LDSI W2021 Project Report

Name: Syed Husain Mustafa
Gloss ID: tum_ldsi_28

Summary

I have trained a machine learning model that can produce correct annotation for a given legal statement with a precision of 85% on an unseen development set. In order to generate such a model 113 out of the provided 141 annotated Board of Veteran's Appeals (BVA) decisions are used for training purposes. The sentences from the 113 cases are featurized in two different ways. I make use of the FastText library to produce 100-dimensional sentence embeddings for the first approach, and rely on the TFIDF Vectorizer from the SpaCy package to produce 3082-dimensional TFIDF vectors for the second approach. In total 24 models were trained using these two different feature sets, out of which the Logistic Regression trained on extended TFIDF features produces the highest F1-Score (86%) on the development set and a 78% F1-Score on the test set, the second best model trained on extended sentence embeddings features is the Radial SVM model that achieves an F1-Score of 84% on the development set, and 73% on the test set.

1. Introduction

In the legal domain, digitization of legal texts has provided an avenue to incorporate modern NLP techniques to extract insightful details for future cases. In this context, a taxonomy of sentence types usually found in the US Board of Veterans' Appeals (BVA) is used to annotate sentences from unseen cases. The development of Machine Learning (ML) models that allow one to automatically label sentences as per a specific taxonomy is highly relevant to legal practitioners in isolating information that may aid them in exercising their profession in a more efficient manner.

In this project an NLP pipeline is proposed for training machine learning models on annotated Board of Veterans' Appeals decisions.

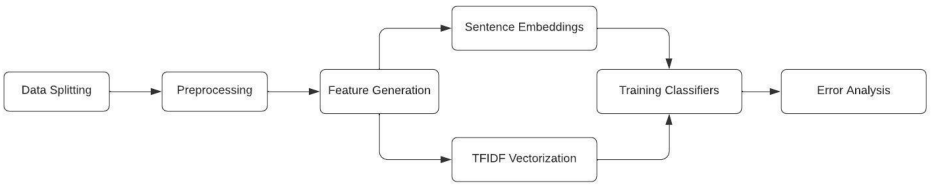


Fig. 1.1

Section 2 elaborates on how the data is prepared. Section 3 elaborates on sentence segmenting unlabeled BVA decision. In Section 4, I address the process of generating custom word embeddings using FastText as well as examine semantic similarities between word vectors and reason as to why some are plausible while others are not. In Section 5 the process of creating two separate features sets, i.e., the TFIDF Featurization and Sentence Embeddeding Featurizatoin are discussed, followed by the application of these feature sets to 24 classifiers. Section 6 then addresses the top three annotation types which are most often misclassified by the best performing classifier. Sections 7 and 8 address possible avenues for future research and how to access the code in this project.

2. Data Preparation

2.1 Dataset Splitting

In total 540 BVA cases are provided in the ‘ldsi_w21_curated_annotations_v2.json’ file. The data structure for which is illustrated below.

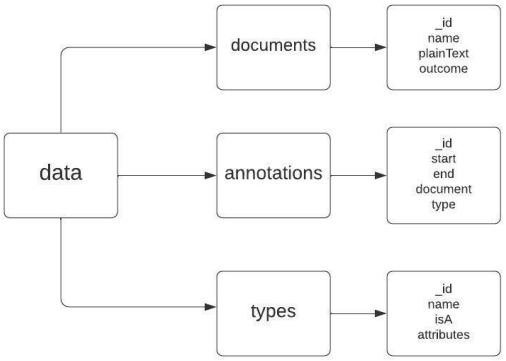


Fig. 2.1.1: “ldsi_w21_curates_annotations_v2.json” structure

Only 141 cases are annotated of which 14 cases are separated to form a development set, which is used to check the performance of the classifers trained on the 113 cases that form the train set. A further 14 cases are separated to form a test set which is used to measure the performance of the best classifier. The unique case ID’s of the test & development sets are given below:

Test Case IDs
61aea57397ad59b4cfc41399, 61aea55d97ad59b4cfc412be, 61aea56f97ad59b4cfc4134b, 61aea55c97ad59b4cfc41297, 61aea57197ad59b4cfc41377, 61aea55e97ad59b4cfc412d8, 61aea57397ad59b4cfc4138e, 61aea55c97ad59b4cfc412ae, 61aea55e97ad59b4cfc412d4, 61aea57497ad59b4cfc413c4, 61aea57497ad59b4cfc413b6, 61aea55c97ad59b4cfc412a0, 61aea57197ad59b4cfc4136e, 61aea55f97ad59b4cfc41334

Dev Case IDs
61aea57097ad59b4cfc4135a 61aea55d97ad59b4cfc412bf, 61aea55e97ad59b4cfc412ee, 61aea56f97ad59b4cfc41343, 61aea55c97ad59b4cfc4129d, 61aea55c97ad59b4cfc4129f, 61aea57397ad59b4cfc4139e, 61aea57197ad59b4cfc4136b, 61aea55d97ad59b4cfc412c1, 61aea57497ad59b4cfc413ba, 61aea57497ad59b4cfc413cc, 61aea57297ad59b4cfc41381, 61aea55e97ad59b4cfc412df, 61aea55c97ad59b4cfc4129e

In total the corpus of 141 documents contains 15,349 annotated sentences, of which 1,440 form the development set, 1,524 form the test set, and 12,385 form the train set.

2.2 Deciding on a Sentence Segmentater

A prerequisite for producing the word embedding model is being able to segment the nearly 3 million sentences present in the 30,000 BVA decision accurately. For sentence segmentation we compare the performance of two approaches. The first is the standard sentence segmenter available in the SpaCy language processing library. The second is the “text2sentence” functionality present in the law-specific sentence segmenter package named “Luima”.

The Standard SpaCy sentence segmenter is found to produce a total of 12,859 sentences. The true number of annotated sentences in the train-set however is only 12,385. On closer examination it is observed that there are instances of erroneous sentence segmentation.

Error-1: The model fails to identify “NO.” as an abbreviation, and instead designates the point(.) as the end of a sentence. Similarly the model fails to identify “Fed.” as an abbreviation in all instances, i.e., sometimes regarding it as a sentence end, while in other instances as an abbreviation. (All errors are illustrated in “LDSI-Project-SHM”.)
Error-2: The model erroneously splits a single contiguous block of citations into two sentences.
Error-3: The segmenter identifies a set of annotations as a single sentence, this defies legal and English language conventions.

```
SegmenterResult1[2] # Error4:
                    # Identifies multiple annotations as one sentence
                    # Ends at "Introduction" despite the absence of stop symbol


REPRESENTATION

Appellant represented by:       Oregon Department of Veterans' Affairs

ATTORNEY FOR THE BOARD

Christopher M. Collins, Associate Counsel



INTRODUCTION
```

Fig. 2.2.1: Error in Sentence Segmentation

To evaluate the performance of the sentence segmeter on the entire train-set I use a custom made function called “true_positive()” which counts the number of predicted sentence start and end indices that are within 3 characters of the actual number of sentence start and end indices. This function takes as input a list of true sentence start and end positions dubbed “true_annotations” (shown in Fig. 2.2.2) as well a list of predicted sentence start and end positions that correspond to the results of the standard SpaCy sentence segmenter. To obtain the indices for each sentence in SpaCy the “sent.start_char”, and “sent.end_char” attributes of each sentence element produced by “nlp(document).sents” is used.

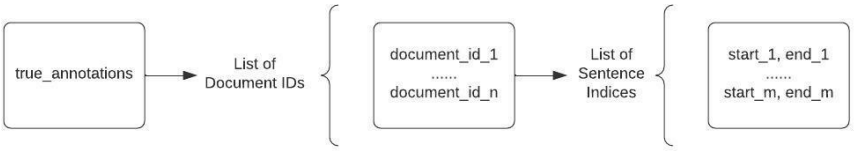


Fig. 2.2.2: “true_annotations” data structure

In the case of the standard SpaCy segmenter only 6,307 of the predicted sentences meet the +/- 3 character criteria. The false negative rate is then the number of unmatched true sentences (12,385-6,307) and the false positive rate is the number of unmatched predicted sentences (12,859-6,307). The following equations are then employed:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad (1)$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (2)$$

$$F1 - Score = \frac{2.\ Precision.\ Recall}{Precision + Recall} \quad (3)$$

The performance of the standard SpaCy Segmenter is illustrated below:

Precision	Recall	F1-Score
49.047359825802936	50.9245054501413	49.96830930122009

In SpaCy it is possible to incorporate domain-specific abbreviations so as to achieve lower instances of false sentence segmentations. To accomplish this I have used the “nlp.tokenizer.add_special_case()” functionality.

```
# Improved SpaCy Segmenter on display
nlp.tokenizer.add_special_case('Vet. App.', [{ORTH: 'Vet. App.'}])
nlp.tokenizer.add_special_case('Fed. Reg.', [{ORTH:'Fed. Reg.'}])
nlp.tokenizer.add_special_case('Supp. ', [{ORTH: 'Supp. '}])
nlp.tokenizer.add_special_case('St. Petersburg', [{ORTH: 'St. Petersburg'}])
SegmenterResult2 = list(nlp(train_documents[0]['plainText']).sents)
```

Figure 2.2.2: Adding special abbreviations

The improved SpaCy sentence segmenter generates only 11,806 sentences, whereas the actual number is 12,385. This indicates that our hard mandate on the model to only recognize the above mentioned cases as abbreviations has failed to identify sentence endings where these terms appear at the end. This suggests that adding these cases may be to the detriment of the Segmenter model.

Error 1: It is observed that this model still treats some groups of sentences, particularly headings as belonging to one sentence.
Error 2: The approach of adding special tokens does not account for all abbreviations, as such even after adding the “Fed. Reg.” other terms such as “Fed. Cir.” are erroneously detected as sentence endings.

The performance of the improved SpaCy sentence segmenter is illustrated below:

Precision	Recall	F1-Score
58.77519905132983	56.02745256358498	57.36844280930924

A law-specific sentence segmenter named “Luima” is employed next. This model generates a total of 12,992 sentences from the train-set, which is far more than the actual 12,385 sentences. On further examination the following errors are observed.

Error 1: “Case Header” across multiple case decisions is identified as 5 different sentences starting at “Citation Nr.”, “Decision Data”, “Archive Data”, “Docket No.”, & “Date”.
Error 2: “Case Footer” is also identified as two sentences.
Error 3: A commonly understood abbreviation “v.” is mistaken for sentence end.

The aforementioned errors put into perspective the excess sentences generated by the Luima model. On further analysis the Luima model produces the following metrics.

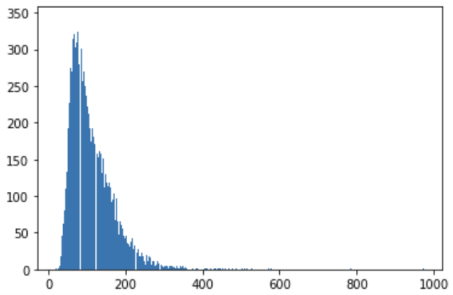
Precision	Recall	F1-Score
82.95104679802957	87.01655228098505	84.93517752295385

Given the better metrics I have used Luima in subsequent sections, despite the model’s shortcomings is identifying multiple annotations where only one exists, the much higher Recall metric suggests that the model is successful in correctly identifying many of the true sentence indices.

3 Preprocessing

3.1 Splitting Unlabeled Data

In this section the Luima sentence segmenter is used to segment 30,000 unlabeled BVA decisions. The lengths of each decision in terms of the number of sentences is stored in the file named “NumberOfSentencesPerDecision.json”. As per Luima, the corpus of 30,000 decisions comprises of 3,360,495 sentences.



3.1.1 Histogram 1

The above graph depicts the number of sentences in each document along the x-axis, and the frequency of such documents along the y-axis. It can be observed that the median BVA decision has close to 99 sentences, while the longest decision has 974 sentences.

3.2 Sentence-Wise Preprocessing

To process each sentence I have developed a tokenizer that works in two steps, i.e., “from_sent_to_unrefined_tokens()” followed by “from_unrefined_to_refined_tokens()”. The “tokenize_corpus()” function takes as input the corpus of 30,000 decisions. The “corpus” is a list of dictionaries with two keys namely (name, plainText). Each individual decision is segmented into sentences using the”text2sentence()” functionality and then passed to the “from_sent_to_unrefined_tokens()” function, where each sentence is tokenized using the SpaCy tokenizer. All space characters of the form “\s” are removed from the sentence, followed by the removal of all non-alpanumerics, done in line with the Manual Tokenization depicted in the “LDSI_W21_Classifier_Workshop” notebook. The subsequent list of tokens is passed to the “from_unrefined_to_refined_tokens” function where all strings of the form “^(19|20)\d{2}$” are treated as a token of type “<YEAR>”, and all alpha-numerics of the form “"^[0-9]{1,3}[a-z]$"” are treated as <ALP-NUM>. Furthermore all punctuations and spaces are removed, and all numbers are tokenized as “<NUM{len(t)}>” . Lastly all words are lower cased before being appended to the list of “clean_tokens”. These are then passed back to the “tokenize_corpus” function which stores the list of tokens per sentence that are alteast 5 tokens long in a list called “tokens_of_sentences”. After iterating over one entire document this list of tokenized sentences is appended to the “tokens_of_corpus”. The stored decision are shuffled and the sentences within each decision are again shuffled and stored in “TokenisedSentenceOfCorpus_Luima.json”. Below is a screenshot that illustrates how “tokens_of_corpus” is structured.

```
In [270]: tokenized_fpath = 'TokenizedSentencesOfCorpus_Luima.json'
          tokenized_corpus_luima = json.load(open(tokenized_fpath))

In [575]: tokenized_corpus_luima[0][5]

Out[575]: ['<NUM2>', 'usca', '<NUM4>', '<NUM4>', '<NUM2>', 'cfr', '<NUM4>']
```

Figure 3.2.1 : Tokenized Sentence 5 of Decision 0

3.3 Tokenizing Unlabeled Data

The count of tokens per each of the ~3Million sentences is then plotted on a histogram. In the below diagram the x-axis represents the length of each sentence in terms of the number of tokens, while the y-axis represents the frequency of sentences of such length.

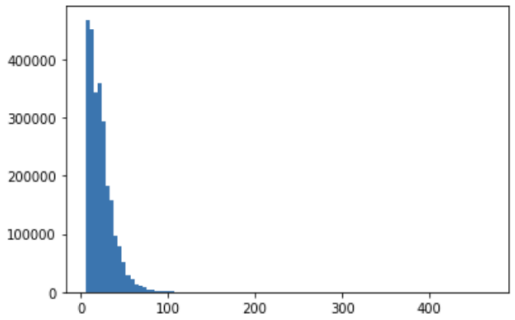


Fig. 3.3.1 Histogram 2

In total there are 59,429,980 tokens (~60 million tokens), where the median sentence length is about 20 tokens long, and the longest sentence in 30,000 decisions is 467 tokens long.

4 Feature Generation

4.1 Generating Custom FastText Embeddings

In this section we employ the list of randomly sorted tokenized sentences stored in "TokenizedSentencesOfCorpus_Luima.json" to train a custom FastText Embedding model. All words that occur less than 20 times are not considered for training purposes and the resultant embeddings are of 100-dimension. The model is trained for 15 epochs.Once the model has been trained it is saved to "model_luima.bin" which has been used to prepare the sentence embedding in the subsequent section.

4.2 Evaluating Custom Embeddings Manually

After training, the model uniquely identifies 26,356 tokens in the vocabulary as determined by the "len(model.words)" command. In order to check that the model adequately captures token distribution in the corpus I have checked the nearest neighbors of the following strings, "veteran", "v.", "argues", "ptsd", "granted", "korea", "holding", & "also" as per the project instructions, as well as three strings of my own choice "soviet", "insomnia", and "suicidal".

In the context of the string "veteran" the closest neighbors appear to be "the", "appellant", "s", and "his". Upon examining any decision using "random.choice()" it is evident why the word "the" possess the highest cosine similarity with "veteran" as "the veteran" is a common phrase used to refer to the person who has appealed the BVA to reverse a decision by a lower court, in this context "appellant" being the word with the second highest cosine similarity with "veteran" is also in line with all the cases that I have examined so far. In reference to injuries, or reparations that are owed to the veteran the high cosine similairity to the apostrophe (') also seems plausible as phrases like "the veteran's claim", "the veteran's plea", etc affirm the close proximity of these terms. However less similar strings like "jbm" and "bhat" as nearest neighbors of "veteran" does not seem plausible. When examining the words similar to "bhat" and "jbm" I find terms such as "jjh", "bhatti", "mjs", "sjm". It might be the case that "jjh", "mjs" and other similar strings were abbreviations that have since lost meaning as a consequence of preprocessing where the stop punctuation symbols were removed, alternatively these words may just be artifacts generated as a consequence of how FastText learns words as a collection of characters[1]. As word embeddings are learnt as sum of n-gram character embeddings, this would suggest that the string "veteran" appears in

close proximity with words that have “b”, “h”, “a”, “t”, “j”, “b”, and “m” in their make-up, however any further analysis in this regard is beyond the scope of this report.

The string “v.” often appears when there is a reference to a prior judgement, in this regard appearance of strings as “noting” and “versions” seems plausible. However less plausible is the occurrence of “deterioration”, “recoving” and non-word strings such as “vaopgcrec”, “vaopgrec” and “vaopgcprec”. The nearest neighbors of the next word, “argues” mostly seem plausible. The word “contends” demonstrates a high cosine similarity with “argues” as these words can be used interchangeably in many contexts, i.e.; “The Board notes that the veteran does not contend combat participation.”, is one example where “argues” may be substituted to provide a nearly equivalent sentence. Other words such as “asserts”, “asserted”, “contending”, “alleges”, and “maintains” may also be substituted for “argues”. The word with the least cosine similarity in the list of nearest neighbors is “contention” which is used in reference to a heated argument, however it is notably similar to the word “argues” although not exactly substitutable - justifying the lower cosine similarity. The nearest neighbors of “ptsd” being the typo “pstd” is also understandable, subsequent strings such as “depressive”, “anxiety”, “psychiatric”, “stressors”, “bipolar” also seem plausible. However words such as “mdd”, “mst” do not seem plausible, unless ofcourse these are typos of “mad” or some other word semantically similar to “ptsd”, or infact artifacts created by fastText. In examining the neighbors of “granted”, “grants”, “warranted”, “connection”, “entitlement”,“incurred”, and “diagnosis” all seem plausible, as such words are used in the “Conclusion of Law” to outline on what basis a request for appeal is being granted. The word “denied” being the semantic opposite in meaning also makes some sense as it appears within a similar context. Within the context of military service abroad the neighboring words of “korea” being “dmz”(Demilitarized zone), “korean”, “tour”,”stationed”, “taegu”(a town in South Korea), and “okinawa” (U.S. military base used to lauch offensive into Korea) seem plausible. However other strings such as “panama” and “germany” seem anomalous, unless ofcourse there are certain outlier cases of Veteran’s serving in WW2 in west Germany that were later relocated to Korea, or naval officers stationed in Panama deployed to Okinawa or Korea.
The neighboring words of “holding” being “vet”, “app”, and strings of the form “<YEAR>holding” also seem plausible given that the term mostly occurs when the current case decision is substantiated by a prior decision where some party “X” contended (versus) for a position against party “Y”. Some sentences that substantiate these finding are “See Barr, 21 Vet. App. at 307 (holding that….” and “See Nieves-Rodriguez v. Peake, 22 Vet. App. 295, 304 (2008) (holding that most of the probative ….” . The next word examined in the notebook is “also” which has neighbors such as “<YEAR>”, “mcknight”, “prinicpi” and other names of people that appear in cases which are often referred to inorder to determine precedence. For example “Kahana v. Shinseki, 24 Vet. App. 428 (2011); see also Jandreau, 492 F.3d at 1376-77.” is one context in which “also” appear along side a famous case as well as a token of type <YEAR>.

Next I examine the list of semantically similar words generated by the fastText model for “soviet”, which include “russian”, “japanese”, “viet”, “iranian” and “libyan” which all seem plausible given the context of the USSR’s involvement in the offensive against the Japanese in WW2, and their involvement in the Vietnam war. The involvement of the Soviets in the Iran-Iraq war also lends plausibility to the word “iranian” appearing in a similar context as “Soviet”. The neighbours of “injury” being “injuries”, “disease”, “disability”, and “incurred” all seem to be plausible. Other words such as “aggravted” also make sense in the context of “Conclusion of Law” when the court determines that a certain injury was or was not caused or aggravated by military service undertaken by the appellant. Lastly I examine the neighbors of “suicidal” which are “suicidality”, “ideation”, “homocidal”, “paranoia”, “delusions” and “hallucinations” which all seem plausible. Given that the term “suicidality” as defined by the American Psychological Association [2] takes into consideration suicidal “ideation” it is reasonable that these words would possess a high cosine similarity. The term “homocidal”

also makes sense within the context of “suicide” as psychological evaluation of veterans are often cited during a hearing indicating that he/she has had/not had “suicidal or homicidal ideation”.

5. Training Classifiers

5.1 TFIDF Featurization

Based on the classifier workshop code the TFIDF vectors of shape (1,3082) are produced for each sentence in the training set. The shape after applying “spacy_tfidf_vectorizer” is (12385, 3082) for the train set, (1440, 3082) for the dev set, and (1524, 3082) for the test set. Two more feature vectors are appended to each of these 3 feature sets that capture the normalized start position of each sentence within a decision, as well as the normalized number of tokens per sentence. The normalized number of tokens per sentence is determined by extracting the average and standard deviation in number of tokens per sentence for each of the 3 data sets, i.e.; train, test and development. The final feature sets are shaped as (12385, 3084) for training set, (1440, 3084) for development set and (1524, 3084) for test set, respectively.

5.2 Word Embedding Featurization

The FastText model trained on ~60 million words from the 30,000 unlabeled BVA decisions is used to produce 100 dimensional word embedding vectors. The sentence embedding vectors are obtained by averaging out the word embeddings of all tokens present in a given sentence. This results in a feature set of shape (12385, 100) for the training set, (1440, 100) for the development set and (1524, 100) for the test set, respectively. As in Section 5.1, these feature vectors are expanded by appending two vectors, the first for normalized sentence start, and the second for normalized number of tokens in a sentence. Resulting in feature sets of shapes (12385, 102), (1440, 102), and (1524, 102).

5.3 Model Training

The above two feature sets which will be referred to as (a) (“TFIDF Extended Features”) and (b) (“Word Embedding Extended Features”) going forward are applied to 12 different classifier models. Namely, Radial, Poly, & Linear SVM Kernels, Random Forest with 25 trees and depth of 10, 16, & 24, Random Forest with 50 trees and depth of 10, 16, & 24, Random Forest with 100 trees and depth of 20, Decision Tree with depth of 20, and a Logistic Regression Classifier.

The best performing model is the “Logistic Regression Classifier” trained on (a) which achieves Precision, Recall and F1-Scores of (85%, 86%, 85%) on the development set, and (78%, 81%, 78%) on the test set. The best model trained on (b) achieves (84%, 85%,84%) Precision, Recall and F1-Score on the development set, and (74%, 75%, 73%) on test set. A detailed table of Precision, Recall, and F1-Score metrics for all 24 models is available under Section 5.3 of the “LDSI-Project-SHM” notebook.

It was observed in 10 out of the 12 models that were trained that those models trained on (b) generally resulted in higher metrics than those trained on (a). This trend was not observed in the case of the Decision Tree and Logistic Regression Classifiers where the models trained on (a) performed (3-5)% higher than those trained on (b).

6. Error Analysis

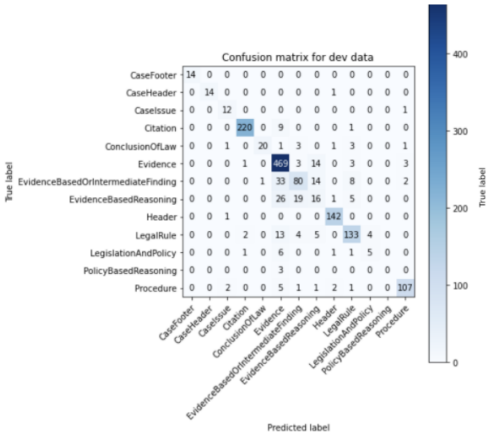


Figure 6.1: Best (a) Model (#1) on dev set

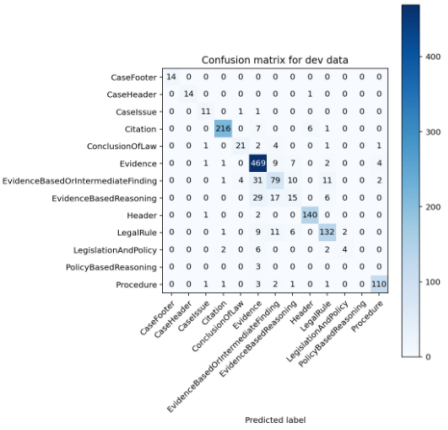


Figure 6.2: Best (b) Model (#2) on dev set

As per the confusion matrix of the best performing model, the most misclassified types are “PolicyBasedReasoning” (100% misclassified), “EvidenceBasedReasoning”(76.2% misclassified) and “Legislation&Policy” (64.3% misclassified). To survey the best models’ mispredictions I make use of the “prediction_errors” function introduced in the Classifier Workshop.

On executing the “prediction_errors” function for predicted label “PolicyBasedReasoning” I did not obtain any output, on close examination of Figure 6.1 it is evident that the model is unable to detect any “PolicyBasedReasoning” types, however 3 instances from the development set are misclassified as “Evidence”, given this I have examined statements that have been misclassified as “Evidence” whereas they were infact “PolicyBasedReasoning” inorder to make a determination why the model was unable to make the correct prediction. As a first example I have considered the statement, “*The new rating criteria for intervertebral disc syndrome were subsumed in the aforementioned amended rating schedule for spine disabilities.*”(case 1533564.txt) This statement on a first read appears to be statement of fact and should hence fit the “Evidence” type, however the appearance of the words “amended” indicates some relation to a legislation or policy. This subtle hint indicates how the sentence even out of context relates to some policy and may possibly be of type “PolicyBasedReasoning”.

In examining statements falsely classified as “EvidenceBasedReasoning” I have come upon a statement regarded as a “LegalRule” which I believe has been mis-annotated. The statement is, “*The Board is required to give consideration to all of the evidence received since the most recent denial of the claim in the March 1996 rating decision in light of the totality of the record.*” (case 1206738.txt) This statement I believe is an instance of “PolicyBasedReasoning”. A conventional legal rule statement is meant to outline abstract rules without any determination about whether the rule is fulfilled. This statement stipulates certain action as a consequence of a policy change that took place in 1996. Another instance of a ‘false-positive’ is the following statement, “*Indeed, any such action would result only in delay*.” (case 1014839.txt) This statement is not a legal determination nor is a finding explicitly mentioned by the Board (“the Board finds that..”). As per my understanding the model has correctly predicted the statement to be of the type “EvidenceBasedReasoning”, whereas it is labeled as “EvidenceBasedOrIntermediateFinding.”

For the type “LegislationAndPolicy” there are some instances of false positives which one would not remotely associate with a “LegilationAndPolicy” type, however on closer examination it becomes evident how the model made such a mistake. For example the statement “*I. Veterans Claims Assistance Act of 2000 (VCAA)*” (case 0636017.txt) is clearly a “Header”, however a model trained to identify mentions of legilation along with a <YEAR> token as “LegislationAndPolicy” makes the mistake of identifying such a statement incorrectly. Considering another example, “*The Veterans Claims Assistance Act of 2000 (VCAA), codified*

in pertinent part at 38 U.S.C.A. ﷺ 5103, 5103A (West 2002), and the pertinent implementing regulation, codified at 38 C.F.R. ﷺ 3.159 (2011), provide that VA will assist a claimant in obtaining evidence necessary to substantiate a claim but is not required to provide assistance to a claimant if there is no reasonable possibility that such assistance would aid in substantiating the claim." (case 1203188.txt) This statement employs the use of alpha-numeric characters and <NUM2> tokens which are typically associated with statements that mention a Legislation or Policy with reference to the specific section of Title 38 under the U.S. Code, hence in that regard it is understandable why and how the model would associate this particular statement with the label of "LegislationAndPolicy". However in my own understanding it is also possible that the annotator has mislabeled the statement, as a "LegislationAndPolicy" statement contains information about when the Policy was enacted and where it can be found, whereas this is not a strict requirement for a "LegalRule" type.
Outside the top 3 most misclassified classes, I believe there are instances where the annotator has incorrectly labeled types. For example, as per my best model the following statement has been annotated as "Evidence". *"Normal forward flexion of the thoracolumbar spine is zero to 90 degrees, extension is zero to 30 degrees, left and right lateral flexion are zero to 30 degrees, and left and right lateral rotation are zero to 30 degrees."* (case 0610579.txt) The annotator has labelled this as "LegislationAndPolicy". As per the semantic types defined in the "LDSI Annotation Guidelines (WS2021)" the aforementioned statement being a statement of fact ought to be classified as "Evidence" and not as "LegilstionAndPolicy".

7. Discussion & Lessons Learned

In the process of implementing this project I have identified some data processing steps that could be improved upon to achieve higher performance metrics. With regard to the sentence segmenters used, it is evident that the standard and augmented SpaCy sentence segmenters which perform at around 50% F1-Score are not reliably. The Luima model is able to achieve a higher F1-Score however some improvement is needed to be able to process multi-line annotations as a single annotation With regard to the dataset, I have found multiple instances where the annotator has mislabeled the statements, it is my understanding that a dataset of higher quality can help immensely improve performance. Furthermore in training 24 models, I have observed that in 20 out of 24 models, the Sentence Embedding features often lead to better results, the models where the opposite is true are often simpler models, such as the DecisionTree and Logistic Regression. This would suggest there is some benefit to training the embedding model on 30,000 documents, however since the best performing model is one based on TFIDF and is easier to train, the cost benefit ratio does not seem to favor the embedding vector approach. To ascertain the usefulness of the Embedding Vector Approach I would suggest pursuing two different approaches, (a) Scaling up of embedding vector dimensions so as to incorporate more information and be dimensionally comparable to TFIDF vectors so as to better determine performance of models trained on such feature vectors. (b) Adding embedding vector features to the TFIDF feature set to see if there is any improvement in performance metrics. Another way to improve performance would be to account for context around the sentence. As is the case with legal documents where evidence statements are often followed by reasoning statements, or how legal rules are often followed by citations. One can incorporate some percentage of preceding sentence vectors to encode information that may be beneficial in determining the right annotation. Lastly I would suggest applying deep neural networks to the above mentioned augmented feature.

8. Code Instructions

The code is available within the "LDSI-Project-SHM" Jupyter Notebook. The "analyze.py" file is prepared from functionalities first tested in the aforementioned notebook. The requirements to run analyze are present in the "requirements.txt" file.

References

[1] Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov; Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* 2017; 5 135–146. doi: https://doi.org/10.1162/tacl_a_00051

[2] "APA Dictionary of Psychology", *Dictionary.apa.org*, 2022. [Online]. Available: https://dictionary.apa.org/suicidality. [Accessed: 05- Mar- 2022].