LAWNICS

Documentation

Lawnics: Search Engine

Overview:

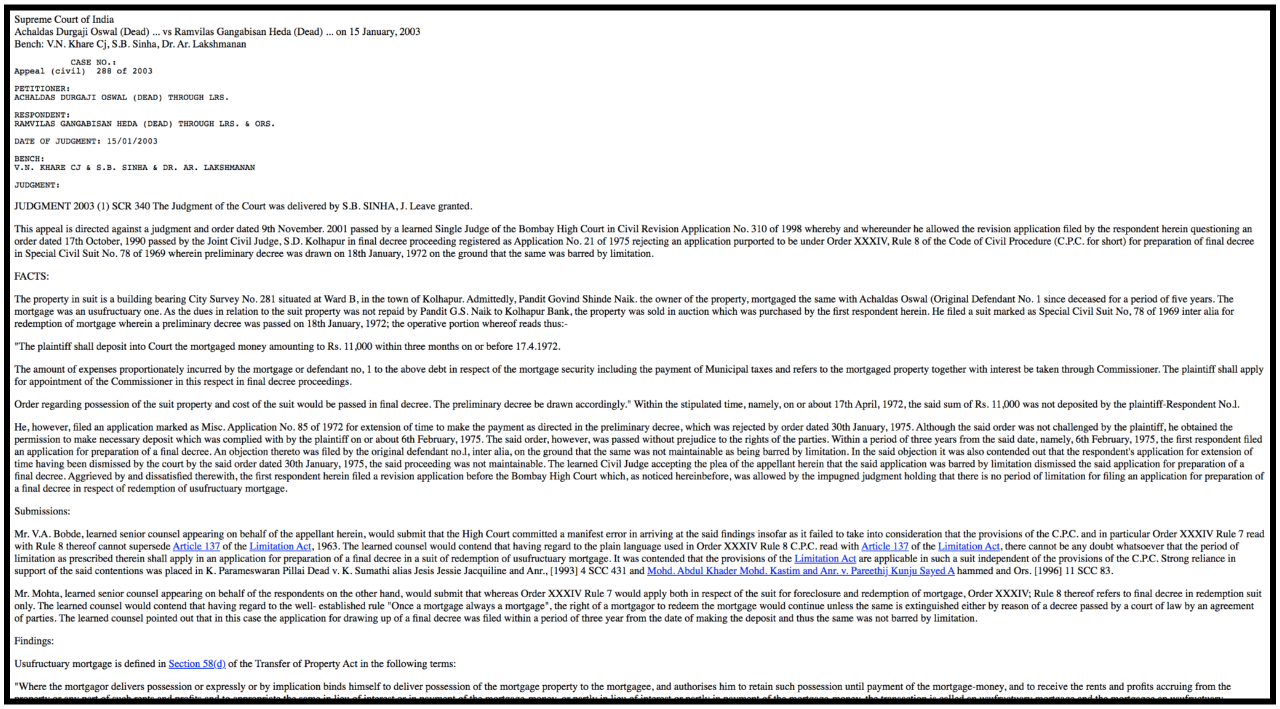
Lawnics helps the next generation lawyers by transforming their work over digital workspace by building AI assistants that help lawyers, organizations, government and Judiciary to make their services faster, smoother and cost effective. Consequently, transforming 300 year old work practices for the digital age. Lawnics are a cloud based legal operating software that leverages artificial intelligence, intuitive designs, and smart processes to make legal practice more meaningful.

Getting started:

The legal data is available in the form of HTML and JSON files. Cleaning the data as required and extracting proper entities is the first task. In this document, the term *entity/entities* will be referred to as *Acts, Articles, Sections, Cases, and Citations*. The term, *tag* and *label* will be interchangeably used for the 'B-'(beginning) and 'I-'(inside) tags as required by the BERT model. For more details on the BERT model and the required input format, refer to these links:

1. Original paper
2. NER with BERT in Spark-NLP
3. Data preparation in CoNLL format

Here is an example of how the raw data(HTML page) looks:



Supreme Court of India
Achaldas Durgaji Oswal (Dead) ... vs Ramvilas Gangabisan Heda (Dead) ... on 15 January, 2003
Bench: V.N. Khare Cj, S.B. Sinha, Dr. Ar. Lakshmanan

```
           CASE NO.:
Appeal (civil)  288 of 2003

PETITIONER:
ACHALDAS DURGAJI OSWAL (DEAD) THROUGH LRS.

RESPONDENT:
RAMVILAS GANGABISAN HEDA (DEAD) THROUGH LRS. & ORS.

DATE OF JUDGMENT: 15/01/2003

BENCH:
V.N. KHARE CJ & S.B. SINHA & DR. AR. LAKSHMANAN

JUDGMENT:
```

JUDGMENT 2003 (1) SCR 340 The Judgment of the Court was delivered by S.B. SINHA, J. Leave granted.

This appeal is directed against a judgment and order dated 9th November. 2001 passed by a learned Single Judge of the Bombay High Court in Civil Revision Application No. 310 of 1998 whereby and whereunder he allowed the revision application filed by the respondent herein questioning an order dated 17th October, 1990 passed by the Joint Civil Judge, S.D. Kolhapur in final decree proceeding registered as Application No. 21 of 1975 rejecting an application purported to be under Order XXXIV, Rule 8 of the Code of Civil Procedure (C.P.C. for short) for preparation of final decree in Special Civil Suit No. 78 of 1969 wherein preliminary decree was drawn on 18th January, 1972 on the ground that the same was barred by limitation.

FACTS:

The property in suit is a building bearing City Survey No. 281 situated at Ward B, in the town of Kolhapur. Admittedly, Pandit Govind Shinde Naik. the owner of the property, mortgaged the same with Achaldas Oswal (Original Defendant No. 1 since deceased for a period of five years. The mortgage was an usufructuary one. As the dues in relation to the suit property was not repaid by Pandit G.S. Naik to Kolhapur Bank, the property was sold in auction which was purchased by the first respondent herein. He filed a suit marked as Special Civil Suit No, 78 of 1969 inter alia for redemption of mortgage wherein a preliminary decree was passed on 18th January, 1972; the operative portion whereof reads thus:-

"The plaintiff shall deposit into Court the mortgaged money amounting to Rs. 11,000 within three months on or before 17.4.1972.

The amount of expenses proportionately incurred by the mortgage or defendant no, 1 to the above debt in respect of the mortgage security including the payment of Municipal taxes and refers to the mortgaged property together with interest be taken through Commissioner. The plaintiff shall apply for appointment of the Commissioner in this respect in final decree proceedings.

Order regarding possession of the suit property and cost of the suit would be passed in final decree. The preliminary decree be drawn accordingly." Within the stipulated time, namely, on or about 17th April, 1972, the said sum of Rs. 11,000 was not deposited by the plaintiff-Respondent No.l.

He, however, filed an application marked as Misc. Application No. 85 of 1972 for extension of time to make the payment as directed in the preliminary decree, which was rejected by order dated 30th January, 1975. Although the said order was not challenged by the plaintiff, he obtained the permission to make necessary deposit which was complied with by the plaintiff on or about 6th February, 1975. The said order, however, was passed without prejudice to the rights of the parties. Within a period of three years from the said date, namely, 6th February, 1975, the first respondent filed an application for preparation of a final decree. An objection thereto was filed by the original defendant no.l, inter alia, on the ground that the same was not maintainable as being barred by limitation. In the said objection it was also contended out that the respondent's application for extension of time having been dismissed by the court by the said order dated 30th January, 1975, the said proceeding was not maintainable. The learned Civil Judge accepting the plea of the appellant herein that the said application was barred by limitation dismissed the said application for preparation of a final decree. Aggrieved by and dissatisfied therewith, the first respondent herein filed a revision application before the Bombay High Court which, as noticed hereinbefore, was allowed by the impugned judgment holding that there is no period of limitation for filing an application for preparation of a final decree in respect of redemption of usufructuary mortgage.

Submissions:

Mr. V.A. Bobde, learned senior counsel appearing on behalf of the appellant herein, would submit that the High Court committed a manifest error in arriving at the said findings insofar as it failed to take into consideration that the provisions of the C.P.C. and in particular Order XXXIV Rule 7 read with Rule 8 thereof cannot supersede Article 137 of the Limitation Act, 1963. The learned counsel would contend that having regard to the plain language used in Order XXXIV Rule 8 C.P.C. read with Article 137 of the Limitation Act, there cannot be any doubt whatsoever that the period of limitation as prescribed therein shall apply in an application for preparation of a final decree in a suit of redemption of usufructuary mortgage. It was contended that the provisions of the Limitation Act are applicable in such a suit independent of the provisions of the C.P.C. Strong reliance in support of the said contentions was placed in K. Parameswaran Pillai Dead v. K. Sumathi alias Jesis Jessie Jacquiline and Anr., [1993] 4 SCC 431 and Mohd. Abdul Khader Mohd. Kastim and Anr. v. Pareethij Kunju Sayed A hammed and Ors. [1996] 11 SCC 83.

Mr. Mohta, learned senior counsel appearing on behalf of the respondents on the other hand, would submit that whereas Order XXXIV Rule 7 would apply both in respect of the suit for foreclosure and redemption of mortgage, Order XXXIV; Rule 8 thereof refers to final decree in redemption suit only. The learned counsel would contend that having regard to the well- established rule "Once a mortgage always a mortgage", the right of a mortgagor to redeem the mortgage would continue unless the same is extinguished either by reason of a decree passed by a court of law by an agreement of parties. The learned counsel pointed out that in this case the application for drawing up of a final decree was filed within a period of three year from the date of making the deposit and thus the same was not barred by limitation.

Findings:

Usufructuary mortgage is defined in Section 58(d) of the Transfer of Property Act in the following terms:

"Where the mortgagor delivers possession or expressly or by implication binds himself to deliver possession of the mortgage property to the mortgagee, and authorises him to retain such possession until payment of the mortgage-money, and to receive the rents and profits accruing from the

Figure 1.0 : Sample HTML file

Requirements:

1. Python 3.7.4
2. Jupyter notebook/google collab

   Main Packages:

   1. *BeautifulSoup* (for HTML handling)
   2. *Pandas* (for data manipulation)
   3. *nltk* (for tokenizing document)
   4. *re* (for regex and matching patterns)
   5. *json* (for annotated file conversion to CoNLL)

Foundation of Web Scraping:

The hyperlinks in the HTML file allowed for extracting the required entities.

1. As you can see in figure 1.0, the hyper linked entities are: *Article 137, Limitation Act, Mohd. Abdul Khader Mohd. Kastim and Anr. v. Pareethij Kunju Sayed A, Section 58(d).*
2. The raw data from the HTML was cleaned and special characters, punctuations, and any other characters that were not classified as words were also removed.
3. The tags were extracted using the BeautifulSoup package, mentioned in '*Requirements',* by extracting all the paragraph('p') tags and associated 'a' tags for the links. A python dictionary would then be formed consisting of all the entities:

    {'Article 137': 'O',

    'Limitation Act': 'O',

    'Mohd. Abdul Khader Mohd. Kastim and Anr. v. Pareethij Kunju Sayed A': 'O',

    'Section 58(d)': 'O',

    And other entities in the HTML in a similar way…….. }
4. After this, a python dictionary of entities was created by tagging the entities appropriately using keywords. For example, any *Article* entity always has the word 'Article' in it, the same with *Sections* and *Acts*. *Citations* and *Cases* were handled differently. For cases, the keyword "vs." or "v" was used to find the *Case* entity. For citations, regex was used to identify each type of citation from the Journal: SCC, AIR, SCR, Crime, ITR, ACR, SCALE, CS, ELT, ILR, JT. The resultant dictionary had the entity name as the key and the start index(of that entity in the original raw data), end index(of that entity in the original raw data), and the type of entity as the value. For example:

   a. {'citations': [[23263, 23280, 'Cite'], [34262, 34277, 'Cite'], [34280, 34297, 'Cite']]}

5. Next, using the start and end index of a particular entity in the original raw data, the first word of the entity was tagged as “B-(name of entity)” tag and the rest of the entity words till the end index were tagged as “I-(name of entity)” tags. The non-entity words were tagged as ‘O’ (other meaning) tags.
6. This was then converted to a dataframe with two columns: token name and respective tags.

This was the foundation of the data scraping model.

For BERT training, POS tags were also added in the source training file using NER.

The resultant data-frame was then converted to tsv format with approximately size of 25MB outside of the data scraping code in google collab (refer to the links in ‘*Getting Started’*). This was the final training dataset used for version 1.0.

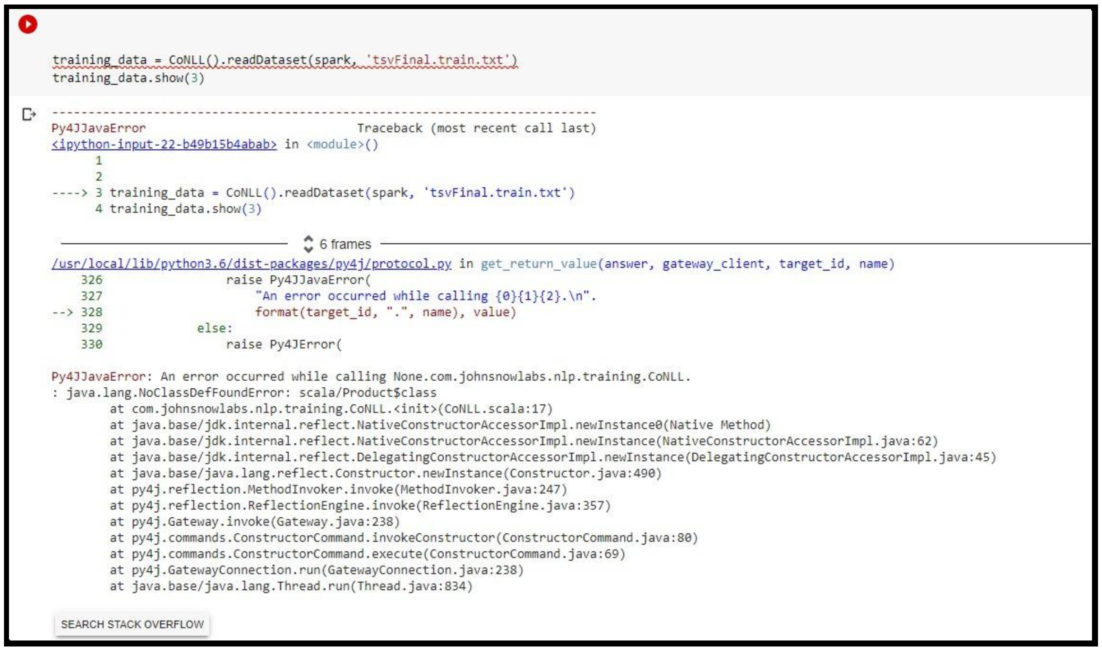
Report of the first BERT training 1.0:

1. Google collab was used for training since spark package requirements failed to work on local systems.

   Requirements for training and testing:

   1. java-8
   2. pyspark 2.4.4
   3. spark-nlp-2.7.4
   4. sklearn
   5. pre-trained variant of BERT: *‘bert_base_cased’*

2. While training, the training dataset was not being read by the ‘CoNLL’ method of the ‘sparknlp.training’ package. The error was as mentioned in figure 1.1. This issue was raised on GitHub.

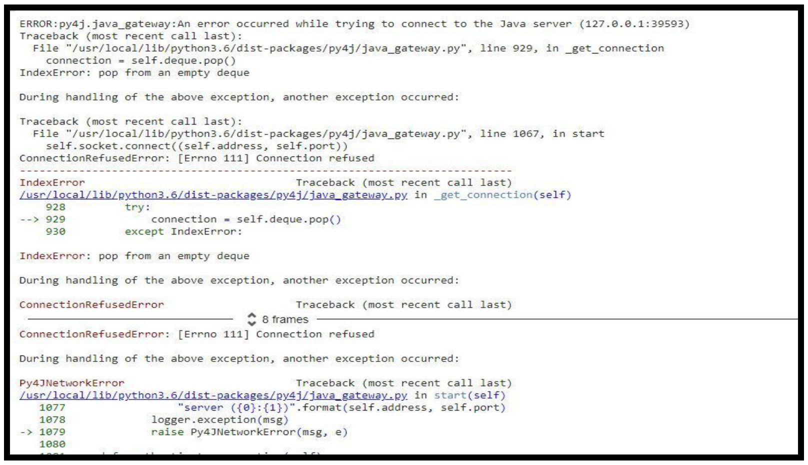


```
training_data = CoNLL().readDataset(spark, 'tsvFinal.train.txt')
training_data.show(3)
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
<ipython-input-22-b49b15b4abab> in <module>()
      1
      2
----> 3 training_data = CoNLL().readDataset(spark, 'tsvFinal.train.txt')
      4 training_data.show(3)

6 frames
/usr/local/lib/python3.6/dist-packages/py4j/protocol.py in get_return_value(answer, gateway_client, target_id, name)
    326                 raise Py4JJavaError(
    327                     "An error occurred while calling {0}{1}{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:
    330                 raise Py4JError(

Py4JJavaError: An error occurred while calling None.com.johnsnowlabs.nlp.training.CoNLL.
: java.lang.NoClassDefFoundError: scala/Product$class
	at com.johnsnowlabs.nlp.training.CoNLL.<init>(CoNLL.scala:17)
	at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
	at java.base/jdk.internal.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
	at java.base/jdk.internal.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
	at java.base/java.lang.reflect.Constructor.newInstance(Constructor.java:490)
	at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:247)
	at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
	at py4j.Gateway.invoke(Gateway.java:238)
	at py4j.commands.ConstructorCommand.invokeConstructor(ConstructorCommand.java:80)
	at py4j.commands.ConstructorCommand.execute(ConstructorCommand.java:69)
	at py4j.GatewayConnection.run(GatewayConnection.java:238)
	at java.base/java.lang.Thread.run(Thread.java:834)
```

Figure 1.1 : Incorrect Input Error

3. This issue was resolved by updating the input dataset and correcting minor errors according to the CoNLL 2003 format using this link.
4. After the training data was in the required CoNLL format, a RAM issue was encountered in the same ‘CoNLL’ method of the ‘sparknlp.training’ package. The error was as mentioned in Figure 1.2.



```
ERROR:py4j.java_gateway:An error occurred while trying to connect to the Java server (127.0.0.1:39593)
Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/py4j/java_gateway.py", line 929, in _get_connection
    connection = self.deque.pop()
IndexError: pop from an empty deque

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/py4j/java_gateway.py", line 1067, in start
    self.socket.connect((self.address, self.port))
ConnectionRefusedError: [Errno 111] Connection refused
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/py4j/java_gateway.py in _get_connection(self)
    928         try:
--> 929             connection = self.deque.pop()
    930         except IndexError:

IndexError: pop from an empty deque

During handling of the above exception, another exception occurred:

ConnectionRefusedError                    Traceback (most recent call last)
                                8 frames
ConnectionRefusedError: [Errno 111] Connection refused

During handling of the above exception, another exception occurred:

Py4JNetworkError                          Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/py4j/java_gateway.py in start(self)
   1077                 "server ({0}:{1})".format(self.address, self.port)
   1078             logger.exception(msg)
-> 1079             raise Py4JNetworkError(msg, e)
   1080
```

Figure 1.2 : RAM issue

5. This issue was then resolved by reducing the input data size and upgrading RAM in collab from 12GB to 25GB. Optimal file size was found to be a dataset of 40 case files.
6. Class imbalances were found as well with "O-" being a lot more than the entity tags.
7. This class imbalance issue was resolved after extracting paragraphs with entities and discarding the ones with no entities(hyperlinks). Each sentence in CoNLL is one para.
8. Pipeline fitting took approximately 45 minutes:

   ner_model = ner_pipeline.fit(training_data)
9. Model was saved in collab and later downloaded as separate files in the local system to save time and effort in pipeline fitting.
10.
```
ner_prediction_pipeline = Pipeline(
  stages = [
        document,
        sentence,
        token,
        bert,
        loaded_ner_model,
        converter])
```

11. Model was very less accurate and was not able to detect many of the entities. This was tried manually by giving some text input in the Light Pipeline to check for prediction.

    ner_pipelineFit = ner_prediction_pipeline.fit(empty_data)

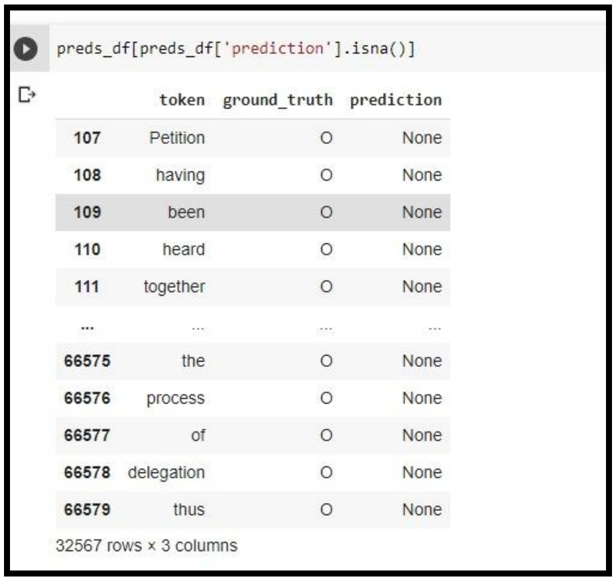
    ner_lp_pipeline = LightPipeline(ner_pipelineFit)

    parsed = ner_lp_pipeline.annotate(txt)

    The *'txt'* is the text file or string passed to check for prediction.

12. During testing, due to null value prediction as seen in figure 1.3, the classification report could not be generated.

13. In figure 1.4, the classification report is shown after removing the null values.

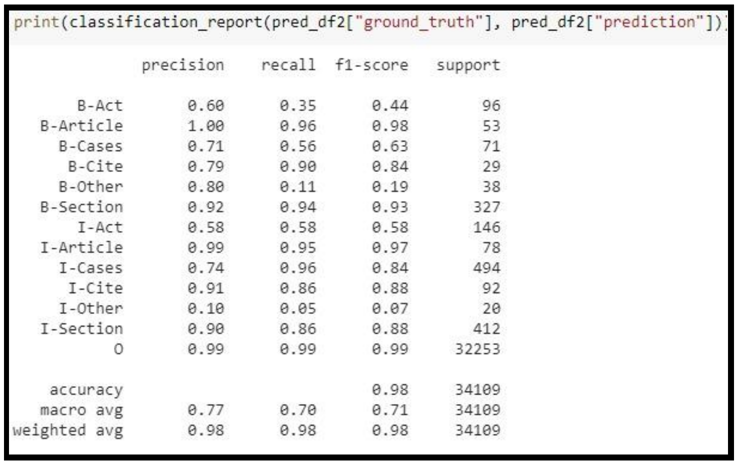


```
preds_df[preds_df['prediction'].isna()]
```

	token	ground_truth	prediction
107	Petition	O	None
108	having	O	None
109	been	O	None
110	heard	O	None
111	together	O	None
...	...	...	...
66575	the	O	None
66576	process	O	None
66577	of	O	None
66578	delegation	O	None
66579	thus	O	None

32567 rows × 3 columns

Figure 1.3



```
print(classification_report(pred_df2["ground_truth"], pred_df2["prediction"]))

              precision    recall  f1-score   support

       B-Act       0.60      0.35      0.44        96
   B-Article       1.00      0.96      0.98        53
     B-Cases       0.71      0.56      0.63        71
      B-Cite       0.79      0.90      0.84        29
     B-Other       0.80      0.11      0.19        38
   B-Section       0.92      0.94      0.93       327
       I-Act       0.58      0.58      0.58       146
   I-Article       0.99      0.95      0.97        78
     I-Cases       0.74      0.96      0.84       494
      I-Cite       0.91      0.86      0.88        92
     I-Other       0.10      0.05      0.07        20
   I-Section       0.90      0.86      0.88       412
           O       0.99      0.99      0.99     32253

    accuracy                           0.98     34109
   macro avg       0.77      0.70      0.71     34109
weighted avg       0.98      0.98      0.98     34109
```

Figure 1.4 : Classification report

Report of BERT training 2.0:

1. Google collab was used for training here as well.
2. Same requirements as BERT training 1.0.
3. The problem in the earlier version was due to input data:
    a. Null values were due to larger sentences which confused the BERT model.
    b. Punctuations were removed due to which accuracy decreased when tested on real unclean data with punctuations.
4. Changes in approach:
    a. Changes were made to document cleaning.
    b. Changes were made to create_entities() method.
    c. Punctuations were left in after cleaning.
    d. For entity extraction, it should be extracted with its respective punctuation. For example: In figure 1.0, 'Limitation Act, 1963' is hyperlinked till the word 'Act'. By version 1.0, 1963 would not be tagged as a part of the Act even though it was.

For this issue, pattern matching was added for Acts and Citations. The new entity tags for *‘Limitation Act, 1963’* and *‘(2015) 7 SCC 58’* were:

i. [‘Limitation’, ‘B-Act’] , [‘Act’, ‘I-Act’], [‘,’, ‘I-Act’], [‘1963’, ‘I-Act’]

ii. [‘(’, ‘B-Cite’], [‘2015’, ‘I-Cite’], [‘)’, ‘I-Cite’], [‘7’, ‘I-Cite’], [‘SCC’, ‘I-Cite’] , [‘58’, ‘I-Cite’]

iii. The same format was followed with sections and other entities.

e. Sentences were split using:

i. *‘PunktSentenceTokenizer’* and *‘PunktTrainer’* of the nltk package.

ii. For sentence splitting, the sentence should not start with any abbreviation words and not split any collocations. Due to this, the *PunktSentenceTokenizer* was updated with some common legal words and abbreviations.

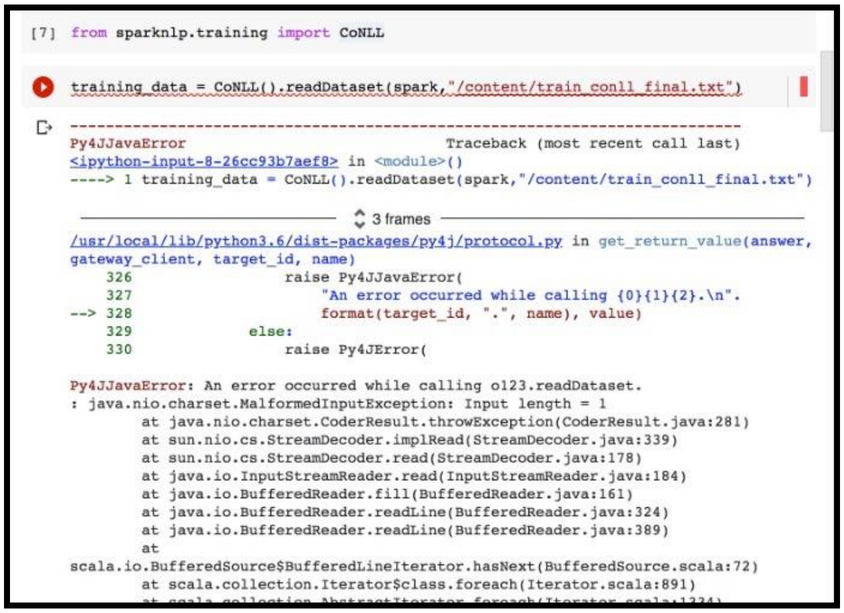
5. Data was generated in the CoNLL format in the original script itself instead of exporting it as a csv and then converting it to tsv and finally to CoNLL.
6. Error occurred during loading the input files in the ‘CoNLL’ method of the ‘sparknlp.training’ package.
7. This error was tried by, first, reducing the input data. This did not work. Upon further investigation, the cause of the error was found as special characters.
8. Unix commands were used to remove the special characters found in the CoNLL file. FYI: Only special characters(Non-ASCII characters) were removed which caused errors in the ‘CoNLL’ method of the ‘sparknlp.training’ package during loading input with punctuations being intact.

9. The following unix commands were used:

    a. bash-3.2$ LC_ALL=C tr -dc '\0-\177' <hasSpecialChars.txt>tempFile.txt

       bash-3.2$ tr -cd '\11\12\15\40-\176'<tempFile.txt>doesNotHaveSpecialChars.txt

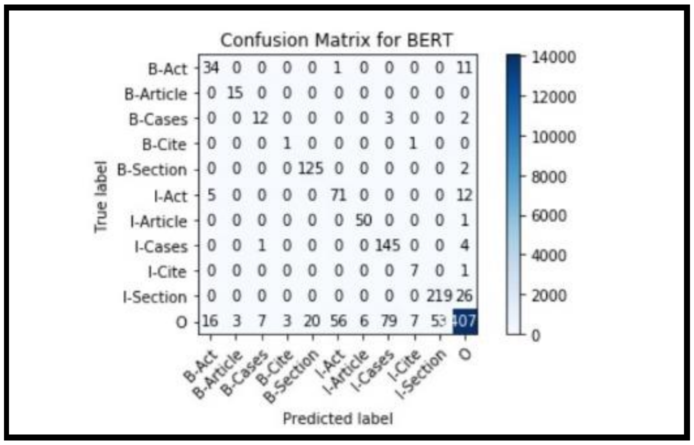
10. After the above changes, the input was accepted in the CoNLL method.
11. There was an error during pipeline fitting as shown in figure 2.0.



```
[7] from sparknlp.training import CoNLL

training_data = CoNLL().readDataset(spark,"/content/train_conll_final.txt")

---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
<ipython-input-8-26cc93b7aef8> in <module>()
----> 1 training_data = CoNLL().readDataset(spark,"/content/train_conll_final.txt")

                              3 frames
/usr/local/lib/python3.6/dist-packages/py4j/protocol.py in get_return_value(answer,
gateway_client, target_id, name)
    326                 raise Py4JJavaError(
    327                     "An error occurred while calling {0}{1}{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:
    330                 raise Py4JError(

Py4JJavaError: An error occurred while calling o123.readDataset.
: java.nio.charset.MalformedInputException: Input length = 1
	at java.nio.charset.CoderResult.throwException(CoderResult.java:281)
	at sun.nio.cs.StreamDecoder.implRead(StreamDecoder.java:339)
	at sun.nio.cs.StreamDecoder.read(StreamDecoder.java:178)
	at java.io.InputStreamReader.read(InputStreamReader.java:184)
	at java.io.BufferedReader.fill(BufferedReader.java:161)
	at java.io.BufferedReader.readLine(BufferedReader.java:324)
	at java.io.BufferedReader.readLine(BufferedReader.java:389)
	at
scala.io.BufferedSource$BufferedLineIterator.hasNext(BufferedSource.scala:72)
	at scala.collection.Iterator$class.foreach(Iterator.scala:891)
```

Figure 2.0 : Memory error

12. This was resolved by reducing the input size to 150 files.
13. Pipeline was fit and the model was trained.
14. Null value predictions still persisted.
15. In figure 2.1, the confusion matrix is shown for this version.



	B-Act	B-Article	B-Cases	B-Cite	B-Section	I-Act	I-Article	I-Cases	I-Cite	I-Section	O
B-Act	34	0	0	0	0	1	0	0	0	0	11
B-Article	0	15	0	0	0	0	0	0	0	0	0
B-Cases	0	0	12	0	0	0	0	3	0	0	2
B-Cite	0	0	0	1	0	0	0	0	1	0	0
B-Section	0	0	0	0	125	0	0	0	0	0	2
I-Act	5	0	0	0	0	71	0	0	0	0	12
I-Article	0	0	0	0	0	0	50	0	0	0	1
I-Cases	0	0	1	0	0	0	0	145	0	0	4
I-Cite	0	0	0	0	0	0	0	0	7	0	1
I-Section	0	0	0	0	0	0	0	0	0	219	26
O	16	3	7	3	20	56	6	79	7	53	407

Figure 2.1 : Confusion Matrix

Report of BERT training 3.0 (IN PROGRESS):

Issues:

1. The sentence splitting was not appropriate and created longer than 110 words sentences which confused BERT while training.
2. The rule based annotations were not efficiently tagging all the entities in the file since some of them were not hyperlinked.
3. When converting from JSON to CoNLL, there exists an issue of labels being converted appropriately.

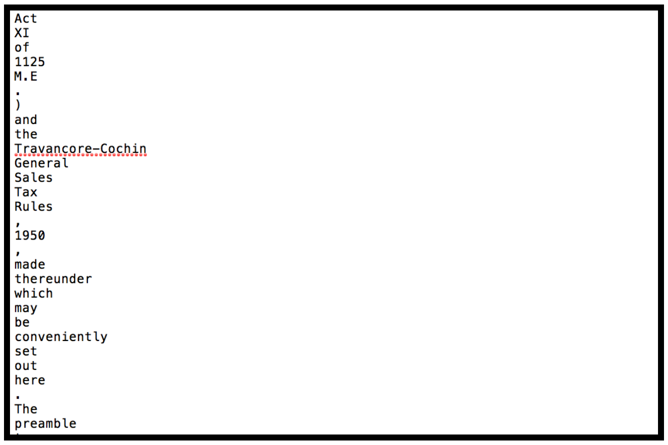
Resolve:

1. For the first issue, sentence segmentation models are being tried and tested. (In progress)
2. For the second issue, the Doccano annotation tool is being used.

About Doccano:

1. The input should be a CoNLL file with no pre-existing tags since it creates overlapping.
2. Doccano does not accept special characters.

3. Uploading CoNLL files was done in batches. Some files in batches were not uploaded since there were special characters in them. Figure 3.0 shows the input format.
4. Some special characters had to be replaced manually and most of them were removed using the unix commands mentioned in version 2.0 section 9.
5. The output of the annotated file is JSON-L and JSON-text tables.
6. Python script was written to convert JSON to CoNLL format(accepted by BERT).



```
Act
XI
of
1125
M.E
.
)
and
the
Travancore-Cochin
General
Sales
Tax
Rules
,
1950
,
made
thereunder
which
may
be
conveniently
set
out
here
.
The
preamble
```

Figure 3.0 : with each token on a different line