GSoC `20 PROPOSAL

ORGANISATION

ACCORD PROJECT

PROJECT NAME

NATURAL LANGUAGE LEGAL CLAUSE DETECTION

By **FREYA MEHTA**

PERSONAL INFORMATION

1. Name and Contact Information

- Name: Freya Mehta
- Email Address: freyamehta03@gmail.com
- Github ID: https://github.com/freyamehta99
- LinkedIn: https://www.linkedin.com/in/freya-mehta/
- Contact Number: +919664701912
- Accord Project Slack handle - Freya Mehta

2. Education

- Institute: International Institute of Information Technology, Hyderabad
- Field of Study: Computer Science and Engineering (Bachelors of Technology) with Honors in Machine Translation and Natural Language Processing
- Location & TimeZone: Telangana, India UTC + 5:30

3. Academic Interests and Why is it that I am interested in the Accord Project?

I am currently pursuing my B.Tech in Computer Science and Engineering + Honors Research in Computational Linguistics at IIIT-Hyderabad, India. A good portion of our academic focus is on Artificial Intelligence and additionally my honors domain is on Machine Translation and Natural Language Processing which I find a really interesting area to work on. So working with Accord Project for this particular project of Legal Clause Detection will help me nurture my Natural Language Processing skills as well as give me a chance to give back to the community with some solid contribution.

I've worked with C++ and Python intensely in a lot of projects and I take keen interest in deep learning as well. I usually love building fun models/tools. The details of my work experience can be found in my resume.

4. Non-Summer of Code Plans

I have my college vacations during the months of Google Summer of Code, I would be able to devote around 40 woman hours a week. I had a vacation plan for a week that lies in between but due to the current COVID-19 scenario I might have to cancel it so I have distributed the work accordingly in the timeline. If not selected I plan to complete my research on sentiment analysis and entity extraction as part of my honors.

DETAILED PROJECT PLAN AND WORKFLOW

1. Project Title

Natural Language Legal Clause Detection

2. Project Mentors

Dan Selman and Adriaan Pelzer

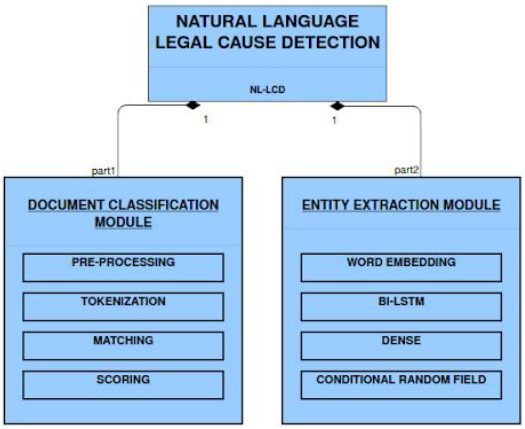
3. Project Goal

The goal of the project is to develop a tool which can detect instances of a specific type of clauses and help users replace the natural language part in their input contract with the instance of Accord Project template clause. Users can easily convert their existing legal documents into professional executable Accord Project contract/clauses by using this tool which can identify the value of Accord Project template text automatically.

4. Abstract

Automating contract classification and extractions is a hard task as it is not very straightforward - it is about both identification and localisation that is to identify and classify legal terms and clauses.

This project would primarily be divided into two main parts:



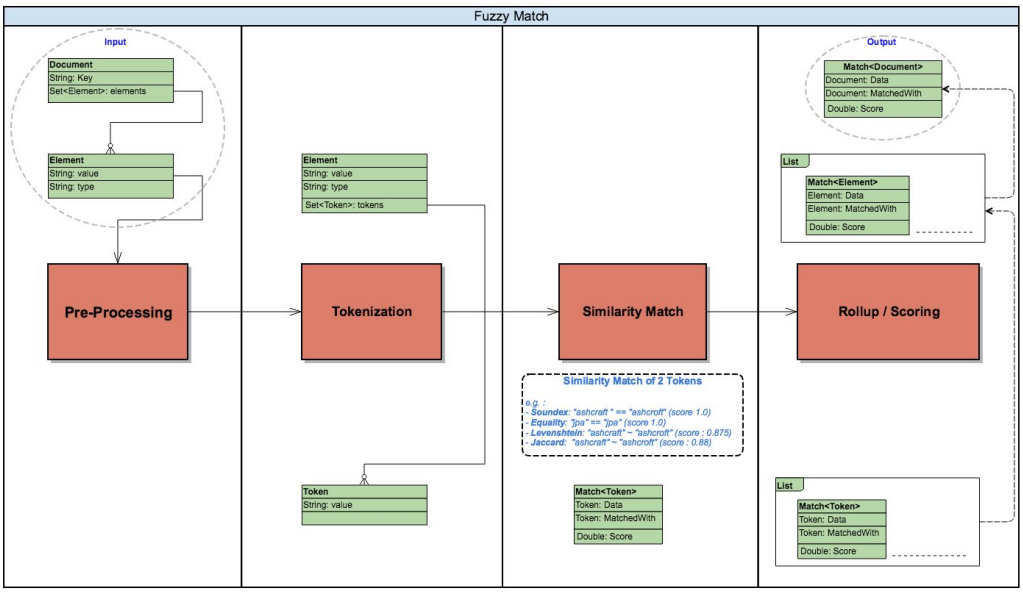
1. **Document Classification Module** : The model maintains a database of unique annotated documents. It takes a new document as input and predicts the annotation for the new document if such a template exists in our database. This part of the model will use a Fuzz-like match algorithm for text similarity. The fuzz-like match algorithm will help us match and categorize documents with similar templates. Automating contract classification and extractions is a hard task as it is not very straightforward - it is about both identification and localisation that is to identify and classify legal terms and clauses.

2. **Entity Extraction Module** : Every Accord Project template text has specific instances whose values the model will be extracting from the input document. After classifying the type of clause/contract, the model would know what instances it has to extract the values of. This part of the model will use Bidirectional Long Short Term Memory (BiLSTM) which would operate on word embeddings and POS tag.

5. Detailed Workflow

Document Classification Module

Four Stages of Fuzz-like match algorithm :-



1) Pre-Processing:

We write a function which will transform the document text from one string to another like follows:

- ❏ Lower Case - Converts all characters to lowercase
- ❏ Trim - Removes leading and trailing spaces
- ❏ Eliminate Special Chars - Removes all characters (like (,), “, etc) except alpha and numeric characters and spaces
- ❏ Eliminate Duplicate words - Remove overlapping words and keep only unique words.

2) Tokenization:

We write a function which will break down each sentence of the document into a stream of token objects for comparison.

- ❏ Word : Breaks down an element into words (anything delimited by space " "). e.g. "delivery and acceptance processes" -> ["delivery" "and" "acceptance" "processes"]

3) Matching:

We write a function which gives us a match probability between two tokens on a scale of 0 to 1.

- Levenshtein: Gets the Levenshtein distance score using apache commons similarity library

  ```
  Token left = new Token("Delivery");
  Token right = new Token("Deliver");
  double result = MatchFunction.levenshtein().apply(left, right);
  Result -> 0.8
  ```

- Soundex: Compares 2 token strings soundex values. Uses apache commons codec library to get soundex values. (Binary result)

  ```
  Token left = new Token("Delivery");
  Token right = new Token("Deliver");
  double result = MatchFunction.soundex().apply(left, right);
  Result -> 1.0
  ```

  ```
  Token left = new Token("minneapolis");
  Token right = new Token("minnesota");
  double result = MatchFunction.soundex().apply(left, right);
  Result -> 0.0
  ```

- Equality: Does a string equals of 2 token strings. reduces the complexity by not performing matches against the elements which have a very low probability to match by creating "Search Groups". (Binary result)

  ```
  Token left = new Token("Delivery");
  Token right = new Token("delivery");
  double result = MatchFunction.equality().apply(left, right);
  Result -> 1.0
  ```

  ```
  Token left = new Token("Delivery");
  Token right = new Token("Deliver");
  double result = MatchFunction.equality().apply(left, right);
  Result -> 0.0
  ```

- Jaccard: This function calculates jaccard distance (Intersection over Union**)** gets the Jaccard score using apache commons similarity library

  ```
  Token left = new Token("Delivery");
  Token right = new Token("delivery");
  double result = MatchFunction.jaccard().apply(left, right);
  Result -> 0.75
  ```

- JaroWinkler: This function calculates Jaro-Winkler distance (Intersection over Union**)** gets the Jaccard score using apache commons similarity library

```
Token left = new Token("Delivery");
Token right = new Token("delivery");
double result = MatchFunction.jarowinkler().apply(left, right);
Result -> 0.96
```

4) **Scoring:**

We write a function which calculates the score of the document match using exponential weighted average. This combines the benefits of weights and exponential increase when calculating the average scores. We perform this matching and based on the score we match a document with multiple existing lists of documents.

`matchService.ApplyMatchFunction(document Document, list<document> MatchedWith)`

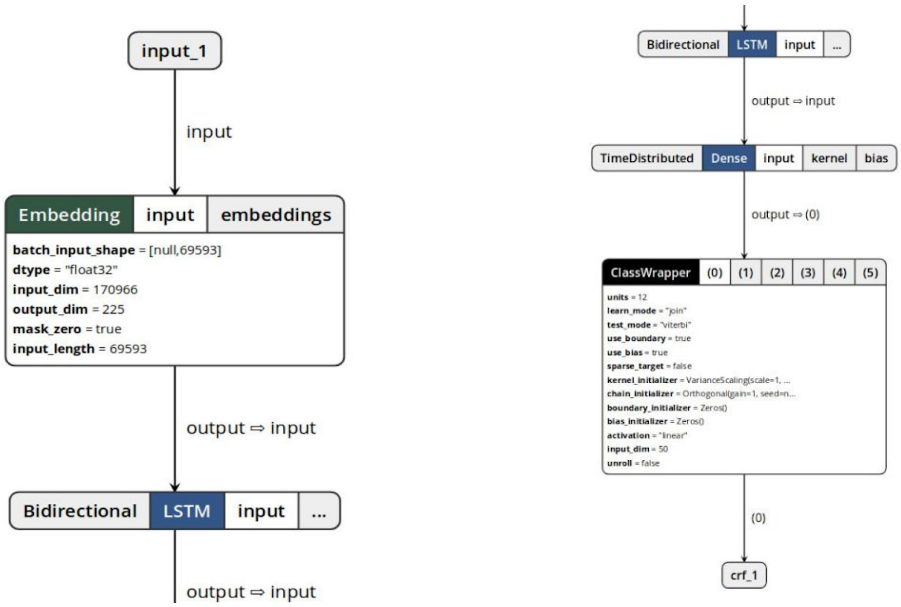
Based on the annotated list of matched documents, we classify the type of clause/contract of the input document. We make a probability distribution of the score of the document matched and classification tag. The tag with highest probability is selected.

Entity Extraction Module

We perform the entity extraction sub-model on the input document for extracting the values of the instances of the type of classified clause/contract. The model uses pre-trained word embeddings and POS tag embeddings which can be obtained by applying WORD2VEC (skip-gram model) to training contracts. The core dataset I need should contain contracts annotated with clause headings such that our model can learn to identify them. The model requires annotations of contract elements for example Contracting Parties (like Buyer, Supplier and Seller), Amount, StartDate, EffectiveDate, TerminationDate, ContractPeriod, LoanDuration, LoanAmount, etc. The entity list for each contract/clause can be found (scraped) from the contract data elements of Accord Project Template Library grammer. After gathering and preprocessing this data, we develop the model. After experimenting with several algorithms like linear SVMs and manually written post-processing rules, I chose a deep learning model which would use bidirectional Long Short Term Memory (BiLSTM) which would operate on word embeddings and POS tag. I have implemented a BiLSTM on a small experimental labelled contract dataset which you can find here and found results that outperform the hybrid of linear SVM and manual post-processing rules. The link to this experiment implementation is https://github.com/freyamehta99/Element-Extraction-using-BiLSTM-RNN.

Since the model is too huge to upload on github, I have uploaded it on this gdrive.

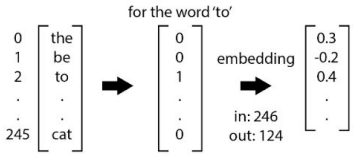
Four Stages of this sub-module (Excluding input):-



If this image is not clear, you can refer to this.

1) **Embedding Layer:**

Depending on the task the word embedding for a certain word varies. For us the word 'act' has legal meaning but for other use cases it becomes a verb. An embedding layer works as a mapping f:X→y *where X is our vocabulary and y is 225(e.g.) dimensional embeddings*. So when the embedding layer receives a word, it maps it to its specific embedding and passes the embedding to the next layer.



In the example above the embedding layer receives an input for the word to in the form of an one hot encoding of length 246(e.g. i.e. vocabulary size) and outputs a 124 dimensional vector.

keras.layers.Embedding(input_dim, output_dim, embeddings_initializer='uniform', embeddings_regularizer=None, activity_regularizer=None, embeddings_constraint=None, mask_zero=False, input_length=None)

2) **Bi-LSTM:**

BiDirectional means that the sequence fed to LSTM twice, once in the forward direction and once in the backward direction and the outputs are concatenated and sent to the next layer.

Separate extractors will be built for each contract entity (e.g. LoanDuration) and apply the extractor for each corresponding element type separately so that it can improve the model's accuracy by focusing on identifying a single element.

3) **Dense:**

This is a fully connected layer applied in conjunction with the TimeDistributed wrapper. This wrapper applies a layer to every temporal slice of an input i.e every word in sequence has this layer applied to it.

```
keras.layers.TimeDistributed(layer)
```

4) **Conditional Random Field (CRF):**

The advantage of CRFs is that they take into consideration the entire sequence i.e they learn constraints between the labels of the words for e.g the crfs are capable of learning that between two words that are contract title (e.g. Seller and Buyer) it is not possible to have the StartDate entity. So the CRF takes the entire output from the dense layer for the entire sequence and based on the features it outputs the most likely sequence of tags.

6. Tools to be used

For the project, I plan on using python. Among the other options that I explored for the project are Java, C and C++. But for projects of huge scale such as this one, python seems to be the most convenient due to a lot of features for example it is open-source. Additionally, python has a built-in package called json, which can be used to work with JSON data very easily.

The model will be implemented in keras using its functional model API.

The layers will be implemented using https://keras.io/layers/ where each embedding, bi-LSTM (tf.keras.layers.Bidirectional), dense, and CRF. Python gives an advantage here. All these tools and libraries are open-source.

7. Timeline

Goals for the various phases:

PHASE	OBJECTIVE OF PHASE
Community Bonding Period	• Creation of a training and testing dataset containing wide varieties of clauses and contracts. • Get to know the community well and talk to my mentors to know more and more implementation details of the project • Play around more with Cicero parse which extracts data from text. • Get more familiar with the markdown transform and CommonMark DOM. • Get good understanding of all the modules, all the intricacies of binding each module and a detailed report of the modules
Phase 1	Building and Training/Testing Document Classification Module
Phase 2	Building Entity Extraction Module
Phase 3	Training and Testing Entity Extraction Module

Week-Wise Goals:

WEEK	DATES	TASKS
Community Bonding Period (May 4 - May 31)		
1	June 1 - June 7	• Setting up the repository and installing all the additional dependencies required. • Search for better datasets (found https://www.lawinsider.com/ yet) • Preprocess the dataset (training and testing) for document classification. Implement PreProcessFunction.
2	June 8 - June 14	• Implement the components, domain, utilities of the fuzz-like matcher. • Write unit-tests for each.

3	June 15 - June 21	• Implement the main functions TokenizerFunction, MatchingFunction, and ScoringFunction • Write unit-tests for each function.
4	June 22 - June 28	• Integrate all the functions and train the model with the training dataset. • Most of the time will be consumed for rigorous testing on the testing dataset and bug fixing. • Measure accuracy and improve.
PHASE 1 EVALUATIONS (June 29)		
5	June 29 - July 5	• Preprocess the dataset for entity extraction.
6	July 6 - July 12	• Prepare dimensional embeddings and dimensional Part-of-Speech tags • Implement the word embedding layer
7	June 13 - July 19	• Study and explore RNNs, Bi-LSTM, and BILSTM-(LSTM)-LR • Implement the Bi-LSTM layer. This layer is quite complex to code hence an entire week is allocated for it.
8	July 20 - July 26	• Implement the Dense layer and the Conditional Random Field layer
PHASE 2 EVALUATIONS (July 27)		
9	July 27 - August 2	• Training the model with training dataset and test. • Most of the time will be consumed for rigorous testing on the testing dataset and bug fixing. • Calculate the accuracy, precision, recall & f1 scores and improve.
10	August 3 - August 9	• Integrate both the sub-modules into one model. • Look for a good optimizer for the model (found adam optimizer - from keras.optimizers import Adam yet) and improve the performance of the model.
11	August 10 - August 16	If things went fine till now, then I would be done with my target till now. This period will basically act as a buffer period. • Complete any left-over tasks • If there is time left then I would like to work on improving the module by adding features • I would be in touch with the mentors and if there are any further issues or bugs that are left unresolved then I would work on them. • Further refine tests. • Write documentation for the whole project.
12	August 17 - August 23	This period will basically act as a buffer period. • Complete any left-over tasks

		• If there is time left then I would like to work on improving the module by adding features • I would be in touch with the mentors and if there are any further issues or bugs that are left unresolved then I would work on them. • Further refine tests. • Write documentation for the whole project.
FINAL	August 24 - August 31	Submit final work product and final evaluation
FINAL EVALUATIONS (August 31)		

Total estimation of woman hours: 40 hours/week

Note:

- Open to changing the timeline according to the mentor.
- Holidays and other stuff are also considered while writing a project plan.
- Buffer time periods have been kept multiple times, to cope up with any unexpected changes.

8. How I will successfully complete the project

- I am confident of completing this project as the project suits my interests and I have a fairly good experience with python. I have previously worked on projects that have strict deadlines and have a good prior experience with open source.
- I believe I have enough fuel to get started on my goals, I plan to devote my entire summer for the completion of the project
- I would push my work to the remote fork frequently so that the mentors can review my progress whenever required. I would always be in constant communication with my mentors to update them on my current progress and seek their guidance whenever I am stuck in any issue.
- Even after GSoC ends, I will actively contribute to Accord and be available if anyone has any questions/issues regarding my code.

9. Thanking Note to organisation

I would like to thank all the members of the Accord Project organization, to give me this opportunity to write this proposal and for providing the possibility to work with you. I will look forward to every feedback from organisation members reviewing this document,and would be glad to discuss/change accordingly.