

NER ON LEGAL DATASET

PREPARED BY

Rishabh Singhal (2018BTechCSE013)

Sanyukta Tanwar (2018BTechCSE026)

FACULTY GUIDE

Dr. Kshitiz Verma



**Department of Computer Science engineering
Institute of Engineering and Technology (IET)
JK Lakshmipat University Jaipur**

May 2022

NER ON LEGAL DATASET

Practice School-2

Submitted in partial fulfilment of the requirements

for the degree of

Bachelor of Technology in Computer Science Engineering

By:

Rishabh Singhal (2018BTechCSE013)

Sanyukta Tanwar (2018BTechCSE026)

Faculty Guide:

Dr. Kshitiz Verma



**Department of Computer Science engineering
Institute of Engineering and Technology (IET)
JK Lakshmipat University Jaipur**

May 2022

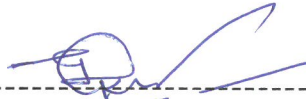
CERTIFICATE

This is to certify that the Practice School-2 project work entitled "**NER ON LEGAL DATASET**" submitted by **Rishabh Singhal (2018BTechCSE013)** **Sanyukta Tanwar (2018BTechCSE026)**, towards the partial fulfillment of the requirements for the degree of **Bachelor of Technology in Computer Science Engineering** of JK Lakshmipat University Jaipur is the record of work carried out by them under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for Practice School-II examination.



Dr. Kshitiz Verma

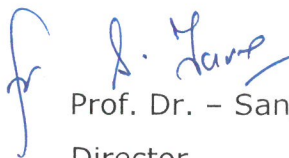
Associate Professor,
Department of Computer Science
Engineering
Institute of Engineering & Technology
(IET)
JK Lakshmipat University Jaipur



Dr. Gireesh Kaushik

Assistant Professor,
Department of Computer Science
Engineering
Institute of Engineering & Technology
(IET)
JK Lakshmipat University Jaipur

Date of Submission 27-05-2022



Prof. Dr. - Sanjay Goel

Director

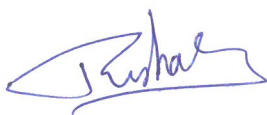
Institute of Engineering & Technology
JK Lakshmipat University Jaipur

ACKNOWLEDGEMENTS

We would like to express our heartfelt gratitude to Dr. Kshitiz Verma, Assistant Professor, Department of Computer Science & Engineering for his valuable time and guidance that made the project work a success.

Thanking all our friends and all those who have helped us carrying out this work directly or indirectly without whom completion of this project work was not possible.

We would also like to thank sincerely to Prof. Dr Sonal Jain, Head of Computer Science Department and Prof Dr Sanjay Goel, Director, Institute of Engineering and Technology, JK Lakshmipat University for giving us a platform to carry out the project.



Sincerely yours,

Rishabh Singhal (2018BTechCSE013)

Sanyukta Tanwar (2018BTechCSE026)

ABSTRACT

For decades, scholars have been fascinated with named entity recognition (NER), a fundamental problem in Natural Language Processing (NLP). Machine translation, information retrieval, and question answering are all examples of downstream NLP and artificial intelligence applications that use NER. Our project's goal is to derive strong word embedding from a corpus of Indian judicial documents and then perform named entity recognition (NER). Researchers have put forth a lot of effort to develop meaningful word embeddings, which can be used to make analogies, do semantic analysis, and perform a variety of other language-related tasks. Our model's goal isn't to provide the best precision throughout the entire language corpus; rather, it should be able to provide more exact conclusions on legal documents than other models trained on more general data. We have tried different NLP model for this and train it on judicial documents. We have used different models which are Bi-LSTM, BERT and Spacy on the legal dataset and analyzed the results for the best output on the legal dataset and then tried to modify the models based on what we've learned in order to develop a more refined model. The key challenge in the legal arena is that the data differs significantly from the general corpus. As a result, the generic model does not produce acceptable results. As a result, we customized the tags and test different models to see which one performs best.

List of Figures

Figure 1.1 Example explaining Named Entity Recognition	1
Figure 2.1 Text file of Supreme Court of India Judgment	4
Figure 2.2 Output from Spacy model as applied to the judgment	4
Figure 4.1 Bidirectional LSTM Diagram	9
Figure 5.1 Bi-LSTM Learning Curve	14
Figure 5.2 BERT Learning Curve	14

List of Tables

Table 1 Results of different models	13
Table 2 Score of different models	13

Contents

CERTIFICATE	III
ACKNOWLEDGEMENTS	IV
ABSTRACT	V
LIST OF FIGURES	VI
LIST OF TABLES	VI
1. INTRODUCTION	1
1.1 LITERATURE REVIEW	2
2. RESEARCH PROBLEM	4
2.1 RESEARCH OBJECTIVE	4
2.2 STANDARD LIBRARIES DON'T WORK	4
2.3 DIFFERENT DOCUMENT MAY HAVE DIFFERENT FORMAT	5
2.4 ENTITY RECOGNITION IN LEGAL DOMAIN	5
3. METHODOLOGY	7
3.1 DATASET	7
3.2 LOAD THE DATASET	7
3.3 DATA PREPROCESSING	7
3.4 BIO TAGGING	8
4. MODEL AND ARCHITECTURE	9
4.1 BI-LSTM	9
4.2 BERT	10
5. RESULTS	13
6. SUMMARY & FUTURE SCOPE OF WORK	15
6.1 SUMMARY	15
6.2 CONCLUSION	15
REFERENCES	16
ANNEXURE I	
CODE	18

Some of the major differences are:

- Technical terms that are unfamiliar to the general public are used in legal English.
- In the legal text, there are other common words that have unique meanings.
- The word order used in several cases is noticeably different when compared to regular English.

In the legal world, a major task in proceeding with any case is to read and understand that case file as well the files with similar records. It is tough to process the legal documentations for automated tasks. Named entity recognition can be used to solve this problem and provide an overview of the file. The goal of our research is to find an efficient method for Named Entity Recognition (NER). We used two models that are bidirectional LSTM and BERT. These models showed different behavior over legal text.

1.1 Literature Review

There are various rule-based NER systems that can provide 88 percent to 92 percent f-measure accuracy for English and include lexicalized grammar, gazetteer lists, and a list of trigger words [23]. The main drawbacks of rule-based approaches are that they necessitate a great deal of experience and grammatical knowledge of the target language or domain and that they are not transferrable to other languages or domains. To obtain high-level linguistic knowledge, ML-based approaches for NER make use of a huge amount of NE annotated training data.

The Hidden Markov Model was used, giving the improvement of 2.53% on F-measure [21], Maximum Entropy [9], and Conditional Random Field [17] are the most commonly utilized ML approaches for the NER challenge. Different ML methods are also used in combination. To create a NER system, Srihari et al. (2000) integrate Maximum Entropy, Hidden Markov Model, and customized rules. For recognizing names, NER systems employ gazetteer lists[19]. Gazetteer lists are used in both the linguistic [23] and the ML-based approaches[19, 21]. Hand-crafted rules are used in the linguistic method, which necessitates the involvement of professional linguists. Some modern systems use machine learning to discover context patterns, reducing the amount of manual effort. The authors in [8]

CHAPTER 1: INTRODUCTION

Natural language processing(NLP) using deep learning are being increasingly used in the legal domain, for legal text segmentation [14], legal subject categorization [15, 16], legal judgment prediction and analysis[2], legal information extraction[7], legal question answering[10, 11], etc. Modern neural NLP methods typically rely on pre-trained word embeddings or pre-trained language models. Word embedding is a way of representing text and documents. A numeric vector input that represents a word in a lower-dimensional space is known as Word Embedding or Word Vector. It permits words with comparable meanings to be represented in the same way. A 50-valued word vector can express 50 distinct features[18]. We need sequence labeling for preprocessing data in all of the above NLP applications. Sequence labeling can be accomplished in a variety of ways; one of the most common data preprocessing tasks is named entity recognition (NER).

Gopal Ramnath Lives near Agra Cantt Railway Station
B-Per I-Per B-loc I-loc I-loc I-loc

Figure 1.1: Example explaining Named Entity Recognition.

NER includes recognizing key information in a text and categorizing it into a set of predetermined categories. An entity is essentially the object that is mentioned or referred to repeatedly throughout the text. As we can see in Fig. 1 Gopal Ramnath is a person entity and Agra Cantt Railway Station is a location entity. We can further classify an entity based on its beginning, inside and outside respectively. We apply the BIO/IOB tagging format (short for beginning, inside, and outside). The B- prefix before a tag indicates that it is at the start of a entity, while the I- prefix indicates that it is within a entity. A token with an O tag does not belong to any entity.

Our focus is on the legal dataset where we have different judgment files in different formats of unstructured data. Also, the legal text has different characteristics compared to generic corpora, such as specialized vocabulary, peculiar syntax, semantics based on considerable domain-specific knowledge, and so on[4].

used a combination of grammatical and statistical methodologies to generate high-precision NE extraction patterns. They offered a method for learning lexical patterns for Indian 4 languages . In Greek legislation, researchers tested LSTM-based approaches for named entity recognition, demonstrating the utility of such techniques in foreign language contexts. In Resource description framework(RDF), they established and used a new language for representing textual references. They also looked at entity linking between textual references and entities from third-party datasets, as well as creating a new dataset for Greek geo-landmarks[3].

The author in [5] used Support Vector Machine(SVM) and Logistic regression in their work of extraction contract elements. They also employed deep learning to improve tasks by adding a CRF layer on top of the BILSTM or stacking an extra LSTM on top of the BILSTM. When techniques are examined for their capacity to extract multi-token contract parts, the stacked BILSTM-LSTM misclassifies fewer tokens, while the BILSTM-CRF combination performs better. They also improved the querying platform for Greek legislation (Nomothesia) by integrating Name Entity Recognition and the Bi-LSTM Bi-LSTM LR neural model.

CHAPTER 2: RESEARCH PROBLEM

2.1 Research Objectives

Our goal is to do NER on the judgments of Supreme Court of India and various High Courts in India. The failure of the standard models to do even basic NER on legal data promoted us to research the topic. We substantiate the failure of the standard model through the examples provided in the following subsections.

2.2 Standard libraries don't work :

Allahabad High Court
Rajendra Pratap . vs Prathma Bank Thru Chairman And ... on 28 March, 2018Bench: Vivek Kumar Birla
HIGH COURT OF JUDICATURE AT ALLAHABAD
?AFR
Court No. - 30
Case :- WRIT - A No. - 11373 of 2013
Petitioner :- Rajendra Pratap .
Respondent :- Prathma Bank Thru Chairman And Ors.
Counsel for Petitioner :- Siddharth Khare Ashok Khare
Counsel for Respondent :- P.K.Singhal S.C.
Hon'ble Vivek Kumar Birla J.
Heard Sri Siddharth Khare, learned counsel for the petitioner as well as Sri P.K. Singhal, learned counsel for the respondents-Bank and perused the record.
Pleading between the parties have been exchanged and with their consent, the present petition is being decided finally at the admission stage itself.
Present petition has been filed challenging the impugned order dated 30.6.2012 passed by the General Manager of the Prathma Bank and the order dated 27.12.2012 passed by the Chairman, Prathma Bank, Moradabad. A further prayer has been made to direct the respondents to forthwith reinstate the petitioner on his post with all consequential benefits of continuity of service with all arrears of salary.

Figure 2.1:Text file of Supreme Court of India Judgment

```
['PROCEEDINGS\n\n    Petition(s', 'ORG']  
['29', 'CARDINAL']  
['WP', 'ORG']  
['the High Court', 'ORG']  
['VERSUS', 'ORG']  
['RAJESH KUMAR & ORS', 'ORG']  
['Respondent(s', 'ORG']  
['O.T.', 'GPE']  
['today', 'DATE']  
['KHANWILKAR\n\n                                HON'BLE DR", 'ORG']  
['Tushar Mehta', 'PERSON']  
['Anindita Pujari', 'PERSON']  
['AOR', 'ORG']  
['Ms.', 'ORG']  
['Kavita Bhardwaj', 'PERSON']  
['Deepanwita Priyanka, Adv', 'ORG']  
['Harsha Garg', 'PERSON']  
['Respondent(s', 'ORG']
```

Figure 2.3: Output from Spacy model as applied to the judgment

Our research is on legal data which is different from the general corpus. This problem can be explained by examples using Fig. 2 and Fig. 3. When we apply the standard model of Spacy for NER[1], it performs well over a general corpus, but when it is used for legal judgments(Shown in Fig. 2), it has several issues. As we can see in Fig. 3 there is a lot of mistagging. The model tags Adv. and HON'BLE as ORG, but they refer to an advocate and a judge, respectively. Section 13, Chapter 2, and Constitution are all tagged as LAW, which is insufficient for the legal processing. We may need more specific tags on the Sections, Articles, etc. Also, it didn't fetch the petitioner because in this Fig. 2 we can see the text is not written in a single line and the data is not in a structured format. So these examples justify how the standard model of Spacy for NER doesn't give good results in the legal context. This problem may not be solved even by the use of the custom Spacy model[20] because if we see in Fig. 2 and Fig. 3, there are two problems one for proper tagging and the second for data extraction. Custom Spacy can solve the problem of tagging as we can customize tags but the problem of data extraction will not be solved.

2.3 Different documents may have different formats:

Also, if we read the format of files to extract information, we'll have a lot of problems since we'll have to create separate models for different formats and we might not capture all of the information we need. For example, In a judicial judgment sections X, Y, and Z appear. The format reading is incapable of extracting all the sections which are in the text. It only reads the standard format and fails to capture complete information.

2.4 Entity Recognition in legal judgement:

NER should enable us to read the files in different formats, even when data is in an unstructured format i.e all formats are not alike. Therefore, we need to develop our own NER model, we'll require manually tagged data to build a custom NER model. When we discuss a specific domain, we may use specific tags. In legal domain, tags for judges, advocates, sections, and so on are required.

- Person: In the legal judgments name of persons which are stated in the text(e.g., Judge, Advocate, Respondent, Petitioner, etc.).

- Organization: In judgment sometimes petitioner or the respondent is a public or private organization (e.g. National Green Tribunal(NGT), National Insurance Company, Rotomac Pens Limited, etc.).
- Geopolitical Entity: In legal terms we have some important locations (e.g. court, crime location, etc.)
- Reference to the Law: Any mention of Indian law is frowned upon (e.g., Sections, Articles, Chapters, Acts, etc.)

CHAPTER 3: METHODOLOGY

We have different judgement files from different courts of India that follow different formats. Firstly we will do parts of speech(POS) tagging and remove the words which are not useful. Then we manually tagged the data in legal context which was required to train our models. BERT and Bi-LSTM are used to perform NER and find the best result.

3.1 Dataset:

Our dataset contains text files of Indian judicial judgments. The dataset contains judgments from several Indian courts, including the Supreme Court of India and the majority of the country's high courts. With approximate 6.1 millions files, we have data from 1947 to 2018. However, we realized that it is not trivial to generalize NER to legal dataset. Hence, we could only use 60 files to define the tags and build the model for NER on legal dataset and the results presented in this paper correspond to these 60 files.

3.2 Load the dataset:

To read the dataset we have used the “open” method and then used readlines to fetch lines from the txt files. After reading the text file which was classified in different courts in India now our task was to preprocess it and tag the data as required for our model to train.

3.3 Data Preprocessing:

After reading the text files of the judgments. Our task is to preprocess it and tag the data as required for our model to train. Tagging is a type of classification that is defined as the automatic assigning of a description to each token. The descriptor here is called a tag, and it can represent a number of things, including parts of speech, semantic information, and so on[13]. Part of speech(POS) tagging is a typical Natural Language Processing process that includes categorizing words in a text (corpus) into one of several parts of speech based on the definition and context of the word[22]. In POS tagging, there are two key steps: first, we must tokenize the word, and then we apply the nltk.pos tag to the tokenized words. So, after POS tagging, we only kept proper nouns, nouns, and numbers from the corpus and exported our data to a CSV file. The main element now is proper name tagging, which

we need to train our model and we manually tagged 20000 words to get a total of 28 tags where 8 are specific to legal data which include section(*B-SEC*, *I-SEC*), article(*B-ART*, *I-ART*), judge(*B-JUG*, *I-JUG*), court(*B-CRT*, *I-CRT*) and remaining 20 were the general tags which include *B-STATE*, *B-COUNTRY*, *I-ORG*, *B-DAY*, *B-CITY*, *I-CITY*, *B-PLC*, *I-PLC*, *I-PER*, *B-TIM*, *I-TIM*, *B-ORG*, *B-PER*, *O*, *I-STATE*, *B-DATE*, *I-DATE*, *B-YEAR*, *B-MONTH*, *B-MONEY*. We tagged the data in BIO format.

3.4 BIO-Tagging:

The main problem is that if we tag the data with traditional format, the model may not be able to distinguish between a person and a judge. BIO format helps here. For example, Hon'ble Manoj Gupta is a judge's name, and Hon'ble appears before the judge's name in Indian judgments. A normal format model would tag Manoj Gupta as a person, but we need a specific tag, so we can use the BIO format to tag Hon'ble as B-Judge and Manoj and Gupta as I-Judge. When it comes to section numbers, if we have Section 14 in our data, we require it to be SEC NUM. However, 14 might be classified as DATE. Also BIO format can solve many problems for example we have advocates from two side but it is difficult to understand for normally tagged data who is the advocate of applicant and who is for respondent. So BIO format can help us in identifying the relation.

CHAPTER 4: MODEL AND ARCHITECTURE

4.1 Bidirectional LSTM:

Bidirectional LSTM is just a combination of LSTM units in both forward and backward order to save the context of the previous and next LSTM unit. This trait of bidirectional LSTM helps in the preservation of near context of word. Long short term memory, or LSTM, was created to address the problem of vanishing gradients in recurrent neural networks. LSTM solves this problem in part by employing three gates: read gate, forget gate, and write gate. The forget and output gates determine whether fresh information should be kept or discarded. The model decision is made using the LSTM block's memory and the output gate's condition. The output is then fed again into the network as an input, resulting in a recurrent sequence. LSTM works effectively on long sequences because of this structure

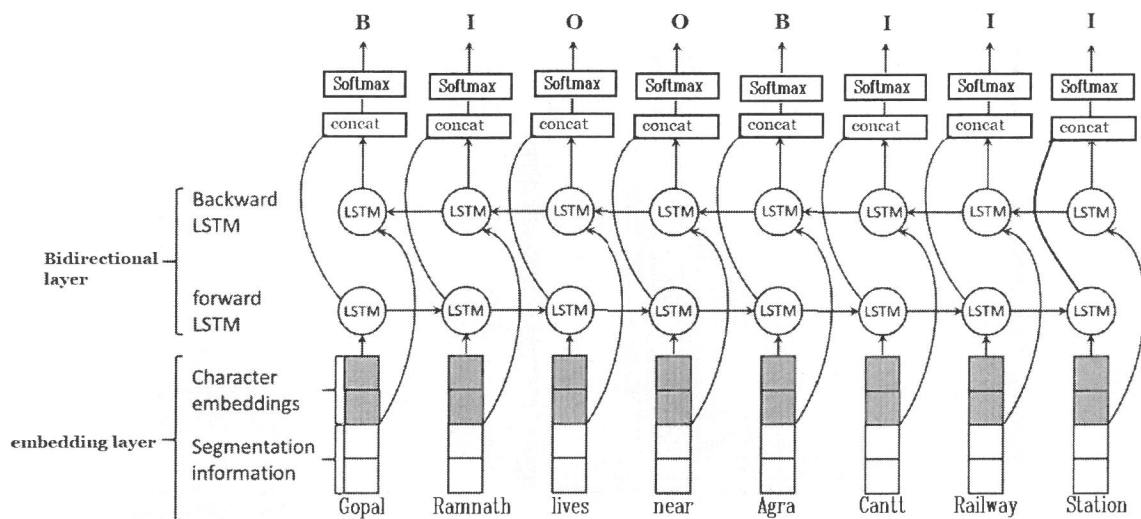


Figure 4.1: Bidirectional LSTM Diagram

Our Bi-LSTM model is divided into the following three layers:

- Embedding layer / Input layer :

This layer is responsible for creating input word embeddings for the Bi-LSTM model. It's done by defining the padded sequence's dimension. As the maximum length of a sentence in our data is 75 the padded sequences will have a maximum length of 75 words. After the network has been trained, the embedding layer will convert each token into an n-dimensional vector (word embeddings). We've set the value of n to 128. The dimensions of the embedding layer's output are (None,

75, 128). The batch sizes are represented by the first argument, 'None.' They display "None" if it is not specified, showing that the model can handle any batch size.

- Bi-LSTM layer:

The output from the preceding embedding layer is used in this layer (75, 128). We used 75 LSTM units in this layer. We'll have forward as well as backward outputs because this is a bidirectional LSTM. Before passing it on to the next layer, it concatenates these outputs. As a result, double the number of outputs to the following layer. It becomes $256(128 * 2)$ in our situation.

- TimeDistributed Dense layer:

We're working with a Bi-LSTM architecture, which means we're anticipating output from each input sequence. As an example, in the sequence (a1 b1, a2 b2, ..., an bn), a and b represent the sequence's inputs and outputs, respectively. Dense(fully-connected) operation across every output over every time step is possible with the TimeDistributeDense layers. This layer will result in a final output of dimension (75,28). The number of words for which tags have been predicted is represented by 75 units, while the probability of each tag for a particular word is represented by a distribution of 28 units.

4.2 BERT:

BERT is a Transformer-based language model that has been pre-trained on big corpora. A task-specific layer is built on top of BERT for each new task, and the model is trained together using task-specific data. For binary violation, multi-label violation, and case importance, we put a linear layer on top of BERT with a sigmoid, softmax, or no activation respectively. We truncate our case descriptions to BERT's maximum length, which affects its performance. This also highlights a significant shortcoming of BERT when it comes to processing large texts, which is a prevalent feature of legal text processing. Since BERT uses absolute position embeddings, it's best to pad the right side of the inputs rather than the left[6].

Masked language modelling (MLM) is used to train BERT. MLM gives BERT a sentence and for optimizing the weights within BERT to produce the identical text on the other side. However, we must mask a few tokens before giving BERT the input sentence. Thus we give BERT a sentence and ask it to reproduce it.

Tokenization: Tokenization is breaking down sentences into words, or tokens, while also removing specific characters, such as punctuation. BERT can conveniently implement it using the BertTokenizerFast class from a pre-trained BERT base model. We utilized the "bert-base-cased" model with "75 words" as the maximum length of an input sequence, post padding to extra PAD tokens at the beginning of sentences smaller than the maximum length of the sequence and set truncation to "TRUE" to restrict tokens that exceed the maximum length of the sequence.

[CLS] and [SEP] tokens are appended to the beginning and end respectively of the tokenized sentence after each word in the input text is tokenized. These tokens serve as an input representation for classification tasks and are used to discriminate between two input sentences.

The length of the input sequence does not match the length of the original label after the tokenization process. The BERT tokenizer incorporates a word-piece tokenizer, which splits a single word into one or more meaningful sub-words. After the tokenization procedure, there were two big issues. The first was the insertion of special tokens such as PAD, CLS, and SEP from BERT, and the second was the splitting of some tokens into sub-words. The "word ids" method from the tokenization was used to fix this problem because it does not give specific word ids to special tokens from BERT and only provides a word id to the first sub-word while assigning "-100" as an id to the rest of the sub-words. The Token Embeddings layer will turn each wordpiece token into a 512-dimensional vector representation.

The three embedding layers in BERT are as follows:

- Token Embedding layer:

This layer is responsible for determining the embedding for each word in the sequence. The model will learn an "emb-dim" dimensional vector for each word in the sequence during training. For each token, you can think of it as a vector look-up. The members of the vectors are considered as model parameters, and they, like any other weights, are optimized via back-propagation.

- Segment Embedding layer:

This layer helps BERT to contextually distinguish each sentence in the input. There are two input sentences, example, I enjoy learning NLP and I enjoy learning DBMS. The model is simply fed the pair of input texts that have been

concatenated. These sentences are represented in the model Segment embeddings layer by two vectors. All tokens belonging to input 1 are assigned to the first vector (index 0), whereas all tokens belonging to input 2 are assigned to the second vector (index 1). To produce final output of BERT, these vectors are transmitted through a linear output layer.

- Position Embedding layer:

Transforms exist in this layer of BERT encoding of the sequential nature of their inputs from the Embedding layer. As a result, both Happy and Good will have identical position embeddings because they are the first words in the given sentence Happy new year and Good morning.

Modeling and Training:

"BertForTokenClassification" class has been used to model the BERT model. This class is a model that encapsulates the BERT model and adds linear layers that operate as token-level classifiers on top of it. The number of unique tags defined in the dataset is set as the output of each token classifier. We utilised "adam" as the optimizer and trained our model across 50 epochs.

context by detecting "B" and "I" entity of date and judge tag because of its structural property of taking inputs from both the directions.

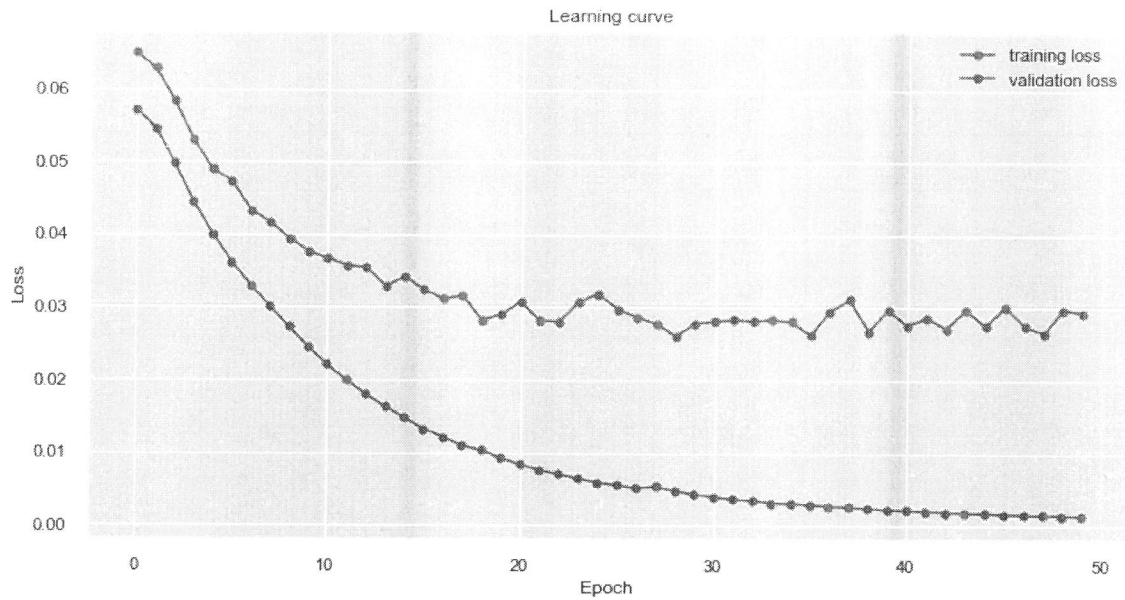


Figure 5.1: Bi-LSTM Learning Curve

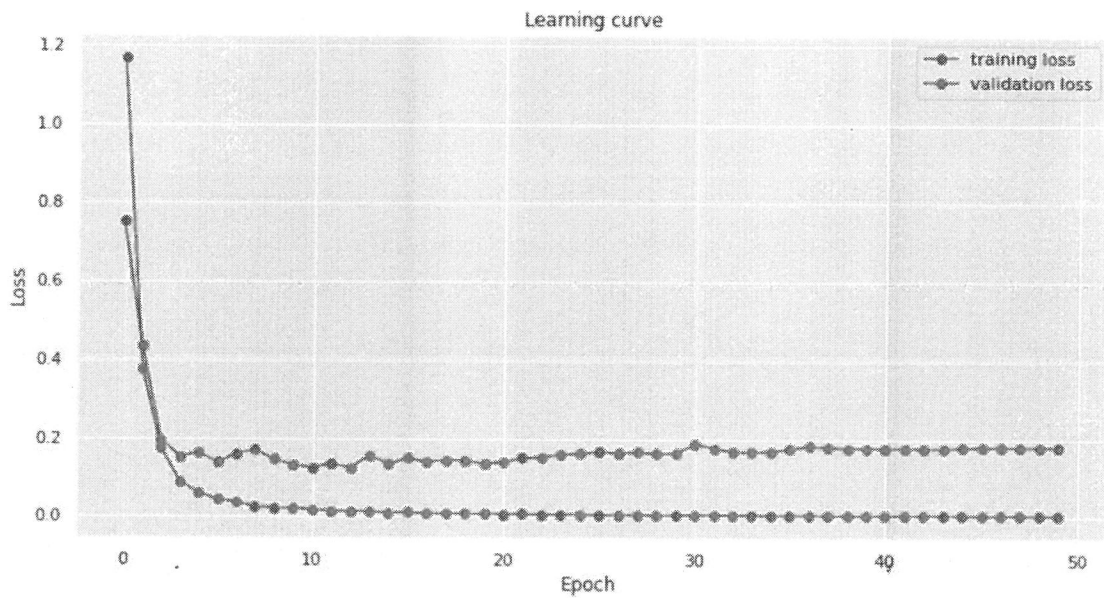


Figure 3.2: BERT Learning Curve

CHAPTER 6: SUMMARY AND FUTURE WORK

6.1 SUMMARY:

This research aims to perform NER on the legal judgments using bidirectional LSTM and BERT models. We used tags specific to legal data in tagging and trained the models. The experiment done on 60 files shows that our model improve the efficiency of NER on the legal judgments. Meanwhile, our model performs well on unstructured data where it signifies the uniqueness of our model. Furthermore, both the models have different outputs and with different accuracy. The main challenge we experienced was tagging the data because the legal domain differs significantly from the general corpus

6.2 FUTURE WORK:

We got the information but now we need to identify how that information is useful or related. For example if we want to find the Advocate of respondent and petitioner we need a relation or if we are able to capture all the sections we can not figure out which is relevant to the particular case. So we can work on this in future. As we have used both BERT and Bi-LSTM and Bi-LSTM works better so further we can add CRF layer in Bi-LSTM model for more efficient result in BIO-tagging.

References

1. Entity recognizer. <https://spacy.io/api/entityrecognizer>
2. Aletras, N., Tsarapatsanis, D., Preotijuc-Pietro, D., Lampos, V.: Predicting judicial decisions of the european court of human rights: A natural language processing perspective. *PeerJ Computer Science* 2, e93 (2016)
3. Angelidis, I., Chalkidis, I., Koubarakis, M.: Named entity recognition, linking and generation for greek legislation. In: JURIX. pp. 1–10 (2018)
4. Chalkidis, I.: Deep neural networks for information mining from legal texts. Ph.D. thesis, Οικονομικόν Πανεπιστήμιο Αθηνών. Σχολή Επιστημών και Τεχνολογίας της... (2021)
5. Chalkidis, I., Androutsopoulos, I.: A deep learning approach to contract element extraction. In: JURIX. pp. 155–164 (2017)
6. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018)
7. Dozier, C., Kondadadi, R., Light, M., Vachher, A., Veeramachaneni, S., Wudali, R.: Named entity recognition and resolution in legal text. In: *Semantic Processing of Legal Texts*, pp. 27–43. Springer (2010)
8. Ekbal, A., Bandyopadhyay, S.: Lexical pattern learning from corpus data for named entity recognition. In: *Proceedings of International Conference on Natural Language Processing (ICON)* (2007)
9. Grishman, R., Borthwick, A.: A maximum entropy approach to named entity recognition (1999)
10. Kim, M.Y., Xu, Y., Goebel, R.: A convolutional neural network in legal question answering. In: *JURISIN Workshop* (2015)
11. Kim, M.Y., Xu, Y., Lu, Y., Goebel, R.: Legal question answering using paraphrasing and entailment analysis
12. Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., Dyer, C.: Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360* (2016)
13. Li, J., Sun, A., Han, J., Li, C.: A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering* 34(1), 50–70 (2020)
14. Loza Mencía, E.: Segmentation of legal documents. In: *Proceedings of the 12th International Conference on Artificial Intelligence and Law*. pp. 88–97 (2009)

15. Loza Mencía, E., Fürnkranz, J.: Efficient multilabel classification algorithms for large-scale problems in the legal domain. In: *Semantic Processing of Legal Texts*, pp. 192–215. Springer (2010)
16. Nallapati, R., Manning, C.D.: Legal docket classification: Where machine learning stumbles. In: *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*. pp. 438–446 (2008)
17. Peng, F., McCallum, A.: Accurate information extraction from research papers using conditional random fields. pp. 329–336 (2004)
18. Rong, X.: word2vec parameter learning explained. arXiv preprint arXiv:1411.2738 (2014)
19. Srihari, R.K.: A hybrid approach for named entity and sub-type tagging. In: *Sixth Applied Natural Language Processing Conference*. pp. 247–254 (2000)
20. Tarcar, A.K., Tiwari, A., Dhaimodker, V.N., Rebelo, P., Desai, R., Rao, D.: Health-care ner models using language model pretraining. arXiv preprint arXiv:1910.11241 (2019)
21. Todorovic, B.T., Rancic, S.R., Markovic, I.M., Mulalic, E.H., Ilic, V.M.: Named entity recognition and classification using context hidden markov model. In: *2008 9th Symposium on Neural Network Applications in Electrical Engineering*. pp. 43–46 (2008). <https://doi.org/10.1109/NEUREL.2008.4685557>
22. Voutilainen, A.: Part-of-speech tagging. *The Oxford handbook of computational linguistics* pp. 219–232 (2003)
23. Wakao, T., Gaizauskas, R., Wilks, Y.: Evaluation of an algorithm for the recognition and classification of proper names. In: *Proceedings of the 16th Conference on Computational Linguistics Volume 1*. p. 418–423. COLING '96, Association for Computational Linguistics, USA (1996). <https://doi.org/10.3115/992628.992701>, <https://doi.org/10.3115/992628.992701>

ANNEXTURE 1:

CODE:

```
import pandas as pd
import spacy
from spacy import displacy
q=[]
NER = spacy.load("en_core_web_sm")
text1=NER(raw_text)
for word in text1.ents:
    print(word.text,"*****", word.label_)
    print([word.text,word.label_])

print(pd.unique(w[1]))

filepath = Path('outspacy.csv')
filepath.parent.mkdir(parents=True, exist_ok=True)
w.to_csv(filepath)
```

```
['Bhatt', 'PERSON']
['U.P. Jal Nigam Thru', 'ORG']
['3 January, 2018', 'DATE']
['Satyendra Singh Chauhan', 'PERSON']
['ALLAHABAD', 'ORG']
['LUCKNOW BENCH', 'ORG']
['2017', 'DATE']
['Bhatt', 'GPE']
['Rana Pratap Marg', 'PERSON']
['Singh', 'GPE']
['Hon'ble Satyendra', 'PERSON']
['Kumar', 'PERSON']
['J.\n\nLearned', 'ORG']
['two weeks', 'DATE']
['Order Date :- 3.1.2018', 'LAW']
```

```
# import modules and download packages
import nltk
nltk.download('words')
nltk.download('punkt')
nltk.download('maxent_ne_chunker')
nltk.download('averaged_perceptron_tagger')
nltk.download('state_union')
from nltk.corpus import state_union
from nltk.tokenize import PunktSentenceTokenizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
```

```

import pandas as pd
from pathlib import Path
a=[]
c=0
with open('3.txt', encoding='utf-8') as f:
    lines = f.readlines()
    for i in range(len(lines)):
        text = word_tokenize(lines[i])
        p=nlk.pos_tag(text)
        q=[]
        for i,j in enumerate(p):
            #print(i,j)

            if(i==0):
                n = ("Sentence: "+str(c),)
                j=n+j
                #print(j)
                q.append(j)
            else:
                n=("NaN",)
                j=n+j
                q.append(j)

        c=c+1
        a.extend(q)

#print(a)
b=pd.DataFrame(a)
print(b)
#filepath = Path('out.csv')
#filepath.parent.mkdir(parents=True, exist_ok=True)
#b.to_csv(filepath)

```

	0	1	2
0	Sentence: 1	Supreme	NNP
1	NaN	Court	NNP
2	NaN	-	:
3	NaN	Daily	JJ
4	NaN	Orders	NNS
...	...	...	...
35199	NaN	SONI	NNP
35200	NaN	]	NN
35201	Sentence: 8057	AR-cum-PS	NNP
35202	NaN	BRANCH	NNP
35203	NaN	OFFICER	NNP

[35204 rows x 3 columns]

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.preprocessing.sequence import pad_sequences
from keras.utils.vis_utils import plot_model
from numpy.random import seed
from itertools import chain
from tensorflow.keras import Model, Input
from tensorflow.keras.layers import LSTM, Embedding, Dense
from tensorflow.keras.layers import TimeDistributed, SpatialDropout1D, Bidirectional
from tensorflow.keras.utils import to_categorical
plt.style.use('seaborn')

```

```

file = r'manual.csv'
data1 = pd.read_csv(file, encoding = "latin1")
data1 = data1.fillna(method = 'ffill')
words = list(set(data1["Word"].values))
words.append("ENDPAD")
num_words = len(words)

```

```

#Grouping Words and tags
tags = list(set(data1["Tag"].values))
num_tags = len(tags)
print(tags)
class Get_sentence(object):
    def __init__(self, data1):
        self.n_sent = 1
        self.data1 = data1
        agg_func = lambda s: [(w, t) for w, t in zip(s["Word"].values.tolist(),
                                                    s["Tag"].values.tolist())]

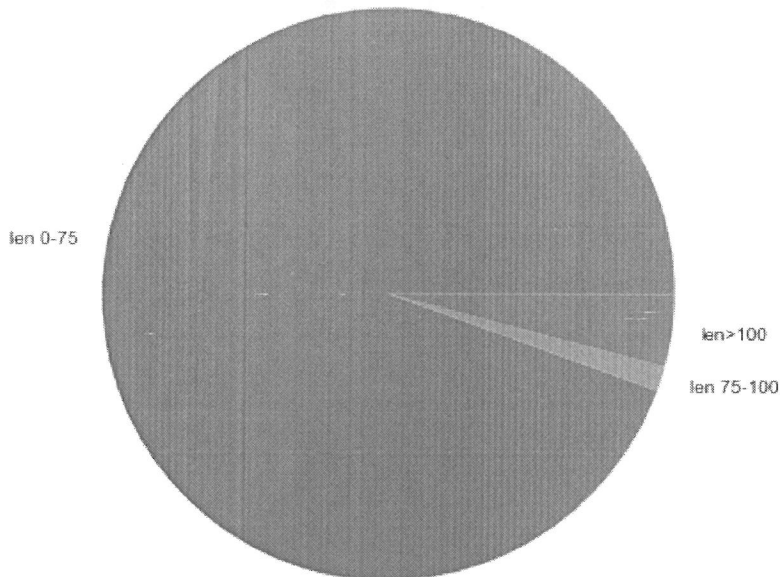
        self.grouped = self.data1.groupby("i>Sentence").apply(agg_func)
        self.sentences = [s for s in self.grouped]
getter = Get_sentence(data1)
sentence = getter.sentences

```

```

#Mapping Length of sentences
plt.figure(figsize=(14,7))
c1=0
c2=0
c3=0
for s in sentence:
    if(len(s)<75):
        c1+=1
    elif(75<len(s) and len(s)<100):
        c2+=1
    else:
        c3+=1
y = np.array([c1,c2,c3])
x = ["len 0-75", "len 75-100", "len>100"]
plt.pie(y, labels=x)
plt.show()

```



```
# Indexing words and tags
word_idx = {w : i + 1 for i ,w in enumerate(words)}
tag_idx = {t : i for i ,t in enumerate(tags)}
print("Toatl tags are" , len(tag_idx))
print(tags)
```

```
#Padding the sequences
#Changing the List of words into 2D numpy arrays.
max_len = 200
X = [[word_idx[w[0]] for w in s] for s in sentence]
X = pad_sequences(maxlen = max_len, sequences = X, padding = 'post', value = num_words - 1)
y = [[tag_idx[w[1]] for w in s] for s in sentence]
y = pad_sequences(maxlen = max_len, sequences = y, padding = 'post', value = tag_idx['0'])
y = [to_categorical(i, num_classes = num_tags) for i in y]
```

```
#Spling training and testing data
x_train,x_test,y_train,y_test = train_test_split(X, y,test_size = 0.1, random_state = 1)
```

```
#Creating the model
#Instantiating a Keras tensor.
input_word = Input(shape = (max_len,))
model = Embedding(input_dim = num_words, output_dim = max_len, input_length = max_len)(input_word)
model = SpatialDropout1D(0.1)(model)
model = Bidirectional(LSTM(units = 100,return_sequences = True, recurrent_dropout = 0.1))(model)
out = TimeDistributed(Dense(num_tags,activation = 'softmax'))(model)
model = Model(input_word,out)
model.compile(optimizer = 'adam',loss = 'categorical_crossentropy',metrics = ['accuracy'])
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 200)]	0
embedding (Embedding)	(None, 200, 200)	350200
spatial_dropout1d (SpatialD ropout1D)	(None, 200, 200)	0
bidirectional (Bidirectiona l)	(None, 200, 200)	240800
time_distributed (TimeDistr ibuted)	(None, 200, 26)	5226

=====
Total params: 596,226
Trainable params: 596,226
Non-trainable params: 0
=====

#Training the model

```
l=model.fit(x_train, np.array(y_train), batch_size = 16, verbose = 2, epochs = 50, validation_split = 0.2)
print("*****")
print(l)
```

```
Epoch 1/50
23/23 - 28s - loss: 0.0572 - accuracy: 0.9866 - val_loss: 0.0650 - val_accuracy: 0.9864 - 28s/epoch - 1s/step
Epoch 2/50
23/23 - 28s - loss: 0.0546 - accuracy: 0.9866 - val_loss: 0.0629 - val_accuracy: 0.9864 - 28s/epoch - 1s/step
Epoch 3/50
23/23 - 27s - loss: 0.0498 - accuracy: 0.9867 - val_loss: 0.0584 - val_accuracy: 0.9867 - 27s/epoch - 1s/step
Epoch 4/50
23/23 - 28s - loss: 0.0445 - accuracy: 0.9873 - val_loss: 0.0530 - val_accuracy: 0.9872 - 28s/epoch - 1s/step
Epoch 5/50
23/23 - 29s - loss: 0.0401 - accuracy: 0.9879 - val_loss: 0.0489 - val_accuracy: 0.9880 - 29s/epoch - 1s/step
Epoch 6/50
23/23 - 29s - loss: 0.0362 - accuracy: 0.9893 - val_loss: 0.0474 - val_accuracy: 0.9886 - 29s/epoch - 1s/step
Epoch 7/50
23/23 - 29s - loss: 0.0330 - accuracy: 0.9911 - val_loss: 0.0433 - val_accuracy: 0.9898 - 29s/epoch - 1s/step
Epoch 8/50
23/23 - 30s - loss: 0.0301 - accuracy: 0.9928 - val_loss: 0.0417 - val_accuracy: 0.9908 - 30s/epoch - 1s/step
Epoch 9/50
23/23 - 30s - loss: 0.0273 - accuracy: 0.9934 - val_loss: 0.0394 - val_accuracy: 0.9912 - 30s/epoch - 1s/step
Epoch 10/50
23/23 - 30s - loss: 0.0245 - accuracy: 0.9941 - val_loss: 0.0376 - val_accuracy: 0.9919 - 30s/epoch - 1s/step
```

```
import seaborn as sns

sns.set(style='darkgrid')

plt.rcParams['figure.figsize'] = (12,6)

# Plot the learning curve.
plt.plot(loss, 'b-o', label="training loss")
plt.plot(vloss, 'r-o', label="validation loss")

# Label the plot.
plt.title("Learning curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()

plt.show()
```

```

#Testing the model
rand_sent = np.random.randint(0, x_test.shape[0]) # get a random sentence*
p = model.predict(np.array([x_test[rand_sent]]))
p = np.argmax(p, axis = -1)
y_true = np.argmax(np.array(y_test), axis = -1)[rand_sent] # get actual tags for random sentence
print("{:20}{:20}\t{}\n".format("Word", "True", "Pred"))
print("-" * 55)
Word=[]
true=[]
PRED=[]
for (w, t, pred) in zip(x_test[rand_sent], y_true, p[0]):
    print("{:20}{:20}\t{}".format(words[w - 1], tags[t], tags[pred]))
    Word.append(words[w - 1])
    true.append(tags[t])
    PRED.append(tags[pred])
DICT={'Word':Word,'True':true,'Pred':PRED}
DF=pd.DataFrame(DICT)
DF.to_csv("out_file.csv")

```

Word	True	Pred

Ttherefore	0	0
,	0	0
this	0	0
appeal	0	0
is	0	0
dismissed	0	0
for	0	0
want	0	0
of	0	0
pecuniary	0	0
jurisdiction	0	0
in	0	0
view	0	0
of	0	0
the	0	0
circular	0	0
no.21	0	0
of	0	0
2015	B-YEAR	B-YEAR
of	0	0
CIT	0	0
(	0	0
^	^	^

```

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

```

```

entities_to_remove = ["B-gpe", "I-gpe", "B-tim", "I-tim", "B-art", "I-art", "B-eve", "I-eve", "B-nat", "I-nat"]
data = data[~data.Tag.isin(entities_to_remove)]
data = data.drop(['POS'], axis=1)
data.head()

```

	Sentence #	Word	Tag
0	Sentence: 1	Thousands	O
1	Sentence: 1	of	O
2	Sentence: 1	demonstrators	O
3	Sentence: 1	have	O
4	Sentence: 1	marched	O

```

class GetSentence(object):
    def __init__(self,data):
        self.data = data
        self.n_sentences = 1
        self.empty = False
        agg_function = lambda s: [(w,t) for w,t in zip(s["Word"].values.tolist(),
                                                       s["Tag"].values.tolist())]
        self.group = self.data.groupby('Sentence #').apply(agg_function)
        self.sentence = [s for s in self.group]

```

```

getter = GetSentence(data)

```

```

sentence = [[word[0] for word in sentence] for sentence in getter.sentence]

```

```

sentence[0]

```

```

['Thousands',
 'of',
 'demonstrators',
 'have',
 'marched',
 'through',
 'London',
 'to',
 'protest',
 'the',
 'war',
 'in',
 'Iraq',
 'and',
 'demand',
 'the',
 'withdrawal',
 'of',
 'troops',
 'from',
 'that',
 'country',
 '.']

```

```

tag_values = list(set(data["Tag"].values))

```

```

tag_values

```

```

['B-per', 'I-geo', 'I-per', 'B-org', 'I-org', 'B-geo', 'O']

```

```

tag_values.append('PAD')

```

```

tag_idx = {t: i for i, t in enumerate(tag_values)}

```

```

tag_idx

```

```

{'B-per': 0,
 'I-geo': 1,
 'I-per': 2,
 'B-org': 3,
 'I-org': 4,
 'B-geo': 5,
 'O': 6,
 'PAD': 7}

```

```
import torch
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler
```

```
torch.__version__
```

```
'1.9.1'
```

```
# !pip install transformers
```

```
from transformers import BertTokenizer, BertConfig
from sklearn.model_selection import train_test_split
```

```
MAX_LEN = 75
BATCH_SIZE = 16
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
print(torch.cuda.device_count())
```

```
1
```

```
tokenizer = BertTokenizer.from_pretrained('bert-base-cased', do_lower_case = False)
```

```
Downloading: 0%|          | 0.00/213k [00:00<?, ?B/s]
```

```
Downloading: 0%|          | 0.00/29.0 [00:00<?, ?B/s]
```

```
Downloading: 0%|          | 0.00/436k [00:00<?, ?B/s]
```

```
def tokenize_preserve(sentences,text_labels):
    tokenized_sentence = []
    labels = []

    for word, label in zip(sentences,text_labels):
        tokenized_word = tokenizer.tokenize(word)
        n_subwords = len(tokenized_word)
        tokenized_sentence.extend(tokenized_word)
        labels.extend([label]*n_subwords)

    return tokenized_sentence, labels
```

```
def tokenize_preserve(sentences,text_labels):
    tokenized_sentence = []
    labels = []

    for word, label in zip(sentences,text_labels):
        tokenized_word = tokenizer.tokenize(word)
        n_subwords = len(tokenized_word)
        tokenized_sentence.extend(tokenized_word)
        labels.extend([label]*n_subwords)

    return tokenized_sentence, labels
```

```
tokenized_texts_and_labels = [
    tokenize_preserve(sent, labs)
    for sent, labs in zip(sentence, labels)
]
```

```

tokenized_texts_and_labels = [
    tokenize_preserve(sent, labs)
    for sent, labs in zip(sentence, labels)
]

tokenized_text = [token_label_pair[0] for token_label_pair in tokenized_texts_and_labels]
labels = [token_label_pair[1] for token_label_pair in tokenized_texts_and_labels]

len(tokenized_text[0]), len(labels[0])

(25, 25)

from keras_preprocessing.sequence import pad_sequences

input_ids = pad_sequences([tokenizer.convert_tokens_to_ids(txt) for txt in tokenized_text],
                           maxlen=MAX_LEN, dtype='long', value=0.0,
                           truncating='post', padding='post')

tags = pad_sequences([[tag_idx.get(1) for l in lab] for lab in labels],
                       maxlen=MAX_LEN, dtype='long', value=tag_idx['PAD'],
                       truncating='post', padding='post')

attention_masks = [[float(i != 0.0) for i in ii] for ii in input_ids]

tr_input, val_input, tr_tag, val_tag = train_test_split(input_ids, tags, random_state=45, test_size=.1)

tr_masks, val_masks, _, _ = train_test_split(attention_masks, input_ids, random_state=45, test_size=.1)

tr_input = torch.tensor(tr_input)
val_input = torch.tensor(val_input)
tr_tag = torch.tensor(tr_tag)
val_tag = torch.tensor(val_tag)
tr_masks = torch.tensor(tr_masks)
val_masks = torch.tensor(val_masks)

train_data = TensorDataset(tr_input, tr_masks, tr_tag)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=BATCH_SIZE)

valid_data = TensorDataset(tr_input, tr_masks, tr_tag)
valid_sampler = SequentialSampler(valid_data)
valid_dataloader = DataLoader(valid_data, sampler=valid_sampler, batch_size=BATCH_SIZE)

from transformers import BertForTokenClassification, AdamW

model = BertForTokenClassification.from_pretrained(
    'bert-base-cased',
    num_labels=len(tag_idx),
    output_attentions = False,
    output_hidden_states = False)

from transformers import get_linear_schedule_with_warmup

epochs = 50
max_grad_norm = 1.0

# Total number of training steps is number of batches * number of epochs.
total_steps = len(train_dataloader) * epochs

# Create the Learning rate scheduler.
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=0,
    num_training_steps=total_steps
)

```

```

#storing loss values
loss_values, validation_loss_values = [], []

#TRAINING AND VALIDATION

for _ in trange(epochs, desc= "Epoch"):
    #/\==>TRAINLOOP(ONEPASS)<==\//
    model.train()
    total_loss=0 #so it resets each epoch

    for step, batch in enumerate(train_dataloader):
        batch = tuple(t.to(device) for t in batch)
        b_input_ids, b_input_mask, b_labels = batch #also the order in train_data/val_data

        model.zero_grad() #clearing previous gradients for each epoch

        outputs = model(b_input_ids, token_type_ids=None,
                        attention_mask=b_input_mask, labels=b_labels)#forward pass

        loss = outputs[0]
        loss.backward() #getting the loss and performing backward pass

        total_loss += loss.item() #tracking loss

        torch.nn.utils.clip_grad_norm(parameters=model.parameters(), max_norm=max_grad_norm)
        #^^^ preventing exploding grads

        optimizer.step() #updates parameters

        scheduler.step() #update Learning_rate

    avg_train_loss = total_loss/len(train_dataloader)
    print('Average train loss : {}'.format(avg_train_loss))

    loss_values.append(avg_train_loss) #storing Loss values if you choose to plot Learning curve

    #/\==>VALIDATION(ONEPASS)<==\//
    model.eval()

    eval_loss, eval_accuracy = 0, 0
    nb_eval_steps, nb_eval_examples = 0, 0
    predictions, true_labels = [], []

    for batch in valid_dataloader:
        batch = tuple(t.to(device) for t in batch)
        b_input_ids, b_input_mask, b_labels = batch

```

```

        with torch.no_grad():
            outputs = model(b_input_ids, token_type_ids=None,
                            attention_mask=b_input_mask, labels=b_labels)

            logits = outputs[1].detach().cpu().numpy()
            label_ids = b_labels.to('cpu').numpy()

            eval_loss += outputs[0].mean().item()
            predictions.extend([list(p) for p in np.argmax(logits, axis=2)])
            true_labels.extend(label_ids)

        eval_loss = eval_loss / len(valid_dataloader)
        validation_loss_values.append(eval_loss)
        print('Validation loss: {}'.format(eval_loss))

    pred_tags = [tag_values[p_i] for p, l in zip(predictions, true_labels)
                 for p_i, l_i in zip(p, l) if tag_values[l_i] != 'PAD']

    valid_tags = [tag_values[l_i] for l in true_labels
                  for l_i in l if tag_values[l_i] != 'PAD']

    print('Validation Accuracy: {}'.format(accuracy_score(pred_tags, valid_tags)))
    print('Validation F-1 Score: {}'.format(f1_score([pred_tags], [valid_tags])))

```

```

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

sns.set(style='darkgrid')

plt.rcParams['figure.figsize'] = (12,6)

# Plot the Learning curve.
plt.plot(loss_values, 'b-o', label="training loss")
plt.plot(validation_loss_values, 'r-o', label="validation loss")

# Label the plot.
plt.title("Learning curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()

plt.show()

```

```
test_sentence = """Mark Elliot Zuckerberg is an American internet entrepreneur. He is known for co-founding the social media web:
```

```

tokenized_sentence = tokenizer.encode(test_sentence)
input_ids = torch.tensor([tokenized_sentence]).cuda()

```

```

with torch.no_grad():
    output = model(input_ids)
    label_indices = np.argmax(output[0].to('cpu').numpy(),axis=2)

```

```
tokens = tokenizer.convert_ids_to_tokens(input_ids.to('cpu').numpy()[0])
```

```
new_tokens, new_labels = [], []
```

```

for token, label_idx in zip(tokens, label_indices[0]):
    if token.startswith('##'):
        new_tokens[-1] = new_tokens[-1] + token[2:]
    else:
        new_labels.append(tag_values[label_idx])
        new_tokens.append(token)

```

```

names = []
locs = []
orgs = []

```

```

curr_name = ""
curr_loc = ""
curr_org = ""
for i in range(len(new_tokens)):
    if new_labels[i] != '0':
        leb = new_labels[i]
        if leb == "B-per":
            if curr_loc != "":
                locs.append(curr_loc)
                curr_loc = ""
            if curr_org != "":
                orgs.append(curr_org)
                curr_org = ""
            if curr_name != "":
                curr_name += " "
            curr_name += new_tokens[i]

```

```

if leb == "I-geo":
    if curr_name != "":
        names.append(curr_name)
        curr_name = ""
    if curr_org != "":
        orgs.append(curr_org)
        curr_org = ""
    if curr_loc != "":
        curr_loc += " "
    curr_loc += new_tokens[i]

if leb == "B-org":
    if curr_loc != "":
        locs.append(curr_loc)
        curr_loc = ""
    if curr_name != "":
        names.append(curr_name)
        curr_name = ""
    if curr_org != "":
        curr_org += " "
    curr_org += new_tokens[i]

if leb == "I-org":
    if curr_loc != "":
        locs.append(curr_loc)
        curr_loc = ""
    if curr_name != "":
        names.append(curr_name)
        curr_name = ""
    if curr_org != "":
        curr_org += " "
    curr_org += new_tokens[i]

else:
    if curr_name != "":
        names.append(curr_name)
        curr_name = ""
    if curr_loc != "":
        locs.append(curr_loc)
        curr_loc = ""
    if curr_org != "":
        orgs.append(curr_org)
        curr_org = ""

```

```

df = pd.DataFrame()
df["Free flow of Text"] = [test_sentence]
df["Extracted Name"] = [names]
df["Extracted Location"] = [locs]
df["Extracted Organization"] = [orgs]

print(df)

df.to_csv("output1.csv", index=False)

```

INTERNSHIP CERTIFICATE

Date: 27/05/2022

TO WHOMSOEVER IT MAY CONCERN

This is to certify that Rishabh Singhal has done his Research project at JK Lakshmipat University, Jaipur, from 04/01/2022 to 08/05/2022 for the duration of 4 months.

He has worked on a project titled NER on Legal Dataset. This project was aimed at which models works best for NER on legal data. As part of the project, the paper submitted by Rishabh Singhal has been accepted for presentation at JURISIN 2022, a well-known conference in the area of legal informatics.

During his internship he has demonstrated his skills with self-motivation to learn new skills. His performance exceeded my expectations, and he was able to complete the project on time.

I wish him all the best for his upcoming career.



Dr. Kshitiz Verma

Assistant Professor

Department of Computer Science & Engineering

JK Lakshmipat University

Institute of Engineering and Technology
JK Lakshmipat University
JAIPUR (Rajasthan)

INTERNSHIP CERTIFICATE

Date: 27/05/2022

TO WHOMSOEVER IT MAY CONCERN

This is to certify that Sanyukta Tanwar has done her Research project at JK Lakshmipat University, Jaipur, from 04/01/2022 to 08/05/2022 for the duration of 4 months.

She has worked on a project titled NER on Legal Dataset. This project was aimed at which models works best for NER on legal data. As part of the project, the paper submitted by Sanyukta Tanwar has been accepted for presentation at JURISIN 2022, a well-known conference in the area of legal informatics.

During her internship she has demonstrated her skills with self-motivation to learn new skills. Her performance exceeded my expectations, and she was able to complete the project on time.

I wish her all the best for her upcoming career.



Dr. Kshitiz Verma

Assistant Professor

Department of Computer Science & Engineering

JK Lakshmipat University

