

# On Classifying the outcomes of Legal Motions

by

Oluwaseun Quadri Cardoso

A thesis  
presented to the University of Waterloo  
in fulfillment of the  
thesis requirement for the degree of  
Master of Mathematics  
in  
Computer Science

Waterloo, Ontario, Canada, 2024

© Oluwaseun Quadri Cardoso 2024

### **Author's Declaration**

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Abstract

Conflict is inherent to the human condition, and socially acceptable methods of resolving conflict typically begin with dialogue, compromise, or negotiation. When these efforts fail, the legal process, often culminating in the courtroom, becomes the final recourse. Legal practitioners strive to position themselves advantageously by predicting the outcomes of legal disputes, increasingly relying on predictive tools to navigate the complexities of the courtroom.

This thesis investigates the feasibility of predicting the outcomes of legal motion disputes using supervised machine learning methods. While previous research has predominantly utilized expertly hand-crafted features for judicial predictions, this study explores the use of written arguments, known as briefs, as the only basis for prediction. We trained 36 classifiers to predict the outcomes of legal motions and compared their performance to that of a baseline model.

The best-performing classifier achieved an accuracy of 62% on the test dataset. However, statistical analysis reveals that the performance of the top 10 classifiers is not statistically different from the baseline model. These findings suggest that, among the top-performing classifiers, there is no conclusively dominant approach for predicting legal motion outcomes using briefs. The thesis also offers theoretical considerations to explain these results.

## Acknowledgements

Along the journey of writing this thesis, I received tremendous support from faculty members from Uwaterloo.

I want to thank Professor Maura Grossman and Professor Gordon Cormack for their patience, effort and support in helping me complete this thesis. Thank you, Maura, for your sponsorship and the introduction to the field of AI and Law, and thank you, Gordon, for your guidance on conducting research.

I want to thank Pure Math Professor Rahim Moosa for proofreading the mathematical arguments made in this paper and rejuvenating my love for mathematics. He demonstrated that no matter what the subject is *math is math*.

I thank statistician Professor Jeffrey Negrea for proofreading my thesis. He provided incredible insights into machine learning theory and taught me how to turn an argument into proof.

I thank Computer Science Professor YaoLiang Yu for taking time from his busy schedule to explain the nature of Neural Networks. At the time of writing this, no one still understands how Neural Networks works, but you made it look easy.

I want to thank Pritwish Jana, my friend and former roommate. Thanks again for proofreading my thesis.

Lastly, I send my love to my parents for all their support.

## **Dedication**

To those that challenge the status quo.

To those that break the chain of conformity.

To those that believe in the impossible.

To those that move against the crowd.

And to those that dare to dream of light in a dark room.

I dedicate to you all.

# Table of Contents

|                                             |            |
|---------------------------------------------|------------|
| <b>Author’s Declaration</b>                 | <b>ii</b>  |
| <b>Abstract</b>                             | <b>iii</b> |
| <b>Acknowledgements</b>                     | <b>iv</b>  |
| <b>Dedication</b>                           | <b>v</b>   |
| <b>List of Figures</b>                      | <b>ix</b>  |
| <b>List of Tables</b>                       | <b>x</b>   |
| <b>1 Introduction</b>                       | <b>1</b>   |
| <b>2 Background</b>                         | <b>3</b>   |
| 2.1 Legal Motions . . . . .                 | 3          |
| 2.2 Legal Dockets . . . . .                 | 4          |
| 2.3 Historical Context . . . . .            | 5          |
| 2.4 Data Sources . . . . .                  | 8          |
| 2.4.1 PACER . . . . .                       | 8          |
| 2.4.2 RECAP . . . . .                       | 8          |
| 2.5 Supervised Learning . . . . .           | 9          |
| 2.5.1 Supervised Learning Methods . . . . . | 14         |
| 2.6 Feature Engineering: (TF-IDF) . . . . . | 20         |

|          |                                                              |           |
|----------|--------------------------------------------------------------|-----------|
| <b>3</b> | <b>Dataset Creation</b>                                      | <b>21</b> |
| 3.0.1    | Step Zero: Download Dataset . . . . .                        | 21        |
| 3.0.2    | Details and Limitations of Step zero . . . . .               | 22        |
| 3.1      | Step One: Parse Docket files . . . . .                       | 23        |
| 3.1.1    | Details on retrieveMotionsfrom . . . . .                     | 23        |
| 3.1.2    | Details on retrieveBriefsfrom and checkBriefsExist . . . . . | 24        |
| 3.1.3    | Limitations and Compromises of Step one . . . . .            | 25        |
| 3.2      | Step Two: Retrieve and Categorize Briefs . . . . .           | 25        |
| 3.2.1    | Details on categorizeBrief and its Limitations . . . . .     | 25        |
| 3.3      | Step three: Represent Motions as Folders . . . . .           | 26        |
| 3.4      | Step four: Convert Folders to CSVs . . . . .                 | 27        |
| 3.4.1    | Statistics of the Dataset . . . . .                          | 28        |
| 3.5      | On modelling the Dataset: Graphs versus Sets . . . . .       | 28        |
| <b>4</b> | <b>Experimental Design</b>                                   | <b>30</b> |
| 4.1      | Development Stage . . . . .                                  | 30        |
| 4.2      | Data Processing and Feature Engineering . . . . .            | 31        |
| 4.3      | Building Classifiers . . . . .                               | 32        |
| 4.3.1    | Brief Classifiers . . . . .                                  | 33        |
| 4.3.2    | Motion Classifiers: DeepSets . . . . .                       | 36        |
| 4.4      | Adapting Brief-Classifiers to Motion-Classifiers . . . . .   | 39        |
| 4.5      | Baseline Brief Classifiers . . . . .                         | 41        |
| 4.6      | Evaluating Classifier Performance . . . . .                  | 41        |
| 4.6.1    | Statistical Tests . . . . .                                  | 42        |
| <b>5</b> | <b>Results and Discussion</b>                                | <b>43</b> |
| 5.1      | Results . . . . .                                            | 43        |
| 5.1.1    | Performance Results . . . . .                                | 43        |

|          |                                                      |           |
|----------|------------------------------------------------------|-----------|
| 5.1.2    | Comparing classifiers with baseline . . . . .        | 49        |
| 5.1.3    | Comparing classifiers with dummy . . . . .           | 50        |
| 5.2      | Discussions . . . . .                                | 51        |
| 5.2.1    | Per-brief Ensemble vs Per-motion Classifier. . . . . | 51        |
| 5.2.2    | Can we do better? . . . . .                          | 52        |
| <b>6</b> | <b>Conclusion and Future Work</b>                    | <b>56</b> |
| 6.1      | Future Work . . . . .                                | 56        |
| 6.1.1    | Future Improvement on current work . . . . .         | 56        |
| 6.1.2    | Alternative approaches . . . . .                     | 57        |
|          | <b>References</b>                                    | <b>58</b> |
|          | <b>APPENDICES</b>                                    | <b>63</b> |
| <b>A</b> | <b>Graphs and Results</b>                            | <b>64</b> |
| A.1      | Methods and their Performance . . . . .              | 65        |
| A.2      | Methods and their P-value . . . . .                  | 66        |

# List of Figures

|     |                                                                 |    |
|-----|-----------------------------------------------------------------|----|
| 2.1 | A Docket Page From a case: DW TRUMP (22-22628) from RECAP . . . | 5  |
| 2.2 | Two ROC Curves with the Same AUC but Different Shapes . . . . . | 15 |
| 2.3 | A Decision Tree on Buying a Car . . . . .                       | 17 |
| 3.1 | A liberal estimate of eligible dockets . . . . .                | 22 |
| 3.2 | Dataset Folder Structure . . . . .                              | 27 |
| 3.3 | A Graph Representation of Motions . . . . .                     | 29 |
| 4.1 | Illustration of Fusion . . . . .                                | 40 |

# List of Tables

|     |                                                                 |    |
|-----|-----------------------------------------------------------------|----|
| 2.1 | Confusion Matrix . . . . .                                      | 14 |
| 5.1 | Brief classifiers accuracy . . . . .                            | 44 |
| 5.2 | Confusion Matrix and ROC Curve of top ten classifiers . . . . . | 45 |
| 5.3 | Top-10 Most accurate Classifiers . . . . .                      | 48 |
| 5.4 | Bottom-10 least accurate Classifiers . . . . .                  | 49 |
| 5.5 | P-values of top-10 classifiers . . . . .                        | 49 |
| 5.6 | P-values of bottom ten classifiers . . . . .                    | 50 |

# Chapter 1

## Introduction

Divination is an attempt to gain insight into a question or future situation through an occult ritual. From the priestesses of Delphi in ancient Greece to the *Babaaláwo* of the Yoruba people in West Africa [38], humans have a long history of attempting to predict the future. In the modern world, we use more sophisticated tools to make sense of the past, present, and future [23]. Although times have changed, modern humans still share the same curious spirit as the travellers who once sought the prophecies of the oracles from Delphi.

The problem defined in this thesis is simple: Can we use legal briefs to predict the winner of a legal motion dispute? One way to solve this problem is by asking an expert judge to read the briefs and give their predictions; however, this approach is infeasible unless one has unlimited time and resources. Another approach would be to study the law and explicitly write rules for how a motion should be decided based on the briefs presented. Again, unless one has all the time, energy, and resources, writing rules that cover all the complexity and nuances of legal motion disputes is also infeasible. Legal practitioners, such as lawyers and judges, use their experience and knowledge of the law to make educated guesses about the outcome of legal disputes. However, lawyers are human and thus susceptible to biases and errors in judgment [26]. Fortunately, we inhabit the Information Age, characterized by digitization of analog data, the Internet, social media, and massive amounts of data. To exploit the technological advances of our time, we must turn to machines to help us make sense of the past and predict the future.

The problem of predicting the outcome of legal motion disputes is framed as a supervised binary classification problem, where we build, train, and test models to predict the outcome of motions. The inputs to these models are a set of legal briefs. Briefs are written

legal arguments that either support or oppose the motion presented in court. A motion is a formal request made to a judge, and the outcome is either “grant” or “deny,” referring to whether the motion will be granted or denied by the judge.

Two approaches were taken to predict legal motions: per-brief and per-motion predictions. Per-brief prediction refers to using only a single brief involved in a motion dispute to predict the outcome of the motion. In contrast, per-motion prediction refers to using a set of briefs to predict the outcome of a motion. The briefs could contain exclusively the supporting or opposing briefs or both. We investigate whether using a group of individual brief predictions performs better than using a set of briefs for motion outcome classification. Another question we explore is whether we even need a classifier by comparing our classifiers with a heuristic predictor that always predicts the most frequent outcome in the dataset. Lastly, we evaluate the predictability of all our classifiers by comparing their performances to a baseline classifier.

We employed two supervised machine learning approaches to build classifiers: traditional and deep learning classification techniques. Traditional approaches to classification involve the preemptive procedure of defining the features. Identifying the “right” sets of features is a daunting and challenging task that requires both skill and domain expertise, a process known as feature engineering. We used a natural language processing technique called term frequency-inverse document frequency (TF-IDF) to extract features from the briefs. In modern approaches to binary classification, we forgo the idea of feature engineering and leave the work of finding the “right” features for the deep learning algorithms. We included two deep learning classifiers, namely BERT [49] and Deep Sets [50]. After training and testing our classifiers, we conducted statistical tests to discern their respective performance differences. Finally, we consider the theoretical limitations of predicting legal motion disputes with machine learning.

The thesis structure is as follows: Chapter 2 provides a historical background of using AI to predict legal disputes, a light introduction to legal motions, a formalized definition of supervised learning, and descriptions for each classifier used in the study. Chapter 3 provides in-depth information about the source of the dataset used in this study, the data-modelling paradigm and its limitations. In Chapter 4, we explore the study’s experimental design; this includes details of the implementation of the classifiers used and the statistical tests performed on our classifier’s performances. Chapter 5 is the results and discussion section, and finally, in Chapter 6, we state our conclusions and discuss future research questions we wish to explore.

# Chapter 2

## Background

### 2.1 Legal Motions

As Cornell Law School defines it, “a motion is a formal request made by any party for a desired ruling, order, or judgment” [17]. Motions are an important procedural tool in the legal system, as they allow parties to shape the course and outcome of a case before it reaches the trial stage. Motions are used to resolve contested issues and help move the case forward. Some common types of motions include:

- Motion to Dismiss: Requests the case be thrown out due to a lack of legal basis.
- Discovery Motions: Requests related to the exchange of information and evidence between the parties.
- Motion for a New Trial: Requests a new trial due to an error or irregularity in the previous trial.
- Motion in limine: Pre-trial Requests to exclude or include specific evidence before the trial begins.

The party making the motion is called the “movant” or “support party,” while the opposing party is the “nonmovant” or “opposing party.” Motions can be made at any stage of a legal case - before, during, or after a trial. The process for filing and responding to a motion typically involves:

1. The supporting party serves written notice and a legal memorandum containing arguments to the opposing party and the court.
2. The opposing party has the opportunity to file a written response.
3. The supporting party generally files a reply to the opposition's response.
4. The opposing party may also choose to reply to the reply of the supporting party; this is called a sur-reply
5. The supporting party may also choose to reply to the sur-reply of the opposing party; this is referred to as a sur-sur-reply
6. Once written deliberation between the parties is finished, the judge may choose to issue a ruling on the papers or may hold a court hearing to consider oral arguments before issuing a ruling on the motion.
7. The outcome of the ruling can be denied, granted or something in between, e.g. denied in part or granted in part.

## 2.2 Legal Dockets

A legal docket is an official record or summary of all proceedings, filings, and events in a court case. It serves as a chronological list of everything that has happened in the case, including [29]:

- Documents filed by the parties (motions, briefs, etc.)
- Orders and judgments issued by the court
- Scheduled hearings, trials, and other court appearances and transcripts of them
- Verdicts and rulings

The docket essentially functions as the court's public record for tracking a case's progress and history. It allows judges, attorneys, and parties to quickly review what has occurred and what is upcoming. In the United States, court dockets are considered public records that anyone can access, often through paid online databases like PACER for federal cases. Figure 2.1 shows an example of a docket page of a United States Bankruptcy Court case involving Donald Trump from a free online repository called RECAP.

| Document Number | Date Filed   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |              |
|-----------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| 1               | Aug 18, 2022 | Chapter 11 Voluntary Petition for Non-Individual. Order for Relief Entered. Small Business Chapter 11 Plan due by 2/14/2023. Disclosure Statement due by 2/14/2023, Filed by Barry D. Haberman of 254 South Main Street on behalf of DW TRUMP. (Haberman, Barry) (Entered: 08/18/2022)                                                                                                                                                                                                                                                                                                                                                                                                     |              |
|                 |              | Main Document Voluntary Petition (Chapter 11)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Download PDF |
|                 | Aug 18, 2022 | Receipt of Voluntary Petition (Chapter 11)( 22-22628) [misc,824] (1738.00) Filing Fee. Receipt number A15884185. Fee amount 1738.00. (Re: Doc # 1) (U.S. Treasury)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |              |
|                 | Aug 18, 2022 | Auto - docket of credit card                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |              |
|                 | Aug 19, 2022 | Deficiencies Set: Section 521(i) Incomplete Filing Date: 10/3/2022. Schedule A/B due 9/1/2022. Schedule D due 9/1/2022. Schedule E/F due 9/1/2022. Schedule G due 9/1/2022. Schedule H due 9/1/2022. Summary of Assets and Liabilities due 9/1/2022. Statement of Financial Affairs due 9/1/2022. 20 Largest Unsecured Creditors due 9/1/2022. Declaration of Schedules due 9/1/2022. List of all creditors due 9/1/2022. List of All Creditors Required on Case Docket in PDF Format due 9/1/2022. Corporate Resolution due 9/1/2022. Local Rule 1007-2 Affidavit due by: 9/1/2022. Corporate Ownership Statement due by: 9/1/2022. Incomplete Filings due by 9/1/2022, (Walker, Justin). |              |
| 2               | Aug 19, 2022 | Notice of 341(a) Meeting of Creditors 341(a) meeting to be held on 9/22/2022 at 01:00 PM at Office of UST (TELECONFERENCE ONLY) - CHAPTER 11s. (Walker, Justin). (Entered: 08/19/2022)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |              |
|                 |              | Main Document 341(a) Notice (Chapter 11,12) (BNC)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Download PDF |
|                 |              | Attachment 1 Notice of Ch11 Mtg C/P: Notice Recipients                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Download PDF |
|                 | Aug 19, 2022 | Set Deficiency Deadlines                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |              |
| 3               | Aug 21, 2022 | Certificate of Mailing Re: Notice of 341(a) Meeting of Creditors (related document(s) (Related Doc # 2)) . Notice Date 08/21/2022. (Admin.) (Entered: 08/22/2022)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |              |
|                 |              | Main Document Certificate of Mailing - 341(a) Meeting                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Buy on PACER |
| 4               | Aug 22, 2022 | Amended Voluntary Petition Filed by Barry D. Haberman on behalf of DW TRUMP. (Haberman, Barry) (Entered: 08/22/2022)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |              |
|                 |              | Main Document Amended Voluntary Petition                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Buy on PACER |
| 5               | Aug 26, 2022 | Order Scheduling Initial Case Conference Signed On 8/26/2022. With hearing to be held on 9/29/2022 at 10:00 AM at Videoconference (ZoomGov) (SHL) (Ebanks, Liza) (Entered: 08/26/2022)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |              |
|                 |              | Main Document Scheduling Order                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Buy on PACER |
| 6               | Sep 1, 2022  | Matrix Filed by Barry D. Haberman on behalf of DW TRUMP. (Haberman, Barry) (Entered: 09/01/2022)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |              |
|                 |              | Main Document Matrix                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Buy on PACER |
| 7               | Sep 1, 2022  | Schedules filed: Schedule A/B - Non-Individual, Schedule D - Non-Individual, Schedule E/F - Non-Individual, Schedule G - Non-Individual, Schedule H - Non-Individual Filed by Barry D. Haberman on behalf of DW TRUMP. (Haberman, Barry) (Entered: 09/01/2022)                                                                                                                                                                                                                                                                                                                                                                                                                             |              |
|                 |              | Main Document Schedules                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Buy on PACER |

Figure 2.1: A Docket Page From a case: DW TRUMP (22-22628) from RECAP

## 2.3 Historical Context

In our current age, Artificial Intelligence (AI) has permeated itself in almost every industry and every aspect of our lives. From AI and Education to AI and Medicine, applying AI to Law is an inevitable phenomenon. It surprises many that the intersection of AI and Law goes all the way back to the 1970s. One of the first-ever artificial intelligence systems applied to law, specifically focusing on tax law, was introduced by L. Thorne McCarty [33] in 1977. At the time, his paper titled, “*Reflections on TAXMAN*” was considered a significant breakthrough. The TAXMAN leveraged a rule-based approach where a specific tax legal domain represents a collection of rules.

There are various forms of AI used in Law. These forms can be partitioned into symbolic AI and machine learning approaches. This thesis uses machine learning as the only AI apparatus for predicting the outcome of legal disputes. However, outstanding scientists have made significant contributions by using symbolic AI to determine the outcome of legal disputes. In legal disputes, the opposing parties each provide arguments to persuade the judge that their side is in the right. An area in symbolic AI called computational models of argumentation in law, or simply legal argumentation theory, aims to determine the winners of such disputes by modelling the arguments themselves and using logical rules to infer (which differs from the notion of prediction in machine learning) the winner. This contrasts with our approach, where we build a classifier that maps the input to the target space; in legal argumentation, the arguments themselves are the focus of emulation.

Phan Minh Dung is famous for his significant contribution to the field of symbolic AI in the area of argumentation theory. His seminal work on the acceptability of arguments [20], published in 1995, produced an abstract argumentation framework that played a fundamental role in studying and implementing argumentation mechanisms in AI. Dung’s framework represents disputes as directed graphs in that the nodes are the arguments, and the arrows represent the relationship between arguments. The limitations of this paradigm are that the abstract nature and complexity of this framework make it difficult to apply directly to real-life situations. However, there have been several extensions proposed to Dung’s argumentation framework including, but not limited to, Structured [6, 37], Value-Based [4], Abstract Dialectical [11] and Probabilistic Argumentation Frameworks [30]. With the combination of modern natural language processing (NLP) techniques, there have been recent developments of practical applications that use the argumentation theory to solve real-world problems including, but not limited to crime reporting,[43], medical decision support [21], environmental policy making [3] and legal reasoning [41]. The beauty of the symbolic AI approach is that it is very explainable and transparent. However, in creating a symbolic AI system, engineers must know the rules pertaining to legal disputes or at least have an expert involved in the implementation process. This is where machine learning (ML) approaches outshine symbolic AI. With ML, we can extract the rules (or patterns) within a dataset that determines the winner of a legal dispute.

One of the earlier uses of machine learning for predicting case outcomes was published in 1974 by MacKaay and Robillard [32]. They used the k-nearest neighbours (K-NN) algorithm to produce a classifier that could predict the outcomes of Canadian tax cases using 46 manually crafted encodings of cases in terms of legally relevant factors. Some of these factors include circumstances surrounding the purchase and sale of the property, use of the property during ownership by the private party, the private party’s intention and whether the tax appeal board had upheld the taxpayer’s claim. Another prominent

use of ML in legal dispute prediction is that of Katz et al. (2014) [28]. They used a classical supervised learning model to predict the outcome of US Supreme Court cases. Like MacKaay et al., they used expertly coded features to train their model. The authors aimed to answer two related but distinct questions: “*Will the Court affirm or reverse the lower court’s judgement?*” and “*How will each individual Justice vote on the question before him or her?*” The former question is both relevant and similar to the question we investigate in this thesis; none of the classifiers we created predict how a specific judge would decide on a motion. The authors further stated three criteria that their classifier would have to satisfy. The first criterion is that the model must be usable independent of the year the data was collected; in other words, as stated by the authors, the model should be *general*. The second criterion is the classifier produced must be generalizable or (as stated by the author) *robust*, and finally, the last criterion is that the classifier must be *fully predictive*, which means that the classifier must be able to predict the vote of each individual justice without using (leaky) information about the votes of other justices.

Our research is similar to both Katz’s and MacKaay’s research. We all use classical supervised machine learning methods to predict the outcome of legal disputes, and our models are also *fully predictive* and *general*, but only some of our classifiers are *robust*. Although we differ in the types of legal disputes, we share the same problem class. However, we differ in many other aspects. The first difference is that both Katz’s and MacKaay’s experiments used experts to encode the relevant legal features of the case in the creation of their dataset. No expert human intervention was used to create our dataset. We used the raw text documents and converted them to vectorized features using TF-IDF for all our classifiers except for BERT. Reliance on experts to encode relevant features can be time-consuming and expensive. Thus, our thesis showcases a cheaper alternative to judicial decision prediction. Another difference is that we used both classical and deep learning supervised classifiers and compared their differences in performance. Deep learning models, specifically Large Language Models (LLM), don’t need feature engineering because they rely on raw text. Wei et al. (2023) [49] compared the performances of fine-tuned LLMs: DistilBERT for document-level and snippet-level classification. Despite the ubiquitous use of LLMs for text classification, Yamen et al. [48] suggest that classical supervised ML methods like Support Vector Machines (SVM) perform as well and better in some text classification problems.

There are several types of law, each with its own judicial process. Using supervised ML, researchers have attempted to predict judicial decisions within each domain of law, to name a non-exhaustive list: Human Rights Law [2, 35, 31, 39]; Constitutional Law and Supreme Court decisions [42, 36, 46]; and Criminal Law [13, 44]. These papers report relatively similar performance results, ranging from 55% to 75% accuracy. However, Sert et

al. [36] stood out because their classifier achieved an 85% to 97% in predicting whether an individual application will be granted or dismissed. While the aforementioned contributions are under the broad umbrella of using “predicting judicial outcome”, our thesis focuses on predicting judicial decisions on legal motions, not entire legal cases. The paper titled “Case-level Prediction of Motion Outcomes in Civil Litigation” by McConnell. et al. [34] uses classical ML classifiers to predict the outcomes of motions in civil litigation, specifically focusing on motions to strike in tort and vehicular cases in Connecticut. This paper relates most to our thesis because it predicts legal motions, and they used TF-IDF as one of their feature engineering techniques.

## 2.4 Data Sources

### 2.4.1 PACER

PACER [1], established with the mission of providing electronic access to court records, stands as a cornerstone in the realm of legal information dissemination. Since its inception, PACER has revolutionized how researchers access and retrieve court documents, offering a vast repository of case-related materials spanning various American jurisdictions and legal proceedings. The primary cost associated with accessing documents on PACER is the fee per page viewed. As of when this paper was written, the fee is set at \$0.10 with a maximum of \$3.00 per document. To put things in perspective, our dataset contains about 2000 briefs; if we paid for each document at \$3.00 each, our total would be about \$6000 dollars. Thus, the cost of access is the limiting factor that motivates us to look for alternative data sources.

### 2.4.2 RECAP

Complementing PACER’s infrastructure is RECAP [22], an innovative initiative to augment public access to court documents while circumventing the financial barriers associated with traditional access methods. The core of RECAP is a set of free browser extensions for Firefox, Chrome, and Safari. When installed, these extensions automatically upload any documents or dockets downloaded from PACER to a central repository hosted by the non-profit Free Law Project. This crowdsourced approach has allowed RECAP to amass millions of PACER documents over the years and to make them available to the public for free. The only limitation of RECAP is that not all the available documents that exist

in PACER can be found in RECAP. As a result of RECAP being crowd-sourced, only popular or relevant documents are bought from PACER and shared with RECAP. Thus, not all the documents listed in a docket in PACER will be available in RECAP.

## 2.5 Supervised Learning

To solve the problem of legal motion prediction, we turn to a machine learning paradigm called **Supervised Learning**. Supervised learning can be broadly defined as an algorithmic method for constructing predictors, in our case classifiers, from data. To illuminate the intricacies of the supervised Learning method and differentiate it from other machine learning methods like Reinforcement and Unsupervised learning, We must first describe the relevant components that a typical supervised learning approach would contain. For the remainder of this thesis, we will use a colloquial term, “*learner*,” to refer to any typical supervised learning algorithm. The formalization presented below was adapted from two well-known theoretical machine learning textbooks: “Patterns, predictions, and actions” by Moritz Hardt and Benjamin Recht and “Understanding machine learning” by Shai Shalev-Shwartz and Shai Ben-David [23, 45]. The theoretical formalization below is only necessary to understand the concepts, arguments and proofs discussed in the section 5.2.2.2.

### The learner’s input

A typical supervised learning approach would contain the following:

- **Domain set:** This represents the set of all the possible objects we may wish to use for prediction. Since we want to predict the outcome of legal motions, our domain set,  $\mathcal{X}$ , would be a set where each element is a tuple containing all the relevant briefs for a particular motion. Since the learner is a computer program, machine learning practitioners usually represent the domain set as a vector space. In our case, an element in the domain set would be a matrix, where each column vector represents a document, and the whole matrix represents the set of relevant briefs. In the following sections, we will discuss a feature engineering technique where we convert the legal briefs, which are represented as long texts, into vectors.
- **Target set:** This is the set of the objects we want to predict. In our case, what we want to predict is whether a motion outcome would be “*granted*” or “*denied*”; thus,

the target set would be a two-element set  $\{\text{"granted"}, \text{"denied"}\}$ . To meaningfully represent the target set to the learner, we assigned the numbers 1 and 0 to represent *"granted"* and *"denied"* respectively; thus target set,  $\mathcal{Y}$ , would be  $\{1, 0\}$ . Colloquial instances of the target set are known as *target labels* or *truth values*.

- **Dataset:** The learner uses this dataset to train and produce a classifier. The dataset is a finite sequence of pairs from the domain and target set,  $\mathcal{X} \times \mathcal{Y}$ . Let us denote  $\mathcal{S}$  as the dataset,  $x \in \mathcal{X}$ ,  $y \in \mathcal{Y}$  and  $n$  is the number of pairs in  $\mathcal{S}$ , then  $\mathcal{S} = ((x_1, y_1), (x_2, y_2), \dots, (x_n, y_n))$ . The individual element in the domain set would be the set of briefs for a particular motion, represented as a matrix. At the same time, the target labels would be the decision on the motion, represented as being 0 or 1. An important caveat to mention is that the availability of the target labels in the training dataset is what makes this machine learning approach *supervised*.

### The learner's output

The learner uses the training dataset to produce an output called a *predictor*. A predictor is a function  $h(x)$  that maps elements from the domain set,  $\mathcal{S}$ , to elements in the target set  $\mathcal{Y}$ . The outputs of predictors are called *prediction labels*. A *classifier* is a specific type of predictor that maps inputs into a finite set with discrete values as elements. Since the outcome of a motion is binary,  $\{\text{granted}, \text{denied}\}$  then our predictor has only two elements to map the inputs into; thus, we refer to the outputs of the learners used our experiments as binary classifiers.

### Assumptions on data-generation model

The data-generation model explains and makes assumptions about the training data-generation process. We first assume that the training dataset instances were generated from the same probability distribution, and the learner does not know anything about the probability distribution,  $\mathcal{D}$ , that the training dataset was sampled from. In our case, the data points would be the briefs and motion orders, and these data points are drawn from the same probability distribution,  $\mathcal{D}$ . In other words, the data points are generated from the same *environment*, which in our case is PACER. Technically, PACER did not create the data points; instead, court proceedings from different states in the U.S.A. generated the data points, and PACER is merely publishing the aftermath. Since American states don't share all the same laws and regulations regarding handling legal motions, one could

argue that the data points are not drawn from the same environments. That being said, we assumed that all the data points were generated from the same environment.

Two possible assumptions describe the relationship between the input and the targets: deterministic and stochastic. The deterministic assumption states that there exists some *true function*,  $f : \mathcal{X} \rightarrow \mathcal{Y}$  s.t.  $y^i = f(x^i) \mid \forall (x^i, y^i) \in \mathcal{S}$ , that can correctly map all the elements from and to the domain and target set respectively. The learner doesn't know the true function, but the existence of some true function is vital because it is precisely what the learner is trying to reproduce. On the other hand, the stochastic assumption states that the relationship between the inputs and target is probabilistic. In other words, in a stochastic regime, there is no true function; instead, an event's outcome is described as a conditional probability  $\Pr(y_i|x_i)$ . For the remainder of this chapter, we use the deterministic paradigm to describe supervised learning concepts; however, in subsequent chapters, we provide arguments and evidence that suggest the relationship between briefs and the judicial outcome of a motion is **stochastic**.

## Measure of error

To summarize the concepts mentioned above, the learner takes as input a training dataset,  $\mathcal{S}$ , to produce a classifier usually called the **hypothesis**. The two fundamental assumptions are: the domain instances of the training dataset are sampled from the same, but unknown probability distribution,  $\mathcal{D}$ , and there exists some true function,  $f$ , that *correctly* maps the domain instances to elements in the binary set  $\{0, 1\}$ .

Now that we have a hypothesis,  $h$ , The next question to consider is how our hypothesis,  $h$ , compares to the true function  $f$ . As mentioned earlier, the only information the learner has is the training set  $\mathcal{S}$ ; the learner has no direct access to  $f$ . To solve this issue, we have to investigate the properties of the target labels. The target labels serves as a conduit to  $f$ , because it is assumed that  $\forall (x^i, y^i) \in \mathcal{S} \ y^i = f(x^i)$ . In other words, comparing  $h(x^i)$  to  $f(x^i)$ , is equivalent to comparing the prediction and target labels,  $h(x^i)$  to  $y^i$ . In summary, to effectively analyze the measure of failure or error of our hypothesis, we can rely on the target labels from the training set. The following points below briefly describe some fundamental concepts in supervised learning that formalize the notion of comparing  $h$  to  $f$ .

- **Loss Functions:** To quantify the notion of a mismatch between a prediction and target value,  $h(x) \neq f(x)$ . A loss function takes two inputs and returns a real number, e.g.  $loss(h(x^i), y^i)$ . The output of a loss function is what we define as the *error* or *loss*.

- **Population Risk:** Let's denote  $X$  to be a random variable that represents some measured information of elements in the domain set  $\mathcal{X}$ , that is mapped by some *true function*,  $f_X$ , to 0 or 1. Then the risk of an hypothesis can be defined as the expected value of the loss function of the random variable  $X$ . [23]

$$Risk_{population}(h) := \mathbb{E}[\text{loss}(h(X), f_X(X))] \quad (2.1)$$

Alternatively, suppose  $Mismatches = \{x : h(x) \neq f_X(x)\}$  and the loss functions as an indicator function of  $x$ :

$$\text{loss}(x) = \begin{cases} 1 & x \in Mismatches \\ 0 & otherwise \end{cases} \quad (2.2)$$

Then, the risk of a hypothesis can also be defined as the probability that the hypothesis does not predict the correct label on a random data point generated by the aforementioned distribution, " $\mathcal{D}$ " [45, page. 34].

$$Risk_{population}(h) = \Pr(Mismatches) \quad (2.3)$$

- **Empirical Risk:** As mentioned before, we don't have access to  $\mathcal{D}$ ,  $f$  or  $f_X$ . The only environment we have access to is the dataset set  $\mathcal{S} = ((x_1, y_1), (x_2, y_2), \dots, (x_n, y_n))$ . The dataset is only a sample of the entire population of a datapoint; thus, we can define the empirical risk on the dataset  $\mathcal{S}$  as:

$$Risk_{\mathcal{S}}[h] := 1/n \sum_{i=1}^n [\text{loss}(h(x_i), y_i)] \quad (2.4)$$

- **Dataset Splits:** The dataset,  $\mathcal{S}$ , is split into two partitions called *training* and *testing* dataset. The training dataset is used by the learner to train or fit the classifiers; the testing set is reserved exclusively for evaluating the classifier. The empirical risk of a hypothesis on the training dataset is called the *training error*. Likewise, the empirical risk on the testing dataset is called the *testing error*. In some cases, the training set is further partitioned into two datasets; one is still called the training set, and the other is called the *validation dataset*. The validation dataset's function is to tune the learners' hyper-parameters.
- **Optimization problem:** When embarking on a supervised machine learning project, the hopeful aim is to produce a hypothesis  $h^*$ , from a hypothesis class  $\mathcal{H}$  such that:

$$h^* = \arg \min_{h \in \mathcal{H}} Risk_{\mathcal{S}_{population}}[h] \quad (2.5)$$

However, since the learner has no access to  $\mathcal{S}_{population}$ , the tangle aim of the learner is thus to produce  $h^*$ , from  $\mathcal{H}$  such that:

$$h^* = \arg \min_{h \in \mathcal{H}} Risk_{\mathcal{S}_{train}}[h] \quad (2.6)$$

- **Generalization Gap:** The generalization gap of the hypothesis  $h$  with respect to a population dataset  $\mathcal{S}_{population}$  can be defined as [8]:

$$\Delta_{Gen}(h) = Risk_{\mathcal{S}_{population}}[h] - Risk_{\mathcal{S}_{train}}[h] \quad (2.7)$$

Typically most people don't have access to  $\mathcal{S}_{population}$ , so we substitute  $\mathcal{S}_{population}$  for the complete dataset,  $\mathcal{S} = \mathcal{S}_{train} \cup \mathcal{S}_{test}$ . And, since  $\mathcal{S}_{train} \subset \mathcal{S}$ , then it suffices to say that the **empirical generation gap** is the same as the empirical risk on the testing dataset. Therefore, the empirical generalization gap can be defined as:

$$\Delta_{EGen}(h) = Risk_{\mathcal{S}_{test}}[h]. \quad (2.8)$$

- **Overfitting and Underfitting:** Given the population dataset,  $\mathcal{S}_{population}$ , we say that the hypothesis,  $h$ , **overfits** the training data,  $\mathcal{S}_{train}$ , if there exists another hypothesis  $\hat{h}$  such that :

$$Risk_{\mathcal{S}_{train}}[h] < Risk_{\mathcal{S}_{train}}[\hat{h}] \text{ and } Risk_{\mathcal{S}_{population}}[h] > Risk_{\mathcal{S}_{population}}[\hat{h}] \quad (2.9)$$

Lastly, we say that  $h$  **underfits**  $\mathcal{S}_{train}$  if there exist an  $\hat{h}$  such that :

$$Risk_{\mathcal{S}_{train}}[h] > Risk_{\mathcal{S}_{train}}[\hat{h}] \text{ and } Risk_{\mathcal{S}_{population}}[h] > Risk_{\mathcal{S}_{population}}[\hat{h}] \quad (2.10)$$

Since we don't have access to  $\mathcal{S}_{population}$ , we can substitute it for the testing dataset  $\mathcal{S}_{test}$ .

Table 2.1: Confusion Matrix

|                 | Predicted Positive  | Predicted Negative  |
|-----------------|---------------------|---------------------|
| Actual Positive | True Positive (TP)  | False Negative (FN) |
| Actual Negative | False Positive (FP) | True Negative (TN)  |

## Measure of performance

Besides the empirical risk of our hypothesis and accuracy, there are other ways to measure the capability of our classifier. Since a prediction can either be right or wrong, we can thus expect four different outcomes. The definition of these outcomes can be illustrated with a Confusion Matrix:

Several other performance metrics can be derived from the aforementioned outcomes. The ones relevant to this study are listed and defined below.

- **True Positive Rate (Recall or TPR):**  $\frac{TP}{TP+FN}$
- **False Negative Rate (FNR):**  $\frac{FN}{FN+TP}$
- **False Positive Rate (FPR):**  $\frac{FP}{TP+TN}$
- **True Negative Rate (TNR):**  $\frac{TN}{FP+TN}$
- **Precision:**  $\frac{TP}{TP+FP}$
- **ROC and AUC:** The receiver operating characteristic curve is a curve on the TPR-FPR plane (see 2.2). It shows the performance of a classification model at all the classification thresholds. The area under the ROC curve is referred to as AUC. AUC provides an aggregated measure of performance across all possible classification thresholds. Like Accuracy, AUC doesn't tell us the whole story; for example, in the diagram below, you will see the ROC for the classifiers A and B. The TPR of B reduces linearly with respect to its FPR; however, between points p and q, the TPR of classifier A is stagnant with respect to changes in FPR.

### 2.5.1 Supervised Learning Methods

In our attempt to classify legal motions, we used six supervised learning algorithms, namely: (1) Logistic Regression, (2) Support Vector Machines, (3) Random Forest Trees, (4) K-Nearest Neighbours, (5) Deep Sets and (6) BERT. This section will briefly overview the

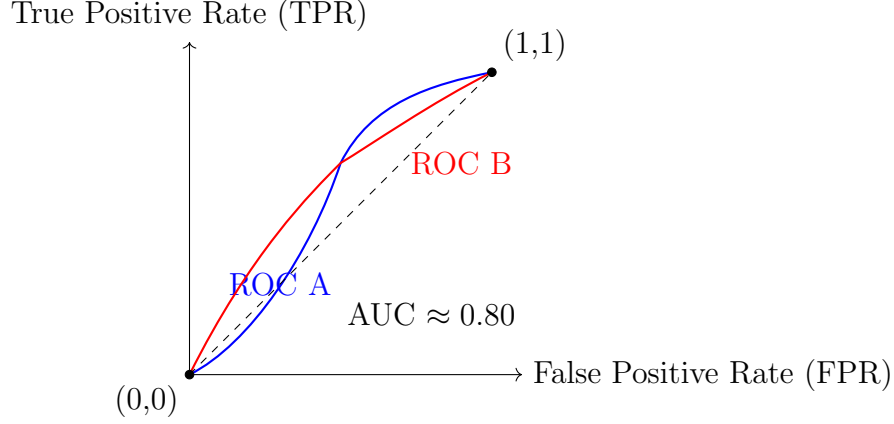


Figure 2.2: Two ROC Curves with the Same AUC but Different Shapes

methodologies and principles behind each of these algorithms and provide a table highlighting their strengths and limitations.

## Logistic Regression

Logistic regression (LR) is a binary classification algorithm that predicts a categorical dependent variable based on one or more independent variables. Logistic regression, as described and adapted from Bishop’s textbook [9], is a simple and interpretable classifier that assumes a linear relationship between features and the log odd of the outcome. Logistic regression can be defined as the composition of a sigmoid function,  $\phi$ , over an affine function,  $a$ . The sigmoid function is also known as a “squashing” function because it maps the entire natural number line to  $[0,1]$ . Let  $\mathbf{w}$  and  $b$  be the weights and  $x$  the input; then the sigmoid function can be defined as:

$$\phi(a) = \frac{1}{1 + \exp(-a)}$$

$$a(x) = \mathbf{w}^T x + b$$

Given that dataset,  $\mathcal{D}$ , where each element is a tuple containing elements from  $\mathcal{X} \times \mathcal{Y}$ , the value produced by  $\phi(a)$  can be interpreted as the probability that the data point,  $x$ , is associated with label 1. The closer  $\phi(a)$  is to 1, the more likely that  $x$  will be *granted*, and vice versa if the closer  $\phi(a)$  is to 0, it is more likely that  $x$  will be *denied*. The loss function of logistic regression called the *binary cross-entropy error function*, is defined as:

$$\text{loss}(h(x^i), y^i) = y^i \log(h(x^i)) + (1 - y^i) \log(1 - h(x^i)) \quad (2.11)$$

## Linear Support Vector Machines

As described by Shalev-Shwartz and Ben-David [45], in Linear Support Vector Machines (SVM) is a binary classification algorithm that aims to find the optimal hyperplane separating two classes. Linear SVM does not use an activation function like logistic regression; instead, it produces a linear hyperplane,  $\langle w, x \rangle + b$ , and uses it as a decision function. The decision function is simply  $\text{sign}(\langle w, x \rangle + b)$ .

To produce the optimal decision function, SVM aims to define a distance, called a *margin*, between the hyperplane and the closest set of points to said hyperplane. A *Hard-SVM* is a type of SVM that constructs a hyperplane that separates the dataset with the largest possible margin. Hard-SVM assumes that the training set is linearly separable. Discovering the appropriate hyperplane is formulated as a quadratic optimization problem:

$$(\mathbf{w}, \mathbf{b}) = \arg \min_{(w,b): |w|=1} |w|^2 \text{ s.t } \forall i, y_i(\langle w, x_i \rangle + b) \geq 1 \quad (2.12)$$

Unlike the values in the target set, SVM expects the target values,  $y_i \in \{1, -1\}$ . If the dataset is not linearly separable, then *Soft-SVM* will be the appropriate choice. In Soft-SVM, the optimization problem can be defined as:

$$(\mathbf{w}, \mathbf{b}) = \arg \min_{(w,b): |w|=1} |w|^2 + C \sum_{i=1}^n \max(0, 1 - y_i(\langle w, x_i \rangle + b)) \quad (2.13)$$

The variable  $C$  is a regularization parameter that controls the trade-off between maximizing the margin and minimizing the classification error. The expression  $\max(0, 1 - y_i(\langle w, x_i \rangle + b))$  is colloquially called the *hinge loss*.

## Random Forest Trees

Our discussion on *Random Forest Trees* (RFT) first begins with a brief introduction to *Decision Trees*. As described in Hastie's textbook [24], decision tree is a tree-based model that makes decisions by following a tree-like structure of questions and answers. The

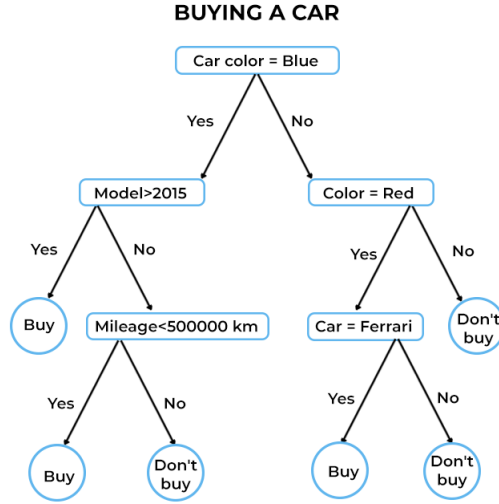


Figure 2.3: A Decision Tree on Buying a Car

questions are posed in the non-terminal nodes, and the answers are the paths to the lead node. The figure 2.3, illustrates the use of a decision tree to create rules for buying a car.

A decision tree is created through a process called recursive partitioning. Within a data point,  $(x_i, y_i)$ , from dataset  $\mathcal{D}$ , let's suppose that each domain point,  $x_i$ , is a vector with  $p$  number of features,  $x_i = [x_i^1, x_i^2, \dots, x_i^p]$  and the two regions produced by a split are defined as  $R1(j, s) = \{x | x^j \leq t\}$  and  $R2(j, s) = \{x | x^j > t\}$ .

The recursive learning algorithm first starts out at a single node tree (the root node), ascribes a feature,  $x_i^t$ , and a threshold,  $t$ , to that node (as the question) and branches (or splits) to two new nodes; with each containing regions (as the answers). This process is done recursively until prematurely stopped by some rule, e.g. number of nodes, depth of tree or error threshold exceeded.

Finding the optimal set of questions on each node is NP-hard. To mitigate against this, we use greedy algorithms to find a *good enough* decision tree. The greedy algorithm iteratively scans through each point in the dataset for the best feature and threshold pair  $(x_i^t, t)$  that produces regions that minimize some loss function, colloquially called a *criterion* or *purity function*. Let node,  $m$ , representing the region,  $R_m$ , with  $N_m$  data points and  $k$  be the class we wish to predict. The two popular loss functions are:

$$\mathcal{S}_k = \{(x, y) \in R_m | y = k\}$$

$$p_{mk} = \frac{\mathcal{S}_k}{|R_m|}$$

Gini Impurity:

$$\sum_k p_{mk}(1 - p_{mk}) \quad (2.14)$$

Cross-entropy:

$$-\sum_k p_{mk} \log(p_{mk}) \quad (2.15)$$

Finally, the random forest trees (RFT) algorithm, introduced by Leo Breiman in 2001 [10], is a slightly modified version of the ensemble method called *bagging*. In classification, an RFT is a *meta-classifier*, where a predefined number of decision trees cast a vote on the class an input belongs to. Each decision tree is trained from a bootstrapped sample of the original training dataset, and the number of variables of each tree,  $d$ , is a function of the total number of features,  $p$ , from a data point,  $x_i = [x_i^1, x_i^2, \dots, x_i^p]$ .

## K-Nearest Neighbors

KNN as described by Hastie and Tibshirani [24] is a simple algorithm unlike other classifiers discussed thus far, KNN doesn't use a loss function to "learn" from them during training. In fact, KNN doesn't have a training phase; it simply stores the dataset in memory and uses it for the prediction of new labels. This algorithm assumes that instances of the domain set can be compared with each other by a metric function. An example of a common metric function is distance, particularly the Euclidean distance between points. Given a query  $x_t$  and a training set, the algorithm tries to find an ordered sequence of  $k$  training points  $[x_1, \dots, x_k]$  closest in the distance to  $x_t$  and then classifies  $x_t$  by using a majority vote among the  $K$  neighbours.

## Deep Sets

Deep Sets is a *permutation invariant* neural network architecture designed to process sets. Introduced by Zaheer et al. [50] in 2017. Deep Sets provide a framework for learning functions that operate on sets of inputs rather than fixed-dimensional vectors. The essence of Deep sets in the context of binary classification can be summarized as the use of a shared encoding function,  $\phi$ , to map each element of a set to vectors in a latent space, where these vectors are aggregated with a permutation invariant function and finally, the aggregated

outputs are then mapped by a decoder,  $\rho$ , to a real number,  $p \in [0, 1]$  representing probability. Since we are dealing with binary classification, the last activation function is a sigmoid function; thus, like logistic regression, the loss function is also a binary cross-entropy error function.

As a result of Deep Sets being a set-based function, we can directly map motions to decisions by representing a motion as a set of briefs. There are three key components of Deep Sets:

- a) Encoder ( $\phi$ ): A neural network that processes each element of the set independently.
- b) Aggregation function: A permutation-invariant function (e.g., sum, max, mean) that combines the encoded elements.
- c) Decoder ( $\rho$ ): A neural network that processes the aggregated representation to produce the final output.

Given  $M = \{br_1, br_2, \dots, br_m\}$ , where  $M$  is a set containing vector representation of briefs, the general form of a Deep Set is:

$$f(M) = \rho\left(\sum_{br \in M} \phi(br)\right) \quad (2.16)$$

## Bert

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based machine learning model for natural language processing developed by Google. Introduced in 2018 by Devlin et al. [19], it is known as the first Transformer-based language model that revolutionized natural language understanding. BERT is primarily used to generate text or word-embeddings. Word embeddings can be used as a feature engineering tool for downstream classification tasks. However, it is also possible to adapt BERT for classification or regression tasks by augmenting the model with an additional output layer. For classification, the output layer will include an activation function like a sigmoid; thus, the entire model can be *fine-tuned* for motion classification using labelled data from the dataset.

## 2.6 Feature Engineering: (TF-IDF)

The brief documents in their raw form are represented as a string of text. Only Language Models like BERT are equipped to take in strings as input; all the other classifiers require that the input be a numerical vector. Therefore, we turn to feature engineering techniques to make use of the remaining classifiers. This thesis's only feature engineering method is Term Frequency-Inverse Document Frequency (TF-IDF).

TF-IDF is widely used in various natural language processing techniques, such as text classification, information retrieval, document summarization, and keyword extraction. Briefs will thus be represented as vectors in a high dimensional space, where each vector component corresponds to a term in the vocabulary of the collection. Adapted from the textbook [12] by Cormack et al. TF-IDF can be describe as such: given  $N$  is the total number of documents in the corpus,  $N_t$  is the number of documents in the corpus containing term  $t$  and  $f_{t,d}$  the number of occurrences of  $t$  within the document  $d$ ; we assign TF-IDF weights to a brief vector as so:

$$TF(t, d) = \begin{cases} \log(f_{t,d}) + 1, & f_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases} \quad (2.17)$$

$$IDF(t) = \log(N/N_t) \quad (2.18)$$

$$TF-IDF(t, d) = TF(t, d) \times IDF(t) \quad (2.19)$$

# Chapter 3

## Dataset Creation

With the proliferation of electronic systems facilitating the storage and dissemination of legal documents, researchers now have unprecedented opportunities to explore the intricate workings of the legal system. Central to this endeavour are platforms such as PACER (Public Access to Court Electronic Records) and RECAP, which are pivotal in allowing this research to exist. To build motion classifiers, we need the set of supporting briefs and opposing briefs of a motion filed during a case. We were able to retrieve about 6 million folders from RECAP. A folder represents a unique docket, where each folder contains documents like motions, order and their other associated briefs and artifacts. Lastly, each folder contains a docket file, which can be an XML or JSON file. The docket files were useful because they contained information, allowing us to easily identify the files in the folder. The creation of the dataset used for the experiment was done in a four-step process.

### 3.0.1 Step Zero: Download Dataset

Founded in 1996, the Internet Archive is a non-profit digital library that aims to provide universal knowledge access. It also offers special tools that allow us to download the RECAP Dataset. We downloaded the dataset on March 7th, 2020, and retrieved 6,603,434 dockets. Here is the link to the [archive](#) of federal court documents. Access to the dataset is legal.

### 3.0.2 Details and Limitations of Step zero

PACER is the primary system for filing legal documents in federal cases in the United States. Thus, the dataset is exclusively composed of US federal cases. We retrieved the dockets via the Internet Archive; all of the documents in the dockets were donated by individuals who bought said documents from PACER. Therefore, there is a selection bias because only the documents people found "interesting" or "relevant" to buy made it into RECAP. It is also important to mention that only some dockets were eligible for motion scrapping; out of 6,603,434 dockets, only 251,625 were eligible dockets containing documents relevant to our study. Figure 3.1 illustrates our investigation of eligible dockets.

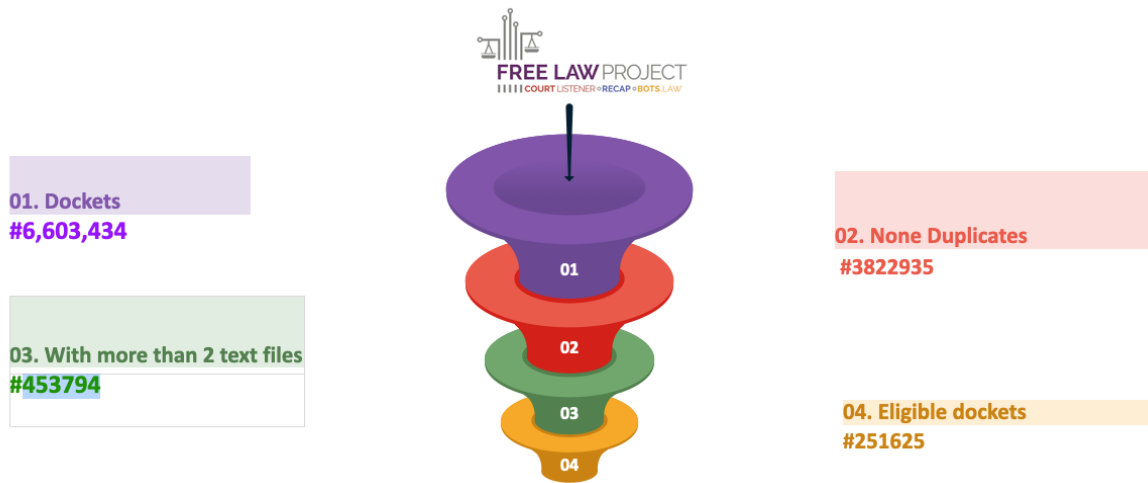


Figure 3.1: A liberal estimate of eligible dockets

The filtration process can be described as follows: out of 6,603,454 dockets only 3,822,935 docket were unique. Only 453,794 dockets contained more than 2 text files files. This criteria was used because, we expect to have at least 2 documents for motion prediction. One for support and the other for opposition. Lastly, only 251,625 unique dockets contained at least 2 documents that contain keywords that could classify them as either a support or opposition brief.

Therefore, selection bias and a relatively small docket sample suggest that our dataset may not be a good representative sample of the population of dockets in PACER.

## 3.1 Step One: Parse Docket files

The first step is to find all the motions mentioned in the docket file, retrieve all the briefs associated with each motion, and determine the decision of the motion, e.g. "grant" or "deny." We compiled a list of paths to all the docket files into a .txt file called, 'dockets'.

---

**Algorithm 1** find motions

---

**Require:** dockets: List of paths to docket files

**Require:** csv: Empty csv

```
1: for docket in dockets do
2:   motions  $\leftarrow$  retrieveMotionsfrom(docket)
3:   for motion in motions do
4:     briefs  $\leftarrow$  retrieveBriefsfrom(motion, docket)
5:     briefs  $\leftarrow$  checkBriefsExist(briefs)
6:     if briefs isn't empty then
7:       addMotionAndBriefs(csv)
8:     end if
9:   end for
10: end for
11: return csv
```

---

The algorithm above is an abstraction of the code used to parse the docket, but the essence is the same. The subsections below provides more details into what the functions **retrieveMotionsfrom()**, **retrieveBriefsfrom()** and **retrieveBriefsThatExist()** accomplish.

### 3.1.1 Details on retrieveMotionsfrom

To understand how the functions work, the important facts about docket files and folders are mentioned below:

- Docket folders only contain one docket file. However, there may be multiple replicas of the docket file each with different extensions: ".Json" or ".XML"
- Information regarding the chronological order of motions and all their respective briefs and related artifacts can be found in the docket file.

- Only documents relevant to a docket will be present in the docket folder.
- Each entry in the docket file has a unique and identifiable number, called a docket entry.
- Each document in a docket folder has the docket entry embedded in its file name.

Depending on whether the docket is a ".XML" or ".JSON" file, **retriveMotionsfrom()** retrieves all the docket numbers of all the motions available in that docket. The description section of the docket entries allowed us to identify motions and the orders of said motions. To be able to find the motions, we first had to find the decisions for motions. We used regex expressions to identify the docket entries that mentioned decisions made on a motion. To find motions that were granted or denied by the judge, we would use the regex expressions: 'granting\s+(\d+)\s+motion' and 'denying\s+(\d+)\s+motion' respectively.

The aforementioned regex expression can be translated as if the docket description contains a phrase that starts with the word 'granting' or 'denying' followed by a number and then finally the word 'motion'. The file associated with the docket entry is an order. With the motions and their judgements captured, we can now use that information to find the briefs associated with that motion. The regex isn't designed to capture motions that were granted in part or denied in part.

### 3.1.2 Details on retriveBriefsfrom and checkBriefsExist

Now that we have the set of motion identification numbers, we can proceed to find the briefs associated with them. This is where the function **retriveBriefsfrom()** comes in. It takes in a motion number as well as the docket file as arguments. Like the **retriveMotionsfrom()**, we use a regex to identify the docket entries that mention a motion's identification number. Typically, the docket entries that contain the supporting or opposing briefs to motions follow a similar pattern.

After analyzing a few relevant docket descriptions, I developed a regex this regex expression to identify the docket entries that contained the identification numbers for the briefs: '\w+.\*\s{motion}\sjoint\s+motion' or '\w+.\*\s{motion}+\s+motion'. For context, the regex expression '{motion}' is a variable. Thus, the motion number is substituted into the regex expression. An explanation of the regex expression is as follows: if there exists a description that contains a phrase that contains a motion number followed by the word "motion" or "joint motion," then treat this document entry as a candidate brief to that motion.

Finally, `checkBriefsExist()` simply checks if the brief is in the folder's current folder. If it is, then the motion, its briefs and other miscellaneous information like jurisdiction type, court, docket identification number, cause, nature of suit, etc., are added to the csv.

It is worth mentioning that the aforementioned regex expression is so broad that the function `retriveBriefsfrom()` would match the same docket entries that would match orders and other types of briefs that are not supporting or opposing briefs. We handled the categorization of briefs in the next step.

### 3.1.3 Limitations and Compromises of Step one

By using a fixed set of regex expressions to identify relevant documents, we implicitly impose an assumption that all the docket descriptions follow a certain pattern. This assumption is not true. There isn't a fixed standard for describing the document on the document section of a docket. Thus, the the regex expression we used can capture some, but not all, of the relevant documents in the docket.

## 3.2 Step Two: Retrieve and Categorize Briefs

The resultant CSV file from the previous script is just an intermediary step. Although we have the briefs, we do not yet know what type of category the brief belongs to: "support" or "opposition." Therefore, the purpose of the second step is to group the briefs into various buckets: "supporting," "opposing," "motion," and "order."

### 3.2.1 Details on categorizeBrief and its Limitations

We used the information in the docket entry's description section to aid us in categorizing briefs. We employed a very simple heuristic for categorizing briefs. Based on a set of keywords, we iterated over every word in the description, and a 'match' occurs when the word from the description is included in the set of keywords. Once we find a match, we set the brief type to the keyword that got matched. The set of keywords is as follows: 'support,' 'opposition,' 'reply,' 'response,' 'ruling,' 'recommendation,' 'report,' 'declaration,' and 'order.' Typically, briefs that are 'responses' are from the opposition team, while briefs that are 'replies' are from the supporting team; we thus further categorized briefs that matched with 'reply' and 'response' as 'support' and 'opposition,' respectively.

---

**Algorithm 2** create motion and brief csvs.py

---

**Require:** docketCsv: The resultant csv from **find-motions.py**

**Require:** motionCsv, briefCsv: Empty csvs

```
1: for docket in documentCsv do
2:   for motion in docket do
3:     motion_info ← extractInfofrom(motion)
4:     briefs ← motion_info['briefs']
5:     for brief in briefs do
6:       brief_type ← categorizeBrief(brief)
7:       brief_info ← extractInfofrom(brief)
8:       brief_info['type'] ← brief_type
9:       add2BriefCsv(brief_info)
10:    end for
11:    add2motionCsv(motionCsv)
12:  end for
13: end for
14: return motionCsv, briefCsv
```

---

This rule is designed to identify briefs whose descriptions follow a **specific** pattern, where the brief type is declared within the first set of words. However, not all brief descriptions follow this pattern. In fact, several nuances exist in the docket entry description that can not be accounted for with this heuristic. To be able to exhaustively account for all the different ways of describing the brief in the docket's entry, we need to employ a machine-learning approach that can learn the nuances and thus correctly classify each brief.

### 3.3 Step three: Represent Motions as Folders

This step was generously implemented by Prof. Cormack. Once the briefs have been categorized, the next step in our pipeline is to extract the text from the PDF and convert each to a .txt file. We then organized the briefs in accordance with their respective motion in a hierarchy of folders. The top-level folder, named 'testset' in 3.2 comprises of a folder named 'files', and two CSV files named 'index-test.csv' and 'index-train.csv'. The folder, files, contains numbered sub-folders, and each sub-folder represents a motion. The files index-test.csv and index-train.csv contain information about the set of sub-folders in files that should be used for testing and training, respectively. Each subfolder in files,

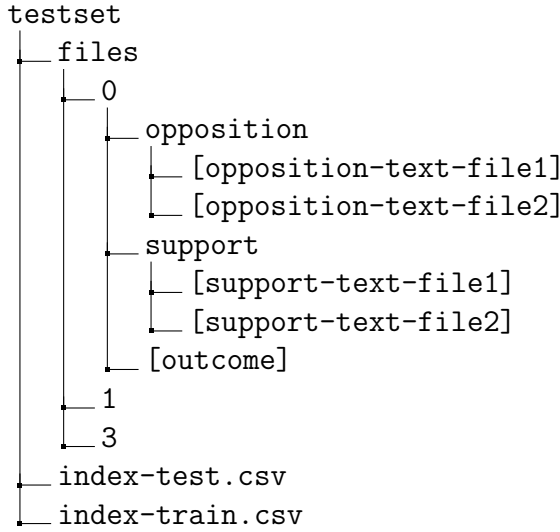


Figure 3.2: Dataset Folder Structure

motion-folder, contains three items of relevance. The opposition and support folder and the outcome. The opposition and support folder contains all the “opposing” and “supporting” briefs, respectively. Lastly, the outcome file is simply an empty file, but its name contains the information of the judgment made on the motion. For example, the outcome file could be named ‘grant’ or ‘deny’. The outcomes of the motions can also be seen in the index files, e.g., ‘index-test.csv.’ or ‘index-train.csv.’ Before the experiments to take place and to preserve fairness, the outcome file is deleted from each folder and outcomes in the index-test.csv is replaced with a “TBD” text.

### 3.4 Step four: Convert Folders to CSVs

The final step in the data pipeline used the *testset* folder produced by the process in step 3 to produce two CSV files called ‘paired\_testset.csv’ and ‘testset.csv’. Each row in the ‘paired\_testset.csv’ can be interpreted as a motion, where the set of briefs relevant to a particular motion is contained in a single row. However, in ‘testset.csv,’ each row contains information about a single brief. Thus, the ‘paired\_testset.csv’ is used for motion predictions, while the ‘testset.csv’ is used for brief predictions.

### 3.4.1 Statistics of the Dataset

| Data Type                | Number of examples |
|--------------------------|--------------------|
| Motions                  | 1175               |
| Supporting briefs        | 1465               |
| Opposing briefs          | 1368               |
| Average brief per motion | 2.4                |
| Granted briefs           | 1364               |
| Denied briefs            | 1469               |
| Granted motions          | 554                |
| Denied motions           | 621                |

## 3.5 On modelling the Dataset: Graphs versus Sets

Let  $G = (V \cup U, E)$  denote a bipartite graph,  $V$  the set of supporting briefs,  $U$  the set of opposing briefs and  $E$  the set of edges connecting the vertices from  $V$  and  $U$  respectively. Thus each vertex in the graph represents a brief. This bipartite graph representation of motions is both intuitive and useful. Taking advantage of the motions' bipartite properties would be an obvious step in building classifiers for motion prediction. Graph-based models, like Graph Neural Networks, would be the most appropriate machine learning tool for building motion classifiers.

However, in reality, most of the motions in the dataset had incomplete vertices. For example, a motion could have 6 briefs, but we could only recover 4 of the 6 briefs available. Because of the loss of the connecting edges that represent the order of the briefs. I thus decided to formalize the motions not as a graph,  $G$ , but instead as a tuple,  $S$ , of supporting and opposing briefs:  $S = (V, U)$ . Figure 3.3 shows the contrasting representations of motions,

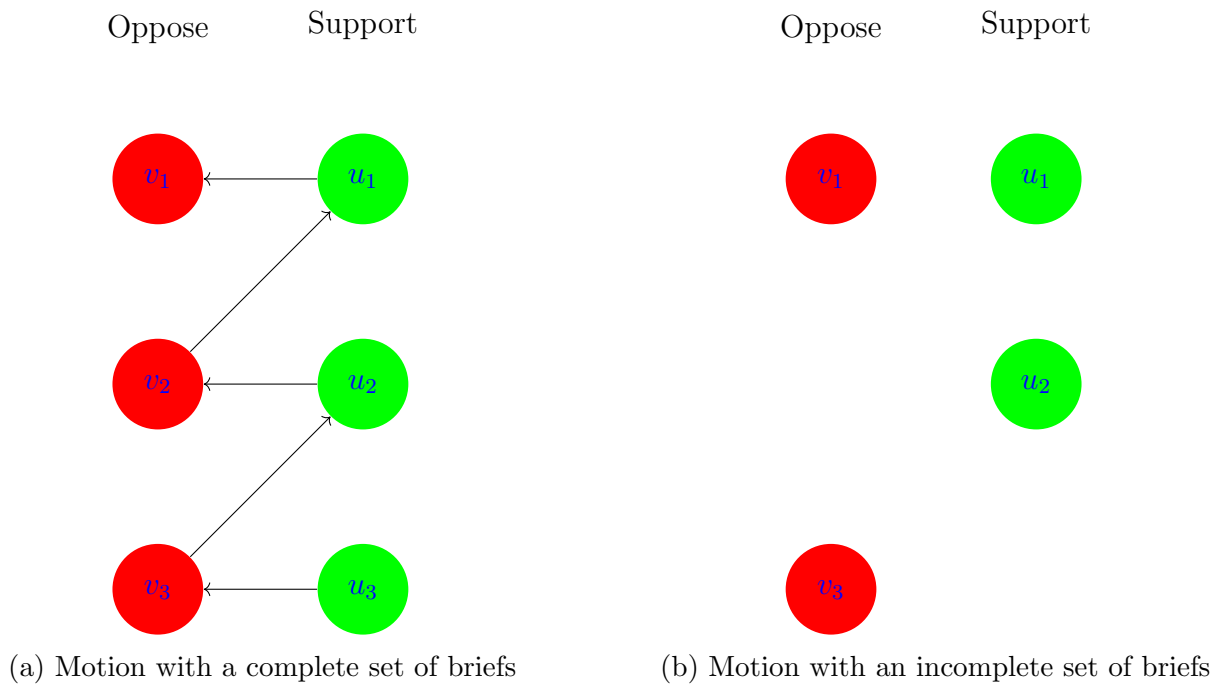


Figure 3.3: A Graph Representation of Motions

# Chapter 4

## Experimental Design

The experiment’s main objective was to build *motion classifiers*, a classifier that uses a set of briefs to predict the outcome of a legal motion. These are three hypotheses we aim to investigate:

- Our classifiers will surpass the baseline classifier.
- Our classifiers will outperform a binary classifier that always predicts the most frequent outcome in the dataset.
- Deep Sets will outperform our ensemble-adapted brief classifiers.

This chapter begins by describing the feature engineering technique done on the dataset. After this, we provide details on two types of models used in this experiment: brief and motion classifiers. We discuss the type of models, implementations, hyper-parameter settings, and the training strategy used to create each classifier. Next, we provide an in-depth overview on the techniques and tools used to evaluate the performance of the classifiers. Finally, we end this chapter by providing a brief overview of the statistical tools used for various hypothesis tests.

### 4.1 Development Stage

As mentioned in the previous chapter, we curated a dataset with 1175 Motions. The final experiment consisted of a train-test dataset split of 675 and 500 motions for the testing and

training dataset, respectively. For the development stage, we used a sample of 500 motions, called the *development dataset*, to create and debug the experiment scripts and to set up some classifiers' hyper-parameters. During the development stage, 20% of the development dataset was set aside to form the *Validation* dataset, which was used for hyperparameter tuning and model selection. In the development stage, we also conducted experiments to decide on the baseline classifier for the final experiment.

## 4.2 Data Processing and Feature Engineering

As discussed in section 3.4, The data creation process resulted in two datasets named *pairedtestset* and *testset*. The *pairedtestset* consists of rows that contain two sets of briefs, one for support and the other for opposition. Thus, the *pairedtestset* is appropriate for motion. The *testset* consists of rows of single briefs, each labelled as a support or opposition brief. Thus, the *testset* is appropriate for brief classifiers.

Both datasets contain information in the form of text. Most of the classifiers we built rely on numerical input features. To convert text into a numerical format, we used a well-known feature engineering technique called Term Frequency-Inverse Document Frequency, also known as TF-IDF. TF-IDF is a powerful technique widely used in natural language processing and information retrieval tasks.

Converting brief documents into TF-IDF feature vectors was achieved with classes from Scikit-learn's [40] library. Specifically, the module, `SKLEARN.FEATURE_EXTRACTION.TEXT` contains utilities for processing text data, including vectorizers that convert text into numerical feature vectors that can be used as input for machine learning models. The three Scikit-learn module classes needed to create TF-IDF feature vectors were `TfidfTransformer`, `CountVectorizer`, and `Pipeline`. Finally, we can describe the feature engineering process in these steps:

- a) **Tokenization and Counting:** We needed to preprocess the briefs before converting legal text to TF-IDF features. This involves intermediary steps like tokenization, stop-word removal and stemming/lemmanization. The intermediary step resulted in a string of *tokens*, which were eventually counted. The `CountVectorizer` is the class responsible for accomplishing these intermediate steps. The final output is a sparse matrix, where each row represents a document, each column represents a unique word/token, and the values are the counts of those tokens in each document.

- b) **Applying TF-IDF Transformation:** The sparse matrix from the previous step is then parsed into an instantiated object of the [TfidfTransformer class](#), resulting in a new matrix containing the TF-IDF values instead of token counts. The number of columns in the matrix is equal to the total number of distinct tokens, known as *vocabulary*. This implies that the number of columns in the TF-IDF matrix or, in other words, the total number of TF-IDF features is a function of the initial set of briefs provided to the CountVectorizer object. Thus, if the training and testing datasets have a different vocabulary, then the training and testing datasets will yield different numbers of TF-IDF features. This is an issue because classifiers built with the training dataset may not be compatible with TF-IDF features from the test dataset.
- c) **Creating a reusable transformation pipeline:** To solve the aforementioned problem and for replication purposes, we used the [Pipeline class](#) to preserve the initial conditions for creating TF-IDF vectors from the training dataset and to save the transformation pipeline for future reuse. With the Pipeline object, we can consistently create TF-IDF features with the same vocabulary as the training dataset. In other words, the Pipeline object will return the same number of features regardless of the dataset's content passed into it.

We created three TF-IDF transformation pipelines: *Support-pipe*, *Opposition-pipe* and *Both-pipes*. The Support-pipes were created exclusively from briefs that were as support, Opposition-pipes from opposition briefs, and the Both-pipe was created with both support and opposition briefs. While most of the classifiers are designed to only predict the legal motion outcomes based on a single brief, a particular classifier, DeepSet, is designed to take in a collection of briefs. Thus, Both-pipes serve TF-IDF vectors to DeepSet classifiers. More details about DeepSets will be discussed in the following sections. Finally, once the transformation pipes are created, they are saved for later use.

## 4.3 Building Classifiers

In this section, we begin by exploring the differences between brief and motion classifiers and why brief classifiers are the stepping stones to motion classifiers. A particular brief classifier, BERT, had inherent limitations that caused sub-optimally performance; we provide details for why this is the case and the measures we used to cope with said limitations. Next, we discuss our approach to adapting brief classifiers to become motion classifiers. Finally, we end this section by going into the details of the implementation, architecture

design, and training strategy on DeepSets. We created multiple brief classifiers; some were trained only with supporting briefs, some with opposing briefs, and the rest were trained on both. Although the brief classifiers were trained on specific types of briefs, the vocabulary in the corpus of supporting briefs likely differs from opposing ones. Thus, the classifier may be able to discriminate between the two types of briefs. However, just because a classifier was trained on a specific kind of brief does not predispose the classifier to disproportionately predict one outcome over the other.

### 4.3.1 Brief Classifiers

Brief classifiers are models trained on *single briefs* to predict motions. Single briefs are individual briefs selected out of the set of briefs that represents a motion. Thus, Per-brief classifiers predict the outcome of a motion with just a single brief. Brief classifiers are not motion classifiers, it predicts the outcome of the motion that the brief belongs to. Motion classifiers consider all the briefs associated with a motion. We do not intend to use brief classifiers to replace motion classifiers; rather, we intend to use brief classifiers as a stepping stone in creating motion classifiers. In the next section, we will delve into two strategies we used to adapt brief classifiers into motion classifiers. The following algorithms were used to create brief classifiers: Stochastic Gradient Descent (SGD), Random Forest, K-nearest neighbours (KNN), Logistic Regression, linear support vector machine (LinearSVC), and Deep Bidirectional Transformers (BERT).

### Building, Training and Saving Classical Machine Learning Classifiers

All the brief classifiers, excluding BERT, were implemented with Python’s Scikit-learn library (version 1.4); thus, details about the implementation and descriptions of the hyperparameters can be investigated on their official website [40]. Below are the specific hyperparameters for each of the Scikit-learn models used in this experiment:

- **SGDClassifier:** `loss='hinge', penalty='l2', alpha=1e-3, random_state=42, max_iter=5, tol=None`
- **RandomForestClassifier:** `n_estimators=1500, criterion="entropy", oob_score=True, max_features=1`
- **KNeighboursClassifier:** `n_neighbors=5, algorithm='auto', p=2, metric='minkowski'`

- **LogisticRegression:** solver='liblinear', dual = False, tol = 0.0001, C = 1, fit\_intercept = True, intercept\_scaling= 1
- **LinearSVC:** random\_state=42, tol=1e-5 , penalty= "l2", loss: "squared\_hinge", dual= True, C = 1, fit\_intercept= True

Each classifier was fitted on the full training dataset, no further partitioning to derive the validation set was needed. After this, the brief classifiers were saved for future use.

## Building, Training and Saving BERT

The methodology employed in this code involves fine-tuning a pre-trained language model, specifically the BERT model, for a sequence classification task using the Hugging Face Transformers library and the PyTorch deep learning framework. HuggingFace has many pre-trained models in its online repository. Using [HuggingFace Transformers library](#), we can download and use pre-trained Large Language Models for sequence classification by specifying the model's *checkpoint* in HuggingFace's AutoModelForSequenceClassification API. Pre-trained models, especially large language models like BERT, often exhibit cutting-edge capabilities like open-ended generation, multi-task learning, and few-shot learning. Training BERT from scratch was an option for us, but the benefits of pre-trained models in speed, performance, and cost-effectiveness made them a compelling choice over training from scratch.

**Data Preparation for BERT:** The dataset is first split into training and validation datasets. Unlike the classical machine learning classifiers, the briefs don't need to undergo any feature engineering process. Each brief is loaded and tokenized using the AutoTokenizer from the Hugging Face library. Once the datasets have been tokenized, we encapsulate them in a Pytorch Dataloader. A Dataloader combines a dataset and a sampler, providing an iterable over the given dataset.

**Building Classifiers:** We followed the standard training loop method for training Neural Networks. As shown in Algorithm (3), the training loop involves iterating over each batch in the training data loader, computing predictions, calculating the loss, performing backpropagation, and updating the model parameters using the Adam optimizer. The training loop would run for each epoch; in our case, the training loop will be run 50 times. As shown in Algorithm (4), at every epoch, the learner would retrieve an updated BERT classifier from the training loop and then evaluate it by recording its accuracy, recall, and precision on the validation dataset. The validation metrics are important because we selected and saved the model with the highest validation accuracy across the epochs to be

used to evaluate the test dataset. Finally, we produced **two BERT classifiers**, one trained entirely on supporting briefs while the other on opposing briefs. The hyperparameters used in this study are detailed below:

- Model Checkpoint: "bert-base-uncased"
- Learning Rate:  $lr = 1e-5$
- Optimizer: AdamW optimizer from PyTorch
- Number of Epochs:  $num\_epochs = 50$
- Number of Training Steps:  $num\_training\_steps = num\_epochs * len(train\_dataloader)$
- Loss Function: Cross entropy loss
- Device: NVIDIA RTX 6000 GPU with 28 Gigabyte RAM

**Limitations and Compromises with BERT:** In the context of the Hugging Face Transformers library, a token refers to the basic unit of text on which a model operates. It is the smallest meaningful unit that the model understands and processes. Using BERT to predict legal motion has **one big limitation**; the context length of BERT is only in **512 tokens**, and the average brief token length is 6000 with a variance of 7000. Here are two potential solutions to this predicament:

- a) Create or download a new Large language model with a higher context length.
- b) Summarize or compress the content of the brief to fit 512 tokens.

However, the first approach is infeasible. Most large language models consume massive amounts of computing power; for example, we tried running the experiment with two HuggingFace LLMs with large context lengths: Longformer and Mistral. Each required more than 43 gigabytes of GPU memory to conduct training. Our server's memory capacity is insufficient to keep up with the computational demands of these models. When conducting this experiment, we could not find any LLMs with a large enough context length to fit in 28 GB RAM. However, we are hopeful that in the near future, computationally cheaper LLMs with more considerable context lengths will be readily available. There is promise in the second approach, where we use a large language model to summarize the briefs into a text with a small enough context length to fit into BERT. There exist readily available

online LLMs that could solve this problem for us. However, this approach is outside the scope of this study. We compromised by truncating only the first 512 tokens of the briefs. This means that, on average, less than 8% of the content contained in a single brief would be considered for legal motion prediction. We suspected that this limitation will have a huge impact on the model’s performance.

---

**Algorithm 3** TrainingLoop

---

```

1: for each batch in the train dataset do
2:   inputs, targets  $\leftarrow$  batch
3:   predictions  $\leftarrow$  classifier(inputs)
4:   loss  $\leftarrow$  lossFuction(predictions,targets)
5:   gradients  $\leftarrow$  backpropagation(loss)
6:   classifier  $\leftarrow$  AdamOptimizer(classifier, gradients)
7: end for
8: return model

```

---



---

**Algorithm 4** Learner

---

**Require:** train dataset, test dataset

**Require:** classifier

```

1: bestValidAcc  $\leftarrow$  0
2: for each epoch from 1 to num_epochs do
3:   classifier  $\leftarrow$  TrainingLoop(train dataset)
4:   validAcc = EvaluateModel(classifier, testing dataset )
5:   if validAcc > bestValidAcc then
6:     bestValidAcc  $\leftarrow$  validAcc
7:     SaveModel(classifier)
8:   end if
9: end for
10: return classifier

```

---

### 4.3.2 Motion Classifiers: DeepSets

The DeepSets model is designed to process sets of arguments using a combination of convolutional and fully connected layers. Unlike the aforementioned brief classifiers, DeepSets is the only classifier that classifies motions directly. DeepSet achieves this property because it can consume a variable amount of briefs. However, we have to set the maximum number

of briefs that the DeepSet classifier can consume, called *max\_brief\_len*. The expected maximum number of briefs was set to the maximum number of briefs contained in a motion within the training dataset. In the training dataset the maximum number of supporting, opposing and both contained in a single motion was, 8, 7, and 10 respectively. DeepSets consists of two main components: Phi and Rho ( $\phi$  &  $\rho$ ). Details on the components of the architecture are displayed in algorithms (5) and (6).

**Data Preparation and Classifier Building:** Like BERT, 20% of the training dataset was used for evaluation, but unlike BERT, Deepsets can't process raw texts. We make use of the three TF-IDF transformation pipelines: *Support-pipe*, *Opposition-pipe* and *Both-pipes*, discussed earlier in section 3.4. We convert the briefs to TF-IDF vectors using these transformation pipes. If a particular motion has less than *max\_brief\_len* briefs, the remaining brief would be set to a zero vector. Once all the raw text had been transformed into vectors (TF-IDF or zero), we loaded the vectors into the dataloader.

We used the same learner in Algorithms (3) and (4) to create our hypotheses; the only difference is that we used the DeepSets as the model and a dataloader with TF-IDF vectors. Finally, **three types of training datasets were created**: one exclusively from supporting briefs, one exclusively on opposing briefs, and the other with both supporting and opposing briefs. Thus, three different DeepSets models were created.

The WANDB library [7] is used for logging and tracking the training process, including hyperparameters and metrics. We used WANDB to log both the creation processes of the BERT and DeepSets classifiers. Train-loss and testing-loss graphs will be presented in the results chapter.

**Training Configuration:** The training configuration was set up using the following parameters:

- Input\_size, classify1 : 31452, 262
- hidden1, hidden2, hidden3: 6290, 1572, 524
- Learning Rate: lr = 1e-4
- Optimizer: Adam optimizer from PyTorch
- Number of Epochs: num\_epochs = 300
- Loss Function: Binary Cross Entropy
- Device: NVIDIA RTX 6000 GPU with 28 Gigabyte RAM

---

**Algorithm 5** Architecture

---

**Require:**  $input\_size$ ,  $hidden1$ ,  $hidden2$ ,  $hidden3$ ,  $classify1$

```
1:  $\phi \leftarrow \text{Sequential}:$   
2:    $\text{Conv1d}(input\_size, hidden1, 1)$   
3:    $\text{BatchNorm1d}(hidden1)$   
4:    $\text{ReLU}()$   
5:    $\text{Conv1d}(hidden1, hidden2, 1)$   
6:    $\text{BatchNorm1d}(hidden2)$   
7:    $\text{ReLU}()$   
8:    $\text{Conv1d}(hidden2, hidden3, 1)$   
9:    $\text{BatchNorm1d}(hidden3)$   
10:   $\text{ReLU}()$   
11:  $\rho \leftarrow \text{Sequential}:$   
12:   $\text{Linear}(hidden3, classify1)$   
13:   $\text{BatchNorm1d}(classify1)$   
14:   $\text{ReLU}()$   
15:   $\text{Linear}(classify1, 1)$   
16:   $\text{Sigmoid}()$ 
```

---

---

**Algorithm 6** Forward Pass

---

**Require:**  $x$

```
1:  $out \leftarrow \phi(x)$   
2:  $out \leftarrow \text{mean}(out)$   
3: return  $\rho(out)$ 
```

---

**Justification For Architecture Configuration:** Prior to the official training of this model, we used the *development dataset* to conduct experiments to figure out a viable set of hyperparameters for the study. However, some of the choices made on the configuration of the DeepSet architecture were influenced by a lack of computational memory. As mentioned, the device only had 28 GB of RAM, and deep learning models are notorious for consuming a lot of memory. To compensate for this hard limit, we made some compromises to the model. For example, Phi ( $\phi$ ) consists of a sequence of three convolutional layers, each followed by batch normalization and ReLU activation. Using convolutional layers instead of linear ones is primarily because of memory and computation consumption. Convolutional layers utilize local connectivity and parameter sharing. These two properties drastically reduce the number of trainable parameters in the model and thus allow us to stack more layers.

## 4.4 Adapting Brief-Classifiers to Motion-Classifiers

**Prediction Files:** All hypotheses were saved in a folder during training. After all the classifiers have been trained, we can evaluate the hypotheses’ performance on the testing dataset. We wrote a script that produced a CSV file where each row contains four columns: the classifier’s **prediction** (“grant” or “deny”), the **truth** value, the **score** that represents the classifier’s *confidence* that “grant” is the correct value and the **brief path or the motion number**.

As mentioned previously, brief classifiers are stepping stones to creating motion classifiers. The prediction files from brief classifiers contain information about the motion to which the briefs belong. The information is stored in the *brief path* column; with this information, we can pool together all the predictions related to a particular motion. We used approaches to aggregate predictions of brief classifiers:

- **First-Brief:** In this method, we simply picked the prediction of the earliest **available** filed brief. Information pertaining to the order of the briefs was contained in the brief’s path name. The prediction of the earliest available brief dictates the prediction of the entire motion. In a perfect scenario, where all the motions contained the complete set of briefs, then with this approach, the prediction of the first supporting brief would represent the prediction for the whole motion. However, if the supporting brief is not in the dataset, then we use the next available brief prediction.
- **Fusion:** The fusion approach averages the prediction scores of all the briefs belonging to a motion. We used the score column in the brief prediction file in this method.

The score could be a probability or a log-likelihood, depending on the classifier. If the score was a probability, we converted the score to log-likelihood scores. After which, we added the scores together. If the aggregated value was negative (less than 0), we predicted the motion that all the briefs correspond to as “deny”; however, if the aggregated value was 0 or more (positive), we predicted the motion as a “grant.” The notion of the order of briefs within the context of a motion is irrelevant because averaging is a permutation invariant function. Knowing the proportion of supporting and opposing briefs doesn’t give us any information about what the fusion classifier would predict. In essence, the purpose of the supporting brief is to influence the judge to grant the motion, while the opposing brief aims to influence the judge to deny the motion. However, from the perspective of the brief classifier, it maps the briefs to the outcome of a motion. If a judge denies a motion, the learners are only concerned with creating a classifier that can map supporting and opposing briefs to deny, regardless of the purposes of the briefs. Since we are averaging prediction scores, the briefs’ predictions are equally weighted. Figure 4.1 illustrates the fusion approach.

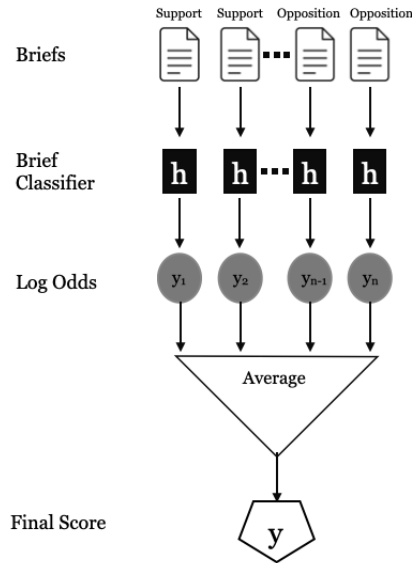


Figure 4.1: Illustration of Fusion

With the Fusion method, we created motion classifiers that aggregated predictions by brief classifiers that were trained solely on supporting briefs, opposing briefs, and lastly,

with both supporting and opposing briefs. Therefore, for each brief classifier, we produced three motion classifiers. We only applied the First-Brief approach to supporting and opposing briefs separately, thus producing two motion classifiers per brief classifier.

## 4.5 Baseline Brief Classifiers

We also included two other implementations of logistic regression and support vector machines (SVM). The logistic regression and SVM were implemented using [Sofia-ML](#) and Thorsten Joachims’s SVMlight package [27], respectively. The SVM and logistic regression hyperparameters are  $C = 10$  and  $\lambda = 0.001$ , respectively. These models also use a particular TF-IDF implementation. All these models plus the TF-IDF pipeline can be set up with Cormack et al. [15] TREC Virtual Machine (TREC-VM). It focuses on providing tools and resources for participants in the Text Retrieval Conference (TREC), particularly in the context of high-stakes information retrieval tasks. These baseline models have been used in AutoTAR [16] and TREC 2015 [14].

In this thesis, we used two baseline motion classifiers as models for comparison.

- a) Dummy: This classifier predicts the judicial decision that occurred most in the dataset. In our case, most motions in the dataset were denied; thus, the dummy classifier would only predict “deny”.
- b) Lrfuse: This is the fusion ensembled motion classifier with logistic regression as its brief classifier. It was picked because it performed best during the development of the experiment pipe.

## 4.6 Evaluating Classifier Performance

After producing prediction files for all the motion classifiers, we are now ready to evaluate the performance of our classifiers. With each prediction file, we calculated a set of standard performance metrics: Accuracy, Area Under the Curve (AUC), Sensitivity and Specificity. The metrics were calculated using Sklean’s [Metric library](#). We also visually represented the confusion matrix and ROC curve for each motion classifier. These visualizations were saved as PNG files for further analysis and inclusion in the thesis.

### 4.6.1 Statistical Tests

We performed two types of statistical tests to evaluate the significance of our results:

- a) **Comparison against a baseline method:** We used a two-sided sign test. This test compared the accuracy of each classifier against the baseline method on a case-by-case basis. The paired observations are a 2-columned matrix, where each column represents a classifier and each row represents a motion, such that at each row, the elements of the columns are either 1 or 0, representing if the model correctly predicted the motion. The null hypothesis is that the median difference between paired observations would be zero. A low p-value ( $< 0.05$ ) would indicate a significant difference in performance between the method and the baseline.
- b) **Comparison against a dummy classifier:** We used a one-sided (greater) sign test. This test compared the accuracy of each classifier against a dummy classifier that always predicts the majority class. The null hypothesis is that the median difference is zero, with the alternative hypothesis being that our method performs better than the dummy classifier. A low p-value would suggest that our method significantly outperforms random guessing based on class distribution.

For both tests, we used the `binomtest` function from the Python library [SciPy](#), which performs an exact test of a simple null hypothesis about the probability of success in a Bernoulli experiment. This approach allows us to assess whether the observed improvements in prediction accuracy were statistically significant or could have occurred by chance. These statistical analyses provide a rigorous framework for comparing the performance of different prediction methods and assessing their practical significance beyond simple metric comparisons.

# Chapter 5

## Results and Discussion

### 5.1 Results

#### 5.1.1 Performance Results

In total, 37 classifiers were used in this study. The naming convention of our classifiers is relatively simple to understand. For example, the name ‘*KNN-tfidf-bothpredictions\_fuse*’ refers to a K-Nearest-Neighbour model that uses both opposition and support briefs for predicting, and the ensemble method is fuse. For names that contain hyphens, ‘-’, the first word before the hyphen symbol is the classifier name, while the last word after the hyphen is the ensemble method. If the word ends with *fuse*, then it’s the fuse method likewise for *first*. The word ‘*oppo*’ indicates that the classifier was trained on and predicts with opposition briefs, likewise ‘*supp*’ refers to supporting briefs, and ‘*both*’ means both opposition and supporting briefs where used.

*RFT-tfidf-bothpredictions\_fuse* is the top-performing classifier in terms of accuracy and AUC, with a score of 62.2% and 66.8% respectively, and both *RFT-tfidf-oppopredictions\_fuse* and *RFT-tfidf-oppopredictions\_first* share the top position in specificity with a score of 0.81. Lrfuse had highest sensitivity score of 58.2%. The performances of all the top-10 classifiers can be viewed in Table 5.3. The three worst-performing classifiers were *KNN-tfidf-supppredictions\_fuse*, *KNN-tfidf-supppredictions\_fuse*, and the dummy classifier, with an accuracy score of 54%, 53.8% and 52.6%. Table 5.4 shows the bottom-10 accurate classifiers.

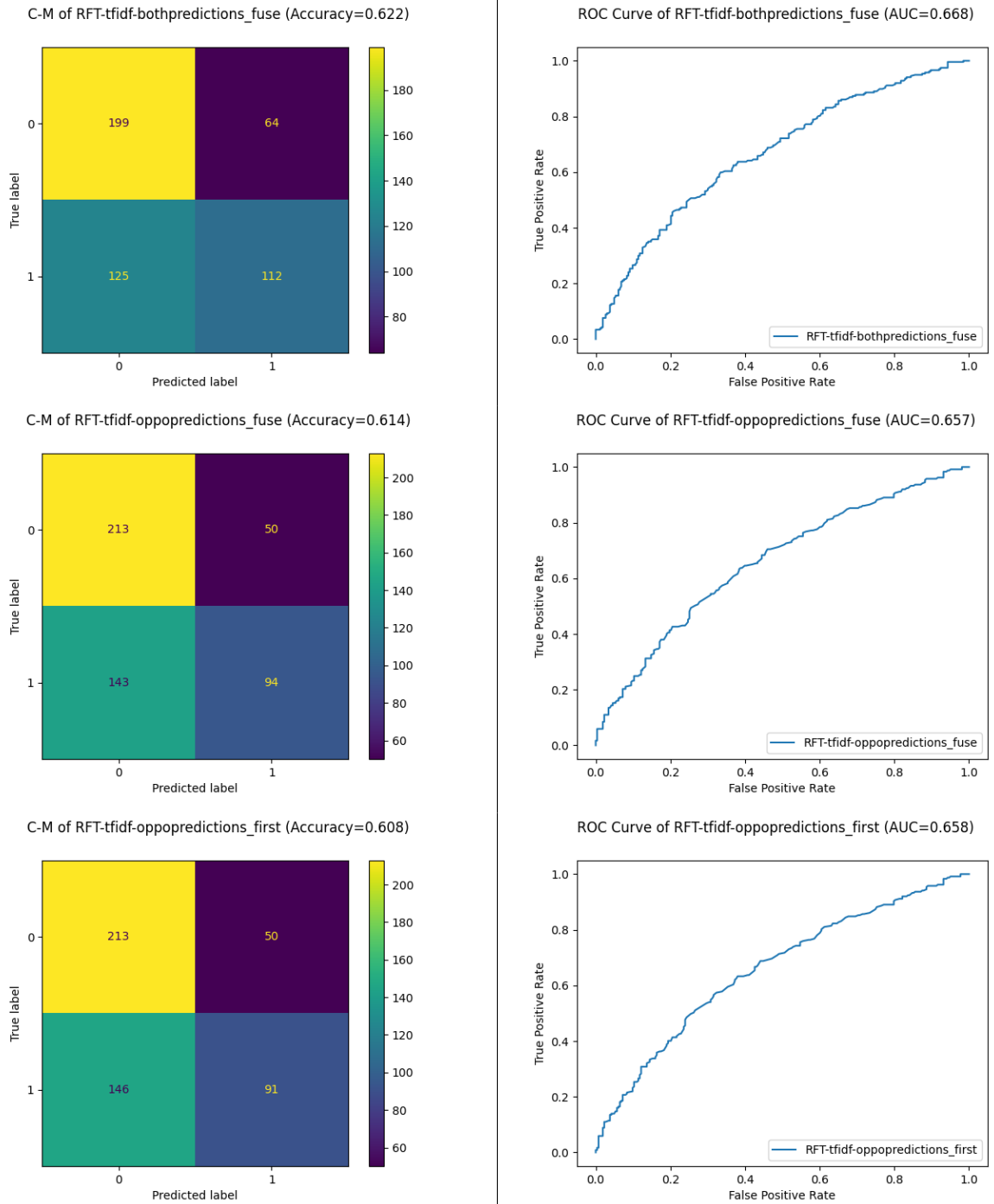
The table 5.1 contains performance metrics for all the brief classifiers used to form the final **Fuse** and **First** adapted motion classifiers, including their accuracy, sensitivity,

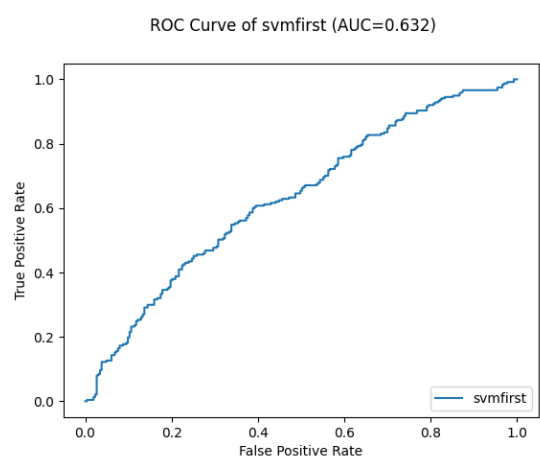
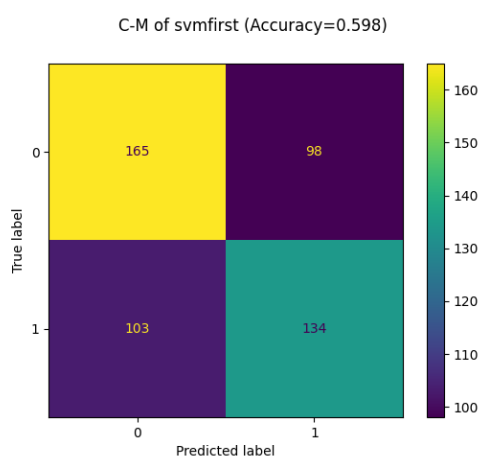
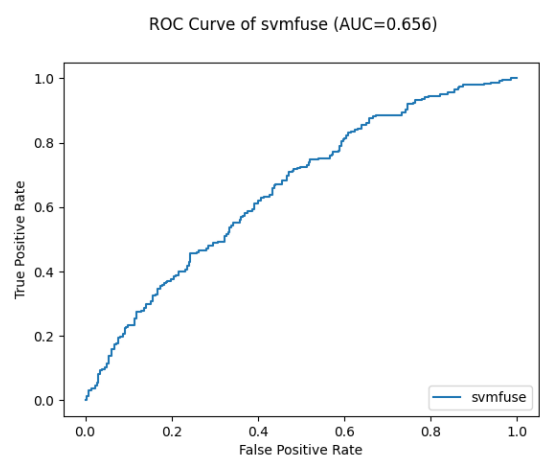
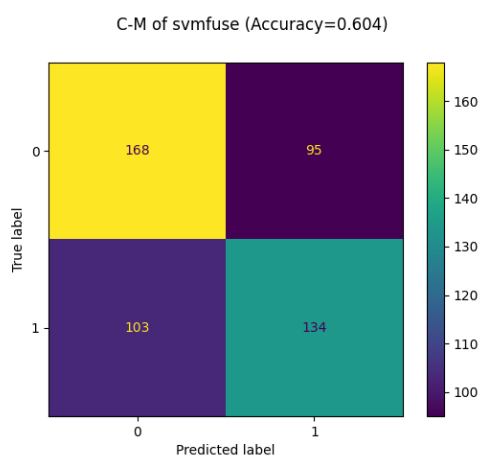
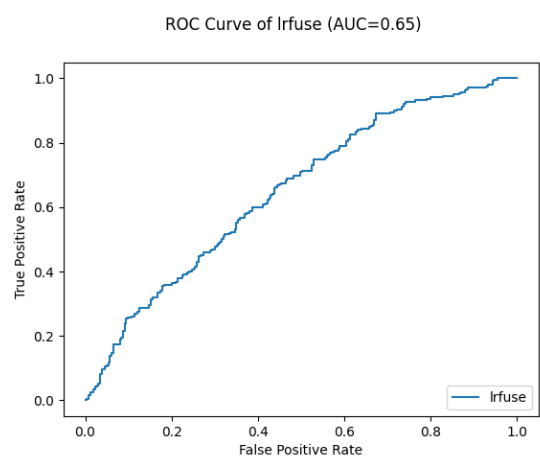
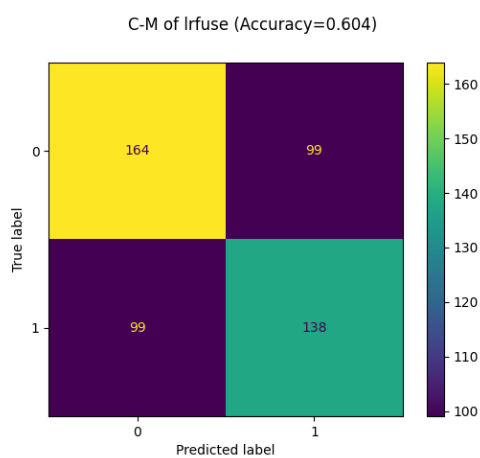
Table 5.1: Brief classifiers accuracy

| Method                                        | Accuracy | Sensitivity | Specificity | AUC   |
|-----------------------------------------------|----------|-------------|-------------|-------|
| svmsupp                                       | 0.591    | 0.565       | 0.614       | 0.626 |
| LinearSVC-tfidf-supppredictions               | 0.591    | 0.569       | 0.611       | 0.629 |
| RFT-tfidf-supppredictions                     | 0.590    | 0.538       | 0.635       | 0.490 |
| RFT-tfidf-oppopredictions                     | 0.587    | 0.390       | 0.765       | 0.466 |
| KNN-tfidf-oppopredictions                     | 0.585    | 0.743       | 0.442       | 0.471 |
| lropo                                         | 0.578    | 0.405       | 0.734       | 0.633 |
| lrsupp                                        | 0.577    | 0.694       | 0.475       | 0.625 |
| svmoppo                                       | 0.576    | 0.527       | 0.620       | 0.638 |
| RFT-sentence_embeddings-oppopredictions       | 0.576    | 0.371       | 0.761       | 0.434 |
| SGD-tfidf-supppredictions                     | 0.575    | 0.593       | 0.560       | 0.614 |
| RFT-sentence_embeddings-supppredictions       | 0.575    | 0.534       | 0.611       | 0.472 |
| Logistic-tfidf-supppredictions                | 0.575    | 0.555       | 0.593       | 0.478 |
| LinearSVC-tfidf-oppopredictions               | 0.574    | 0.516       | 0.627       | 0.623 |
| SGD-sentence_embeddings-oppopredictions       | 0.571    | 0.669       | 0.483       | 0.568 |
| LLM-bert-base-uncased-supppredictions         | 0.569    | 0.461       | 0.662       | 0.587 |
| SGD-sentence_embeddings-supppredictions       | 0.567    | 0.600       | 0.539       | 0.597 |
| Logistic-tfidf-oppopredictions                | 0.562    | 0.449       | 0.664       | 0.424 |
| Logistic-sentence_embeddings-supppredictions  | 0.561    | 0.524       | 0.593       | 0.470 |
| SGD-tfidf-oppopredictions                     | 0.559    | 0.364       | 0.734       | 0.624 |
| LinearSVC-sentence_embeddings-supppredictions | 0.559    | 0.500       | 0.611       | 0.578 |
| LLM-bert-base-uncased-oppopredictions         | 0.555    | 0.561       | 0.550       | 0.573 |
| Logistic-sentence_embeddings-oppopredictions  | 0.553    | 0.449       | 0.647       | 0.440 |
| KNN-tfidf-supppredictions                     | 0.548    | 0.385       | 0.689       | 0.485 |
| LinearSVC-sentence_embeddings-oppopredictions | 0.541    | 0.486       | 0.590       | 0.558 |
| KNN-sentence_embeddings-oppopredictions       | 0.532    | 0.505       | 0.557       | 0.445 |
| KNN-sentence_embeddings-supppredictions       | 0.524    | 0.517       | 0.530       | 0.497 |

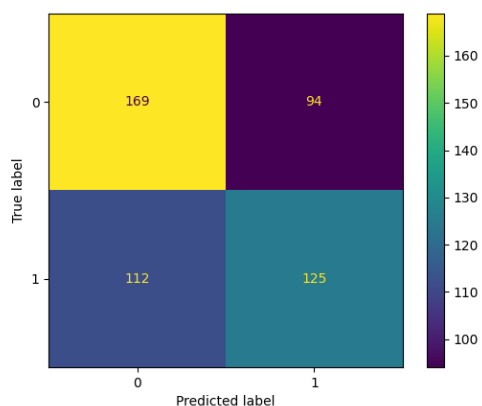
specificity, and area under the curve (AUC). The accuracy of the brief classifiers ranges from 52% to 59%, indicating moderate performance across different models. Sensitivity and specificity values show variability, suggesting that while some classifiers may better identify positive cases (high sensitivity), others perform better at identifying negative cases (high specificity). By comparing these results with motion classifiers' performance in tables 5.4 and 5.3, we can see that our brief classifiers perform relatively the same as the motion classifiers. None of the brief classifiers performed higher than 60%, and only six motion

Table 5.2: Confusion Matrix and ROC Curve of top ten classifiers

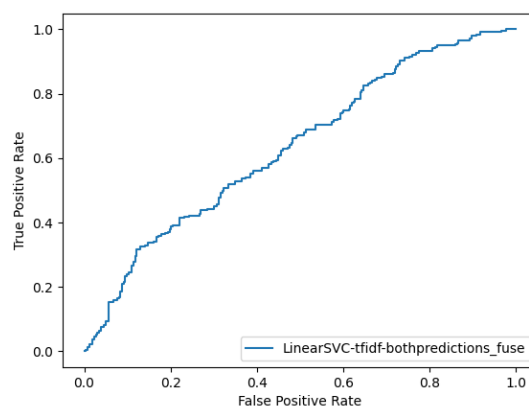




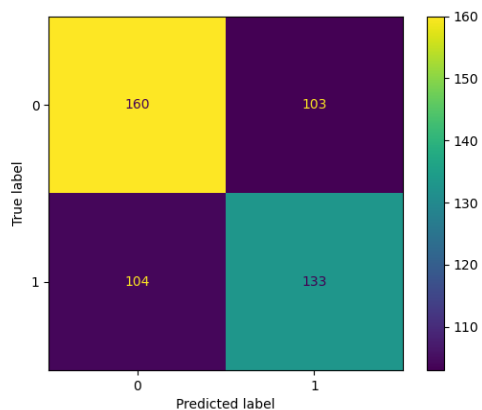
C-M of LinearSVC-tfidf-bothpredictions\_fuse (Accuracy=0.588)



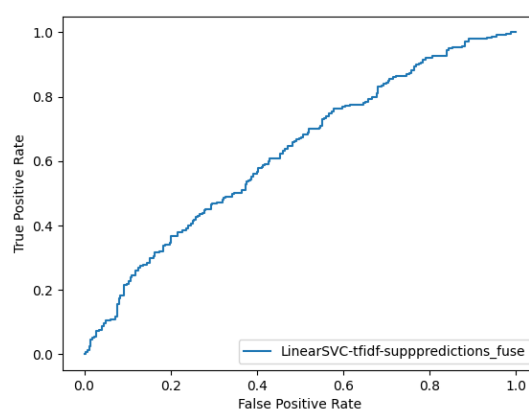
ROC Curve of LinearSVC-tfidf-bothpredictions\_fuse (AUC=0.634)



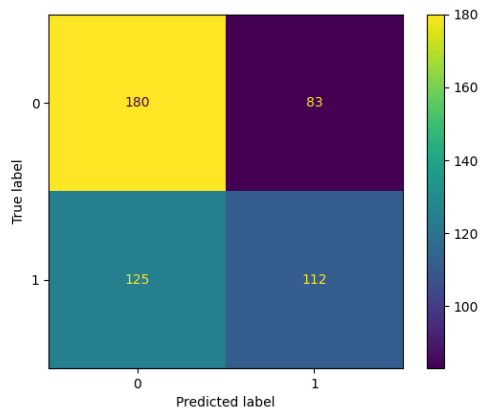
C-M of LinearSVC-tfidf-supppredictions\_fuse (Accuracy=0.586)



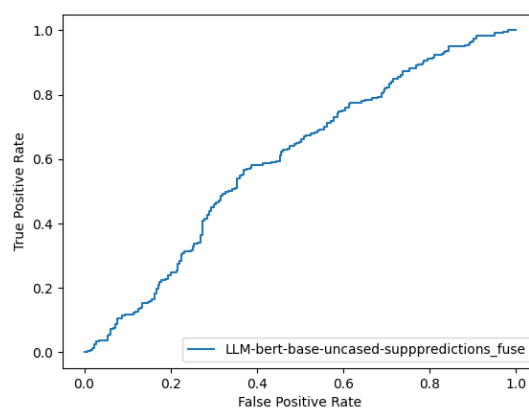
ROC Curve of LinearSVC-tfidf-supppredictions\_fuse (AUC=0.625)



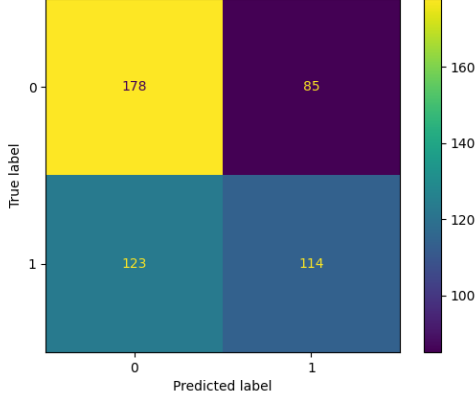
C-M of LLM-bert-base-uncased-supppredictions\_fuse (Accuracy=0.584)



ROC Curve of LLM-bert-base-uncased-supppredictions\_fuse (AUC=0.598)



C-M of LLM-bert-base-uncased-suppredictions\_first (Accuracy=0.584)



ROC Curve of LLM-bert-base-uncased-suppredictions\_first (AUC=0.599)

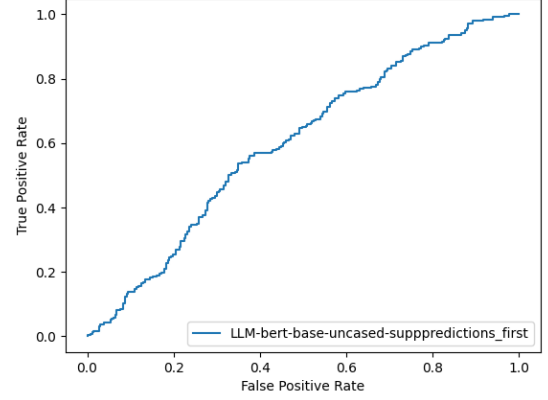


Table 5.3: Top-10 Most accurate Classifiers

| Method                                     | Accuracy | AUC      | Sensitivity | Specificity |
|--------------------------------------------|----------|----------|-------------|-------------|
| RFT-tfidf-bothpredictions_fuse             | 0.622    | 0.667782 | 0.472574    | 0.756654    |
| RFT-tfidf-oppopredictions_fuse             | 0.614    | 0.657233 | 0.396624    | 0.809886    |
| RFT-tfidf-oppopredictions_first            | 0.608    | 0.658372 | 0.383966    | 0.809886    |
| lrfuse (baseline)                          | 0.604    | 0.649532 | 0.582278    | 0.623574    |
| svmfuse                                    | 0.604    | 0.655837 | 0.565401    | 0.638783    |
| svmfirst                                   | 0.598    | 0.631989 | 0.565401    | 0.627376    |
| LinearSVC-tfidf-bothpredictions_fuse       | 0.588    | 0.634099 | 0.527426    | 0.642586    |
| LinearSVC-tfidf-suppredictions_fuse        | 0.586    | 0.625451 | 0.561181    | 0.608365    |
| LLM-bert-base-uncased-suppredictions_fuse  | 0.584    | 0.597712 | 0.472574    | 0.684411    |
| LLM-bert-base-uncased-suppredictions_first | 0.584    | 0.599076 | 0.481013    | 0.676806    |

classifiers outperformed all the brief classifiers. However, it should be noted that there are more briefs than motions; specifically, 620 supporting briefs, 567 opposing briefs and 500 motions were used in the testset.

In tables 5.6 and 5.5, we presented the p-values from the statistical test results as described in Section C. The performance results of all the classifiers can be seen in Appendix A.

Table 5.4: Bottom-10 least accurate Classifiers

| Method                                     | Accuracy | AUC      | Sensitivity | Specificity |
|--------------------------------------------|----------|----------|-------------|-------------|
| SGD-tfidf-oppopredictions_first            | 0.562    | 0.625259 | 0.350211    | 0.752852    |
| Logistic-tfidf-oppopredictions_fuse        | 0.560    | 0.621521 | 0.438819    | 0.669202    |
| DeepSet-tfidf-oppopredictions              | 0.558    | 0.583    | 0.565401    | 0.551331    |
| KNN-tfidf-bothpredictions_fuse             | 0.558    | 0.583065 | 0.637131    | 0.486692    |
| LLM-bert-base-uncased-oppopredictions_fuse | 0.558    | 0.580345 | 0.552743    | 0.562738    |
| DeepSet-tfidf-supppredictions              | 0.544    | 0.591183 | 0.556962    | 0.532319    |
| LLM-bert-base-uncased-bothpredictions_fuse | 0.544    | 0.59383  | 0.527426    | 0.558935    |
| KNN-tfidf-supppredictions_fuse             | 0.540    | 0.551612 | 0.392405    | 0.673004    |
| KNN-tfidf-supppredictions_first            | 0.538    | 0.55028  | 0.367089    | 0.692015    |
| dummy (baseline)                           | 0.526    | N/A      | 0.000000    | 1.000000    |

Table 5.5: P-values of top-10 classifiers

| Method                                      | p-value (baseline) | p-value (dummy) |
|---------------------------------------------|--------------------|-----------------|
| RFT-tfidf-bothpredictions_fuse              | 0.455814           | 0.000184        |
| RFT-tfidf-oppopredictions_fuse              | 0.751191           | 0.000154        |
| RFT-tfidf-oppopredictions_first             | 0.937409           | 0.000350        |
| lrfuse (baseline)                           | 1.000000           | 0.006703        |
| svmfuse                                     | 1.000000           | 0.005935        |
| svmfirst                                    | 0.826405           | 0.010684        |
| LinearSVC-tfidf-bothpredictions_fuse        | 0.402963           | 0.021197        |
| LinearSVC-tfidf-supppredictions_fuse        | 0.385669           | 0.029415        |
| LLM-bert-base-uncased-supppredictions_fuse  | 0.500055           | 0.022336        |
| LLM-bert-base-uncased-supppredictions_first | 0.509419           | 0.023443        |

### 5.1.2 Comparing classifiers with baseline

At the start of this experiment, one of our goals was to develop a method that could beat the baseline, *lrfuse*. A surface-level analysis of the contents of Table 5.3 might make casual observers conclude that the top 10 using the fuse ensemble method on Random Forest Trees (RFT) with both supporting and opposing briefs as inputs is the best approach for motion prediction, they would be correct. However, they would be wrong to infer further that the fuse ensemble method with RFT is superior and would always beat the baseline

Table 5.6: P-values of bottom ten classifiers

| Method                                     | p-value (baseline) | p-value (dummy) |
|--------------------------------------------|--------------------|-----------------|
| SGD-tfidf-oppopredictions_first            | 0.073214           | 0.081036        |
| Logistic-tfidf-oppopredictions_fuse        | 0.044762           | 0.123455        |
| DeepSet-tfidf-oppopredictions              | 0.041501           | 0.172361        |
| KNN-tfidf-bothpredictions_fuse             | 0.063537           | 0.187563        |
| LLM-bert-base-uncased-oppopredictions_fuse | 0.131512           | 0.169451        |
| DeepSet-tfidf-supppredictions              | 0.009503           | 0.308236        |
| LLM-bert-base-uncased-bothpredictions_fuse | 0.047181           | 0.303209        |
| KNN-tfidf-supppredictions_fuse             | 0.013401           | 0.326971        |
| KNN-tfidf-supppredictions_first            | 0.010925           | 0.349899        |
| dummy (baseline)                           | 0.013405           | 1.000000        |

in different conditions. This is because an element of random chance may existed in our experiment. In other words, maybe *RFT-tfidf-bothpredictions\_fuse* was *lucky*.

To investigate the legitimacy of *RFT-tfidf-bothpredictions\_fuse*’s supremacy, we conducted a two-sided hypothesis test that compares the performance of each classifier to the baseline. The results are highlighted in Table 5.6; The p-value of the hypothesis test on *RFT-tfidf-bothpredictions\_fuse* and the baseline is approximately 0.46. This implies that we have **failed to reject** the null hypothesis. Thus, we conclude that there is no significant difference between the performance of *RFT-tfidf-bothpredictions\_fuse* and *lrfuse*. In fact, Table 5.6 also suggest that the performances of all the top-10 most accurate classifiers aren’t statistically different from the baseline.

Another interesting observation is that most of the other methods are not **significantly different** from the baseline. In Table 5.5, It is only after the 10<sup>th</sup> worst performing method, *Logistic-tfidf-oppopredictions\_fuse* and below, that we start to see p-values less than  $\alpha = 0.05$ .

### 5.1.3 Comparing classifiers with dummy

According to the dataset statistics in Section 3.4.1, more denies than grants exist. A causal observer might suggest forsaking building a classifier to instead guess the judge will always deny the motion. We employed a second statistical hypothesis test to investigate this very question. Based on the results in Table 5.5, we can see that the dummy classifier

performed the worst. However, at both popular significant levels  $\alpha_1 = 0.01$  and  $\alpha_2 = 0.05$ , our hypothesis test **failed to reject** the claim that the performance of the other bottom-10 methods are equal to the performance of the dummy classifier.

On the flip side, in Table 5.6 and with  $\alpha_1 = 0.01$ ; only the top 5 classifiers **reject** the null hypothesis. This implies that, except for the top-5 methods, the test shows that all other approaches are not **significantly** better than the dummy classifier.

## 5.2 Discussions

### 5.2.1 Per-brief Ensemble vs Per-motion Classifier.

At the beginning of this experiment, we hypothesized that the non-ensemble approach, Deep Sets, would outperform the per-brief ensemble classifiers. This is because, unlike the ensemble adaptation on the per brief classifiers, Deep Sets is the only model capable of using the entire motion of briefs to make predictions and all the models, except the baseline models and Bert, used the same TF-IDF vectors. Thus, came as a surprise to us that Deep Sets performed relatively poorly. According to table 5.4, two of the three Deep Set models were among the worst ten performing classifiers; the other Deep Set classifier, *DeepSet-tfidf-bothpredictions*, is ranked 24<sup>th</sup> and can be seen in the Appendix at Table A.2. There are two plausible reasons for the performances of Deep Sets:

- **Smaller Latent Space:** In the paper titled: Universal approximation of functions on sets [47], The authors stipulated a theorem that declares the necessary conditions for Deep Sets to be capable of representing arbitrary continuous functions on sets of size  $M$ . The theorem can be paraphrased as follows: given functions  $f$ ,  $\rho$  and  $\Phi$ ; dimensions  $D, A, M, N \in \mathbb{N}$  and  $X \in \mathbb{R}^{D \times M}$ .

$$f : \mathbb{R}^{D \times M} \mapsto \mathbb{R} \quad (5.1)$$

$$\rho : \mathbb{R}^{A \times N} \mapsto \mathbb{R} \quad (5.2)$$

$$\Phi : \mathbb{R}^{D \times M} \mapsto \mathbb{R}^{A \times N} \quad (5.3)$$

**If  $M > N$  then there exist continuous permutation invariant functions  $f$  which can not be expressed as  $f(X) = \rho(\Phi(X))$ .** The author only explicitly

proved when  $D = A = 1$ , but I expect their proofs can be easily extended to the case when for  $D = A > 1$ . However, it is unknown whether this theorem holds for  $D \neq A$ . In our case,  $M = D$ , but  $D > A$ . To be exact  $D$  is 31452 while  $A$  is 6290. Our intuition says since  $D > A$  and  $M = D$ , then the latent space is smaller, and thus, our Deep Set cannot represent the permutation invariant function  $f$ .

- **Curse of high-dimensionality:** When the expected dimensions of the input increase, so does the volume of the input vector space, so much so that the data available appears sparse in the high-dimensional vector space. A consequence of this is that a large amount of data is needed for training increases with respect to the dimensions of the input. That being said, most of the brief classifiers used the same TF-IDF vector as Deep Sets. However, Deep Sets process a set of briefs, while brief classifiers process single briefs. For context, the TF-IDF vectors had 31452 dimensions, while our Deep Set was made to handle a maximum of 8 briefs. The brief classifiers were trained on 1646 briefs during training, while Deep Sets were trained on 675 motions. Finally, according to the statistics of the dataset in section 3.4.1, there was an average of 2.4 briefs per motion. To summarise, the input space of the Deep Set is larger, and the number of examples it used to train was smaller than the brief classifiers.

### 5.2.2 Can we do better?

Given the limited resources, including the incomplete dataset, compromises we made with the architecture of Deep Set, and the truncation of the text with Bert, it was quite a remarkable feat to find a classifier that achieved 62% accuracy on motion prediction. A natural follow-up question would be, can we do better? Another way of knowing if we can do better is by investigating the upper bound of our classifiers' accuracy. This section will briefly explore the theoretical considerations of achieving better results for predicting motions.

**To summarize the subsections below:** We established that in the deterministic and stochastic setting of supervised machine learning, the performance of our classifiers is theoretically bounded. This theoretical result posits a plausible explanation that our top-10 performing classifiers are not statistically different from the *lrfuse* because there exists a theoretical performance limit, and our classifiers **could be** relatively close to that limit. In the stochastic setting, the Bayes optimal predictor sets the theoretical limit. It is unknown what the Bayes error rate is, but we could speculate that it's slightly above

62%. In the deterministic setting, we established that none of our TF-IDF classifiers can represent the true function because the true function is not **permutation invariant**.

### 5.2.2.1 Does the true function exist? - No

Returning to the formalization of supervised learning discussed in the background section, the supervised learning framework presupposes the existence of a deterministic function that we call the true function,  $f$ . It should be noted that an alternative supervised learning framework assumes a stochastic relationship between the inputs and the target instead. To know whether we can do better, we first need to know whether the relationship between inputs and the target,  $x_i$  and  $y_i$ , is deterministic or stochastic. In a study by Danziger et al. [18], they analyzed the decisions of Israeli parole boards and found that the probability of granting parole was about 65% at the start of a session but dropped to nearly zero before a meal break. After the break, the rate of favourable decisions returned to 65%. Demonstrated that judges’ verdicts are more lenient after a meal break, thus coining the term “*hungry judge*” effect. In other words, there are factors besides the arguments presented in court that could affect the case’s outcome, and because our (deterministic) classifiers rely only on the arguments presented to the judge, thus it is possible that the same arguments could have two different outcomes depending on whether the judge is hungry or not, while the classifier would always yield the same output for the same input. Therefore, our answer is simple: There is evidence that suggests the relationship between the input and the target is **not deterministic**, however the input and the output of our classifier is **deterministic**.

### 5.2.2.2 Is there a limit to how well our classifiers could do? - Yes

Before we speculate on the limits of our approach let’s consider the limits of our feature engineering technique.

**Limitations of TF-IDF:** Our brief and motion classifiers aren’t trained from the raw briefs but rather from vector transformation of the brief documents. The only brief classifier trained from raw text is BERT; thus, for the remainder of this section, we exclude BERT when we refer to brief classifiers. In other words, the brief classifiers aren’t trained from text but from TF-IDF vectors.

We will use the following example to explore the limitations of TF-IDF. The sentences “*she said he stole the money*” and “*he said she stole the money*.” both share the exact words, but the sentences convey different meanings. However, the TF-IDF method would

convert all semantically different sentences to the same vector. This means that TF-IDF transformations lose the semantics of the original texts. In other words, TF-IDF transformations are not *injective*. **To conclude**, TF-IDF transformation is a permutation-invariant function that takes text as input; thus, the order of the words in the text and its semantics are lost after transformation.

**Proof: If there was a true function, it isn't permutation invariant.** From the previous subsection 5.2.2.1, we have established that the true function does **not exist**. We have also demonstrated that the TF-IDF transformation is permutation-invariant. Let us consider a **hypothetical scenario** where the relationship between inputs and targets is deterministic.

Let  $T : X \rightarrow Z$  be the TF-IDF transformation function and  $f : X \rightarrow Y$  is the true function. We want to know whether it is possible for there to exist a function  $g$  such that  $f = g \circ T$ .

If we can show that there are two briefs,  $x_1, x_2 \in X$  with the same TF-IDF vector  $T(x_1) = T(x_2)$ , but for which different verdicts should be rendered  $f(x_1) \neq f(x_2)$  then no such  $g$  can exist.

This is true because it leads to the following contradiction: If  $x_1, x_2 \in X$  with  $T(x_1) = T(x_2)$  and if  $f = g \circ T$  then  $f(x_1) = g \circ T(x_1) = g \circ T(x_2) = f(x_2)$ .

Thus, we need only construct hypothetical briefs  $x_1$  and  $x_2$  that meet these properties to conclude that no such function can exist.

### Counter-Examples:

Let's consider the examples below. The sentences  $x_1$  and  $x_2$  are permutations of each other; however, they differ in meaning. Thus, a permutation invariant function would map these sentences to the same vector. In both examples,  $x_1$  admits guilt, while  $x_2$  maintains innocence; thus,  $f$  should map both semantically opposing sentences,  $x_1$  and  $x_2$ , to different verdicts.

- Example 1:  $x_1 = \text{"I am not guilty. I am innocent."} \wedge x_2 = \text{"I am guilty. I am not innocent."}$
- Example 2:  $x_1 = \text{"He is guilty, I am not guilty."} \wedge x_2 = \text{"I am guilty. He is not guilty."}$

The two provided examples would lead to a contradiction because  $g \circ T(x_1) = g \circ T(x_2)$  but  $f(x_1) \neq f(x_2)$  and  $f = g \circ T$ , thus proving that the true function is not permutation invariant. ■

**Possible oppositions to the counterexamples:** While our counterexamples do satisfy the condition in 5.2.2.2, critics could point to the obvious fact that, in practice, legal briefs contain tens, if not hundreds, of pages. And that the possibility that any two or more briefs are a permutation of the other is unlikely. Another critic could say that the admission of guilt is not evidence of guilt and thus may not be admissible in court. Although the truth of said arguments is difficult to determine, our critics’ argument is still valid. Our counterexamples are not the typical arguments or briefs made in court. However, the proof is still valid, and our counterexamples nonetheless satisfy the condition of the proof. The proof opens up the discourse for others to find better counterexamples that occur in real life and satisfy the conditions for the proof. Lastly, the point of the counter-examples is to show that by changing the position of ”*not*” within a sentence, we negate one of the argument’s premises. Thus, it converts a valid deductive argument into an invalid or unsound argument. A true function should be able to distinguish between an invalid and a valid argument, while a permutation invariant function cannot make such a distinction.

### 5.2.2.3 In the stochastic setting, is there also a limit in performance? - Yes

According to Shalev-Shwartz and Ben-David, [45, page. 46], they established that in the stochastic setting, there exists no classifier that can have a lower error than the Bayes optimal predictor,  $f_d$ :

$$f_d = \begin{cases} grant & \mathbf{P}[y = grant|x] \geq 1/2 \\ deny & \text{otherwise} \end{cases} \quad (5.4)$$

This implies that our approach to motion prediction has a performance limit. According to Shalev-Shwartz and Ben-David, it isn’t obvious exactly what that performance limit is because we don’t know the probability distribution  $\mathcal{D}$  (2.5). In other words, we don’t know if the performance limit is 99%, 80% or 65% accurate; we can only speculate.

# Chapter 6

## Conclusion and Future Work

In this work, we scraped dockets from RECAP, produced a dataset that can be used for motion classification, and built and trained 36 motion classifiers. The best-performing classifier, with an accuracy of 62.2%, was Random Forest Trees, trained on both the supporting and opposing briefs. However, our statistical analysis concluded that none of the top 10 classifiers are significantly different from the baseline. Using a significant level of 0.01, only the top-5 classifiers are statistically better than the dummy classifier. In our discussion, we explored the limitations in the architecture of Deep Set, namely that the latent space is small and the ratio of the training data points to the number of features. Lastly, we explored the theoretical limitations of our approach in supervised learning’s deterministic and stochastic settings.

### 6.1 Future Work

#### 6.1.1 Future Improvement on current work

**Lack of Computational Power:** We didn’t have enough memory to use larger language models. Besides BERT, the contents of the brief were converted into TF-IDF vectors, which were used by our classifiers to make predictions. BERT used, at most, two paragraphs per brief for predicting. Therefore, the full capability of large language models was not fully utilized in this study. Our future research would test the hypothesis of using large language models against TF-IDF vectors.

**A Representative Environment:** It would help to expose our models to all the information available to an actual judge. In our case, all the classifiers used supporting

and(or) opposing briefs for prediction. The classifiers needed to be made aware of the rules about a particular motion, the laws of the case’s jurisdiction, and the attachment files accompanying briefs in a docket file. In our dataset, some motions also had an incomplete set of briefs; thus, our classifier made predictions with less information that a judge would be exposed to. By simulating a representative environment for our classifiers, where the classifiers have access to all relevant materials—such as legal precedents, jurisdictional laws, supporting documents and attachment files—our models could better understand the context and nuances of each case. This could help bridge the gap from an *academic exercise* to a commercially viable endeavour.

### 6.1.2 Alternative approaches

We used a myriad of supervised learning algorithms to create classifiers, and our hypothesis tests show that the top ten performing classifiers are statistically indistinguishable, with the best classifier achieving a 62% accuracy. Let’s imagine that our classifiers were to achieve near 99% accuracy on the test set; such a classifier would suggest that the legal motion procedure is, in fact, a deterministic system. However, would such a classifier be useful, specifically to legal practitioners? The briefs are converted to an irreversible 31452-dimensional TF-IDF vector; even if we could accurately predict who would win a motion, we would be able to know why. Thus, the inherent limitation of our approach is explainability.

**An Injective Approach:** We used TF-IDF and discussed its limitations. After we convert a text to TF-IDF vectors, we cannot reconstruct the same text. Thus, having an injective feature engineering approach would allow us to tinker with the arguments made in the briefs, such that we can measure precisely the specific fact, argument or rule that flipped the judicial opinion of motion from a grant to denial.

**A Generative Approach:** A progressive move would be to train a computer to “think” like a judge or a lawyer. Lawyers and Judges are held to specific standards; such standards could be imposed on generative large language models. The main advantage of a generative approach versus a binary classification approach is the quality of the feedback. The only feedback classifiers have during training would be an error signal if they predicted ‘0’ when the accurate answer was ‘1’, while with the generative approach, the feedback could be how close in “spirit” the model’s opinion is with that of the actual judicial opinion. By “spirit” we refer not only to the LLM’s capability to come to the same logical conclusion as a judge but also to the style and degree elaboration of how said LLM came to its conclusion.

# References

- [1] Administrative Office of the U.S. Courts. Public access to court electronic records (pacer), 2024.
- [2] Nikolaos Aletras, Dimitrios Tsarapatsanis, Daniel Preoțiuc-Pietro, and Vasileios Lampsos. Predicting judicial decisions of the European Court of Human Rights: a natural language processing perspective. *PeerJ Computer Science*, 2:e93, 2016.
- [3] Katie Atkinson, Trevor Bench-Capon, and Danushka Bollegala. Explanation in ai and law: Past, present and future. *Artificial Intelligence*, 289:103387, 2017.
- [4] Trevor J. M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *Journal of Logic and Computation*, 13(3):429–448, 2003.
- [5] Vithor Gomes Ferreira Bertalan and Evandro Eduardo Seron Ruiz. Predicting judicial outcomes in the brazilian legal system using textual features. In *DHandNLP@PROPOR*, pages 22–32, 2020.
- [6] Philippe Besnard and Anthony Hunter. *Elements of Argumentation*. MIT Press, 2008.
- [7] Weights & Biases. Weights & biases: The AI developer platform. <https://wandb.ai/site>, 2024. Accessed: 06 June 2024.
- [8] Jeff Bilmes. Underfitting and Overfitting in Machine Learning. [https://people.ece.uw.edu/bilmes/classes/ee511/ee511\\_spring\\_2020/overfitting\\_underfitting.pdf](https://people.ece.uw.edu/bilmes/classes/ee511/ee511_spring_2020/overfitting_underfitting.pdf).
- [9] Christopher M. Bishop. *Pattern Recognition*. Springer, 2006.
- [10] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

- [11] Gerhard Brewka and Stefan Woltran. Abstract dialectical frameworks. In *Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning (KR 2010)*, pages 102–111, 2010.
- [12] Stefan Büttcher, Charles L. A. Clarke, and Gordon V. Cormack. *Information Retrieval: Implementation and Evaluation of Search Engines*. Google Inc. and University of Waterloo, 2009.
- [13] Shihchieh Chou and Tai-Ping Hsing. Text Mining Technique for Chinese Written Judgment of Criminal Case. In Hsinchun Chen, Michael Chau, Shu-hsing Li, Shalini Urs, Srinath Srinivasa, and G. Alan Wang, editors, *Intelligence and Security Informatics*, pages 113–125, Berlin, Heidelberg, 2010. Springer.
- [14] Charles L.A. Clarke, Haotian Zhang, Wu Lin, Yipeng Wang, and Mark D. Smucker. Waterlooclarke at trec 2015. In *Proceedings of the 24th Text Retrieval Conference (TREC 2015)*. National Institute of Standards and Technology (NIST), 2015.
- [15] Gordon V. Cormack. Baseline model implementation for automatic participation in the trec 2015 total recall track. <https://cormack.uwaterloo.ca/trecvm/>, 2015. The software is designed for use with VirtualBox and requires specific system configurations.
- [16] Gordon V. Cormack and Maura R. Grossman. Autonomy and Reliability of Continuous Active Learning for Technology-Assisted Review, April 2015. arXiv:1504.06868 [cs].
- [17] Cornell Law School. Motion. <https://www.law.cornell.edu/wex/motion>, 2023. Accessed: 2024-09-18.
- [18] Shai Danziger, Jonathan Levav, and Liora Avnaim-Pesso. Extraneous factors in judicial decisions. *Proceedings of the National Academy of Sciences*, 108(17):6889–6892, 2011.
- [19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [20] Phan Minh Dung. On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and  $n$ -person games. *Artificial Intelligence*, 77(2):321–357, 1995.

- [21] Kai Essers, Martin Chapman, Nadin Kokciyan, Isabel Sassoon, Talya Porat, Panagiotis Balatsoukas, Peter Young, Mark Ashworth, Vasa Curcin, Sanjay Modgil, Simon Parsons, and Elizabeth I. Sklar. The CONSULT System: Demonstration. In *Proceedings of the 6th International Conference on Human-Agent Interaction*, pages 385–386, Southampton United Kingdom, December 2018. ACM.
- [22] Free Law Project. Advanced recap archive search for pacer, 2024.
- [23] Moritz Hardt and Benjamin Recht. *Patterns, predictions, and actions: Foundations of machine learning*. Princeton University Press, 2022.
- [24] Trevor Hastie and Robert Tibshirani. *The Elements of Statistical Learning*. Springer, 2017.
- [25] Anthony Hunter. A probabilistic approach to modelling uncertain logical arguments. *International Journal of Approximate Reasoning*, 54(1):47–81, 2013.
- [26] Jonas Jacobson, Jasmine Dobbs-Marsh, Varda Liberman, and Julia A. Minson. Predicting Civil Jury Verdicts: How Attorneys Use (and Misuse) a Second Opinion. *Journal of Empirical Legal Studies*, 8(s1):99–119, 2011.
- [27] Thorsten Joachims. SVMlight: Support vector machine. [https://www.cs.cornell.edu/people/tj/svm\\_light/](https://www.cs.cornell.edu/people/tj/svm_light/), 1999. Version 6.02.
- [28] Daniel Martin Katz, Michael J. Bommarito II, and Josh Blackman. Predicting the Behavior of the Supreme Court of the United States: A General Approach, 2014. arXiv:1407.6333.
- [29] Legal Information Institute. Docket. <https://www.law.cornell.edu/wex/docket>, 2022. Accessed: 2024-08-10.
- [30] Hengfei Li, Nir Oren, and Timothy J. Norman. Probabilistic argumentation frameworks. In *International Workshop on Theorie and Applications of Formal Argumentation*, pages 1–16. Springer, Berlin, Heidelberg, 2011.
- [31] Zhenyu Liu and Huanhuan Chen. A predictive performance comparison of machine learning models for judicial cases. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–6, November 2017.
- [32] Ejan Mackaay and Pierre Robillard. Predicting judicial decisions: the nearest neighbor rule and visual representation of case patterns. *Datenverarbeitung im Recht*, 3:302–331, 1974.

- [33] L. Thorne McCarty. Reflections on TAXMAN: An experiment in artificial intelligence and legal reasoning. *Harvard Law Review*, 90(5):837–893, January 1977.
- [34] Devin J. McConnell, James Zhu, Sachin Pandya, and Derek Aguiar. Case-level prediction of motion outcomes in civil litigation. In *Proceedings of the Eighteenth International Conference for Artificial Intelligence and Law (ICAIL’21)*, page 16. ACM, 2021.
- [35] Masha Medvedeva, Michel Vols, and Martijn Wieling. Using machine learning to predict decisions of the European Court of Human Rights. *Artificial Intelligence and Law*, 28(2):237–266, 2020.
- [36] Engin Yıldırım Mehmet Fatih Sert and İrfan Haşlak. Using artificial intelligence to predict decisions of the Turkish Constitutional Court.
- [37] Sanjay Modgil and Henry Prakken. A general account of argumentation with preferences. *Artificial Intelligence*, 195:361–397, 2013.
- [38] Olusegun Olu-Osayomi and Babatunde Adebua. The relevance of the practice of Ifá Divination in Ola Rotimi’s “The gods are not to blame”. *Yoruba Studies Review*, 8:105–129, 05 2023.
- [39] Conor O’Sullivan and Joeran Beel. Predicting the Outcome of Judicial Decisions made by the European Court of Human Rights.
- [40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit. <https://scikit-learn.org/stable/index.html>, 2024. Accessed: 2024-05-22.
- [41] Henry Prakken, Adam Wyner, Trevor Bench-Capon, and Katie Atkinson. A formalism for modeling case-based reasoning with precedents in law. *Artificial Intelligence and Law*, 28(4):439–480, 2020.
- [42] Theodore W. Ruger, Pauline T. Kim, Andrew D. Martin, and Kevin M. Quinn. The Supreme Court forecasting project: Legal and political science approaches to predicting Supreme Court decision making. *Columbia Law Review*, 104(4):1150–1210, 2004.
- [43] Marijn Schraagen, Bas Testerink, Daphne Odekerken, and Floris Bex. Argumentation-driven information extraction for online crime reports. In *Proceedings of International*

*Workshop on Legal Data Analytics and Mining (LeDAM 2018)*, page 5, New York, NY, USA, 2019. ACM. Turin, Italy.

- [44] Rafe Athar Shaikh, Tirath Prasad Sahu, and Veena Anand. Predicting outcomes of legal cases based on legal factors using classifiers. *Procedia Computer Science*, 167:2393–2402, January 2020.
- [45] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.
- [46] Michael Benedict L. Virtucio, Jeffrey A. Aborot, John Kevin C. Abonita, Roxanne S. Aviñante, Rother Jay B. Copino, Michelle P. Neverida, Vanesa O. Osiana, Elmer C. Peramo, Joanna G. Syjuco, and Glenn Brian A. Tan. Predicting decisions of the philippine supreme court using natural language processing and machine learning. Technical report, Advanced Science and Technology Institute, Department of Science and Technology, Quezon City, Philippines, 2023.
- [47] Edward Wagstaf, Fabian B. Fuchs, Michael A. Osborne Martin Engelcke, and Ingmar Posner. Universal approximation of functions on sets. *arXiv preprint arXiv:2107.01959*, 2021. Accessed: 2024-07-31.
- [48] Yasmen Wahba, Nazim Madhavji, and John Steinbacher. A Comparison of SVM Against Pre-trained Language Models (PLMs) for Text Classification Tasks. In Giuseppe Nicosia, Varun Ojha, Emanuele La Malfa, Gabriele La Malfa, Panos Pardalos, Giuseppe Di Fatta, Giovanni Giuffrida, and Renato Umeton, editors, *Machine Learning, Optimization, and Data Science*, volume 13811, pages 304–313. Cham, 2023.
- [49] Fusheng Wei, Robert Keeling, Nathaniel Huber-Fliflet, Jianping Zhang, Adam Dabrowski, Jingchao Yang, Qiang Mao, and Han Qin. Empirical study of llm fine-tuning for text classification in legal document review. *Datenverarbeitung im Recht*, 2023.
- [50] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander J Smola. Deep sets. *arXiv preprint arXiv:1703.06114*, 2017.

# APPENDICES

# Appendix A

## Graphs and Results

## A.1 Methods and their Performance

| Method                                      | Accuracy | AUC      | Sensitivity | Specificity |
|---------------------------------------------|----------|----------|-------------|-------------|
| RFT-tfidf-bothpredictions_fuse              | 0.622    | 0.667782 | 0.472574    | 0.756654    |
| RFT-tfidf-oppopredictions_fuse              | 0.614    | 0.657233 | 0.396624    | 0.809886    |
| RFT-tfidf-oppopredictions_first             | 0.608    | 0.658372 | 0.383966    | 0.809886    |
| lrfuse (baseline)                           | 0.604    | 0.649532 | 0.582278    | 0.623574    |
| svmfuse                                     | 0.604    | 0.655837 | 0.565401    | 0.638783    |
| svmfuse                                     | 0.598    | 0.631989 | 0.565401    | 0.627376    |
| LinearSVC-tfidf-bothpredictions_fuse        | 0.588    | 0.634099 | 0.527426    | 0.642586    |
| LinearSVC-tfidf-supppredictions_fuse        | 0.586    | 0.625451 | 0.561181    | 0.608365    |
| LLM-bert-base-uncased-supppredictions_fuse  | 0.584    | 0.597712 | 0.472574    | 0.684411    |
| LLM-bert-base-uncased-supppredictions_first | 0.584    | 0.599076 | 0.481013    | 0.676806    |
| SGD-tfidf-bothpredictions_fuse              | 0.584    | 0.626125 | 0.489451    | 0.669202    |
| RFT-tfidf-supppredictions_fuse              | 0.584    | 0.636305 | 0.523207    | 0.638783    |
| lrfuse                                      | 0.582    | 0.633401 | 0.679325    | 0.494297    |
| LinearSVC-tfidf-supppredictions_first       | 0.582    | 0.620863 | 0.552743    | 0.608365    |
| KNN-tfidf-oppopredictions_fuse              | 0.578    | 0.583065 | 0.734177    | 0.437262    |
| LinearSVC-tfidf-oppopredictions_first       | 0.578    | 0.625451 | 0.506329    | 0.642586    |
| KNN-tfidf-oppopredictions_first             | 0.576    | 0.579647 | 0.725738    | 0.441065    |
| SGD-tfidf-supppredictions_fuse              | 0.576    | 0.612055 | 0.590717    | 0.562738    |
| LinearSVC-tfidf-oppopredictions_fuse        | 0.574    | 0.620029 | 0.506329    | 0.634981    |
| Logistic-tfidf-supppredictions_fuse         | 0.574    | 0.617044 | 0.552743    | 0.593156    |
| RFT-tfidf-supppredictions_first             | 0.574    | 0.623711 | 0.540084    | 0.604563    |
| SGD-tfidf-supppredictions_first             | 0.572    | 0.608525 | 0.586498    | 0.558935    |
| Logistic-tfidf-bothpredictions_fuse         | 0.572    | 0.628676 | 0.497890    | 0.638783    |
| Logistic-tfidf-supppredictions_first        | 0.570    | 0.612167 | 0.556962    | 0.581749    |
| LLM-bert-base-uncased-oppopredictions_first | 0.568    | 0.584621 | 0.573840    | 0.562738    |
| DeepSet-tfidf-bothpredictions               | 0.568    | 0.594487 | 0.443038    | 0.680608    |
| Logistic-tfidf-oppopredictions_first        | 0.564    | 0.625724 | 0.434599    | 0.680608    |
| SGD-tfidf-oppopredictions_fuse              | 0.562    | 0.621617 | 0.350211    | 0.752852    |
| SGD-tfidf-oppopredictions_first             | 0.562    | 0.625259 | 0.350211    | 0.752852    |
| Logistic-tfidf-oppopredictions_fuse         | 0.560    | 0.621521 | 0.438819    | 0.669202    |
| DeepSet-tfidf-oppopredictions               | 0.558    | 0.583    | 0.565401    | 0.551331    |
| KNN-tfidf-bothpredictions_fuse              | 0.558    | 0.583065 | 0.637131    | 0.486692    |
| LLM-bert-base-uncased-oppopredictions_fuse  | 0.558    | 0.580345 | 0.552743    | 0.562738    |
| DeepSet-tfidf-supppredictions               | 0.544    | 0.591183 | 0.556962    | 0.532319    |
| LLM-bert-base-uncased-bothpredictions_fuse  | 0.544    | 0.59383  | 0.527426    | 0.558935    |
| KNN-tfidf-supppredictions_fuse              | 0.540    | 0.551612 | 0.392405    | 0.673004    |
| KNN-tfidf-supppredictions_first             | 0.538    | 0.55028  | 0.367089    | 0.692015    |
| dummy                                       | 0.526    | N/A      | 0.000000    | 1.000000    |

## A.2 Methods and their P-value

| Method                                      | p-value (baseline) | p-value (dummy) |
|---------------------------------------------|--------------------|-----------------|
| RFT-tfidf-bothpredictions_fuse              | 0.455814           | 0.000184        |
| RFT-tfidf-oppopredictions_fuse              | 0.751191           | 0.000154        |
| RFT-tfidf-oppopredictions_first             | 0.937409           | 0.000350        |
| lrfuse (baseline)                           | 1.000000           | 0.006703        |
| svmfuse                                     | 1.000000           | 0.005935        |
| svmfirfirst                                 | 0.826405           | 0.010684        |
| LinearSVC-tfidf-bothpredictions_fuse        | 0.402963           | 0.021197        |
| LinearSVC-tfidf-supppredictions_fuse        | 0.385669           | 0.029415        |
| LLM-bert-base-uncased-supppredictions_fuse  | 0.500055           | 0.022336        |
| LLM-bert-base-uncased-supppredictions_first | 0.509419           | 0.023443        |
| SGD-tfidf-bothpredictions_fuse              | 0.288784           | 0.024561        |
| RFT-tfidf-supppredictions_fuse              | 0.403470           | 0.029117        |
| lrfirst                                     | 0.272269           | 0.057590        |
| LinearSVC-tfidf-supppredictions_first       | 0.283607           | 0.038666        |
| KNN-tfidf-oppopredictions_fuse              | 0.341294           | 0.081729        |
| LinearSVC-tfidf-oppopredictions_first       | 0.236902           | 0.043610        |
| KNN-tfidf-oppopredictions_first             | 0.310039           | 0.089470        |
| SGD-tfidf-supppredictions_fuse              | 0.150675           | 0.066348        |
| LinearSVC-tfidf-oppopredictions_fuse        | 0.154816           | 0.058695        |
| Logistic-tfidf-supppredictions_fuse         | 0.132902           | 0.067914        |
| RFT-tfidf-supppredictions_first             | 0.202924           | 0.065430        |
| SGD-tfidf-supppredictions_first             | 0.101136           | 0.084087        |
| Logistic-tfidf-bothpredictions_fuse         | 0.088782           | 0.065756        |
| Logistic-tfidf-supppredictions_first        | 0.082054           | 0.088458        |
| LLM-bert-base-uncased-oppopredictions_first | 0.233874           | 0.103360        |
| DeepSet-tfidf-bothpredictions               | 0.101433           | 0.072765        |
| Logistic-tfidf-oppopredictions_first        | 0.077262           | 0.093967        |
| SGD-tfidf-oppopredictions_fuse              | 0.073214           | 0.081036        |
| SGD-tfidf-oppopredictions_first             | 0.073214           | 0.081036        |
| Logistic-tfidf-oppopredictions_fuse         | 0.044762           | 0.123455        |
| DeepSet-tfidf-oppopredictions               | 0.041501           | 0.172361        |
| KNN-tfidf-bothpredictions_fuse              | 0.063537           | 0.187563        |
| LLM-bert-base-uncased-oppopredictions_fuse  | 0.131512           | 0.169451        |
| DeepSet-tfidf-supppredictions               | 0.009503           | 0.308236        |
| LLM-bert-base-uncased-bothpredictions_fuse  | 0.047181           | 0.303209        |
| KNN-tfidf-supppredictions_fuse              | 0.013401           | 0.326971        |
| KNN-tfidf-supppredictions_first             | 0.010925           | 0.349899        |
| dummy                                       | 0.013405           | 1.000000        |