A DEEP LEARNING MODEL TO PREDICT CONGRESSIONAL ROLL CALL VOTES FROM LEGISLATIVE TEXTS

Jonathan Wayne Korn and Mark A. Newman

Department of Data Science, Harrisburg University, Harrisburg, Pennsylvania

ABSTRACT

Developments in natural language processing (NLP) techniques, convolutional neural networks (CNNs), and long-short- term memory networks (LSTMs) allow for a state-of-the-art automated system capable of predicting the status (pass/fail) of congressional roll call votes. The paper introduces a custom hybrid model labeled "Predict Text Classification Network" (PTCN), which inputs legislation and outputs a prediction of the document's classification (pass/fail). The convolutional layers and the LSTM layers automatically recognize features from the input data's latent space. The PTCN's custom architecture provides elements enabling adaptation to the input's variance from adjustment to the kernel weights over time. On the document level, the model reported an average evaluation of 67.32% using 10-fold cross-validation. The results suggest that the model can recognize congressional voting behaviors from the associated legislation's language. Overall, the PTCN provides a solution with competitive performance to related systems targeting congressional roll call votes.

KEYWORDS

Deep Learning (DL), Convolutional Neural Networks (CNNs), Long-Short-Term Memory Networks (LSTMs), Natural Language Processing (NLP), Congressional Roll Call Votes

1. INTRODUCTION

Predicting the status (pass/fail) of congressional roll call votes has been political scientists' goal for decades. There are patterns of congressional voting behavior captured in the legislative text, which has shown significance when predicting congressional votes' status. Understanding the future status of legislation provides vital insights into government and industry matters. Analyzing roll-call data allows insight into information detailing the legislation's vote status and can predict future votes [1].

Other approaches using quantitative roll call data and legislative text have shown success in the past, however only under certain conditions. Their success expresses limitations due to the dimensionality of the data and unforeseen conditions in Congress's complex social environment. For instance, using two separate datasets diminishes a model's flexibility to predict and adapt to the event space's changing conditions.

Political environments are complex social networks that often create noisy data. The scale of topics that the government considers in legislation is a diverse subject matter, and the language is sparse. Most past approaches rely on an attempt to use extra dimensionality from both text and quantitative data. However, using extra dimensionality produces limitations, or predictive bottleneck, in approaches using text and quantitative congressional data. Many situations occur when congressional roll call data is not available or represents uncontrolled conditions creating

issues predicting the event. For instance, a pure expression of mixed but non-partisan support in the voting data could limit existing models' overall accuracy.

A more robust model is required to address the current limitations expressed in prior models addressing the problem discussed above. Statistical modeling seeks to learn the joint probability function from words

contained in the texts [2]. However, it is difficult to obtain this goal because of the "... curse of dimensionality" [2]. However, the rise of deep learning makes it possible for computer systems to recognize patterns in complex text representations. Advancements in natural language processing techniques allow text to convert into tokenized word vector spaces. The conversion provides the proper dimensions to embed the texts into different deep neural networks. A custom hybrid architecture provides the ability to input texts and provide accurate outputs from recognized patterns. Convolutional neural networks (CNNs) and long- short term memory neural networks (LSTMs) can recognize these intricate patterns within the data's dimensions. Each layer in the network provides benefits in recognizing temporal and spatial features from abstract data representations. Mainly, CNNs reflect successful results in identifying abstract data patterns mainly because of development in max-pooling layers.

However, due to long lag periods from the data's complexity, the CNN architecture cannot alone capture the text's patterns. A primary reason for implementing LSTM layers in the model's architecture is to overcome the long lag time problem that occurs when processing high dimensional data. Overcoming long lag periods allows for the neural network's depth to continue, enabling minute features to be recognized.

Combining CNN and LSTM algorithms provide an architecture capable of recognizing features in sparse word vector spaces over long periods, known as a Convolutional Long-Short Term Memory Neural Network (C-LSTM). Adaptable layers during kernel initialization help filter out the non-significant features from the inputs dimensional space. The adaptability in each layer of the network provides stability in the predictions and robustness against the dynamic social environment creating the data. The custom architecture ensures the network depth is suitable for accurate prediction from highly sparse text samples. The paper presents a custom deep learning solution to accomplish the binary classification of legislative texts.

1.1. Organizational Structure

The remainder of the paper is organized into the following sections, including Section 2. Literature Review, Section 3. Methods and Materials, Section 4. Results, Section 5. Conclusion, and Section 6. Future Work. Section 3. Methods and Materials includes two sections including 3.1. The Data, and 3.2. The Deep Learning Approach. Section 3.1. includes Section 3.1.1. Pre-processing text and 3.1.2. Equal Distribution of Samples. Section 3.2. includes 3.2.1. The Custom PTCN Architecture and 3.2.2. The PTCN Modelling Process.

2. LITERATURE REVIEW

A focus for quantitative political scientists is using roll-call data to understand legislative voting behaviors better. Few models have attempted to use supplementary data such as the text of legislation to understand congressional voting trends better.

In 1991, research expressed that spatial positions captured in roll call data is stable and contains reliable features to recognize voting patterns [3]. The party discipline is present in the spatial

dimensions of roll-call votes of the legislator's ideal points captured in the data [4]. Spatial dimensions in the context of the domain refer to the probabilistic word occurrence contained across the text samples. Researchers utilized a method using Bayesian simulation models to capture the ideal points of the legislators. The approach allows researchers to ensure the beliefs incorporated into the inputs' dimensional space for roll call analysis [5]. The method enables researchers to handle the increasing complexity in higher-dimensional contexts [5]. Policy ideas are standard features in legislation that provide insight into legislators' behavior similar to quantitative roll call data [6].

In 2011, Gerrish and Blei introduced a probabilistic model capable of inferring a person's political position for specific topics [7]. The model focuses on capturing the individual representative's view on specific political interests from the text alone. The authors have used 12 years of congressional legislative data in their experiment to capture significant patterns. The patterns express the lawmakers' vote behavior and on which type of document [7]. Gerrish and Blei integrated the analysis of text into quantitative models such as the ideal

point model with success inferring "... ideal points and bills' locations..." from roll call data leading to the prediction of legislative vote status (pass/fail) [8]. The integration of the bill texts into the ideal point model helped mitigate a limitation of only predicting on vote data alone, which may, at times, be inconsistent in its availability. The authors developed a supervised ideal point topic model capable of predicting pending bills using votes. It also is a method of exploring the connection between language and political support [8]. Other models developed by Gerrish and Blei to predict the "...inferred ideal points using different forms of regression on phrase counts." [8]. However, lead existing models cannot predict when mixed but non-partisan support is present in the data. Many existing models cannot expand beyond one-dimensional limitations, such as the ideal point model [8]. The authors also based their model's performance on the baseline that 85% of all votes are 'yea', limiting the model's performance results. The authors reported their ideal topic model predicted 89% of the votes with 64 topics, and their L2 model predicted 90% [8]. In sequential predictions, both their models predicted 87% and 88.1% accurate at predicting future votes, respectively [8]. Their study only attempts to understand a few topics with a one-dimensional political space, which creates a predictive bottleneck [8]. Interestingly, the ideal point model reflects representatives' preferences, constituency preferences, or any other feature indicating a preference for particular legislation [1]. In 2013, Spatio-temporal modeling expressed success in using text to predict congressional roll call votes with relative success to the ideal points model [9]. In 2015, ideal points were redefined as two characterizations, including word and vote choice [10]. These characterize the ideal points and the dimensions of the policy. The researchers utilize Sparse Factor Analysis to combine both votes and textual data to estimate the ideal points [10].

In 2016, Yang and others introduced hierarchical attention networks for document classification [11], the results outperformed previous models by a large margin, which can be indicated in the author's results using the Yelp 2013, Yelp 2014, Yelp 2015, IMBD review, yahoo Answer, and Amazon review datasets to test their algorithm and compare it to other methods [11]. On average, the HN-ATT performed with about a 70% accuracy rate across the tasks. They were generalizing the network to address multiple types of tasks that limit their ability to identify the text's critical features, such as the political ideology in the congressional legislative texts across spatial and temporal dimensions. A similar approach that uses multi-dimensional bill texts to predict roll calls' status is Kraft, Jain, and Rush's approach established in 2016. Using an embedding model with prepared bill texts, they competed with Gerrish and Blei's approach. Mainly the approach utilized ideal vectors rather than ideal points [12]. The approach relies on quantitative data that leads to the same predictive bottlenecks as prior models due to the complex event space.

In 2016, the Gov2Vec method expressed success in capturing opinions from legal documents. The text's transformation allows words to be embedded in a model to learn the representations of individuals' opinions [13]. In 2017 it became possible to use quantitative data and text to predict if a bill will become law. The researcher states that they always perform better than using text or content individually [14]. Interestingly, the author conducted three experiments, including "...text-only, text, and context, or context-only..." to test the predictive power of each type of model [14]. Nay's approach uses a language model that provides a prediction on the text's sentence level, providing a probability of the sentence contributing to the bill text's (pass/fail) status [14]. The data included a few performance measures and two data conditions spanning 14 years.[14]. Nay concluded that a text alone approach is fundamental for better results. For newer data, the use of bill text outperforms bill context-only. Context-only only outperforms bill text alone for older data [14]. The most important finding is that text adds predictive power to the model. However, the approach relies on two data sources, which creates a predictive bottleneck. As the most successful approach to date using text alone, the model can predict a 65% accuracy [14].

A hybrid C-LSTM Neural Networks architecture can help mitigate the predictive bottleneck in existing models by capturing more features from the bill texts alone. In statistical language modeling, the primary goal is learning. The main objective to learn is the "... joint probability function ..." of each sequence of words contained in the language [2]. However, there is difficulty in this type of task because of the curse of dimensionality. By learning distributed representations of language, the curse mitigates. The main issue is addressing the variance from the training data to the testing data. Past research discovered "... the similarities

between words to obtain generalization from training sequences to new sequences..." [15] [16] [17] [18]. Much of this type of work is thanks to contributions by Schutze, 1993, where vector-space representations for words can be learned based on the probability of the word co-occurring in documents [19]. Most of the experimental workings can be summed up from learning distributed feature vectors to represent their similarities between words, which is discussed in [15] [17] [20]. Past research has shown that the hybridization of CNNs and LSTMs tasked to solve text classification problems is successful [21]. The combination of the two different types of neural networks builds a C-LSTM [21]. In 2015, Zhou et al. introduced a successful C- LSTM in sentence and document modeling [21]. The CNN extracts a sequence of high-level phrase representations, which are in-return fed through LSTM layers set to obtain the sentence-representations [21]. Overall, the hybrid model performed better than existing systems in classification tasks. The results indicated that the local features of phrases and the sentence's global/temporal semantics are recognizable by their model [21]. In 2019, a team of researchers took advantage of convolutional recurrent neural networks to tackle text classification tasks [22]. Their experiments showed that the method of using a C-LSTM achieves better success than other networks.

3. METHODS AND MATERIALS

The following sections discuss key components to developing the PTCN and the associated results referred to in Section IV. The methodology is summarized below in Figure 1.

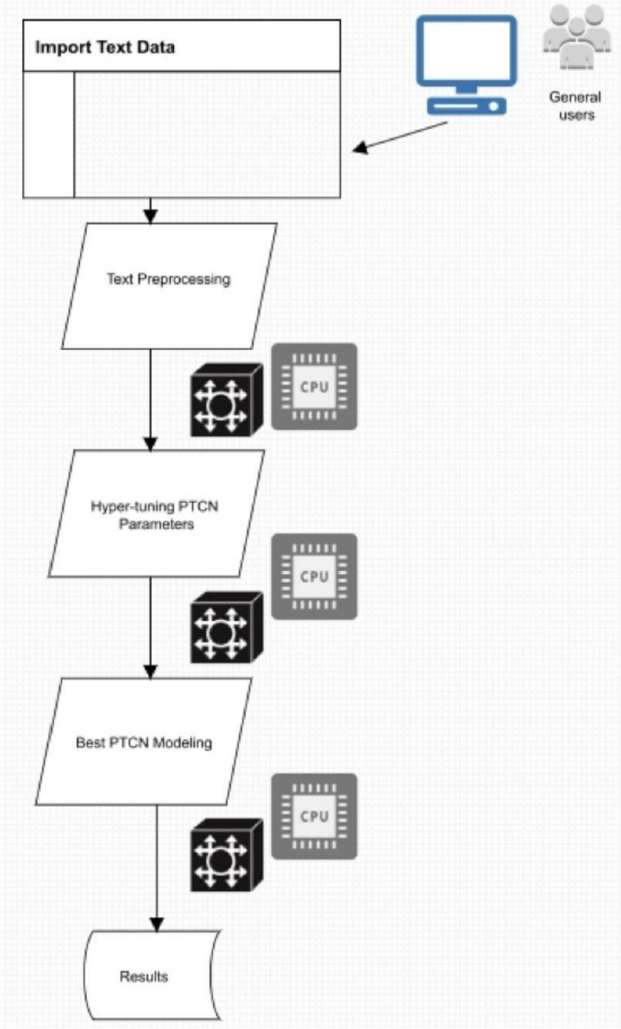

Fig. 1: Visualization of the methodology.

3.1. The Data

The original data is extracted and organized from (https://www.Govtrack.us) consisting of legislative texts and associated quantitative roll call data. The original data contained samples from the year 2000 to 2019, including 3668 samples from the house and senate. Refer to Figure 2 for an example of original legislative texts. Not all the samples from the original population will be included in the experiment due to limiting factors as discussed below.

```
List of 3668
 $ :Classes â€˜readtextâ€™ and 'data.frame':    1 obs. of  2 variables:
  ..$ doc_id: chr "bill.txt"
  ..$ text  : chr "[Congressional Bills 106th Congress]\n[From the U.S. Government Printing Off
 $ :Classes â€˜readtextâ€™ and 'data.frame':    1 obs. of  2 variables:
  ..$ doc_id: chr "bill.txt"
  ..$ text  : chr "[Congressional Bills 106th Congress]\n[From the U.S. Government Printing Off
 $ :Classes â€˜readtextâ€™ and 'data.frame':    1 obs. of  2 variables:
  ..$ doc_id: chr "bill.txt"
  ..$ text  : chr "[106th Congress Public Law 408]\n[From the U.S. Government Printing Office]\
```

Fig. 2: Legislative text samples.

3.1.1. Pre-Processing the Text

A series of NLP techniques are required to supervise the custom C-LSTM training to classify the legislative text based on voting status. The text samples contain noise, which is in the form of stop words, uppercasing, special characters, NA values, punctuation, numbers, and whitespace. The removal of noise from the text samples helps mitigate the C-LSTM from recognizing patterns within the texts that create bias predictions.

The C-LSTM is a deep learning algorithm that automatically extracts features from an input vector. Each text undergoes augmentation using the following conditions, including converting data types from character to a string, lower case conversion, stop-words removal using the "SMART" function, punctuation removal, number removal, white-space removal, and document stemming. The above augmentation resulted in Figure 3.

```
[1] "congression bill th congress govern print offic engross amend senat ea senat unit state ju
[2] "congression bill th congress govern publish offic refer senat rfs th congress d session se
[3] "th congress public law govern print offic doc docid publ page notif feder employe antidisc
[4] "congression bill th congress govern publish offic refer senat rfs doc th congress d sessio
[5] "congression bill th congress govern print offic refer senat rfs th congress d session sena
[6] "congression bill th congress govern print offic calendar senat pcs calendar th congress d
```

Fig. 3: Sample of pre-processed Legislative texts.

After pre-processing, the text goes through a tokenization and vectorization step resulting in each word, symbol, or any other character represented as a unique number. For instance, the word 'bill' is represented by the value of 3109 across all the documents. Note the text is limited to 10000 max features during the tokenization process. Refer to Figure 4 for an example of the vectorized vocabulary from the legislative documents.

```
$congression
[1] 11548

$bill
[1] 3109

$th
[1] 4535
```

Fig. 4: Sample of tokenized and vectorized Vocabulary.

The pre-processing of the text resulted in vectorized legislative documents, as seen in Figure 5. A method of padding is implemented to transform all the texts to the same length. The dimensions of the text are converted into a tokenized, vectorized, and padded format with a max length of 10,000.

```
[[1]]
  [1]  220  699  536   83   82  870   48 4045   11  155 4654  155   12    3   66
 [16] 1371  699   61  122  540    2   11  268  165  257    2  113  116  448    4
 [31]  340    9   45  790   11   26   80  145   20    1  564   16    2  785   44
 [46]   46  340    9  116    2    5 4262  165    1  268  165  257    2   11    2
 [61] 2316 4262  165   46    5  482 3046 1005    1  268  165  257    2   11   29
 [76]  133   37   40    1   99  214  268  165  370  133  175  483    1   99   21
 [91]  268  165  370    5  248    1  268  165  257    2   11  299   59   29   24
[106]   14    7   79  395  162   40   67  185  429   62  208    5 5798  448  140
[121]    9    1  268  165   40 1233   11    2  890   32  526    5  356 1221   13
[136]    1  268  165  257    2  890   11    7   26  435    9  235   20  243
[151]    7    7  243   13   26    7   20    7    7  243   13   26    7   20
[166]   20    7  356 1221  133  229   33 1221  827   40  498  133  392    1   21
[181]  133  871 2717  479  356 1221 2459  133  395   40  849  280  229  543   44
[196]   46  340  733  514    1  127  229   18   49  448   46  340  395  214    9
[211]   29   40  113  303  395  643 1341  247  280  447 1454  133 2717  356    9
```

Fig. 5: Sample of tokenized, vectorized, and padded text.

3.1.2. Equal Distribution of Samples

The labeled legislative text samples are balanced into an equal distribution of each voting status to reduce bias. It is essential to mitigate bias in the data. Deep neural networks transform inputs over extensive credit assignment pathways (CAPs). The more complicated the dimensions of an input, then the more complex the CAPs.

Each vote's status was captured by determining the number of "yea" or "Aye" responses reached for each legislation. Each translates into a value of 1, representing a vote to pass the document. All other responses are considered a vote against the document, labeled as a value of 0. Interestingly, the number of documents labeled one is greater than 0 labeled documents. Note that some votes require a special condition to pass, which is more than 2/3 of the votes. The study ignores the special voting condition because it only focuses on an equal distribution of (pass/fail) text representations.

An equal distribution of the voting statuses in the sample is necessary to mitigate bias in the network for either classification. The text samples are randomly selected from each class to represent an equal distribution, which reduces the original number of samples. Each class of vote is represented equally by 98 randomly selected legislative texts. 98 is the maximum number of 0 labeled samples available in the data due to the behavior of congress. Each text is a max length of 10,000 features. It should be noted that most of the legislative texts are close or higher to the maximum number of features. Custom features in the PTCN's architecture are implemented to deal with the small number of samples during training as discussed below.

3.2. The Deep Learning Approach

The following section discusses the custom deep learning architecture and modelling process implemented in the study.

3.2.1. The Custom PTCN Architecture

The sequential deep learning model is a stack of different layers set with several parameters, including dropout rate, hidden convolutional nodes, LSTM hidden nodes, L1 regularization rate, L2 regularization rate, batch size, input max length, max features, embedding dimensions, leaky Relu rate, kernel size, epochs, max-pooling size, learning rate, and validation split. In Figure 6, an example of the PTCN models architecture:

Model

Layer (type)	Output Shape	Param #
embedding_9 (Embedding)	(None, 1000, 200)	12000000
conv1d_63 (Conv1D)	(None, 1000, 8)	8008
leaky_re_lu_54 (LeakyReLU)	(None, 1000, 8)	0
conv1d_64 (Conv1D)	(None, 1000, 32)	1056
leaky_re_lu_55 (LeakyReLU)	(None, 1000, 32)	0
conv1d_65 (Conv1D)	(None, 1000, 16)	1040
max_pooling1d_18 (MaxPooling1D)	(None, 250, 16)	0
conv1d_66 (Conv1D)	(None, 250, 16)	528
leaky_re_lu_56 (LeakyReLU)	(None, 250, 16)	0
conv1d_67 (Conv1D)	(None, 250, 8)	264
conv1d_68 (Conv1D)	(None, 250, 16)	272
leaky_re_lu_57 (LeakyReLU)	(None, 250, 16)	0
conv1d_69 (Conv1D)	(None, 250, 8)	264
leaky_re_lu_58 (LeakyReLU)	(None, 250, 8)	0
max_pooling1d_19 (MaxPooling1D)	(None, 62, 8)	0
batch_normalization_v1_9 (BatchNorm	(None, 62, 8)	32
dropout_18 (Dropout)	(None, 62, 8)	0
lstm_9 (LSTM)	(None, 32)	5248
leaky_re_lu_59 (LeakyReLU)	(None, 32)	0
dropout_19 (Dropout)	(None, 32)	0
dense_9 (Dense)	(None, 2)	66
activation_9 (Activation)	(None, 2)	0

Total params: 12,016,778
Trainable params: 12,016,762
Non-trainable params: 16

Fig. 6: PTCN architecture

The first layer in the model is the embedding layer, which embeds the tokenized words. At the next layer, the inputs pipe into a 1-dimensional convolutional layer with only a 4th of the set convolutional hidden nodes. The inputs are representations of lengthy and highly sparse word vector spaces, making the model weight's critically important. The model requires a method to adjust to the high variance between samples due to the low number of samples. The control of the kernels will help the model steadily learn significant features contained in the text.

Throughout the convolutions, the model can explore deeper to recognize more significant features. A variance scaling kernel initializer provides an "initializer capable of adapting its scale to the shape of weights. " [23]. Variance scaling is a kernel initialization strategy that encodes an object without knowing the shape of a variable [23]. The Convolutional layers' initializer makes the deep network adapt to the input weights. It is important to regularize the kernels when utilizing an adaptive initializer. Setting a duel regularization technique that deploys L1 and L2 regularization helps mitigate high fluctuations while

batching samples through the layers. L1 is a Lasso Regression (LR). L2 is Ridge Regression. The main difference between the two methods is the penalty term. The layer includes strides set at 1L through the convolutions are an "...an integer or list of a single integer, specifying the stride length of the convolution" [24]. The convolutional layer is activated using a Leaky Relu function. The leaky Relu function allows for a "... small gradient when the unit is not active ...", providing "... slightly higher flexibility to the model than traditional Relu. " [25].

The first convolutional layer extracts lower level features from the inputs due to the decrease in the hidden nodes. The reduction of the number of transformations provides control to the adaptable features initializing the weights. The second layer pipes the inputs through another 1-dimensional convolutional layer with the same parameters set, except the number of hidden nodes set to 32. Increasing the number of hidden nodes provides more transformations extracting higher-level features from the inputs. The second convolutional layer is activated using the leaky Relu function. The next layer in the stack is another convolutional layer set at half the set number of hidden nodes. Reducing the number of nodes and following with a max-pooling layer helps mitigate overfitting during training. In the study, the max-pooling layer is set to 4. The following are two more layers of 1-dimensional convolutional layers set to half the set hidden nodes and a 4th of the set hidden nodes.

All parameters are set the same as the prior convolutional layers. A second max-pooling layer, batch normalization, and dropout layer help mitigate overfitting further. The next layer is an LSTM layer set at 32 hidden nodes. Variance Scaling kernel initializers and L1 and L2 kernel regularizers control the model's exploration of the feature space. The LSTM layer is activated using a Leaky Relu function. A dropout layer of 0.5 is in the stack before the output layer. The output layer is activated using a sigmoid function. The model compiles using a loss function of binary cross-entropy. The PTCN uses a stochastic gradient descent (SGD) optimizer. The hyper-tuning sessions determine the learning rate.

3.2.2. The PTCN Modelling Process

To ensure the model is producing the best results, the parameters of the PTCN are hyper-tuned. The data is split 80/20% for training and validation of the model during the hyper-tuning sessions. The hyper-tuning model's performance evaluates a random selection of 100 samples. The best parameters are captured and set for the final model training. Once the final parameters are identified, the PTCN is trained for a final training session. The final training session of the PTCN uses the best parameters identified during the hyper-tuning sessions, and 10-fold cross-validation is implemented to evaluate the model's performance.

4. RESULTS

After implementing the model using 10-fold cross-validation, the PTCN Model averaged 67.32% evaluation accuracy with a standard deviation of 9.11. The best model performed at 76.32% evaluation on fold 6, as depicted below in Figure 7.

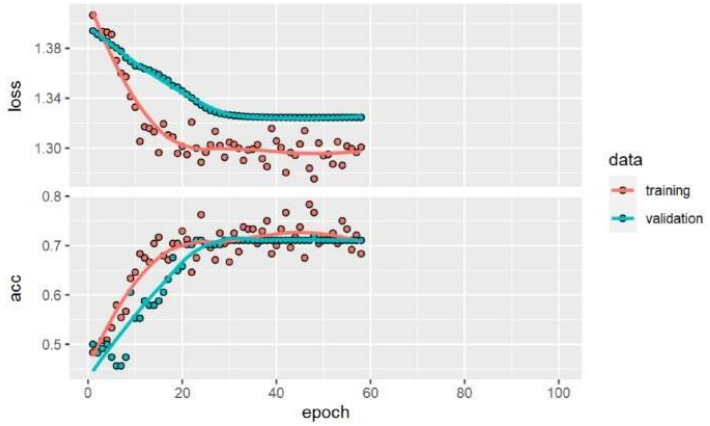

Fig. 7: Sample of PTCN training and validation performance including early stopping.

The minimum evaluation accuracy reports at 44.74% on the fold 2. However, the learning curve in Figure 7 suggests that the parameters are well trained, mainly the lack of samples diminishes the overall performance of the PTCN. The word vector spaces are reliable distributions of language representations for determining roll call vote behavior in congress [26]. Note, the evaluations 95% confidence interval is 59.76% to 88.56% on fold 6. The model's generalization is measured by further understanding its performance through the resulting confusion matrix. Here, the model expresses an accuracy with mitigated bias. On fold 6, Table I shows that the model did not significantly express bias towards either class when tested across a random selection of samples. Even with the small sample size for training the model is able to effectively distinguish between the classes as indicated in Table 1. The adaptability of the model's network to adjust to the inputs shape appears to extract significant features from even a small sample as indicated by folds 3 through 10 in Table 2. Even with a small sample set, the texts were 1000s of words in length. The sheer length of the documents may have been an ample space to extract patterns from the texts due to the available variance. It is likely that the performance of the model on fold 1 and 2 is an indication that the samples lacked ample feature space limiting the PTCN capabilities to adapt to the input's dimensions and recognize significant patterns.

Table 1. Sample of confusion matrix of PTCN evaluation dataset.

	Class 0	**Class 1**
Class 0	15	5
Class 1	4	14

On fold 6, the best measures of accuracy, precision, recall, and F1 score report, 76.32, 75, 78.95, 76.69 respectively. Each fold's performance measures are shown below in Table II showing that the PTCN across all the folds performed relatively well compared to existing approaches.

Table 2. PTCN evaluation metrics across each fold.

	Accuracy	**Precision**	**Recall**	**F1**
Fold1	65.00	80.00	40.00	53.33
Fold2	44.74	45.45	52.63	48.78
Fold3	73.68	76.47	68.42	72.22
Fold4	72.50	71.43	75.00	73.17
Fold5	62.50	61.90	65.00	63.41
Fold6	76.32	75.00	78.95	76.69
Fold7	65.00	65.00	65.00	65.00
Fold8	68.42	65.22	78.95	71.43
Fold9	72.50	73.68	70.00	71.29
Fold10	72.50	71.43	75.00	73.17

The model's training/validation/evaluation indicates that each follows a similar pattern, which suggests that overfitting could easily get handled. Even though the evaluation accuracy shows significant variance across the folds, it provides evidence of the algorithm's robustness. Increasing the number of samples should account for the limitation expressed in fold 1 and 2 from Table 1. However, the overall training, validation, and evaluation of the algorithm indicates the PTCN competes with past models. In turn, the results express that high accuracies predicting congressional roll call votes are possible; however, the accuracy may be bounded lower than deterministic accuracy [27].

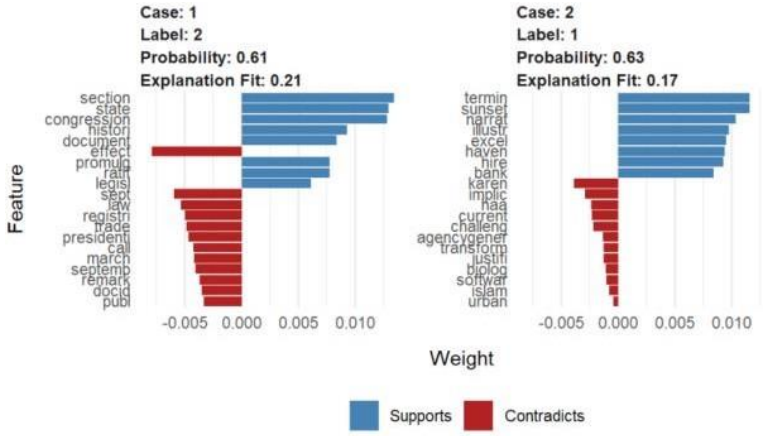

Fig. 8: Sample of the importance of features for each class.

Understanding the features that contribute or contradict each class help determine possible forms of noise in the inputs. A sample of the network's embedded features is depicted in Figure 8. The top-number of features in each class, as depicted in Figure 8, may help explore for noise. Any noise identified should be removed later for further experimentation to help fine the model to the problem.

5. CONCLUSION

The PTCN near performs current state-of-the-art systems in predicting congressional roll call votes. Event circumstances with no available quantitative roll call data or topic complexity do not affect PTCN's predictions.

The PTCN architecture is adaptable to language's sparse nature, allowing it to adjust to complex language representations. The individual legislative texts' overall length helped provide the model with enough information to recognize significant features still. The different types of legislative documents structure and content should be further experimented with to gauge if the model's accuracy scales. Including a larger sample of texts reflecting yea's and nay's may also improve the PTCN's overall capabilities. Another method to improve the model's accuracy is testing a larger range of regularization values on the kernel regularizers. Even though the inputs in these tasks are sparse, and the language may be ambiguous, the PTCN design expresses strong capabilities in recognizing sophisticated features from the data representations transformed into word vector spaces. Mainly, the PTCN recognizes word probabilities of occurrence, similar to other approaches discussed; however, it may recognize features related to the texts' structure or other significant features.

The custom design of the network filters through lower and higher-level features from layer to layer. The innovative stack expresses the ability to adapt to inputs and recognize common and high-level features from complex input dimensions. Further research should shed light on the stability of the model's accuracy and overall reliability.

6. FUTURE WORK

In the future, the PTCN should be tested with various types of problems, texts, and input shapes. Specifically, the PTCN should be tested on other types of binary text classification problems. Testing other types of texts and inputs is critical to understand the generalization of the model.

For the particular task discussed in this paper an example of future works should include shorter length texts, or even samples of greater length, or even samples of larger distributions or lower distributions. Testing the adaptability of the network to various input dimensions is critical to further understand the capability of the PTCN. Also, as more legislative texts become available the current state of the PTCN will be updated by training, validating, and evaluating the model with the inclusion of new samples. The project may be transcribed from R to Python programming language to take advantage of the faster runtimes during training of the PTCN. The research is included publicly in Mind Mimic Labs GitHub repository [28].

REFERENCES

[1] J. D. Clinton, "Using roll call estimates to test models of politics," Annual Review of Political Science, vol. 15, pp. 79–99, 2012, doi: 10.1146/annurev-polisci-043010-095836.

[2] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," Journal of machine learning research, vol. 3, no. Feb, pp. 1137–1155, 2003.

[3] K. T. Poole and H. Rosenthal, "Patterns of congressional voting," American journal of political science, pp. 228–278, 1991, doi: 10.2307/2111445.

[4] N. McCarty, K. T. Poole, and H. Rosenthal, "The hunt for party discipline in congress," American Political Science Review, vol. 95, no. 3, pp. 673–687, 2001, doi: 10.2139/ssrn.1154137.

[5] S. Jackman, "Multidimensional analysis of roll call data via bayesian simulation: Identification, estimation, inference, and model checking," Political Analysis, vol. 9, no. 3, pp. 227–241, 2001, doi: 10.1093/polana/9.3.227.

[6] J. Wilkerson, D. Smith, and N. Stramp, "Tracing the flow of policy ideas in legislatures: A text reuse approach," American Journal of Political Science, vol. 59, no. 4, pp. 943–956, 2015, doi: 10.1111/ajps.12175.

[7] S. Gerrish and D. M. Blei, "How they vote: Issue-adjusted models of legislative behavior," in Advances in neural information processing systems, 2012, pp. 2753–2761, Accessed: Jul. 11, 2019. [Online]. Available: https://semanticscholar.org/paper/0c5b7dde400a8786f32d58ad704f8e1ffd135031.

[8] S. Gerrish and D. M. Blei, "Predicting legislative roll calls from text," in Proceedings of the 28th international conference on machine learning (icml-11), 2011, pp. 489–496, Accessed: Jul. 11, 2019. [Online]. Available: https://semanticscholar.org/paper/62e14dec73970514a5e3f81b059d63b34e9ad37c.

[9] E. Wang, E. Salazar, D. Dunson, L. Carin, and others, "Spatio-temporal modeling of legislation and votes," Bayesian Analysis, vol. 8, no. 1, pp. 233–268, 2013, doi: 10.1214/13-ba810.

[10] I. S. Kim, J. Londregan, and M. Ratkovic, "Voting, speechmaking, and the dimensions of conflict in the us senate," Unpublished paper, 2015.

[11] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," Proceedings of NAACL-HLT 2016, vol. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 1480–1489, 2016, doi: 10.18653/v1/N16-1174.

[12] P. Kraft, H. Jain, and A. M. Rush, "An embedding model for predicting roll-call votes," in Proceedings of the 2016 conference on empirical methods in natural language processing, 2016, pp. 2066–2070, doi: 10.18653/v1/d16-1221.

[13] J. J. Nay, "Gov2Vec: Learning distributed representations of institutions and their legal text," in Proceedings of the first workshop on nlp and computational social science, 2016, pp. 49–54, doi: 10.18653/v1/w16-5607.

[14] J. J. Nay, "Predicting and understanding law-making with word vectors and an ensemble model," PloS one, vol. 12, no. 5, p. e0176999, 2017, doi: 10.1371/journal.pone.0176999.

[15] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," Computational linguistics, vol. 18, no. 4, pp. 467–479, 1992.

[16] F. Pereira, N. Tishby, and L. Lee, "Distributional clustering of english words," in Proceedings of the 31st annual meeting on association for computational linguistics, 1993, pp. 183–190, doi: 10.3115/981574.981598.

[17] T. R. Niesler, E. W. Whittaker, and P. C. Woodland, "Comparison of part-of-speech and automatically derived category-based language models for speech recognition," in Proceedings of the 1998 ieee international conference on acoustics, speech and signal processing, icassp'98 (cat. No. 98CH36181), 1998, vol. 1, pp. 177–180, doi: 10.1109/icassp.1998.674396.
[18] L. D. Baker and A. K. McCallum, "Distributional clustering of words for text classification," in Proceedings of the 21st annual international acm sigir conference on research and development in information retrieval, 1998, pp. 96–103, doi: 10.1145/290941.290970.
[19] H. Schütze, "Word space," in Advances in neural information processing systems, 1993, pp. 895–902.
[20] R. Kneser and H. Ney, "Improved clustering techniques for class-based statistical language modelling," 1993.
[21] C. Zhou, C. Sun, Z. Liu, and F. Lau, "A c-lstm neural network for text classification," arXiv preprint arXiv:1511.08630, 2015, [Online]. Available: https://arxiv.org/ abs/1511.08630.
[22] R. Wang, Z. Li, J. Cao, T. Chen, and L. Wang, "Convolutional recurrent neural networks for text classification," 2019 International Joint Conference on Neural Networks (IJCNN), pp. 1–6, 2019.
[23] "VarianceScaling." Accessed: Jul. 23, 2020. [Online]. Available: https://keras.rstudio.com/reference/initializer variance scaling.html.
[24] "Strides." Accessed: Jul. 23, 2020. [Online]. Available: https://keras.io/api/layers/convolution layers/convolution2d/.
[25] "LeakyRelu." Accessed: Jul. 23, 2020. [Online]. Available: https://keras.io/api/layers/activation layers/leakyrelu/.
[26] M. Taddy, "Document classification by inversion of distributed language representations," in Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 2: Short papers), 2015, pp. 45–49, doi: 10.3115/v1/p15-2008.
[27] T. Martin, J. M. Hofman, A. Sharma, A. Anderson, and D. J. Watts, "Exploring limits to prediction in complex social systems," in Proceedings of the 25th international conference on world wide web, 2016, pp. 683–694, doi: 10.1145/2872427.2883001.
[28] "GitHub mind mimic labs." Accessed: Jul. 27, 2020. [Online]. Available: https://github.com/MindMimicLabs/classification-congressional-votes.
[29] "GitHub." Accessed: Jun. 04, 2018. [Online]. Available: https://github.com.
[30] R Core Team, "R: A language and environment for statistical computing." R Foundation for Statistical Computing, Vienna, Austria, 2019, Accessed: Dec. 15, 2019. [Online]. Available: https://www.R- project.org.
[31] RStudio Team, "RStudio: Integrated development environment for r." RStudio, Inc., Boston, MA, 2015, Accessed: Jun. 30, 2018. [Online]. Available: https://www.rstudio.com.