# ANLP PROJECT MID SUBMISSION

ContractNLI: A Dataset for Document-level Natural Language Inference for Contracts

**Team Number**
8

**Team Name**
DB

**Team Members**

1. Swetha Vipparla (2020101121)
2. Shubh Agarwal (2020101131)
3. Ayush Agrawal (2020101025)

# EXPLORATORY DATA ANALYSIS

The dataset has 17 hypotheses annotated on 607 non-disclosure agreements (NDAs). The hypotheses are fixed throughout all the contracts including the test dataset.

The dataset is provided as JSON files.

The information in the dataset is as follows:

| **Attribute** | **Explanation** |
|---|---|
| `text` | The full document text |
| `spans` | List of spans as pairs of the start and end character indices. |
| `annotation_sets` | It is provided as a list to accommodate multiple annotations per document. Since we only have a single annotation for each document, you may safely access the appropriate annotation by `document['annotation_sets'][0]['annotations']`. |
| `annotations` | Each key represents a hypothesis key. choice is either Entailment, Contradiction or NotMentioned. `spans` is given as indices of spans above. `spans` is empty when choice is NotMentioned. |
| `labels` | Each key represents a hypothesis key. hypothesis is the hypothesis text that should be used in NLI. |

There are 3 JSON files in the dataset, one for train, test, and dev. Each JSON file comes with supplemental information.

| Information | Explanation |
|---|---|
| id | A unique ID throughout train, development and test datasets. |
| file_name | The filename of the original document in the dataset zip file. |
| document_type | One of search-pdf (a PDF from a search engine), sec-text (a text file from SEC filing) or sec-html (an HTML file from SEC filing). |
| url | The URL that we obtained the document from. |

Dataset Distribution:

| Dataset | No. of Contracts (Documents) | Total number of Spans |
|---|---|---|
| Train | 423 | 32895 |
| Dev | 61 | 5102 |
| Test | 123 | 10061 |

# Baselines & Metrics

We have implemented the following six baselines from scratch according to their descriptions given in the paper. We have calculated metrics like accuracy, F1 score, Precision @ 80% Recall etc. as applicable for each of the baseline models.

**Objectives**

There are two objectives for which the baseline models are used:

I. **NLI task**: Given a contract and a hypothesis, we have to find whether the hypothesis is an entailment of, or contradiction to or not related to the contract. For this task, the metrics calculated are:

   A. Accuracy
   B. F1 score for Entailment
   C. F1 score for Contradiction

II. **Evidence Identification**: Given a span of a contract and a hypothesis which might be an entailment or contradiction, we have to find whether the given span is an evidence for the hypothesis inference. For this task, the metrics calculated are:

   A. Mean Averaged Precision (mAP)
   B. Precision at 80% Recall

**Models**

1. **Random Baseline**

   **Objective**: Evidence Identification Only
   **Description**: It outputs 0 or 1 randomly for a given span-hypothesis pair where 1 indicates the span is an evidence for the given hypothesis and 0 indicates it is not.

| **Metric** | **Value Obtained** |
|---|---|
| mAP | 0.031 |
| Precision at 80% Recall | 0.032 |

2. **Majority-Vote**:

   **Objective**: NLI Only
   **Description**: This is a majority vote oracle classifier which learns the majority label assigned to each hypothesis over all the contracts. Then given a test set, it just outputs the majority vote for the hypothesis without considering the document given.

| **Metric** | **Value Obtained** |
|---|---|
| Accuracy | 86% |
| F1 score for Entailment | 0.401 |
| F1 score for Contradiction | 0.085 |

3. **Doc TF-IDF+SVM** :

   **Objective**: NLI Only
   **Description**: We use the `TfidfVectorizer` class of scikit-learn to learn vocabulary and `idf` (inverse document frequency) from the documents present in train.json. To construct the training data, we form pairs of document and hypothesis vectors by concatenating them with the target label being one of entailment, contradiction or not mentioned. After constructing the training data, we fit the SVM classifier (with a linear kernel) on this dataset. We then construct the testing data in a similar fashion taking documents from test.json and then perform predictions on the test set.

| **Metric** | **Value Obtained** |
|---|---|
| Accuracy | 71.2% |
| F1 score for Entailment | 0.677 |
| F1 score for Contradiction | 0.188 |

4. **Span TF-IDF+Cosine**:

   **Objective**: Evidence Identification Only
   **Description**: Like the Doc TF-IDF + SVM baseline, we use the `TfidfVectorizer`

class to learn vocabulary and idf from documents present in train.json. Then, we calculate the probability of each span being an evidence by calculating the cosine similarity between the vectors of span and hypothesis generated by the TfidfVectorizer.

| **Metric** | **Value Obtained** |
|---|---|
| mAP | 0.357 |
| Precision at 80% Recall | 0.060 |

5. **Span TF-IDF + SVM**:

   **Objective**: Evidence Identification Only
   **Description**: In this model, an SVM model is trained to predict the probability of the span being evidence for the hypothesis. The input to the SVM is the vector of span and hypothesis concatenated together and output is a binary target where 1 indicates that the span is evidence of the hypothesis and 0 indicates it is not.

| **Metric** | **Value Obtained** |
|---|---|
| mAP | 0.811 |
| Precision at 80% Recall | 0.301 |

6. **SQuAD BERT**:

   **Objective**: Evidence Identification Only
   **Description**:Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowd workers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable. SQuAD BERT is basically BERT that is fine tuned for SQuAD.
   As mentioned in the paper, the hugging face's SQuAD BERT implementation is used. The synonymous implementation in the library is BertForQuestionAnswering. Given a text and a question(hypothesis), the model outputs the start and end index of the span of the answer in the text. This is done by calculating the probabilities for each start and end spans and then obtaining the argmax of them.

**Issues Faced**:
However, we faced issues in implementing this model since there is no code available for the same and the implementation mentioned in the paper is incomplete. Specifically, the computation of the probabilities over different context windows proved to be complex to implement without further information on the methods of tokenisation used by the authors for this task in particular as well as their pre-processing and post processing scripts.

# Main Implementation

The main implementation of the paper is the **Span NLI BERT** model.

The main idea of the model is as follows:

- Previous methods used Transformers for span identification by predicting start and end tokens. This approach had challenges, as it required the model to detect span boundaries and gather evidence simultaneously.
- Dividing documents into fixed contexts could lead to issues like spans being split across contexts or lacking sufficient context.
- Span NLI BERT is introduced as a multi-task Transformer model to address these issues.
- Instead of predicting start and end tokens, it uses special `[SPAN]` tokens to represent spans and performs multi-label binary classification.
- Document splitting is done with dynamic stride sizes to ensure each span has at least one context without splitting.

We have started implementing the model. The first step is to make dynamic contexts for the model whose algorithm is given in the paper.

The pseudocode outlines a procedure to extract overlapping textual contexts from a given sequence of tokens. It operates by considering span boundaries represented in the list B, ensuring that each context contains a minimum of n surrounding tokens and does not exceed a maximum context length `l`. The algorithm iteratively processes the span boundaries, constructs contexts that meet these criteria, and appends them to an output list named `contexts`. It dynamically updates the start and end indices to ensure overlapping contexts are generated, and it removes already processed span boundaries to avoid duplication.