Classifying US Supreme Court Legal Opinions

605.744 Information Retrieval
Semester Project

Tom Stuckey
May 2016

Table of Contents

1 Problem and Motivation

US Supreme Court legal opinions are interesting documents. Aside from the legalese, they present an interesting problem space with respect to text classification. Supreme Court legal opinions are fairly complex documents. Although they have a semi-standard form, the phrasal morphology (e.g. “this court reverses” used synonymously with “is reversed”) and additional challenges of sectional dependence (i.e. it matters which section certain phrases occur in the legal opinions) require additional logic when attempting to perform an automatic classification.

This project focuses primarily on developing model(s) to classify these Supreme Court legal opinions and secondarily on presenting some rudimentary information with regard to the President and political party responsible for appointing the justice presenting the legal opinion. Section 2 provides an overview of some related research. Section 3 describes the approach taken. Section 4 provides the results and concluding discussion.

2 Relevant Legal and Scientific Research

In preparing for the project, several related efforts regarding legal classification, in particular, and opinion/sentiment processing, in general, were examined. First, some work from Stanford looked particularly promising in its goal to apply machine learning to legal docket classification, but its overall results were that machine learning using bag-of-words features performed poorly.[1] Second, research aligned with extracting rules from regulations looked promising, but, upon investigation, it was more focused on inferring meaning by parsing sentences and inferring context.[2] Lastly, three bodies of research focused on sentiment analysis by Dr. Lillian Lee from Cornell were examined. *A Sentimental Education* looked at various methodologies infer the sentiment for given span of text.[3] *How Opinions are Received* took a deep view of Amazon reviews to attempt determine sentiment and plagiarism (e.g. bot generated, paid review, etc).[4] *Thumbs Up* looked at several different mechanisms for leveraging supervised machine learning and found that Support Vector Machines worked the best in their case.[5]

[1] Nallapati, R., & Manning, C. D. (2008). Legal docket-entry classification. *Proceedings of the Conference on Empirical Methods in Natural Language Processing - EMNLP '08*. doi:10.3115/1613715.1613771

[2] Wyner, A., & Peters, W. (n.d.). On rule extraction from regulations. Retrieved April 13, 2016, from http://www.researchgate.net/publication/266177190_On_Rule_Extraction_from_Regulations

[3] Pang, B., & Lee, L. (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. Retrieved April 16, 2016, from http://www.cs.cornell.edu/home/llee/papers/cutsent.pdf

[4] Danescu-Niculescu-Mizil, C., Kossinets, G., Kleinberg, J., & Lee, L. (2009). How opinions are received by online communities. *Proceedings of the 18th International Conference on World Wide Web - WWW '09*. doi:10.1145/1526709.1526729

[5] Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up? sentiment classification using machine learning techniques. Retrieved April 16, 2016, from http://www.cs.cornell.edu/home/llee/papers/sentiment.pdf

3 Approach

This project began with the general goal of being able to classify legal opinions. Prior to in-depth research, this naively appeared to be a straightforward problem as each legal opinion would be analogous to a normal trial decision where the justice either rules in-favor of the plaintiff or against plaintiff. Accordingly, the initial project vision had much broader scope with simple binary classification of the legal opinions using Support Vector Machines (SVM) and Naive-Bayes predictions based on the political alignments of the various Supreme Court justices. However, after investigating the situation more thoroughly, it was determined the complexities Supreme Court opinions/decisions can fall in number of different classes, and the project had to be rescoped to more appropriately address this. The following subsections describe the data, provide an overview of the preparation activities, and provide an overview of the classification activities.

3:1 Data Overview

Each US Supreme Court legal opinions has two main areas: the syllabus and the opinion. The syllabus provides background on the overall case outlining its origin and path to the Supreme Court. The opinion section contains the *main opinion* and *disposition* and can additionally contain *concurring opinion(s)* and *dissenting opinions(s)*. The *main opinion* contains the majority justices' rationale backing the official *disposition*. *Concurring opinions* are for justices who agree with the majority *disposition* but who have a different legal rationale than the majority. *Dissenting opinions* are where any justices who disagree with the majority *disposition* record their legal perspective.[6]

Given the above overview courtesy of the American Bar Association, the text classification of the legal opinions would require additional logic to ensure the right terms were coming out of the correct section of the opinion document. Moreover, what about the actual *disposition* itself? The introduction above identifies different sections of each opinion document, but what are the potential values, or classifications, in the *disposition*?

Returning to the American Bar Association reference, there are three types of *dispositions* possible: 1. Affirm: uphold the lower court's ruling 2. Reverse/Void/Vacate: overturn the lower court's ruling 3. Remand: send it back to the lower court for retrial.[7] Unfortunately, from the perspective of text classification, there is another class, the Supreme Court can also decide to *reverse and remand* the case; this means the lower court's original decision is reversed, but that the lower court has to retry the case.

[6] American Bar Association. (2012, September). How to read a Supreme Court opinion. Retrieved March 19, 2016, from http://www.americanbar.org/publications/insights_on_law_andsociety/13/fall_2012/how_to_read_a_ussupremecourtopinion.html

[7] American Bar Association. (2012, September). How to read a Supreme Court opinion. Retrieved March 19, 2016, from http://www.americanbar.org/publications/insights_on_law_andsociety/13/fall_2012/how_to_read_a_ussupremecourtopinion.html

At this point, it appears there are four distinct classifications possible disposition in a Supreme Court legal opinion: affirm, reverse, remand, or reverse and remand. In contemplating the technical implications of pursuing a multi-class identifier and after a recommendation from the project advisor, it was decided pursue a binary model for each class instead of one single multi-class model. Given this decision, and in examining the additional potentials within a given opinion section, it was decided to additionally include an additional class: *unanimous*; this class represented whether or not the opinion was a unanimous decision by the justices or a split decision. Hence, for each opinion, this project attempted to classify each opinion as being relevant / irrelevant for each of the following classes:

- Unanimous
- Affirm
- Reverse
- Remand
- Reverse and remand

A discussion of the development of the features for each of these classes is located in the *Preparation Overview* section after a description of how the opinions were collected.

3:2 Preparation Overview

Each year, the US Supreme Court receives requests to hear upwards of 7,000 cases and hears arguments on about 80 of these per year.[8] Over the period from 2002 to 2015, they issued opinions on an average of 76 opinions per year. These opinions are made available on the US Supreme Court website at: http://www.supremecourt.gov/opinions/opinions.aspx. For this project, it was desirable to both maximize the dataset and minimize errors, so automation was built to pull all the Supreme Court legal decisions readily available from the website, to convert from PDF format to text format, and to remove duplicate opinions; these scripts are available in Appendix B Data Preparation.[9]

This automation yielded 996 textual opinion documents from 2002 through 2015. Next, it was time to identify feature vectors for each classifier. Identifying features for each class was accomplished through an iterative routine of manual inspection of a subset of the legal opinions and grep pattern matching across the whole data set to maximize valid inclusion and minimize conflation. The original goal was to identify single words as features in the opinion section of each legal opinion document, but this proved to be excessively conflationary. Accordingly, it was necessary to expand into larger n-gram phrases as features. Continuing the iterative routine mentioned earlier with n-grams, with n = 2-to-4, the candidate features converged as illustrated in Table 3-1.

[8] Frequently asked questions - Supreme Court of the United States. (n.d.). Retrieved May 6, 2016, from http://www.supremecourt.gov/faq.aspx#faqgi9

[9] Special thanks to project advisor, Dr. Paul McNamee for insights on the source, request mechanism, and PDF-to-text utility.

Table 3-1 Classes and Corresponding Features

Unanimous	Affirm	Reverse	Remand	Reverse and Remand
"for a unanimous court"	"court affirms"	"court reverses"	"court remands"	"reverses and remands"
	"is affirmed"	"is reversed"	"is remanded"	"reverse and remand'
		"court vacates"		"reversed and remanded"
		"is vacated"		

The final step in data preparation was randomize the 996 legal opinion document into an appropriate categories for training, validation, and test. While this could have been accomplished by arbitrarily assigning the data, that could have introduced bias by oversampling from specific times. As a mitigation, I leveraged the *shuf* utility across a defined enumeration of 70% train, 15% dev, and 15% test in a *bash* script to randomly place the opinions into the appropriate categories. This script is available in Appendix B Data Preparation.

3:3 Classification Overview

The actual classification routines can be separated into four areas creating the feature vectors, determining document relevance for training and validation data sets, executing the SVM^{Light} processing routines, and executing the results summarization.

3:3:1 Creating Feature Vectors

Creating the feature vectors for each of the five classes, *unanimous*, *affirm*, *reverse*, *remand*, *reverse and remand*, was a matter of codifying logic in a program to process each file against the list of features for each class (listed in Table 3-1) and to write one row per document across each of the five files (one file per class) with the occurrence of each feature.

Java was selected as the programming language for the application. Even though the features were n-grams, one compromise made for simplicity was to look for occurrences of each feature in a single line of each document. For example, if the feature "reverses and remands" from the *reverse and remand* class spanned two lines in a given opinion document, the application would not record this occurrence of the feature for the document in the *reverse and remand* class file. The biggest challenge was refining the algorithm to correctly identify the various parts of each legal opinion. Small variances in format necessitated substantially more logic to correctly identify things such as the Case Name, the name of the justice providing the main opinion, and

the true start of the disposition section. A listing of this application is available in Appendix C Feature Vector Generation.

3:3:2 Determining Training and Validation Document Relevance

Ideally, determination of individual document relevance for each the five classes would have been done by lawyers or paralegals. Unfortunately, due to time and budget constraints, this was not feasible for this project, although the notion of leveraging Amazon Mechanical Turk for paralegal services was briefly entertained.[10] As workaround, approximation via automation was leveraged. As a compromise in this workaround, during processing of documents for feature vector creation (as described in Section 3:3:1 Creating Feature Vectors), if a document contained any of the feature vectors for a given class, that training or validation document[11] was marked as relevant for that class. This was a significant decision as expert review would have been vastly preferable. This logic was codified in the Feature Vector Generation application package, and the listing is available in Appendix C Feature Vector Generation.

3:3:3 Executing SVM^{Light} Processing

After the feature vectors had been created for the training, validation, and testing categories for each of the five classes, the binary classifier models had to be build and verified. Foremost, SVM^{Light} was selected as the machine learning application. Next, given the number of models to be built, verified, and leveraged for predictions and the different categories of data (train, validation, test) involved, it was decided to build supporting automation with *bash* scripts to standardize the execution of the build, validation, and prediction runs. A listing of this automation is available in Appendix D SVM^{Light} Invocation.

3:3:4 Executing Summarization

After the models were built and validated and the predictions were made, the decision was made to provide a summary overview across all of the test documents to provide a one-stop-shop for each case name, justice who provided the majority opinion, appointing President and political party, number of words in the legal opinion, and the classes to which each legal opinion belongs. The listing for this application is available in Appendix E Summarizer Routine.

[10] https://www.mturk.com/mturk/welcome
[11] Testing documents were left marked as 0 to indicate to SVM^{Light} that they had not yet been categorized.

4 Results and Discussion

In Section 3:3:2, the decision to automatically determine document relevance for both training and validation data based on the presence of ***any*** of that class's feature vectors was described. As a consequence, during validation of the 142 documents, the SVM^{Light} validation routines scored extraordinarily high with respect to precision and recall; in fact, only one document in the validation dataset for the *reverse* class was incorrectly marked (recall dropped to 97.44% for the validation of this class). For the other classes, as expected as a consequence of the automatic relevance determination, validation showed 100% precision and recall.

Appendix A Summary Results contains the output of the Summarizer Routine described in Section 3:3:4. For each document in the test set, it lists the case name, justice who provided the majority opinion, appointing President and political party, number of words in the legal opinion, and the classes to which each legal opinion belongs. From the perspective of text classification, 90 of the 151 legal opinion documents, 59%, were placed in at least one class. Unfortunately, of those 90 opinions placed in at least one class, 38 (42%) were placed in conflicted classes, e.g. "reverse" and "remand" separately instead of "reverse and remand" OR, more egregiously "affirm" and "remand." Consequently, this project would, at best, be considered an initial capability in need of refinements in the relevance markings, feature selection, and single-line limitation for feature comparison.

Appendix A Summary Results

GRANHOLM, GOVERNOR OF MICHIGAN, ET AL. v.|Anthony Kennedy|Ronald Regan (Republican)|23643 affirm WHITFIELD v. UNITED STATES|unknown|unknown|3858 affirm GONZALES, ATTORNEY GENERAL, ET AL. v. RAICH ET|John Paul Stevens|Gerald Ford (Republican)|26469 affirm : remand : unanimous MCCREARY COUNTY, KENTUCKY, ET AL. v. AMERI-|David Souter|George H. W. Bush (Republican)|25748 affirm REHABILITATION AND CORRECTION, ET AL. v.|Stephen Breyer| Bill Clinton (Democrat)|6840 remand KOONS BUICK PONTIAC GMC, INC. v. NIGH|Ruth Bader Ginsburg|Bill Clinton (Democrat)|10337 reverse JAMA v. IMMIGRATION AND CUSTOMS|Antonin Scalia|Ronald Regan (Republican)|13479 affirm SMITH v. MASSACHUSETTS|Antonin Scalia|Ronald Regan (Republican)|6955 remand PETITIONER v. MISSISSIPPI|unknown|unknown|2166 GONZALES, ATTORNEY GENERAL, ET AL. v. O|John Roberts|George W. Bush (Republican)|8226 affirm : remand UNITED STATES v. GEORGIA ET AL.|Antonin Scalia|Ronald Regan (Republican)|4250 remand : reverse SCHEIDLER ET AL. v. NATIONAL ORGANIZATION FOR|Stephen Breyer| Bill Clinton (Democrat)|6401 reverse DAIMLERCHRYSLER CORP. ET AL. v. CUNO ET AL.|John Roberts|George W. Bush (Republican)|8618 SAN REMO HOTEL, L. P., ET AL. v. CITY AND COUNTY|John Paul Stevens|Gerald Ford (Republican)|11397 ARTHUR ANDERSEN LLP v. UNITED STATES|William Rehnquist|Richard Nixon (Republican)|4832 remand : reverse RICKY BELL, WARDEN v. GARY BRADFORD CONE|unknown|unknown|4966 ANZA ET AL. v. IDEAL STEEL SUPPLY CORP.|Anthony Kennedy|Ronald Regan (Republican)|13485 remand : reverse JOSE ERNESTO MEDELLIN, PETITIONER v. DOUG|unknown|unknown|12310 UNITED STATES v. OLSON ET AL.|Stephen Breyer| Bill Clinton (Democrat)|2104 remand : reverse TEXACO INC. v. DAGHER ET AL.|Clarence Thomas|George H. W. Bush (Republican)|2767 reverse PAUL ALLEN DYE v. GERALD HOFBAUER, WARDEN|unknown|unknown|1205 remand : reverse SAMSON v. CALIFORNIA|Clarence Thomas|George H. W. Bush (Republican)|8723 affirm PHILIP MORRIS USA v. WILLIAMS, PERSONAL REPRE-|Stephen Breyer| Bill Clinton (Democrat)|6793 ET AL. v. PERRY, GOVERNOR OF TEXAS, ET AL.|unknown|unknown|45623 affirm KIRCHER ET AL. v. PUTNAM FUNDS TRUST ET AL.|David Souter|George H. W. Bush (Republican)|7413 remand MOHAWK INDUSTRIES, INC., PETITIONER v.|unknown|unknown|206 JOSE PADILLA v. C. T. HANFT, UNITED STATES|unknown|unknown|1050 JONES v. BOCK, WARDEN, ET AL.|John Roberts|George W. Bush (Republican)|9476 NORFOLK SOUTHERN RAILWAY CO. v. SORRELL|John Roberts|George W. Bush (Republican)|9528 remand CAREY, WARDEN v. MUSLADIN|Clarence Thomas|George H. W. Bush (Republican)|5268 reverse JEFFREY JEROME SALINAS v. UNITED STATES|unknown|unknown|231 ENVIRONMENTAL DEFENSE ET AL. v. DUKE ENERGY|David Souter|George H. W. Bush (Republican)|8427 remand : reverse POWEREX CORP. v. RELIANT ENERGY SERVICES,|Antonin Scalia|Ronald Regan (Republican)|9026 remand 05A1233 v.|unknown|unknown|1203 JOHN DOE ET AL. v. ALBERTO R. GONZALES,|unknown|unknown|2362 KENTUCKY RETIREMENT SYSTEMS ET AL. v. EQUAL|Stephen Breyer| Bill Clinton (Democrat)|10663 reverse<u>and</u>remand : reverse BEGAY v. UNITED STATES|Stephen

Breyer| Bill Clinton (Democrat)|10664 GONZALEZ v. UNITED STATES|Anthony Kennedy|Ronald Regan (Republican)|12146 affirm NATIONS ET AL. v. CITY OF NEW YORK|Clarence Thomas|George H. W. Bush (Republican)|4560 affirm DEPARTMENT OF CORRECTIONS v. DANIEL|unknown|unknown|1863 remand : reverse RACHEL HAAS, CAROL HAAS, AND RICHARD HAAS v.|unknown|unknown|659 v. WILLIAM WEAVER|unknown|unknown|3343 ASSOCIATION v. BRENTWOOD ACADEMY|unknown|unknown|5942 BOWLES v. RUSSELL, WARDEN|unknown|unknown|7503 affirm PANETTI v. QUARTERMAN, DIRECTOR, TEXAS|Anthony Kennedy|Ronald Regan (Republican)|19089 remand : reverse LOGAN v. UNITED STATES|Ruth Bader Ginsburg|Bill Clinton (Democrat)|5447 BRENDLIN v. CALIFORNIA|David Souter|George H. W. Bush (Republican)|5899 reverse DANFORTH v. MINNESOTA|John Paul Stevens|Gerald Ford (Republican)|17761 remand : reverse SAFECO INSURANCE CO. OF AMERICA ET AL. v.|David Souter|George H. W. Bush (Republican)|10302 OF AMERICA ET AL. v. BROWN, ATTORNEY GENERAL|John Paul Stevens|Gerald Ford (Republican)|8072 remand 06A375 (06–532) v.|unknown|unknown|1652 KEVIN GREEN v. GENE M. JOHNSON, DIRECTOR,|unknown|unknown|318 KNOWLES, WARDEN v. MIRZAYANCE|Clarence Thomas|George H. W. Bush (Republican)|6446 reverse PATRICK MARLOWE v. UNITED STATES|unknown|unknown|565 CO. ET AL. v. UNITED STATES ET AL.|John Paul Stevens|Gerald Ford (Republican)|8633 reverse BRIDGE ET AL. v. PHOENIX BOND & INDEMNITY CO.|Clarence Thomas|George H. W. Bush (Republican)|8924 affirm PATRICK KENNEDY, PETITIONER v. LOUISIANA|unknown|unknown|1338 ROTHGERY v. GILLESPIE COUNTY, TEXAS|David Souter|George H. W. Bush (Republican)|18487 remand : reverse JOE CLARENCE SMITH v. ARIZONA|unknown|unknown|316 CAROLINA STATE BOARD OF ELECTIONS, ET AL. v.|unknown|unknown|17306 UNITED STUDENT AID FUNDS, INC. v. ESPINOSA|Clarence Thomas|George H. W. Bush (Republican)|7306 affirm FLORIDA v. POWELL|Ruth Bader Ginsburg|Bill Clinton (Democrat)|10406 remand : reverse BRUCE WEYHRAUCH, PETITIONER v. UNITED|unknown|unknown|160 UNITED STATES v. COMSTOCK ET AL.|Stephen Breyer| Bill Clinton (Democrat)|20209 remand : reverse JAMAL KIYEMBA ET AL. v. BARACK H. OBAMA,|unknown|unknown|383 UNITED STATES v. MARCUS|unknown|unknown|5151 reverse<u>and</u>remand : reverse ARTHUR ANDERSEN LLP ET AL. v. CARLISLE ET AL.|Antonin Scalia|Ronald Regan (Republican)|4288 remand : reverse BERGHUIS, WARDEN v. THOMPKINS|Anthony Kennedy|Ronald Regan (Republican)|15918 remand : reverse MCDONALD ET AL. v. CITY OF CHICAGO, ILLINOIS,|unknown|unknown|75260 remand : reverse UNITED STATES v. O'BRIEN ET AL.|Anthony Kennedy|Ronald Regan (Republican)|8778 affirm INSTRUCTION v. FLORES ET AL.|Samuel Alito|George W. Bush (Republican)|30014 GROSS v. FBL FINANCIAL SERVICES, INC.|Clarence Thomas|George H. W. Bush (Republican)|9761 reverse<u>and</u>remand ARTEMUS RICK WALKER v. GEORGIA|unknown|unknown|2221 LAWRENCE W. NELSON, AKA ZIKEE v.|unknown|unknown|905 remand : reverse JOHN ROBERTSON, PETITIONER v. UNITED STATES|unknown|unknown|4012 NKEN v. HOLDER, ATTORNEY GENERAL|John Roberts|George W. Bush (Republican)|11286 CONKRIGHT ET AL. v. FROMMERT ET AL.|John Roberts|George W. Bush (Republican)|12912 remand : reverse BLACK ET AL. v. UNITED STATES|Ruth Bader Ginsburg|Bill Clinton (Democrat)|4205 HEMI GROUP, LLC, ET AL. v. CITY OF NEW YORK,|unknown|unknown|11494 remand : reverse LEWIS ET AL. v. CITY OF CHICAGO, ILLINOIS|Antonin Scalia|Ronald Regan (Republican)|4715 remand : reverse FREEMAN v. UNITED

STATES|unknown|unknown|11164 DAVIS v. UNITED STATES|Samuel Alito|George W. Bush (Republican)|11445 JUNIOR UNIVERSITY v. ROCHE MOLECULAR|John Roberts|George W. Bush (Republican)|10204 affirm MILNER v. DEPARTMENT OF THE NAVY|Elena Kagan|Barrack Obama (Democrat)|11496 BOROUGH OF DURYEA, PENNSYLVANIA, ET AL. v.|Anthony Kennedy|Ronald Regan (Republican)|11200 remand : reverse UNITED STATES v. JAMES FORD SEALE|unknown|unknown|785 THOMPSON v. NORTH AMERICAN STAINLESS, LP|Antonin Scalia|Ronald Regan (Republican)|3605 remand : reverse RENICO, WARDEN v. LETT|unknown|unknown|12646 remand : unanimous : reverseandremand DOLAN v. UNITED STATES|Stephen Breyer| Bill Clinton (Democrat)|9436 SNYDER v. PHELPS ET AL.|John Roberts|George W. Bush (Republican)|11792 affirm DEMARCUS ALI SEARS v. STEPHEN UPTON,|unknown|unknown|6973 remand : reverse JAVIER CAVAZOS, ACTING WARDEN v. SHIRLEY|unknown|unknown|6275 remand MARTEL, WARDEN v. CLAIR|Elena Kagan|Barrack Obama (Democrat)|7937 reverse BLUEFORD v. ARKANSAS|John Roberts|George W. Bush (Republican)|10083 DEPARTMENT OF CORRECTIONS, ET AL. v.|unknown|unknown|2163 ROBERT HUBER ET UX. v. NEW JERSEY DEPART-|unknown|unknown|435 HOWES, WARDEN v. FIELDS|Samuel Alito|George W. Bush (Republican)|8574 ZIVOTOFSKY ET UX. v. CLINTON, SECRETARY|John Roberts|George W. Bush (Republican)|12585 remand : reverse MADISON COUNTY, NEW YORK ET AL. v. ONEIDA|unknown|unknown|336 ET AL. v. NOVO NORDISK A/S ET AL.|Elena Kagan|Barrack Obama (Democrat)|13161 remand : reverse STATE PRISON v. JOSEPH E. CORCORAN|unknown|unknown|2048 remand : reverse MILLER v. ALABAMA|Elena Kagan|Barrack Obama (Democrat)|23439 HERB LUX ET AL. v. NANCY RODRIGUES, IN HER|unknown|unknown|699 MARX v. GENERAL REVENUE CORP.|Clarence Thomas|George H. W. Bush (Republican)|11023 affirm SERVICES v. AUBURN REGIONAL MEDICAL|Ruth Bader Ginsburg|Bill Clinton (Democrat)|8135 remand EMPLOYEE BENEFITS PLAN v. MCCUTCHEN ET AL.|Elena Kagan|Barrack Obama (Democrat)|8282 KOONTZ v. ST. JOHNS RIVER WATER MANAGEMENT|Samuel Alito|George W. Bush (Republican)|16345 remand CITY OF ARLINGTON, TEXAS, ET AL. v. FEDERAL|Antonin Scalia|Ronald Regan (Republican)|14927 affirm ALEXANDER VASQUEZ, PETITIONER v. UNITED|unknown|unknown|168 FLORIDA v. JARDINES|Antonin Scalia|Ronald Regan (Republican)|10396 DUANE EDWARD BUCK v. RICK THALER, DIRECTOR,|unknown|unknown|1678 BOWMAN v. MONSANTO CO. ET AL.|Elena Kagan|Barrack Obama (Democrat)|4591 TARRANT REGIONAL WATER DISTRICT v.|Sonia Sotomayor|Barrack Obama (Democrat)|10645 affirm et al. v. STEVE BULLOCK, ATTORNEY|unknown|unknown|301 OCTANE FITNESS, LLC v. ICON HEALTH & FITNESS,|Sonia Sotomayor|Barrack Obama (Democrat)|5221 remand : reverse BG GROUP PLC v. REPUBLIC OF ARGENTINA|Stephen Breyer| Bill Clinton (Democrat)|16079 reverse MCBURNEY ET AL. v. YOUNG, DEPUTY|Samuel Alito|George W. Bush (Republican)|6684 affirm MARACICH ET AL. v. SPEARS ET AL.|Anthony Kennedy|Ronald Regan (Republican)|18948 remand : reverse SEKHAR v. UNITED STATES|Antonin Scalia|Ronald Regan (Republican)|6334 reverse BURT, WARDEN v. TITLOW|Samuel Alito|George W. Bush (Republican)|6052 reverse SANDIFER ET AL. v. UNITED STATES STEEL CORP.|Antonin Scalia|Ronald Regan (Republican)|6639 affirm UNITED STATES v. KEBODEAUX|Stephen Breyer| Bill Clinton (Democrat)|13722 remand : reverse MICHIGAN v. BAY MILLS INDIAN COMMUNITY ET AL.|Elena Kagan|Barrack Obama (Democrat)|20332 affirm UNITED STATES v. WOODS|Antonin Scalia|Ronald

Regan (Republican)|6983 reverse POM WONDERFUL LLC v. COCA-COLA CO.|Anthony Kennedy|Ronald Regan (Republican)|7659 remand REPUBLIC OF ARGENTINA v. NML CAPITAL, LTD.|Antonin Scalia|Ronald Regan (Republican)|5576 affirm SHELBY COUNTY, ALABAMA v. HOLDER, ATTORNEY|John Roberts|George W. Bush (Republican)|24633 reverse KANSAS v. NEBRASKA ET AL.|Elena Kagan|Barrack Obama (Democrat)|19171 PEREZ, SECRETARY OF LABOR, ET AL. v. MORTGAGE|Sonia Sotomayor|Barrack Obama (Democrat)|16698 reverse TRACEY L. JOHNSON, ET AL. v. CITY OF SHELBY,|unknown|unknown|886 COMMUNITY AFFAIRS ET AL. v. INCLUSIVE|Anthony Kennedy|Ronald Regan (Republican)|27897 affirm : remand BRUMFIELD v. CAIN, WARDEN|Sonia Sotomayor|Barrack Obama (Democrat)|17354 remand LANE v. FRANKS ET AL.|Sonia Sotomayor|Barrack Obama (Democrat)|7850 affirm : remand REED ET AL. v. TOWN OF GILBERT, ARIZONA, ET AL.|Clarence Thomas|George H. W. Bush (Republican)|12391 ALABAMA DEPARTMENT OF REVENUE ET AL. v. CSX|Antonin Scalia|Ronald Regan (Republican)|8133 remand : reverse ANTHONY RAY HINTON v. ALABAMA|unknown|unknown|4854 WHITFIELD v. UNITED STATES|Antonin Scalia|Ronald Regan (Republican)|2080 affirm v. SHARIF|Sonia Sotomayor|Barrack Obama (Democrat)|23522 remand RAUL LOPEZ, WARDEN v. MARVIN VERNIS SMITH|unknown|unknown|2870 remand : reverse T-MOBILE SOUTH, LLC v. CITY OF ROSWELL,|Sonia Sotomayor|Barrack Obama (Democrat)|10079 ALABAMA ET AL. v. NORTH CAROLINA|Antonin Scalia|Ronald Regan (Republican)|14814 NEW JERSEY v. DELAWARE|Ruth Bader Ginsburg|Bill Clinton (Democrat)|19878 BAKER BOTTS L.L.P. ET AL. v. ASARCO LLC|Clarence Thomas|George H. W. Bush (Republican)|8366 KING ET AL. v. BURWELL, SECRETARY OF HEALTH|John Roberts|George W. Bush (Republican)|17626 MOTOR VEHICLES BOARD, ET AL. v. TEXAS|Stephen Breyer| Bill Clinton (Democrat)|14080 ARMSTRONG ET AL. v. EXCEPTIONAL CHILD|unknown|unknown|10799 affirm HARRIS v. VIEGELAHN, CHAPTER 13 TRUSTEE|Ruth Bader Ginsburg|Bill Clinton (Democrat)|4916 remand : reverse KANSAS v. CARR|Antonin Scalia|Ronald Regan (Republican)|10724 FEDERAL ENERGY REGULATORY COMMISSION v.|Elena Kagan|Barrack Obama (Democrat)|36782 reverse<u>and</u>remand MELENE JAMES v. CITY OF BOISE, IDAHO, ET AL.|unknown|unknown|594 remand : reverse

We had 90 with at least one classification

Appendix B Data Preparation Routine

Get the PDFs from the Supreme Court website

```
#!/bin/bash
#Get the information from www.supremecourt.gov via wget

directoryPrefix=supremecourt_
script_name=get_data.sh
mypath=../data/datasources

#get to where the script is invoked from
parent_path=$( cd "$(dirname "${BASH_SOURCE}")" ; pwd -P )
cd "$parent_path"
#and then get to the right relative path
cd "$mypath"

#Lets iterate from 2003(oldest available) to 2015 (newest available)
#This is the prep work to make the request
for ((i=3;i<=15;i++));
  do
    if [ "$i" -lt "10" ]
      then
          #put in a leading zero so we make correct HTTP request
          year="0$i"
      else
          year="$i"
    fi
    echo "$year"


    directoryname=$directoryPrefix$year
    #check to see if the directory exists
    if [ -d "$directoryname" ]
      then
        #delete the existing directory
        rm -rf "$directoryname"
    fi
    #create a fresh directory
    mkdir "$directoryname"
    cd "$directoryname"
```

```
    #check to see if there is already an old copy of this script there
    if [ -e "$script_name" ]
      then
      #delete the existing file
      rm "$script_name"
    fi

    if [ "$i" -le "11" ]
      #2003-to-2011 is done one way
      then
        #create the script to make the wget request
        printf "#!/bin/sh\n">$script_name
        printf "wget -r --accept=pdf http://www.supremecourt.gov/opinions/"$year"pdf"
>>$script_name

      #2012-to-2015 is done another way
      else
        #create the script to make the wget request
        printf "#!/bin/sh\n">$script_name
        printf "wget -r --accept=pdf http://www.supremecourt.gov/opinions/slipopinion
/$year">>$script_name
    fi

    chmod +x "$script_name"
    #Ok, let's run the script; note we are just running these sequentially
    ./"$script_name"
    cd ..
done
```

Convert the PDFs to Text with the Apache Tika library

```
#!/bin/bash
#
mypath=../../data/datasources
mydestination=../../data/datasources-converted
tikapath=../../lib/tika-app-1.12.jar

#get to where the script is invoked from
parent_path=$( cd "$(dirname "${BASH_SOURCE}")" ; pwd -P )
cd "$parent_path"
#and then get to the right relative path
cd "$mypath"

find . -iname \*pdf -type d -print0 | while IFS= read -r -d '' valid_directory;
do
  #printf "Valid Directory: ""$valid_directory%s\n"
   find "$valid_directory" -iname \*.pdf -type f -print0 | while IFS= read -r -d '' f
ile;
   do
        rename=`echo "$file" | sed 's/\.pdf/\.txt/'`
        #printf "%s\t""$rename""%s\n"
        filename_only=`basename "$rename"`
        output="$mydestination"/"$filename_only"
        printf "Destination file is ""$output""%s\n"
        java -jar "$tikapath" --text "$file" > "$output"
        printf "$file%s\n"
   done
done
```

Remove Duplicate Text Files

```
#!/bin/bash
#
mypath=../../data/datasources-converted

#get to where the script is invoked from
parent_path=$( cd "$(dirname "${BASH_SOURCE}")" ; pwd -P )
cd "$parent_path"
#and then get to the right relative path
cd "$mypath"

find . -iname \*\(1\).txt -type f -print0 | while IFS= read -r -d '' dupfile;
   do
        rm "$dupfile"
   done
```

Randomize the dataset into Train/Dev/Test

```
#!/bin/bash
#
mypath=../../data/datasources-converted
mydestination=../../data/datasources-partitioned

#get to where the script is invoked from
parent_path=$( cd "$(dirname "${BASH_SOURCE}")" ; pwd -P )
cd "$parent_path"
#and then get to the right relative path
cd "$mypath"

find . -iname \*.txt -type f -print0 | while IFS= read -r -d '' srcfile;
   do
        srcfilename_only=`basename "$srcfile"`
        target=`gshuf -e train train train train train train train train train train
train train train train train dev dev dev test test test -n 1`
        #printf "$mydestination""/""$target""/""$srcfilename_only%s\n"
        #printf "$srcfile%s\n"
        destfile="$mydestination""/""$target""/""$srcfilename_only"
        cp "$srcfile" "$destfile"
   done
```

Appendix C Feature Vector Generation

```
package core;

import org.apache.commons.io.filefilter.FileFilterUtils;
import org.apache.commons.io.filefilter.HiddenFileFilter;
import utilities.MydirectoryWalker;

import java.io.File;
import java.util.List;

/**
 * Stages a list of all the files with a give nsuffix in a given directory
 */
public class DirectoryParser {
    public File startDirectory;
    public List results;

    public DirectoryParser(String top_directory){
        this.startDirectory = null;
        this.results=null;
        try {
            startDirectory = new File(top_directory);
        } catch (Exception e) {
            System.err.println("Failed to find the directory " + top_directory);
            System.exit(1);
            e.printStackTrace();
        }
    }

    /**
     * Get files with a given suffix and store the internally in the resutls list
     * @param suffix
     */
    public void getFilesbySuffix(String suffix){
        MydirectoryWalker mywalker = new MydirectoryWalker(
                HiddenFileFilter.VISIBLE,
                FileFilterUtils.suffixFileFilter(suffix)
        );
        results = mywalker.parseDirectory(startDirectory);
    }
```

```
    /**
     * Get files with a given prefix and store them internally in the results list
     * @param prefix
     */
    public void getFilesbyPrefix(String prefix){
        MydirectoryWalker mywalker = new MydirectoryWalker(
                HiddenFileFilter.VISIBLE,
                FileFilterUtils.prefixFileFilter(prefix)
        );
        results = mywalker.parseDirectory(startDirectory);
    }


}package core;

import utilities.ClassificationType;
import utilities.DocumentInfo;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.HashMap;
import java.util.Map;

/**
 * Handles the Processing of the feature file and maintains the HashMap we'll reuse d
uring
 * pcoesssing of source text
 */
public class FeaturesProcessor {
    HashMap<String, ClassificationType> myClassifications; //key: classification name
; value: classifcation info

    public FeaturesProcessor(String featureFile) {
        myClassifications = new HashMap<>();
        BufferedReader featureVectorBufferedReader = setupInputReader(featureFile);
        processFile(featureVectorBufferedReader);
    }

    /**
     * Create an instance of the BufferedRread to process the feature file
     *
     * @param featureFile
     * @return
     */
```

```
    private BufferedReader setupInputReader(String featureFile) {
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader(featureFile));
        } catch (FileNotFoundException e) {
            System.out.println("FeaturesProcessor: had a problem reading feature file
");
            e.printStackTrace();
        }
        return br;
    }

    /**
     * Process the Feature Vector File and populate the myClassifications HashMap str
ucture
     */
    private void processFile(BufferedReader featureVectorBufferedReader) {
        if (featureVectorBufferedReader == null) {
            return;
        }//just return if the buffered reader is null
        try {
            String lastClassification = null;
            String line = featureVectorBufferedReader.readLine();
            while (line != null) {
                line=line.trim();
                if (line.endsWith(":")) {
                    //features end with a ':'
                    String aClassification = line.substring(0, line.length() - 1);
                    if (FeatureVectorsCreator.DEBUG) {
                        System.out.println("FeaturesProcessor: aClassification is: "
+ aClassification);
                    }
                    //shouldn't need to check as we are just processing one of the co
nfiguration files
                    //at this point, but just for consistency/safety, let's ensure we
 don't already
                    //have the key
                    if (!(myClassifications.containsKey(aClassification))) {
                        myClassifications.put(aClassification, new ClassificationType
());
                        lastClassification = aClassification;
                    }
                }
                if (line.contains("|")) {
                    String[] tmp = line.split("[|]");
```

```
                    Integer featureID = Integer.parseInt(tmp[0]);
                    String feature = tmp[1];
                    if (FeatureVectorsCreator.DEBUG) {
                        System.out.println("FeaturesProcessor: adding to Classificati
on: " + lastClassification +
                                " the feature: " + feature + " with featureID: " + fe
atureID);
                    }
                    myClassifications.get(lastClassification).addFeature(feature,feat
ureID);
                }
                line = featureVectorBufferedReader.readLine();
            }//end while
            featureVectorBufferedReader.close();
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("FeaturesProcessor had a problem: " + e.getMessage());
            System.exit(3);
        }
    }

    /**
     * Clears out the Document feature counts so this instance can be reused for each
     * document processing routine
     */
    public void clearFeatureCounts() {
        for (Map.Entry<String, ClassificationType> classificationEntry : myClassifica
tions.entrySet()) {
            classificationEntry.getValue().hasAnyFeatures = false;//reset the flag
            HashMap<String, DocumentInfo> t_featureHash = classificationEntry.getValu
e().featureHash;
            for (Map.Entry<String, DocumentInfo> featureEntry : t_featureHash.entrySe
t()) {
                featureEntry.getValue().featureCount = 0;//reset the invidual count f
or this feature
            }
        }
    }

}
package core;

public class FeatureVectorsCreator {
    public static boolean DEBUG = false;
    public static String OPERATION = "train";
```

```
    public static void main(String[] args) {
        if (args.length != 5) {
            System.out.println("Usage: FeatureVectorsCreator directory_with_src_texts
 feature-file.txt judge-appointer.txt" +
                    " output_directory train/dev/test ");
            System.exit(2);
        }
        String topDirectory = args[0];
        String featureFile = args[1];
        String judgeAppointerFile = args[2];
        String outputDirectory = args[3];

        if (args[4].toLowerCase().equals("train")){OPERATION="train";}
        if (args[4].toLowerCase().equals("dev")){OPERATION="dev";}
        if (args[4].toLowerCase().equals("test")){OPERATION="test";}

        FeaturesProcessor featuresProcessor = new FeaturesProcessor(featureFile);
        JudgeProcessor judgeProcessor = new JudgeProcessor(judgeAppointerFile);
        DirectoryParser directory = new DirectoryParser(topDirectory);
        directory.getFilesbySuffix(".txt");
        OutputProcessor outputProcessor = new OutputProcessor(outputDirectory, featur
esProcessor);
        new SourceTextProcessor(directory.results, featuresProcessor, judgeProcessor,
 outputProcessor);
    }
}
package core;

import utilities.ClassificationType;
import utilities.DocumentInfo;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.HashMap;
import java.util.Map;

/**
 * Handles the Processing of the Judge Appointer file into  a HashMap
 */
public class JudgeProcessor {
    HashMap<String, String> judgeAppointerMap; //key: Judge Name; value: Appointer na
me and political party
```

```
    public JudgeProcessor(String featureFile) {
        judgeAppointerMap = new HashMap<>();
        BufferedReader featureVectorBufferedReader = setupInputReader(featureFile);
        processFile(featureVectorBufferedReader);
    }

    public class JudgeAppointerPair {
        String judgeFullName;
        String appointerAndParty;
    }

    /**
     * Create an instance of the BufferedRead to process the feature file
     *
     * @param featureFile
     * @return
     */
    private BufferedReader setupInputReader(String featureFile) {
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader(featureFile));
        } catch (FileNotFoundException e) {
            System.out.println("JudgeProcessor: had a problem reading feature file");
            e.printStackTrace();
        }
        return br;
    }

    /**
     * Process the Judge Appointer File and populate the HashMap structure
     */
    private void processFile(BufferedReader featureVectorBufferedReader) {
        if (featureVectorBufferedReader == null) {
            return;
        }//just return if the buffered reader is null
        try {
            String line = featureVectorBufferedReader.readLine();
            while (line != null) {
                String[] tokens = line.trim().split("[:]");
                judgeAppointerMap.put(tokens[0], tokens[1]);
                if (FeatureVectorsCreator.DEBUG) {
                    System.out.println("JudgeProcessor: judge: " + tokens[0] + " || a
ppointer: " + tokens[1]);
                }
                line = featureVectorBufferedReader.readLine();
```

```
            }//end while
            featureVectorBufferedReader.close();
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("JudgeProcessor had a problem: " + e.getMessage());
            System.exit(4);
        }
    }

    private String cleanJudgeName(String name) {
        if ((name.equals("unknown")) ||
                (name.equals("chief justice")) ||
                (name.equals("per curiam"))  ||
                (!name.contains(","))){
            return name;
        }
        String cleanName = name.toLowerCase();
        cleanName = cleanName.substring(0, cleanName.indexOf(","));
        cleanName = cleanName.replace(".", "");
        cleanName = cleanName.trim();
        return cleanName;
    }

    /**
     * Lookup Appointer for a given judge
     *
     * @param judge
     * @return
     */
    public JudgeAppointerPair lookupAppointer(String judge) {
        JudgeAppointerPair judgeAppointerPair = new JudgeAppointerPair();
        judgeAppointerPair.judgeFullName = "unknown";
        judgeAppointerPair.appointerAndParty = "unknown";

        for (Map.Entry<String, String> judgeEntry : judgeAppointerMap.entrySet()) {
            if (judgeEntry.getKey().toLowerCase().contains(cleanJudgeName(judge))) {
                judgeAppointerPair.judgeFullName = judgeEntry.getKey();
                judgeAppointerPair.appointerAndParty = judgeEntry.getValue();
            }
        }
        return judgeAppointerPair;
    }
}
package core;
```

```
import utilities.ClassificationType;

import java.io.*;
import java.nio.file.FileSystems;
import java.nio.file.Files;
import java.util.HashMap;
import java.util.Map;

/**
 * Handles the reference to the output directory and the file handles thereof
 */
public class OutputProcessor {
    public final String SUMMARY_INFO_OUTPUT_PREFIX = "_Summary_info_";
    public final String FEATURE_FILE_DELIMTER = "\t";
    public final String SUMMARY_FILE_DELIMTER = "|";

    BufferedWriter summaryFileBufferedWriter;
    HashMap<String, BufferedWriter> featureFileBufferedWriters;

    public OutputProcessor(String output_directory, FeaturesProcessor featuresProcess
or) {
        //setup the summary outputfile BufferedWriter
        summaryFileBufferedWriter = setupOutputWriter(output_directory + File.separat
or
                + SUMMARY_INFO_OUTPUT_PREFIX + FeatureVectorsCreator.OPERATION );
        featureFileBufferedWriters = new HashMap<>();
        //setup the BufferedWriter for each Classification Type
        for (Map.Entry<String, ClassificationType> entry : featuresProcessor.myClassi
fications.entrySet()) {
            String eachClassification = entry.getKey();
            featureFileBufferedWriters.put(eachClassification,
                    setupOutputWriter(output_directory + File.separator
                            + FeatureVectorsCreator.OPERATION + "_" + eachClassificat
ion.replace(" ","_")));
        }
    }

    /**
     * Setup a BufferedWriter for the given filename
     *
     * @param filename
     * @return reference to the new BufferedWriter
     */
    private BufferedWriter setupOutputWriter(String filename) {
```

```
        BufferedWriter mybufferedWriter = null;
        try {
            Files.deleteIfExists(FileSystems.getDefault().getPath(filename));
            File t_file = new File(filename);//create the handle
            t_file.createNewFile();//touch the file
            mybufferedWriter = new BufferedWriter(new FileWriter(t_file,true));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return mybufferedWriter;
    }

    public void iterateAndOutputFeatureVectors() {
/*        for (Map.Entry<String, ReverseCollectionInfo> entry : docToTerm.entrySet())
 {
            String docID = entry.getKey();
            ReverseCollectionInfo termsInfo = entry.getValue();
            //Write the document information
            featureVectorFileProcessing.writeDocumentInfo(docID, termsInfo);
        }
        featureVectorFileProcessing.closeFile();*/
    }

    /**
     * Writeout the line of text and follow it up with a newline
      * @param value
     */
   public void writeSummaryEntry(String value, Boolean new_line){
       try {
           summaryFileBufferedWriter.write(value);
           if (new_line){
               summaryFileBufferedWriter.newLine();
           }else{
               summaryFileBufferedWriter.write(SUMMARY_FILE_DELIMTER);
           }
       } catch (IOException e) {
           e.printStackTrace();
       }
   }


    /**
     * Find the BufferedWriter for the classifer
```

```
     * @param classifier
     * @return BufferedWriter referecne
     */
    public BufferedWriter getAppropriateBufferedWriter(String classifier){
       BufferedWriter correctBW=null;
        for (Map.Entry<String, BufferedWriter>bwEntry : featureFileBufferedWriters.en
trySet()){
            //Be sure to test for full string equality between the key and the classi
fier
            //as some classifers are subsets of others
            //A contains test vs a equals test will have contents being written to un
intended files
            if (bwEntry.getKey().equals(classifier)){
                correctBW = bwEntry.getValue();
            }
        }
        return correctBW;
    }

    /**
     * Write out a value and follow it up with either a carriage return (if new_line)
 is specified
     * or the delimiter for the feature file
     * @param classifierBufferedWriter
     * @param value
     * @param new_line
     */
    public void writeClassifierEntry(BufferedWriter classifierBufferedWriter, String
value, Boolean new_line){

        if (classifierBufferedWriter==null){return;}//just head back if there isn't a
 BufferedWriter reference
        try {
            if (new_line){
                classifierBufferedWriter.write(value);
                classifierBufferedWriter.newLine();
            }else{
                classifierBufferedWriter.write(value);
                classifierBufferedWriter.write(FEATURE_FILE_DELIMTER);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    public void closeFiles(){
        try {
            summaryFileBufferedWriter.flush();
            summaryFileBufferedWriter.close();
            for (Map.Entry<String,BufferedWriter>bwEntry : featureFileBufferedWriters
.entrySet()){
                bwEntry.getValue().flush();
                bwEntry.getValue().close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

}
package core;

import utilities.ClassificationType;
import utilities.DocumentInfo;

import java.io.*;
import java.util.*;

public class SourceTextProcessor {
    List srcTextFiles;
    FeaturesProcessor featuresProcessor;
    OutputProcessor outputProcessor;
    Integer docCount;
    Integer overallWordCount;

    class CaseInfo {
        String caseName = "none";
        String opinionAuthor = "per curiam"; //default meaning general per the court,
 not Judge specific
        Integer docWordCount = 0;
    }

    public SourceTextProcessor(List files, FeaturesProcessor featuresProcessor, Judge
Processor judgeProcessor,
                               OutputProcessor outputProcessor) {
        this.srcTextFiles = files;
        this.featuresProcessor = featuresProcessor;
        this.outputProcessor = outputProcessor;
        docCount = 0;
```

```
        overallWordCount = 0;

        Iterator<File> myiterator = srcTextFiles.iterator();
        File t_file;
        //Iterate through the documents
        while (myiterator.hasNext()) {
            t_file = myiterator.next();
            if (FeatureVectorsCreator.DEBUG) {
                System.out.println("SourceTextProcessor: Processing source text file:
 " + t_file);
            }
                try {
                    //clear out the counts in the featuresProcessor HashMap before we
 start processing the new file
                    featuresProcessor.clearFeatureCounts();
                    docCount++;
                    BufferedReader srcTextBufferedReader = setupInputReader(t_file.ge
tCanonicalPath());
                    CaseInfo caseInfo = processFile(srcTextBufferedReader);
                    writeOutResults(caseInfo, featuresProcessor, judgeProcessor);
                } catch (IOException e) {
                    e.printStackTrace();
                }
        }//end document iteration
        System.out.println("Total Documents processed: " + docCount);
        System.out.println("Total Words processed: " + overallWordCount);
        outputProcessor.closeFiles();//close the file handles
    }

    /**
     * Create an instance of the BufferedRread to process the source text file
     *
     * @param srcFile
     * @return
     */
    private BufferedReader setupInputReader(String srcFile) {
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader(srcFile));
        } catch (FileNotFoundException e) {
            System.out.println("SourceTextProcessor: had a problem reading  source te
xtfile " + srcFile);
            e.printStackTrace();
        }
        return br;
```

```
    }

    private Integer countWordsinLine(String line) {
        String[] tokens = line.split("[\\s]");
        return tokens.length;
    }

    private CaseInfo processFile(BufferedReader srcTextBufferedReader) {
        if (srcTextBufferedReader == null) {
            return null;
        }//just return if the buffered reader is null
        CaseInfo caseInfo = new CaseInfo();
        try {
            Boolean checkCaseName = true;
            Boolean checkOpinionAuthor = true;
            Boolean inSyllabusSection = true;
            Boolean inOpinionSection = false;

            String line = srcTextBufferedReader.readLine();
            while (line != null) {
                line = line.trim();
                overallWordCount = overallWordCount + countWordsinLine(line); //keep
track of the overall word count as a metric
                caseInfo.docWordCount = caseInfo.docWordCount + countWordsinLine(line
);//keep track of the doc word count as a metric
                if ((inSyllabusSection) && (checkCaseName)) {
                    if ((line.toLowerCase().contains("v.")) &&
                            !(line.toLowerCase().contains("see"))
                            ) {
                        caseInfo.caseName = line;
                        if (FeatureVectorsCreator.DEBUG) {
                            System.out.println("\tSourceTextProcessor: Case Name is:
" + caseInfo.caseName);
                        }
                        checkCaseName = false;
                    }
                }//end checkCaseName

                if ((inSyllabusSection) && (checkOpinionAuthor)) {
                    if (line.toLowerCase().contains("delivered the opinion")) {
                        if (line.toLowerCase().contains("chief justice")){
                            caseInfo.opinionAuthor = "chief justice";
                        }else{
                            if (line.toLowerCase().contains(".")){
                                caseInfo.opinionAuthor = line.substring(0, line.index
```

```
Of("."));
                            }else{
                                caseInfo.opinionAuthor = line.substring(0,line.indexO
f(" delivered the opinion"));
                            }
                        }
                        if (FeatureVectorsCreator.DEBUG) {
                            System.out.println("\tSourceTextProcessor: Opinion Author
 is: " + caseInfo.opinionAuthor);
                        }
                        checkOpinionAuthor = false;
                    }
                }//end checkOpinionAuthor

                if (inSyllabusSection) {
                    if ((line.toLowerCase().startsWith("opinion of the court")) ||
                            (line.contains("per curiam"))) {
                        inSyllabusSection = false;
                        inOpinionSection = true;
                    }
                }//end syllabus processing
                if (inOpinionSection) {
                    //OK we're processing the main body of the opinion now
                    //Iterate through each classification
                    for (Map.Entry<String, ClassificationType> classificationEntry :
featuresProcessor.myClassifications.entrySet()) {
                        HashMap<String, DocumentInfo> t_featureHash = classificationE
ntry.getValue().featureHash;
                        //System.out.println("\t\t\tSoureTextProcessor: looking at cl
ass: "+classificationEntry.getKey());
                        //Iterate through each each feature in each classification
                        for (Map.Entry<String, DocumentInfo> featureEntry : t_feature
Hash.entrySet()) {
                            String thisFeature = featureEntry.getKey();//get the feat
ure text
                            //check to see if the line of source text contains thisFe
ature
                            if (line.toLowerCase().contains(thisFeature)) {
                                Integer currentFeatureCount = featureEntry.getValue()
.featureCount;
                                featureEntry.getValue().featureCount = currentFeature
Count+1;//increment the feature count
                                if (FeatureVectorsCreator.DEBUG) {
                                    System.out.println("\t\t\t\tSourceTextProcessor:
got a match on class: " + classificationEntry.getKey() + " | for feature: " + thisFea
```

```
ture);
                                    System.out.println("\t\t\t\t\t Its featureCount i
s now: "+featureEntry.getValue().featureCount);
                                }
                                classificationEntry.getValue().hasAnyFeatures = true;
//if any features for this classification
                                // are true, make sure this flag is set to true
                            }//end if the feature is contained
                        }//end feature iteration
                    }//end classification iteration
                } //end opinion processing
                line = srcTextBufferedReader.readLine();
            }//end line processing
            srcTextBufferedReader.close();
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("processFile had a problem: " + e.getMessage());
            System.exit(3);
        }
        return caseInfo;
    }

    /**
     * Write out the Results to the files; this starts off with the summary file and
then iterates through each class
     * (one file per class) for all classes
     * @param caseInfo
     * @param featuresProcessor
     * @param judgeProcessor
     */
    private void writeOutResults(CaseInfo caseInfo, FeaturesProcessor featuresProcess
or, JudgeProcessor judgeProcessor) {
        JudgeProcessor.JudgeAppointerPair ja = judgeProcessor.lookupAppointer(caseInf
o.opinionAuthor);

        outputProcessor.writeSummaryEntry(caseInfo.caseName, false);
        outputProcessor.writeSummaryEntry(ja.judgeFullName, false);
        outputProcessor.writeSummaryEntry(ja.appointerAndParty, false);
        outputProcessor.writeSummaryEntry(caseInfo.docWordCount.toString(), true);

        //iterate through the classifiers and write out the info
        for (Map.Entry<String, ClassificationType> classificationTypeEntry : features
Processor.myClassifications.entrySet()) {
            //get the appropriate BufferWriter File Handle
            BufferedWriter bufferedWriter = outputProcessor.getAppropriateBufferedWri
```

```
ter(classificationTypeEntry.getKey());

            if (classificationTypeEntry.getValue().hasAnyFeatures) {
                if (FeatureVectorsCreator.DEBUG) {
                    System.out.println("YES, on doc: " + docCount + ", we had some fe
atures! for class: " + classificationTypeEntry.getKey());
                }
                //had some features
                Integer relevance = getRelevance(true, FeatureVectorsCreator.OPERATIO
N);
                outputProcessor.writeClassifierEntry(bufferedWriter, relevance.toStri
ng(),false);
                iterateThroughFeatures(bufferedWriter, classificationTypeEntry.getVal
ue().featureHash);
                outputProcessor.writeClassifierEntry(bufferedWriter,"",true);
            } else {
                if (FeatureVectorsCreator.DEBUG){
                    System.out.println("NO, on doc: "+docCount+", we had no features
for class: "+classificationTypeEntry.getKey());
                }
                //had no features
                Integer relevance = getRelevance(false, FeatureVectorsCreator.OPERATI
ON);
                outputProcessor.writeClassifierEntry(bufferedWriter, relevance.toStri
ng(),true);
            }
        }//end classification entry iteration
    }

    private void iterateThroughFeatures(BufferedWriter bufferedWriter, HashMap<String
, DocumentInfo> featureHash){
        Map<String,DocumentInfo> sortedFeatureHash=getHashSortedbyCount(featureHash);
        for (Map.Entry<String,DocumentInfo> featureEntry : sortedFeatureHash.entrySet
()){
            DocumentInfo documentInfo = featureEntry.getValue();
            if (documentInfo.featureCount>0) {
                outputProcessor.writeClassifierEntry(bufferedWriter, documentInfo.fea
tureID + ":" + documentInfo.featureCount, false);
            }
        }
    }
    /**
     * Sort regular K,V Hashmap by V
     * @param unsortedMap
     * @return sortedMap
```

```
     */
    private Map<String, DocumentInfo> getHashSortedbyCount(Map<String, DocumentInfo>
unsortedMap) {
        List<Map.Entry<String, DocumentInfo>> list = new LinkedList<Map.Entry<String,
 DocumentInfo>>(unsortedMap.entrySet());
        // Sorting the list based on values
        Collections.sort(list, new Comparator<Map.Entry<String, DocumentInfo>>() {
            public int compare(Map.Entry<String, DocumentInfo> o1, Map.Entry<String,
DocumentInfo> o2) {
                //sort in descending numeric value order
                return o1.getValue().featureID.compareTo(o2.getValue().featureID);
            }
        });

        // Maintaining insertion order with the help of LinkedList
        Map<String, DocumentInfo> sortedMap = new LinkedHashMap<String, DocumentInfo>
();
        for (Map.Entry<String, DocumentInfo> entry : list) {
            sortedMap.put(entry.getKey(), entry.getValue());
        }
        return sortedMap;
    }

    /**
     * Determine the value of relevance to put return based on relevance and the mode
 of operation we are in.
     * Training and Dev mode, we write a 1 or a -1 based on the boolean parameter
     * Test mode, we always write a 0 as SVM will compute it for us
     *
     * @param hadFeatures
     * @param operation
     * @return
     */
    private Integer getRelevance(boolean hadFeatures, String operation) {
        Integer relevant = -1;
        if ((hadFeatures) && operation.equals("train")){
            relevant =1;
        }
        if ((hadFeatures) && operation.equals("dev")){
            relevant =1;
        }
        if (operation.equals("test")){
            relevant=0;
        }
        return relevant;
```

```
    }

}
package utilities;

import java.util.HashMap;

/**
 * Created by thomasstuckey on 4/23/16.
 */
public class ClassificationType {
    public HashMap<String, DocumentInfo> featureHash;//key: feature name;
                                                     // value: counts per Document fo
r each feature
    public Boolean hasAnyFeatures = false;

    public ClassificationType() {
        this.featureHash = new HashMap<>();
    }

    public void addFeature(String feature, Integer featureID) {
            featureHash.put(feature,new DocumentInfo(featureID));
    }

}
package utilities;

/**
 * Created by thomasstuckey on 1/30/16.
 */
public class DocumentInfo {
    public Integer featureID = 0;
    public Integer featureCount = 0;

    public DocumentInfo(Integer t_featureID) {
        this.featureID=t_featureID;
    }

}
package utilities;

import org.apache.commons.io.DirectoryWalker;
import org.apache.commons.io.filefilter.IOFileFilter;

import java.io.File;
```

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

/**
 * Created with IntelliJ IDEA.
 * User: thomasstuckey
 * Date: 7/23/12
 * Time: 9:01 PM
 * To change this template use File | Settings | File Templates.
 */
public class MydirectoryWalker extends DirectoryWalker {

    public MydirectoryWalker(IOFileFilter dirFilter, IOFileFilter fileFilter) {
        super(dirFilter, fileFilter, -1);
    }

    /**Parse the directory by invoking the walk method that searches for file
     * names that match the
     *
     * @param startDirectory
     * @return
     */
    public List parseDirectory(File startDirectory){
        List results = new ArrayList();
        try {
            this.walk(startDirectory, results);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return results;
    }


    /**Adds the file to the results.
     *
     * @param file to be added to the results
     * @param depth
     * @param results the list of files that meet the criteria.
     */
    protected void handleFile(File file, int depth, Collection results){
        results.add(file);
    }
```

```
}
```

Appendix D SVM^{Light} Invocation

Train the Models and Validate the Models

```
#!/bin/bash

./svm_light_osx/svm_learn ../../data/output_data/train_affirm ../../data/learner_data
/model_affirm
./svm_light_osx/svm_learn ../../data/output_data/train_remand ../../data/learner_data
/model_remand
./svm_light_osx/svm_learn ../../data/output_data/train_reverse ../../data/learner_dat
a/model_reverse
./svm_light_osx/svm_learn ../../data/output_data/train_reverse_and_remand ../../data/
learner_data/model_reverse_and_remand
./svm_light_osx/svm_learn ../../data/output_data/train_unanimous ../../data/learner_d
ata/model_unanimous

./svm_light_osx/svm_classify  ../../data/output_data/dev_affirm ../../data/learner_da
ta/model_affirm ../../data/learner_data/predictions_dev_affirm
./svm_light_osx/svm_classify  ../../data/output_data/dev_remand ../../data/learner_da
ta/model_remand ../../data/learner_data/predictions_dev_remand
./svm_light_osx/svm_classify  ../../data/output_data/dev_reverse ../../data/learner_d
ata/model_reverse ../../data/learner_data/predictions_dev_reverse
./svm_light_osx/svm_classify  ../../data/output_data/dev_reverse_and_remand ../../dat
a/learner_data/model_reverse_and_remand ../../data/learner_data/predictions_dev_rever
se_and_remand
./svm_light_osx/svm_classify  ../../data/output_data/dev_unanimous ../../data/learner
_data/model_unanimous ../../data/learner_data/predictions_dev_unanimous
```

Train the Models and get predictions on the Test Set

```
#!/bin/bash

./svm_light_osx/svm_learn ../../data/output_data/train_affirm ../../data/learner_data
/model_affirm
./svm_light_osx/svm_learn ../../data/output_data/train_remand ../../data/learner_data
/model_remand
./svm_light_osx/svm_learn ../../data/output_data/train_reverse ../../data/learner_dat
a/model_reverse
./svm_light_osx/svm_learn ../../data/output_data/train_reverse_and_remand ../../data/
learner_data/model_reverse_and_remand
./svm_light_osx/svm_learn ../../data/output_data/train_unanimous ../../data/learner_d
ata/model_unanimous

./svm_light_osx/svm_classify  ../../data/output_data/test_affirm ../../data/learner_d
ata/model_affirm ../../data/learner_data/predictions_test_affirm
./svm_light_osx/svm_classify  ../../data/output_data/test_remand ../../data/learner_d
ata/model_remand ../../data/learner_data/predictions_test_remand
./svm_light_osx/svm_classify  ../../data/output_data/test_reverse ../../data/learner_
data/model_reverse ../../data/learner_data/predictions_test_reverse
./svm_light_osx/svm_classify  ../../data/output_data/test_reverse_and_remand ../../da
ta/learner_data/model_reverse_and_remand ../../data/learner_data/predictions_test_rev
erse_and_remand
./svm_light_osx/svm_classify  ../../data/output_data/test_unanimous ../../data/learne
r_data/model_unanimous ../../data/learner_data/predictions_test_unanimous
```

Appendix E Summarizer Routine

```
package core;

import java.io.*;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

/**
 * Handles the Processing of the feature file and maintains the HashMap we'll reuse d
uring
 * pcoesssing of source text
 */
public class LearnerFiles {
    HashMap<String,LineNumberReader> learerFileBufferedReaders;

    /**
     * Create LearnerFiles instance and populate the Hashmap with
     * key: filename of predictions for each classifier
     * value: LineNumberReader (variant of Buffered Reader)
     * @param file_list
     */
    public LearnerFiles(List file_list) {
        learerFileBufferedReaders= new HashMap<>();
        Iterator<File> myiterator = file_list.iterator();

        File t_file;
        //Iterate through the documents
        while (myiterator.hasNext()) {
            t_file = myiterator.next();
            try {
                LineNumberReader learnerLineReader = setupInputReader(t_file.getCanon
icalPath());
                //populate the classifier with the classifier name and the buffered r
eader
                learerFileBufferedReaders.put(getClassFromFileName(t_file.getCanonica
lPath()),learnerLineReader);
            } catch (IOException e) {
                e.printStackTrace();
            }
```

```
        }
    }

    /**
     * Extract the name of the classifier from the string name
     * @param filename whole filename being processed
     * @return the name of the classifer (filename with the LEARNER_PREFIX stripped o
ff)
     */
    public String getClassFromFileName(String filename){
       String classifier=null;
       classifier=filename.substring(filename.indexOf(SummarizerProcessor.LEARNER_PRE
FIX));
       classifier=classifier.substring(SummarizerProcessor.LEARNER_PREFIX.length());
       if (SummarizerProcessor.DEBUG){
           System.out.println("LearnerFiles: classifier was: "+classifier);
       }
       return classifier;
   }

    /**
     * Create the LineNumberReader reference for each file
     * @param featureFile
     * @return
     */
    private LineNumberReader setupInputReader(String featureFile) {
        BufferedReader br = null;
        LineNumberReader lnr = null;
        try {
            br = new BufferedReader(new FileReader(featureFile));
            lnr = new LineNumberReader(br);
        } catch (FileNotFoundException e) {
            System.out.println("FeaturesProcessor: had a problem reading feature file
");
            e.printStackTrace();
        }
        return lnr;
    }

    /**
     * Iterate through and close the reader files references
     */
    public void closeFiles(){
        for (Map.Entry<String, LineNumberReader>readerEntry : learerFileBufferedReade
rs.entrySet()){
```

```
            try {
                readerEntry.getValue().close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }


}
package core;

import java.io.IOException;
import java.io.LineNumberReader;
import java.util.Map;

public class LearnerTextProcessor {
    LearnerFiles learnerFiles;
    Integer countOfDocsWithSomeClassification = 0;

    /**
     * Setup the class Instance, process the files, and close the files
     *
     * @param learnerFiles
     * @param summaryFiles
     */
    public LearnerTextProcessor(LearnerFiles learnerFiles, SummaryFiles summaryFiles)
 {
        this.learnerFiles = learnerFiles;
        processFile(summaryFiles);
        learnerFiles.closeFiles();
        summaryFiles.closeFiles();
    }


    /**
     * Iterate through each row of the old summary file, get the classifiers by looki
ng at the same row across
     * each of the test prediction files, and write out the new value in the output s
ummary file
     *
     * @param summaryFiles
     */
    private void processFile(SummaryFiles summaryFiles) {
        if (summaryFiles.oldSummaryBufferedReader == null) {
```

```
            return;
        }//just return if the buffered reader is null
        Integer lineNumber = 0; //for coordinating row across files
        try {
            String oldLine = summaryFiles.readOldLine();
            while (oldLine != null) {
                oldLine = oldLine.trim();
                String classifiers = getValidClassifiersForActiveRow(lineNumber);
                if (!(classifiers == null)) {
                    summaryFiles.writeNewEntry(oldLine + "\t\t" + classifiers, true);
                    countOfDocsWithSomeClassification = countOfDocsWithSomeClassifica
tion + 1;
                } else {
                    //no classifiers so just write out the oldLine
                    summaryFiles.writeNewEntry(oldLine, true);
                }
                lineNumber = lineNumber + 1;
                oldLine = summaryFiles.readOldLine();
            }//end line processing
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("processFile had a problem: " + e.getMessage());
            System.exit(3);
        }
        summaryFiles.writeNewEntry("\n\nWe had " + countOfDocsWithSomeClassification
+ " with at least one classification", true);
    }

    /**
     * Check each classifier prediction file, if the value for a given row for a give
n classifier prediction file is >1,
     * then document aligned to that row is in the classifier denoted by the classifi
ere prediction file name (and key
     * of the HashMap). Iterate across each classifier and concatenate the classifers
 together before returning the
     * concatenated string
     * @param lineNumber
     * @return
     */
    public String getValidClassifiersForActiveRow(Integer lineNumber) {
        String results = null;
        for (Map.Entry<String, LineNumberReader> learnerEntry : learnerFiles.learerFi
leBufferedReaders.entrySet()) {
            String classifier = learnerEntry.getKey();
            LineNumberReader lnr = learnerEntry.getValue();
```

```
            try {
                //increment to the current line number
                lnr.setLineNumber(lineNumber);
                String tmpValue = lnr.readLine();
                Double dblValue = new Double(tmpValue);
                //System.out.print(classifier+" : "+dblValue+" ");
                if (dblValue > 0) {
                    if (results != null) {
                        results = results + " : " + classifier;

                    } else {
                        results = classifier;
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }//end iterator
        return results;
    }

}
package core;

/**
 * Summarize the existing summary file for the test set documents against the results
 of each category
 * in the test predictions output directory
 */
public class SummarizerProcessor {
    public static boolean DEBUG = false;
    static String LEARNER_PREFIX = "predictions_test_"; //this is the file prefix we
will use to search the specified
                                                        //directory for output

    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("Usage: SummarizerProcessor directory_with_learneroutp
ut old_summary_info_file new_summary_info_file");
            System.exit(2);
        }
        String learnerDirectory = args[0];
        String oldSummaryInfoFile= args[1];
        String newSummaryInfoFile= args[2];
```

```
        DirectoryParser directory = new DirectoryParser(learnerDirectory);
        //get the files with the LEARNER_PREFIX
        directory.getFilesbyPrefix(LEARNER_PREFIX);
        LearnerFiles learnerFiles = new LearnerFiles(directory.results);
        SummaryFiles summaryFiles= new SummaryFiles(oldSummaryInfoFile,newSummaryInfo
File);
        new LearnerTextProcessor(learnerFiles, summaryFiles);
    }
}
package core;


import java.io.*;
import java.nio.file.FileSystems;
import java.nio.file.Files;

/**
 * Handles the reference to the output directory and the file handles thereof
 */
public class SummaryFiles {

    BufferedReader oldSummaryBufferedReader;
    BufferedWriter newSummaryBufferedWriter;
    String SUMMARY_FILE_DELIMTER="\t";

    /**
     * Setup the class variables to read the old summary file
     * and to write the new summary file
     * @param oldSummaryFile
     * @param newSummaryFile
     */
    public SummaryFiles(String oldSummaryFile,String newSummaryFile){
        oldSummaryBufferedReader = setupInputReader(oldSummaryFile);
        newSummaryBufferedWriter = setupOutputWriter(newSummaryFile);
    }

    /**
     * Setup a BufferedWriter for the given filename for the new summary file
     *
     * @param filename
     * @return reference to the new BufferedWriter
     */
    public BufferedWriter setupOutputWriter(String filename) {
        BufferedWriter mybufferedWriter = null;
        try {
```

```
            Files.deleteIfExists(FileSystems.getDefault().getPath(filename));
            File t_file = new File(filename);//create the handle
            t_file.createNewFile();//touch the file
            mybufferedWriter = new BufferedWriter(new FileWriter(t_file,true));
        } catch (FileNotFoundException e) {
            System.out.println("SummaryFiles: couln't find file "+filename);
        } catch (IOException e) {
            System.out.println("SummaryFiles: had a problem setting up file");
        }
        return mybufferedWriter;
    }

    /**
     * Create an instance of the BufferedReader to process the old summary file
     *
     * @param readFile
     * @return reference to the BufferedReader
     */
    public BufferedReader setupInputReader(String readFile) {
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader(readFile));
        } catch (FileNotFoundException e) {
            System.out.println("SummaryFiles: had a problem reading file");
            e.printStackTrace();
        }
        return br;
    }

    /**
     * Reads a line of text from the existing summary file
     * @return
     */
    public String readOldLine(){
        String result=null;
        try{
            result=oldSummaryBufferedReader.readLine();
        }catch (IOException e){
            e.printStackTrace();
        }
        return result;
    }

    /**
     * Writeout the line of text to the new summary file and follow it up with a newl
```

```
ine
      * @param value
     */
   public void writeNewEntry(String value, Boolean new_line){
       try {
           newSummaryBufferedWriter.write(value);
           if (new_line){
               newSummaryBufferedWriter.newLine();
           }else{
               newSummaryBufferedWriter.write(SUMMARY_FILE_DELIMTER);
           }
       } catch (IOException e) {
           e.printStackTrace();
       }
   }


    /**
     * Close the file handles
     */
    public void closeFiles(){
        try {
            oldSummaryBufferedReader.close();
            newSummaryBufferedWriter.close();
            }
         catch (IOException e) {
            e.printStackTrace();
        }

    }

}
```