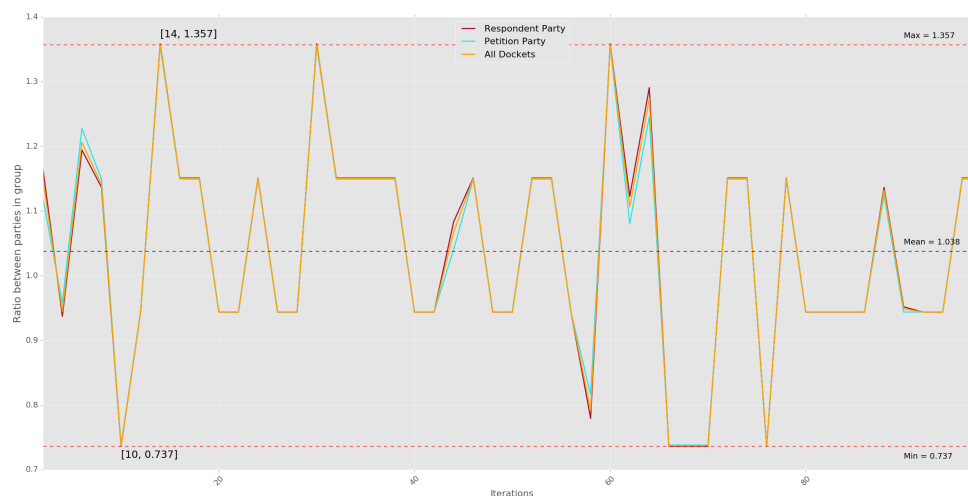


Figure 4.11: Ratio between numbers of docket in two classes created by k-means algorithm split into two groups of docket by winning party

$$F_1 = \frac{2 * t_p}{2 * t_p + t_n + f_n}$$

Accuracy does not reflect uneven performance of the model correctly. It only captures mean of accuracies for each class. Therefore, it is used only as a secondary metric. Instead, F_1 -score is used to select the best model. This metric uses *precision*⁶⁰ and *recall*⁶¹ to better follow the effort to find model providing balanced predictions.

Considering size of the dataset, it is harder to split it into train and test set so cross-validation is used to find the best one. Table 4.6 shows splits for popular configurations of cross-validation with ratio of cases won by Petitioner and Respondent party. It is desirable to keep the ratio as close to 1 as possible to achieve balanced sets. This condition is best satisfied with 7-fold cross-validation. To get the best result for given configuration, model is trained and evaluated three times for each step. In the end, model with the highest F_1 score is considered as the best one for given classifier and parameters. Since other related works do not often provide F_1 score, results also state accuracy of the model and best achieved accuracy for given classifier regardless of any other metric.

⁶⁰Precision measures what portion of predicted samples are true positives

⁶¹Recall measures ratio of true positives to all samples classified correctly

n-Folds	Test set	Train set	Petitioner : Respondent
3	41 (24)	82 (48)	0.585
4	31 (18)	93 (54)	0.581
5	25 (14)	100 (56)	0.560
7	17 (10)	102 (60)	0.588
10	12 (7)	108 (63)	0.583

Table 4.6: Size of train and test set with different stratified n-folds in cross-validation with ratio of Petitioner:Respondent samples in set; format is *petitioner(respondent)* size of the fold

Each classifier is tested on several different subsets of features to evaluate their impact on overall performance of the model. Even though, each feature subset gives slightly different view on samples in dataset, parameters of each classifier are fine-tuned on *BASE* subset of features and evaluated on all subsets. This is necessary pre-condition to be able to reliably compare their results. Selected feature subsets are:

- **BASE** contains basic features *Difference in Sentiment Changes*, *Important Case* and *Has Experience*
- **BASE + Activity** BASE set of features with *Activity index*
- **BASE + LowerCourt** BASE set of features with *lcDisagreement* and *lcDispositionDirection*
- **BASE + Interruptions** BASE set of features with *Interruption difference*
- **BASE + TCI** BASE set of features with *Topic Chain Index*
- **ALL** contains all features

4.4.1 Gaussian Naive Bayes

First evaluated model is *Gaussian Naive Bayes* algorithm for classification as a simple classifier. It is based on assumption that all classes occur independent from each other. This implementation of Naive Bayes algorithm also assumes that likelihood of the features is Gaussian[35]:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Its parameters σ_y and μ_y are estimated using maximum likelihood. Prior probabilities are generated by the algorithm itself. No other parameters are set for

		Predicted				Predicted	
		Res	Pet			Res	Pet
Actual	Res	0.2	0.8	Actual	Res	0.6	0.4
	Pet	0.059	0.941		Pet	0.176	0.824

(a) BASE features

(b) Best subset - BASE + Interruptions

Figure 4.12: Confusion matrix for **Gaussian Naive Bayes** classifier

Features	Best F_1	Acc of Best F_1	Best Acc
BASE	0.78	0.667	0.704
BASE + Activity	0.78 (+0)	0.667 (+0)	0.667
BASE + LowerCourt	0.78 (+0)	0.667 (+0)	0.667
BASE + Interruptions	0.8 (+0.02)	0.741 (+0.074)	0.741
BASE + TCI	0.78 (+0)	0.667 (+0)	0.704
ALL	0.778 (-0.002)	0.704 (+0.037)	0.704

Table 4.7: Results (F_1 score and accuracy) for **Gaussian Naive Bayes** classifier with respect to BASE results

the classifier. Table 4.7 shows best results achieved on trained models with all feature subsets. Surprisingly, all models have better accuracy than baseline. Their F_1 scores are also very similar (except for *BASE + Interruptions* model) which indicates that no other additional features provide useful information to the classifier. Not only the model slightly outperforms *BASE* model, its predictions are also less biased toward one class (see Table 4.12). Both accuracy for both classes is above 0.5, whereas model *BASE* is highly biased toward Respondent winner class. Model using all features together slightly under-perform in comparison to model *BASE + Interruptions* in both criteria. So, the best feature subset for *Gaussian Naive Bayes* classifier is *BASE + Interruptions*.

4.4.2 k-Nearest Neighbor

Next evaluated classifier is k-NN (*k-Nearest Neighbor*) algorithm that implements k-nearest neighbor voting strategy. Instead of making any assumptions about prior probability of each class, it tries to group similar samples that belongs to the same prediction class together. Its strength shines when sample space is dense and samples from classes are easily separable. Furthermore, prediction power of the model tend to also rely on the right combination of multiple parameters (on top of the proper set of features) - *k-neighbors*, *neighbor metric* and *nearest-neighbor algorithm*. Experiments with different settings of parameter *k-neighbor* showed no improvement in F_1 score (see Figure 4.14). Hence, *k-neighbor* is set to 2. Different settings of parameters

		Predicted				Predicted	
		Res	Pet			Res	Pet
Actual	Res	0.1	0.9	Actual	Res	0.7	0.3
	Pet	0.059	0.941		Pet	0.235	0.765

(a) BASE features

(b) Best subset - BASE + All

Figure 4.13: Confusion matrix for **k-Nearest Neighbor** classifier

Features	Best F_1	Acc of Best F_1	Best Acc
BASE	0.762	0.63	0.667
BASE + Activity	0.774 (+0.017)	0.741 (+0.074)	0.741
BASE + LowerCourt	0.731 (-0.026)	0.592 (-0.075)	0.592
BASE + Interruptions	0.769 (+0.012)	0.667 (+0)	0.703
BASE + TCI	0.788 (+0.031)	0.741 (+0.74)	0.741
ALL	0.788	0.741 (+0.037)	0.741

Table 4.8: Results (F_1 score and accuracy) for **k-Nearest Neighbor** classifier with respect to BASE results

nearest-neighbor-algorithm and *neighbormetric* also showed no improvement of F_1 score. This situation hints that sample space is too sparse or too dense for this classifier to better approximate it. Table 4.8 show similar performance to Gaussian Naive Bayes classifier. However, there are some peculiarities. Particularly, the best trained model with *BASE + LowerCourt* features performs even below the baseline for dataset. Even model with *BASE* features overcomes it in both criteria. Models with other feature sets performed slightly better than *BASE* set model in terms of F_1 score. Surprisingly, two feature sets have the same best F_1 score. But, their confusion matrices showed that model based on *BASE + TCI* features is significantly biased toward Petitioner winner class. Thus, model with *BASE + All* features makes more reliable predictions. The best k-NN model is *BASE + All* with parameters $k - neighbors = 2$ and the rest of them are default. [58]

4.4.3 Random Decision Forest

This classifier is already introduced in Section 3.3. As mentioned previously, the classifier creates rules which attempt to approximate the sample space. It proven to be rather effective when working with text-based features in Chapter 3. Fine-tuning the algorithm is not easy so only three advanced parameters *max_feats*, *max_depth* and *n_estimators* are selected for tuning. In order to achieve better results, all parameters should be estimated in two rounds. All three are estimated in *round 1* one after each other. Then,

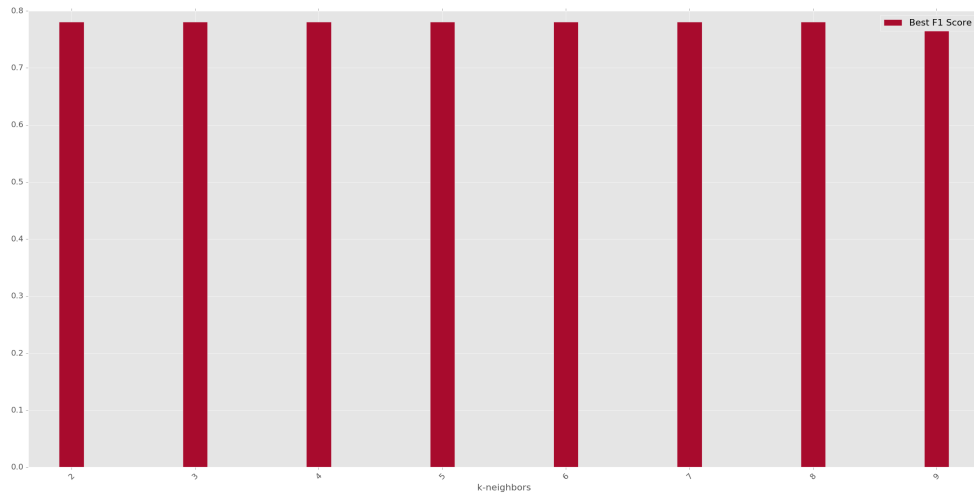


Figure 4.14: Best F_1 score for different settings of parameter k -neighbors for **k-Nearest Neighbor** classifier

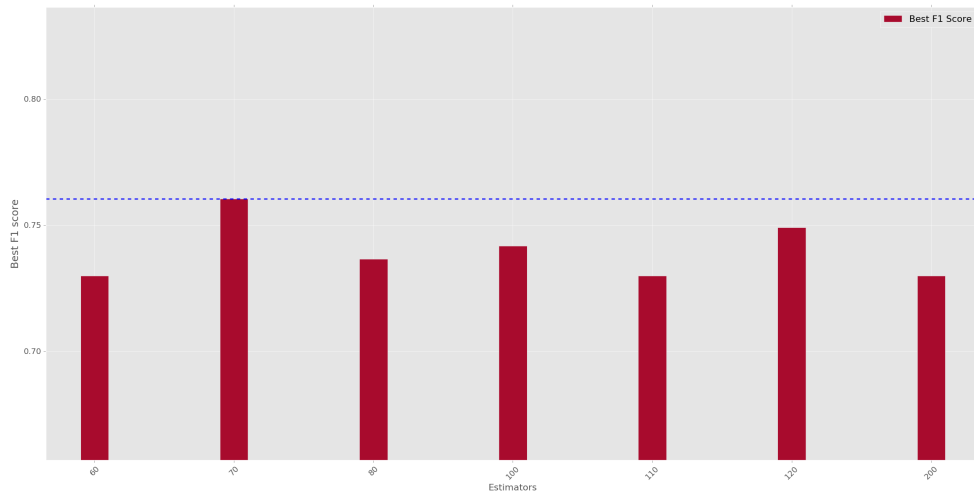


Figure 4.15: Best achieved F_1 score for different settings of n -estimators for **Random Forest Tree** classifier in round 1

the same check on every parameter is done again in *round 2* to ensure their optimal performance.

In *round 1*, we start with parameter *n_estimators*. Figure 4.15 show that after 70 estimators, the gain in F_1 disappears and the models perform worse. At this value, $F_1 = 0.757$. A quick test with higher number of them do not yield any improvement at the moment. Next parameter to estimate is *max - depth*. It affects complexity of individual trees in the random forest. Figure 4.16 shows an interesting behavior of F_1 score. There are three dominant peaks at values 4, 10 and 12 with $F_1 = 0.79$ (peaks at values 10 and 12 have $F_1 = 0.789$). Comparison of their confusion matrices shows that best model with parameter *max - depth* = 10 is more biased toward Petitioner winner class than model with other two parameters. That is the reason why it is selected for further testing. The last parameter, *max_features*, poses restriction on inner structure of each tree. Considering the low number of features, this parameter does not have such a significant. There is no point in using math functions *log₂* or *sqrt* which the algorithm offers, as their results are virtually identical for small numbers. Simpler measure is ratio that determines portion of features which can be used. Due to smaller number of features used by these models (maximum is 11), test shown that best models for each parameter value have approximately the same F_1 score. Given that, it appears that models with higher values achieve more consistent F_1 score results across folds. So, in the end, all features are allowed to be used in every decision tree in random forest.

Round 2 does not provide any improvement after one of the parameters is changed. Best achieved F_1 score does not get better than $F_1 = 0.789$ for model with *BASE* features.

Overall results (see Table 4.9) are very mixed. Clearly, the classifier is sensitive to selected features. Measured F_1 scores suggest that *Activity index*, *LowerCourt* and *Interruptions* are weak on their own. Both F_1 score and accuracy of these models under-perform in comparison to model with only *BASE* features. Even inspection of their respective confusion matrices shows that these models achieve their relatively good F_1 score and accuracy by focusing solely on Petitioner winner class. The only two models that performed better than *BASE* model happened to perform equally good having $F_1 = 0.83$. Also their accuracy is equal (both achieve accuracy 0.741). Given they performed equally in terms of selected metrics, it is not surprising that their confusion matrices are exactly the same. Therefore, it cannot be reliably stated which model is better than the other. However, in long-run, all features together suggest more stable results (using all available features did not cause drop in performance). Therefore, the best settings for random forest tree classifier is a model with parameters *n_estimators* = 70, *max - depth* = 10, *max_features* = *all* that uses all feature set.

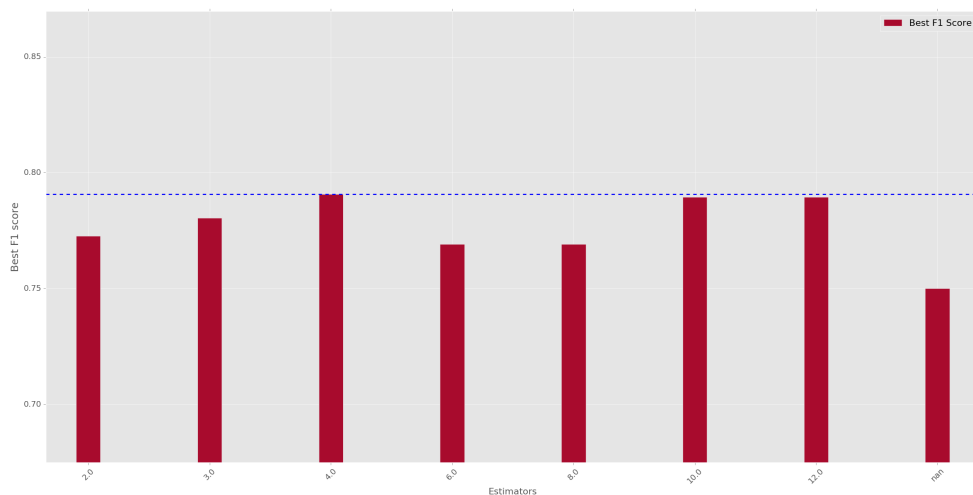


Figure 4.16: Best F_1 score for different settings of parameter $max\text{-depth}$ for **Random Forest Tree** classifier

		Predicted	
		Res	Pet
Actual	Res	0.4	0.6
	Pet	0.118	0.882

(a) BASE features

		Predicted	
		Res	Pet
Actual	Res	0.3	0.7
	Pet	0.0	1.0

(b) Best subset - BASE + All

Figure 4.17: Confusion matrix for **Random Decision Forest** classifier

Features	Best F_1	Acc of Best F_1	Best Acc
BASE	0.789	0.704	0.704
BASE + Activity	0.78 (-0.009)	0.667 (-0.037)	0.667
BASE + LowerCourt	0.769 (-0.02)	0.667 (-0.037)	0.667
BASE + Interruptions	0.78 (-0.009)	0.667 (-0.037)	0.667
BASE + TCI	0.83 (+0.041)	0.741 (+0.037)	0.741
ALL	0.83 (+0.041)	0.741 (+0.037)	0.741

Table 4.9: Results (F_1 score and accuracy) for **Random Decision Forest** classifier with respect to BASE results

4.4.4 Final Evaluation

All evaluated classifiers managed to perform above the baseline defined by distribution of samples in classes in the dataset. Their features are derived from simple NLP metrics. In terms of accuracy, all classifiers achieved the same level - **74.1 %** - which is 10.97 % above the baseline of this dataset. However, each classifier under-performed in a different aspect. The measured results should be understood as demonstration of behavior of the models on given dataset with selected features. Since the dataset is also imbalanced and relatively small, it is harder to select the best training set to achieve stable results. Size of the dataset is given by computational limitations of available working station. Preprocessing of one case file takes between 5-8 minutes⁶² (so, it took approximately 20 hours to extract relevant data).

In terms of F_1 score, the worse model is created by k-NN algorithm for classification. But, when comparing its confusion matrix to the best model created by Random Forest Tree classifier (see Table 4.13 and Table 4.17 respectively), k-NN model appears to provide more stable predictions. The latter mentioned model classifies very few dockets to Respondent winner class. There are no false positives for this class which is alarming due to higher number of false positives in Petitioner winner class. Therefore, k-NN model is considered better model.

This also proves that F_1 score is not an ultimate metric that can compare performance of models from all points of view. Granted, it considers more angles of a model so it is a fast and cheap way to quickly find top n models. But, their final evaluation should be manual to focus on important aspects for the problem at hand.

Comparison of k-NN model and model created by Gauss Naive Bayes algorithm for classification is more straightforward. Again, the latter mentioned has better F_1 score. In this instance, comparison of their confusion matrices (see Table 4.13 and 4.12 respectively) does not give enough clues to claim one of them being better than the other, because they are nearly identical. So, performance of other models (with other subsets of features) is considered (see Table 4.7 and Table 4.8 respectively). Obviously, models based on Gaussian Naive Bayes algorithm perform better with given feature sets. Therefore, this classifier is better in majority of aspects.

The presented results demonstrates that individual simple NLP techniques alone improve performance of the final model only marginally. Although, there are exceptions. Features *Topic Chain Index* and *interruption difference* tend to notably balance results of the model and . On top of this, all of them used together, they rarely worsen performance of a model in a significant way. Only *Activity index* seems to provide very little additional and useful information to model to increase its performance. It is a compound features based on

⁶²CPU: Intel i7 2.4GHz Broadwell, RAM: 8GB DDR3, OS: Windows 10

several other simple NLP metrics. So, this metric may require more analysis to increase its informative value.

Classifier	F_1 score	Accuracy
Gaussian Naive Bayes	0.8	0.741
k-Nearest Neighbor	0.788	0.741
Random Forest Tree	0.83	0.741

Table 4.10: Comparison of performance (F_1 score and accuracy) of best models for each classifier

Conclusion

Goal of this work was two-fold. The main goal was to evaluate behavior of simple NLP techniques when applied on complex problem in discourse analysis. In order to reliably analyze the dataset, a new domain-independent sentence-level sentiment model was proposed as a secondary goal in Chapter 3. Together, these techniques were applied on dataset of Oral Arguments in Supreme Court of the United States to check, if they could be used to create a reliable model that could predict outcome of a case in Chapter 4.

First two chapters of this thesis covered necessary theoretical background to understand its content. Firstly, US Supreme Court was introduced as an institution. The chapter also provided a brief overview of the legal process behind a case and how a case gets heard by the Court. These information helped to understand domain of the problem for the prediction model developed in Chapter 4.

The next chapter focused on all tools and datasets mentioned in the thesis. The first section listed and explained all relevant NLP terms for completeness. Then, it continued discussing every major tool or technique used in the thesis. Each one of them was properly explained with current state of the art trends. It made a case that most of the state of the art approaches are cumbersome and rather complex. Thus, this thesis defended using their simplified versions instead.

Since creating a custom model was the secondary goal of this thesis, the most discussed technology in this section was sentiment analysis. This part emphasized number of challenges which had to be considered when creating a customized model (mainly domain dependence and level of detail). The section also talked over possible ways to overcome them. Their implementation in the custom sentiment model was further described in Chapter 3. The remaining sections of the chapter focused on technological part of the thesis. It listed all used technologies with their respective versions in order to provide all necessary information for anybody to reproduce experiments in thesis. The last section in chapter thoroughly described complementary datasets *General*

Inquirer, *Large Movie Reviews* and *WordNet* used in later chapters in proposed models (sentiment model and model predicting outcome of the US Supreme Court ruling).

Chapter 3 described individual steps required to create a stable and reliable custom sentiment model. It opened up with section that characterized the backbone of the proposed sentiment model - MPQA Dataset. Since sentences in this dataset did not contain explicitly-defined sentiment, the chapter defined key attributes for each relevant sentence in it and developed special ranking algorithm. Then, it ranked all sentences with enough information to create dataset to train and test the customized sentiment model. In this state, the generated dataset contained noise and outliers. There were filtered out in a series of steps supported by detailed analysis of dataset after each step. Despite all the effort, dataset was very imbalanced at that stage. This proven to greatly influence of performance of classifiers, so several *Neutral* and *Positive* sentences were removed to achieve equal number of samples in every class.

Afterwards, we identified features in the dataset which potentially can help to detect proper sentiment in sentence. In comparison to other similar works, the model did not extract token-based features. Instead, it employed two dictionaries to extract meaning of the phrase rather than the word itself. Also, since previous chapter strongly argued that syntax trees are hard to build, it advocated for using n-grams as an approximation. The section explained several other approximations necessary to create the final model.

The last section briefly visited the idea of creating two-step classifier. This idea was quickly abandoned and the section stated reasons why Random Forest Tree ensemble was picked for sentiment model. Then, it described process of fine-tuning its parameters via series of steps to achieve the best overall accuracy and individual accuracy for every class. To prove that final model was useful, it was evaluated on *Large Movie Reviews* corpus.

Lastly, chapter shortly mentioned related works in this area. Then, it stated list of steps to successfully extract text of the hearings from PDF via series of explained steps. Once created, dataset was thoroughly analyzed to devise customized features for the problem like *difference in interruptions* or *difference in sentiment changes*. We picked three fundamentally different classification algorithms to see which approach better handles the simple features and how it reacts to them.

Each proposed classifier was evaluated individually for multiple feature sets to see how well it could approximate the sample space. Since each one of them is based on a different approach, they were fine-tuned separately to reach their potential and the highest F_1 score. These processes were described step by step with relevant views on metrics to allow anybody to replicate their results. Results of the best models for each classifier were compared manually to determine the best fitting model. It turned out that *Gaussian Naive Bayes* classifier overcame the baseline by almost 11 % while still providing reliable results. This result proves that even simple NLP techniques in combination

with simpler classification algorithms bear enough information to predict outcome of a case heard by justices of the US Supreme Court.

On top of that, the results also proved that correct combination of simple NLP techniques was able to improve properties of prediction model. Either they contributed to increased accuracy of the model, or the model provided more reliable results when using them. Only in rare cases, the features deteriorated the final model.

Applications and Future Work

We prove that simple NLP techniques do in fact help to analyze a discourse. There several possible applications of the results and extensions to both parts of this work.

Firstly, the simple NLP techniques mentioned in this work can be applied on other datasets with, preferably, political discourse, like for example ”*Congressional speech data*”⁶³. Further progress in this area may generate other, derivative, metrics. Or, it can just further prove strength of these techniques.

As a matter of fact, the sentiment model can be used for complete different task not related to discourse. The way it was trained, its results should be relevant in for problems which require sentence-level sentiment analysis as well. For example, it may be interesting to see how well it performs on posts from social networks.

Next extension to the work may be improvements of the proposed sentiment model. The simplest improvement may be adding another annotated dataset (ideally from a new domain) to expand its predictive power. Then, adding new dictionaries can also add extra information to the model. Or any internal part of feature extraction process can be replaced, if deemed insufficient. Great candidates for this experiment are algorithms for detection of synonyms and noun phrases. Both of them may significantly influence behavior of Topic-Chain index metric.

Another possibility is to completely replace the customized sentiment model with another model. Currently, there it a number of publicly available sentiment models - for example *AlchemyAPI*⁶⁴. They may provide a relevant benchmark of predictive power for the custom sentiment model. Or, it simply boosts (or sinks) accuracy of this sentiment-based feature in the prediction model for Oral Arguments.

For prediction model, processing more PDF files with Oral Arguments may allow deeper exploration analysis on the dataset to uncover new, relevant, features. Alternatively, bigger dataset provides means to confirm whether the proposed model maintains its predictive power.

⁶³Source at <http://www.cs.cornell.edu/home/lllee/data/convote.html>

⁶⁴AlchemyAPI is part of IBM Watson

CONCLUSION

Lastly, replacing *Tesseract OCR* with another OCR algorithm. Currently, the biggest bottleneck during the processing a PDF file is transformation of PDF file to plain text (takes up 30-50 % of time). Different OCR algorithm may noticeably speed up the preprocessing phase and also increase its overall reliability.

Bibliography

- [1] Jurafsky, D. *Word Meaning and Similarity*. Stanford University.
- [2] Fortune. *The U.S. Supreme Court Will Return with Only 8 Justices*. [cit. 2016-11-13]. Available from: <http://fortune.com/2016/09/30/us-supreme-court-justices/>
- [3] Supreme Court of the United States. *Frequently Asked Questions - Supreme Court of the United States*. [cit. 2016-11-15]. Available from: <https://www.supremecourt.gov/faq.aspx>
- [4] Judicial Learning Center. *How Does the U.S. Supreme Court Work? — The Judicial Learning Center*. [cit. 2016-11-15]. Available from: <http://judiciallearningcenter.org/the-us-supreme-court/>
- [5] Constitution Facts. *Fascinating Facts About The Supreme Court*. [cit. 2016-11-15]. Available from: https://www.constitutionfacts.com/content/supremeCourt/files/SupremeCourt_FascinatingFacts.pdf
- [6] The Beginning of the U.S. Supreme Court. *The Beginning of the U.S. Supreme Court*. [cit. 2016-11-15]. Available from: http://www.socialstudiesforkids.com/articles/ushistory/supreme_court_begins.htm
- [7] American Government. *The Supreme Court in the American System of Government*. [cit. 2016-11-15]. Available from: <http://law2.umkc.edu/faculty/projects/ftrials/conlaw/supremecourtintro.html>
- [8] United States Courts. *Supreme Court Procedures*. [cit. 2016-11-16]. Available from: <http://www.uscourts.gov/about-federal-courts/educational-resources/about-educational-outreach/activity-resources/supreme-1>

BIBLIOGRAPHY

- [9] FindLaw. *Federal vs. State Courts - Key Differences*. [cit. 2016-11-15]. Available from: <http://litigation.findlaw.com/legal-system/federal-vs-state-courts-key-differences.html>
- [10] American Government. *The Supreme Court: What Does It Do?* [cit. 2016-11-15]. Available from: <http://www.ushistory.org/gov/9c.asp>
- [11] Timothy R. Johnson, J. F. S. I., Paul J. Wahlbeck. *The Influence of Oral Arguments on the U.S. Supreme Court*. Master's thesis, [cit. 2016-11-16].
- [12] FindLaw. *How Does the U.S. Supreme Court Work?* [cit. 2016-11-15]. Available from: http://blogs.findlaw.com/law_and_life/2013/10/how-does-the-us-supreme-court-work.html
- [13] New York Times. *Major Supreme Court Cases in 2015*. [cit. 2016-11-16]. Available from: <http://www.nytimes.com/interactive/2015/us/major-supreme-court-cases-in-2015.html>
- [14] CNN. *Top US Supreme Court Decisions Fast Facts*. [cit. 2016-11-16]. Available from: <http://edition.cnn.com/2013/06/21/us/top-u-s-supreme-court-decisions-fast-facts/>
- [15] US Supreme Court. *Frequently Asked Questions (FAQ) — Documents*. [cit. 2016-11-17]. Available from: <https://www.supremecourt.gov/faq-documents.aspx>
- [16] Jurafsky, D. *The Genesis of the Database*. Washington University Law.
- [17] Manning, C. D. *Part-of-Speech Tagging from 97Some Linguistics?* Departments of Linguistics and Computer Science, Stanford University, [cit. 2016-12-02]. Available from: <http://machinelearningtext.pbworks.com/w/file/fetch/48157747/CICLing2011-manning-tagging.pdf>
- [18] World of Computing Magazine. *PARTS-OF-SPEECH TAGGING*. 2009, [cit. 2016-11-28]. Available from: <http://language.worldofcomputing.net/pos-tagging/parts-of-speech-tagging.html>
- [19] Dr. Huong LeThanh. *Part-of-speech tagging*. 2006, [cit. 2016-11-30]. Available from: <http://www.computational-logic.org/iccl/master/lectures/summer06/nlp/part-of-speech-tagging.pdf>
- [20] Greene, B. B.; Rubin, G. M. *Automatic grammatical tagging of English*. Brown University, 1971, [cit. 2016-11-30].
- [21] Brill, E. *A simple rule-based part of speech tagger*. University of Pennsylvania, 1992, [cit. 2016-11-30]. Available from: <http://aclweb.org/anthology/A92-1021>

-
- [22] Ratnaparkhi, A. *A Maximum Entropy Model for Part-Of-Speech Tagging*. Dept. of Computer and Information Science , University of Pennsylvania, [cit. 2016-12-03]. Available from: <http://www.aclweb.org/anthology/W96-0213>
- [23] Kristina Toutanova, C. M., Dan Klein; Singer, Y. *Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network*. Stanford, 2003, [cit. 2016-12-02]. Available from: <http://machinelearningtext.pbworks.com/w/file/48157747/CICLing2011-manning-tagging.pdf>
- [24] Loria, S. *State-of-the-art Part-of-Speech Tagging in TextBlob*. 2014, [cit. 2016-12-02]. Available from: <http://stevenloria.com/tutorial-state-of-the-art-part-of-speech-tagging-in-textblob/>
- [25] Danqi Chen, C. D. M. *A Fast and Accurate Dependency Parser using Neural Networks*. Computer Science Department, Stanford University, 2014, [cit. 2016-12-05]. Available from: <http://cs.stanford.edu/people/danqi/papers/emnlp2014.pdf>
- [26] Text Mining Online. *Dive Into NLTK, Part IV: Stemming and Lemmatization*. 2014, [cit. 2016-12-06]. Available from: <http://textminingonline.com/dive-into-nltk-part-iv-stemming-and-lemmatization>
- [27] *The Porter stemming algorithm*. [cit. 2016-12-07]. Available from: <http://snowball.tartarus.org/algorithms/porter/stemmer.html>
- [28] Fellbaum, C. (editor). Cambridge, MA: MIT Press, 1998.
- [29] Lin, D. *Automatic Retrieval and Clustering of Similar Words*. Department of Computer Science, University of Manitoba.
- [30] George Aaron Broadwell, T. S. S. S. T. U. B. A. E. L. J. T. L., Jennifer Stromer-Galley; Webb, N. *Modeling Socio-Cultural Phenomena in Discourse*. ILS Institute, University at Albany, SUNY, Lockheed Martin Corporation, [cit. 2016-12-05].
- [31] Sentdex. *Sentiment Analysis - What is it?* [cit. 2016-12-16]. Available from: <http://sentdex.com/sentiment-analysis/>
- [32] Browlee, J. *Predict Sentiment From Movie Reviews Using Deep Learning*. Machine Learning Mastery, 2016, [cit. 2016-12-16]. Available from: <http://machinelearningmastery.com/predict-sentiment-movie-reviews-using-deep-learning/>
- [33] Turkish Language. *Learn Turkish with Manisa Turkish*. [cit. 2016-12-18]. Available from: <http://www.turkishlanguage.co.uk/>

- [34] Google. *Efficient Estimation of Word Representations in Vector Space*. [cit. 2016-12-18]. Available from: <https://arxiv.org/abs/1301.3781>
- [35] Scikit. *Naive Bayes*. [cit. 2016-12-07]. Available from: http://scikit-learn.org/stable/modules/naive_bayes.html
- [36] Katti, R. *Naive Bayes Classification for Sentiment Analysis of Movie Reviews*. RPubS, 2016, [cit. 2016-12-18]. Available from: <https://rpubs.com/cen0te/naivebayes-sentimentpolarity>
- [37] Pak, A.; Paroubek, P. *Twitter as a corpus for sentiment analysis and opinion mining*. 2010, [cit. 2016-12-18].
- [38] James Hong, M. F. *Sentiment Analysis with Deeply Learned Distributed Representations of Variable Length Texts*. Stanford University, [cit. 2016-12-18].
- [39] Swati Soni, A. S. *Sentiment Analysis of Customer Reviews based on Hidden Markov Model*. National Institute of Technology Raipur, 2015, [cit. 2016-12-18].
- [40] TheresaWilson, P. H., JanyceWiebe. *Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis*. University of Pittsburgh, [cit. 2016-12-18]. Available from: <http://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/>
- [41] Bruno Ohana, S. J. D., Brendan Tierney. *Domain Independent Sentiment Classification with Many Lexicons*. Dublin Institute of Technology, 2011, [cit. 2016-12-18].
- [42] Sheehan, E. *Optical Character Recognition*. Research Computing Facility at USC, [cit. 2016-12-14]. Available from: <http://www-bcf.usc.edu/~wdutton/comm533/OCR-SHEE.htm>
- [43] Abby. *Why OCR a PDF*. [cit. 2016-12-14]. Available from: <https://abby.technology/en:kb:tip:pdf-ocr>
- [44] Chirag Patel, D. P., Atul Patel. *Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study*. 2012.
- [45] Bird, E. L., Steven; Klein, E. *Natural Language Processing with Python*. O'Reilly Media Inc, 2009, [cit. 2016-11-24].
- [46] Brown Corpus Manual. *Brown Corpus Manual*. [cit. 2016-11-27]. Available from: <http://clu.uni.no/icame/brown/bcm.html>
- [47] Project Gutenberg Literary Archive Foundation. *Project Gutenberg*. 2016, [cit. 2016-11-27]. Available from: <https://www.gutenberg.org/>

-
- [48] University of Pennsylvania. *Penn Treebank*. [cit. 2016-11-27]. Available from: http://www.ling.ohio-state.edu//research/comp/resources/penn_treebank_descrip.html
- [49] Pang, B.; Lee, L. A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts. In *Proceedings of the ACL*, 2004.
- [50] Harvard University. *How the General Inquirer is used and a comparison of General Inquirer with other text-analysis procedures*. [cit. 2016-12-11]. Available from: <http://www.wjh.harvard.edu/~inquirer/3JMoreInfo.html>
- [51] Maas, A. L.; Daly, R. E.; Pham, P. T.; et al. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. Available from: <http://www.aclweb.org/anthology/P11-1015>
- [52] *MPQA 3.0: Entity/Event-Level Sentiment Corpus*. 2015, [cit. 2016-12-2]. Available from: <http://www.computational-logic.org/iccl/master/lectures/summer06/nlp/part-of-speech-tagging.pdf>
- [53] Brownlee, J. *Boosting and AdaBoost for Machine Learning*. [cit. 2016-12-24]. Available from: <http://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/>
- [54] Yang, Y.; Pedersen, J. O. *A Comparative Study on Feature Selection in Text Categorization*. 1997.
- [55] Nasrallah, C. *Predicting the Supreme Court from oral arguments*. 2015, [cit. 2016-12-28]. Available from: <https://sciencecowboy.wordpress.com/2015/03/05/predicting-the-supreme-court-from-oral-arguments/>
- [56] Hawes, T. *COMPUTATIONAL ANALYSIS OF THE CONVERSATIONAL DYNAMICS OF THE UNITED STATES SUPREME COURT*. Cornell University, 2009, [cit. 2016-12-18]. Available from: <http://drum.lib.umd.edu/handle/1903/9999>
- [57] Feldman, A. *Forecasting Votes in Hellerstedt*. Empirical SCOTUS, 2016, [cit. 2016-12-28]. Available from: <https://empiricalscotus.com/2016/03/04/forecasting-votes-in-hellerstedt/>
- [58] Scikit. *k-Nearest Neighbor*. [cit. 2016-01-06]. Available from: <http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

Acronyms

- CSV** Comma-separated values
- OCR** Optical Character Recognition
- OOP** Object-Oriented Programming
- NLP** Natural Language Processing
- PDF** Portable Document Format
- SVM** Support-Vector Machine
- TCI** Topic-Chain Index

Ranking constants MPQA

Here is complete list of constants used in ranking algorithm for MPQA corpora in Chapter 3:

Table B.1: "Intensity" attribute

Annotation	Value
neutral	0
low	0.2
low-medium	0.35
medium	0.5
medium-high	0.75
high	1
high-extreme	1.5
unknown	0

Table B.2: "Expression Intensity" attribute

Annotation	Value
neutral	1.01
low	1.25
medium	1.5
high	2
extreme	2.5
unknown	1.0

Table B.3: "Polarity" attribute

Annotation	Value
positive	1
negative	-1
both	0.9
neutral	0.01
uncertain-positive	0.8
uncertain-negative	-0.8
uncertain-both	0.75
uncertain-neutral	0.05
unknown	0

Table B.4: "Attitude type" attribute

Annotation	Value
other	0
other-attitude	0.001
arguing-neg	-0.75
arguing-pos	0.75
sentiment-pos	0.5
sentiment-neg	-0.5
agree-pos	0.25
agree-neg	-0.25
intention-pos	0.6
intention-neg	-0.6
speculation	0.01
specilation	0.01
unknown	0

Table B.5: "Attitude uncertain" attribute

Annotation	Value
somewhat-uncertain	0.9
very-uncertain	0.5
unknown	1.0

MPQA features list

Table C.1: List of extracted features - Comparison features

# Features	List of Features
5	<ul style="list-style-type: none"> • more words with priorpolarity <i>Positive</i> being <i>strongsubj</i> than words with priorpolarity <i>Negative</i> being <i>strongsubj</i> • more words with priorpolarity <i>Positive</i> being <i>weaksubj</i> than words with priorpolarity <i>Negative</i> being <i>weaksubj</i> • more words with priorpolarity <i>Positive</i> than words with priorpolarity <i>Negative</i> • more words with priorpolarity <i>Positive</i> than words with priorpolarity <i>Neutral</i> • more words with priorpolarity <i>Negative</i> than words with priorpolarity <i>Neutral</i>

Table C.2: List of extracted features - Category Counts

# Features	List of Features
34	<p> <i>Negations</i> # <i>Negative</i> priorpolarity # <i>Both</i> priorpolarity # <i>Neutral</i> priorpolarity # <i>Positive</i> priorpolarity # <i>Negative</i> priorpolarity with type <i>strongsubj</i> # <i>Positive</i> priorpolarity with type <i>strongsubj</i> # <i>Neutral</i> priorpolarity with type <i>strongsubj</i> # <i>Both</i> priorpolarity with type <i>strongsubj</i> # <i>Negative</i> priorpolarity with type <i>weaksubj</i> # <i>Positive</i> priorpolarity with type <i>weaksubj</i> # <i>Neutral</i> priorpolarity with type <i>weaksubj</i> # <i>Both</i> priorpolarity with type <i>weaksubj</i> # <i>All words</i> with type <i>weaksubj</i> # <i>All words</i> with type <i>strongsubj</i> # <i>Hostile</i> words with type <i>weaksubj</i> # <i>Strong</i> words with type <i>weaksubj</i> # <i>Active</i> words with type <i>weaksubj</i> # <i>Passive</i> words with type <i>weaksubj</i> # <i>Positiv</i> words with type <i>weaksubj</i> # <i>Negativ</i> words with type <i>weaksubj</i> # <i>Hostile</i> words with type <i>strongsubj</i> # <i>Strong</i> words with type <i>strongsubj</i> # <i>Active</i> words with type <i>strongsubj</i> # <i>Passive</i> words with type <i>strongsubj</i> # <i>Positiv</i> words with type <i>strongsubj</i> # <i>Negativ</i> words with type <i>strongsubj</i> # <i>Hostile</i> words # <i>Strong</i> words # <i>Active</i> words # <i>Passive</i> words # <i>Positiv</i> words # <i>Negativ</i> words # <i>Yes</i> words # <i>No</i> words </p>

Table C.3: List of extracted features - Negation-related features

# Features	List of Features
40	<p data-bbox="560 517 1228 584">Counts number of words, of it the sentence satisfies following rules:</p> <ul data-bbox="603 622 1126 1794" style="list-style-type: none"> <li data-bbox="603 622 927 656">• <i>Active</i> before negation <li data-bbox="603 685 906 719">• <i>Active</i> after negation <li data-bbox="603 748 943 781">• <i>Passive</i> before negation <li data-bbox="603 810 922 844">• <i>Passive</i> after negation <li data-bbox="603 873 935 907">• <i>Hostile</i> before negation <li data-bbox="603 936 914 969">• <i>Hostile</i> after negation <li data-bbox="603 999 890 1032">• <i>Yes</i> before negation <li data-bbox="603 1061 869 1095">• <i>Yes</i> after negation <li data-bbox="603 1124 880 1158">• <i>No</i> before negation <li data-bbox="603 1187 863 1220">• <i>No</i> after negation <li data-bbox="603 1249 932 1283">• <i>Negate</i> before negation <li data-bbox="603 1312 911 1346">• <i>Negate</i> after negation <li data-bbox="603 1375 1126 1408">• <i>Negative</i> priorpolarity before negation <li data-bbox="603 1438 1106 1471">• <i>Negative</i> priorpolarity after negation <li data-bbox="603 1500 1121 1534">• <i>Positive</i> priorpolarity before negation <li data-bbox="603 1563 1101 1597">• <i>Positive</i> priorpolarity after negation <li data-bbox="603 1626 1112 1659">• <i>Neutral</i> priorpolarity before negation <li data-bbox="603 1688 1091 1722">• <i>Neutral</i> priorpolarity after negation <li data-bbox="603 1751 1078 1785">• <i>Both</i> priorpolarity before negation <li data-bbox="603 1814 1058 1848">• <i>Both</i> priorpolarity after negation

Table C.4: List of extracted features - N-gram

# Features	List of Features
29	<ul style="list-style-type: none"> • number n-grams with length 2, 3 or 4 in which words <i>Negative</i> with priorpolarity <i>positive</i> are before words <i>positive</i> • number n-grams with length 2, 3 or 4 in which words <i>Negative</i> with priorpolarity <i>negative</i> are before words <i>positive</i> • number n-grams with length 2, 3 or 4 in which words <i>Negations</i> before words with <i>positive</i> priorpolarity • number n-grams with length 2, 3 or 4 in which words <i>Negations</i> before words with <i>negative</i> priorpolarity words • number n-grams with length 2, 3 or 4 in which words <i>Negations</i> before words with <i>neutral</i> priorpolarity words • number n-grams with length 2, 3 or 4 in which words <i>Hostile</i> are before words <i>negative</i> priorpolarity words • number n-grams with length 2 or 3 in which words <i>Persist</i> are before words with <i>positive</i> priorpolarity words • number n-grams with length 2 or 3 in which words <i>Pleasur</i> are before words with <i>positive</i> priorpolarity words • number n-grams with length 2 or 3 in which words <i>Weak</i> are before words with <i>positive</i> priorpolarity words • number n-grams with length 2 or 3 in which words <i>Active</i> are before words with <i>positive</i> priorpolarity words

Table C.5: List of extracted features - POS n-grams

# Features	List of Features
52	<p>Number of words before and words after which follow these rules in n-grams with length 4 or 5:</p> <ul style="list-style-type: none"> • Adjectives before <i>Positiv</i> noun • Adjectives or adverbs before <i>Active</i> noun • Adjectives before <i>Strong</i> noun • Adjectives before <i>Negativ</i> noun • <i>Negation</i> adjectives before words with priorpolarity <i>positive</i> • Adjective or adverbs before words with priorpolarity <i>positive</i> • Adjective or adverbs before words with priorpolarity <i>negative</i> • Adjective or adverbs before words with priorpolarity <i>neutral</i> • Adjective before words with priorpolarity <i>positive</i> • Adjective before words with priorpolarity <i>negative</i> • Adjective before words with priorpolarity <i>neutral</i> • <i>Negation</i> adjectives before nouns with priorpolarity <i>positive</i> • Adjective before words <i>Hostile</i> noun

Table C.6: List of extracted features - Other features

# Features	List of Features
1	Punctuation

Command-line script

Standard way to run the script is:

```
python politics.py -c controller-name -t task-name
```

Every *task* may have different set of additional required (or optional) parameters.

D.1 Controller - Inquirer

This controller contains task:

- **read-corpus-raw** which transforms original *General Inquirer* dataset to normalized and clean dataset in format known by this script (see Table D.1 for list of supported parameters)

D.2 Controller - Largereviews

This controller contains two tasks:

- **read-raw-corpus** which transforms original *Large Movie Views* dataset with reviews to a single file in which one line represents one sentence from a review (it takes no additional parameters)

Parameter	Description	Required
-f	Path to input file (default value is path to the dictionary)	False
-fout	Path to output file (default value is path to the dictionary <code>./corpora/processed/</code>)	False

Table D.1: Parameters for task `read-corpus-raw` in controller

- **read-raw-corpus-overall** adds sentiment information from the source dataset to each sentence in normalized dataset (it takes no additional parameters)

D.3 Controller - Mpqa

This controller contains four tasks:

- **read-corpus-sentences-raw** extracts all sentences from *MQPA* dataset and puts them into a single file (see Table D.1 for list of supported parameters)
- **read-corpus-annotations-raw** reads *MQPA* dataset and extracts all annotations with their attributes into a single file (see Table D.1 for list of supported parameters)
- **read-corpus-processed-clues** transforms *MPQA Subjectivity Lexicon* into normalized format understood by this script (see Table D.1 for list of supported parameters)
- **combine-corpora-clues** combines data from *General Inquirer* and *MPQA Subjectivity Lexicon* together into one file (it takes no additional parameters)

D.4 Controller - Pdf

This controller contains five tasks:

- **read-pdf** transforms PDF file into a plain text and produces file with the same name and extension ".plain"; then it strips all unwanted characters away and produces another file with the same name and extension ".clean" (see Table D.2 for list of supported parameters)
- **process-pdf** reads plain text and strips unwanted characters away (in case the source file was not produced by task *read-pdf*); result is saved in file with the same name with extension ".clean" (see Table D.1 for list of supported parameters)
- **extract-parts** reads cleaned text and splits it into turns; result is saved in file with the same name with extension ".dialog" (see Table D.3 for list of supported parameters)
- **generate-sentences** reads data split into turns and further splits them into sentences and the result saves in file with same name and extension ".sentences" (see Table D.3 for list of supported parameters)

Parameter	Description	Required
-f	Path to PDF file	True
-m	Type of PDF file ("bottom-middle" or "top-right")	False

Table D.2: Parameters for task "read-pdf" in controller *Pdf*

Parameter	Description	Required
-f	Path to input file	True

Table D.3: Parameters for tasks in controller *Pdf*

Parameter	Description	Required
-f	Path to input file	True
-m	PDF Mode	True

Table D.4: Parameters for task *all-tasks* in controller *Pipeline*

- **generate-pos** reads data split into sentences, assigns them POS tags and the result saves in file with same name and extension "pos" (see Table D.3 for list of supported parameters)

D.5 Controller - Pipeline

This controller contains three tasks:

- **all-tasks** runs all tasks necessary to read PDF file and output its reports (see Table D.4 for list of supported parameters)
- **all-tasks-dir** runs all tasks necessary to read PDF file and output its reports for a given directory - file *modes.info* is reserved for information about format of each PDF file (see Table D.1 for list of supported parameters)
- **all-tasks-dir-no-pdf** runs all tasks necessary except from reading PDF file and output its reports for a given directory; also only a selected step can be performed on all files in the directory (see Table D.5 for list of supported parameters)

D.6 Controller - Reports

This controller contains four tasks:

Parameter	Description	Required
-f	Path to input file	True
-m	PDF Mode	False
-s	Requested step (e. g. "step-5")	False

Table D.5: Parameters for task *all-tasks-dir-no-pdf* in controller *Pipeline*

Parameter	Description	Required
-f	Path to directory with input files	True

Table D.6: Parameters for task *all-tasks-dir* in controller *Pipeline*

Parameter	Description	Required
-f	Path to input file	True
-o	Optional output directory for report	False

Table D.7: Parameters for tasks *nlp-reports* and *basic-reports* in controller *Reports*

- **nlp-reports** takes in name of file in directory containing processed PDF file and runs all NLP reports on them (see Table D.7 for list of supported parameters)
- **basic-reports** takes in name of file in directory containing processed PDF file and runs all basic reports on them (see Table D.7 for list of supported parameters)
- **merge-preprocessed** walks over all files of the same type in subdirectories of directory "*parsed-data*" and merges them into a single file with name being the extension name (it takes no additional parameters)
- **merge-reports** walks over all files of the same type in subdirectories of directory "*report-data*" and merges them into a single file with name being the extension name (it takes no additional parameters)

D.7 Controller - Sentiment

This controller contains six tasks:

- **prepare-training-data** runs ranking algorithm on normalized MPQA dataset to prepare training data (it takes no additional parameters)
- **normalize-values** normalizes ranked data in *MPQA* dataset to values between $< -1, 1 >$ (it takes no additional parameters)

Parameter	Description	Required
-fin	Input file	True
-fout	Optional output directory	False

Table D.8: Parameters for tasks in controller *Sentiment*

- **extract-features** extracts features for each sentence in file with extension "*sentences*" and stores them in file with the same name and extension "*.features*" (see Table D.8 for list of supported parameters)
- **calculate-sentiment** reads "*.feature*" file and calculates sentiment for each sentence in it; the result is stored in file with the same name and extension "*.sentiment*" (see Table D.8 for list of supported parameters)
- **calculate-per-turn** reads "*.sentiment*" file and calculates sentiment per turn by combining sentences in one turn together; as a result file with same name and extension "*dialog*" is updated with the sentiment of turns (see Table D.8 for list of supported parameters)
- **calculate-sentiment-all-files** calculate sentiment (run tasks *calculate-sentiment* and *calculate-per-turn*) on all files in given director (see Table D.8 for list of supported parameters)

Contents of enclosed CD

	readme.txt	the file with CD contents description
	src	the directory of source codes
	politics	the directory of source codes of the thesis
	tesseract-train ...	the directory of source codes of the trained OCR
	algorithm	
	thesis	the directory of \LaTeX source codes of the thesis
	text	the thesis text directory
	DP_polak_lukas_2017.pdf	the thesis text in PDF format
	DP_polak_lukas_2017.ps	the thesis text in PS format