Semantic Role Labelling For Hindi

Team Logical Parsers

Satya G. - 2023202015
Hrishikesh N. - 2023201012
Nikith D. - 2023201057

Semantic Role Labelling:

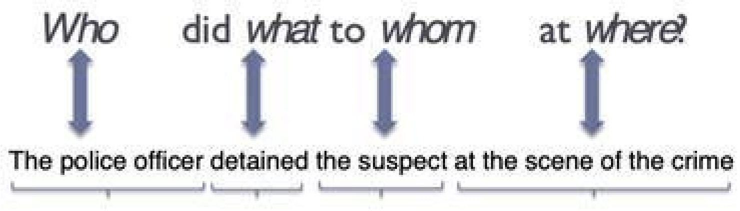
Semantic Role Labelling (SRL) is a natural language processing (NLP) task that assigns semantic roles to words and phrases in a sentence. It indicates the relationship between words and the main predicate or verb. Its role is to extract information about who did what to whom, when, where, and how in a given sentence. Simply put, SRL tries to identify the different roles played by various elements of a sentence corresponding to the action given by the verb. It plays a key role in understanding sentences and the role of specific words in a given sentence. The development of better SRL methods will lead to improvements in machine translation, text summarization, text generation, sentiment analysis, question answering, and information extraction.

SRL can be broken down into different parts, and they are:

- **Predicate Detection**: Identifying the predicate in a given statement.
- **Argument Identification**: Identifying the arguments for the given predicate in the given sentence.
- **Role Labeling**: Assigning roles to the arguments found.

Each of these parts can be considered a challenge for free-order languages (languages where the syntactical structure of a sentence is not fixed and is flexible) like Hindi (*"Ram ne apne dost ko kitaab di."* and *"Ek kitaab Ram ne apne dost ko di."* have the same meaning with different order) Latin, Japanese, and Russian. This project focuses more on the Role labeling aspect of SRL in Hindi.

Example:


Who did what to whom at where?
The police officer detained the suspect at the scene of the crime

The above example shows SRL in English. It shows three labels - **Agent**, **Theme**, and **Location**. In the example, the police officer (**Agent**) is the person detaining (**Predicate**) the suspect (**Theme**) at the scene of the crime (**Location**). Some of the common labels are:

- Agent
- Experiencer

- Theme
- Result
- Location
- Instrument
- Time

SRL in Hindi is implemented with different labels. The previous works on SRL in Hindi, like "Towards
Building Semantic Role Labeler for Indian Languages" (Anwar and Sharma, 2016) and "Enhancing Semantic Role Labeling in Hindi and Urdu" (Gupta and Shrivastava, 2018) use a 2-step approach. They first identify whether a given word is an argument for a given predicate, and if it is, they label it. This helps because there are a lot of words that are not arguments to a given predicate, and if included, induce issues in creating accurate models.

For SRL in Hindi, the labels used are given below:

Label	**Description**
ARG0	Agent, Experiencer, or doer
ARG1	Patient or Theme
ARG2	Beneficiary
ARG3	Instrument
ARG2-ATR	Attribute or Quality
ARG2-LOC	Physical Location
ARG2-GOL	Goal
ARG2-SOU	Source
ARGM-PRX	Noun-Verb Construction
ARGM-ADV	Adverb
ARGM-DIR	Direction
ARGM-EXT	Extent or Comparision
ARGM-MNR	Manner
ARGM-PRP	Purpose

Label	Description
ARGM-DIS	Discourse
ARGM-LOC	Abstract Location
ARGM-MNS	Means
ARGM-NEG	Negation
ARGM-TMP	Time
ARGM-CAU	Cause or Reason

DATASET

Retrieval and Preprocessing

For the dataset, we have chosen the Hindi Propbank.

Hindi Propbank

ProbBank is an annotated corpus consisting of information regarding the structure of Predicates. It involves creating a semantic layer of annotation that adds up the predicate to argument structure to syntactic representations. The semantic role labeling's consistency helps in training of ML systems such as automatic semantic role labelers. Propbank also considers the different senses of the same verb while annotating the semantic roles, which means the same verb might have two different sets of SR labels.

The structure of the Hindi Propbank has a few tags at the start representing the organization and extra features. We have removed those details from the propbank dataset and just considered the <sentence> tag as our starting point.

Each <sentence> tag represents a separate sentence. Within each sentence, there are various annotations indicating the syntactic structure of the sentence <sentence id= 'Number' > represents the unique id of the sentence in the prop bank.
There are a few annotations in the sentence tag that we feel are important to be understood.

1. <fs> - It is the feature structure that provides us with the grammatical information about a word or a phrase. It gives us an idea about the grammatical category, for example, NP(noun phrase) or VP(verb phrase), and syntactic relations.

2. <posn> - Gives us the idea about the positional information, i.e., position in that particular sentence
3. <pbrel> - It indicates the predicate-argument relation
4. <head> - head word of the phrase that we are grabbing from every sentence, and this headword is used for finding the embeddings
5. <drel> - Syntactic relation
6. <name> - name or identifier of the word

Additionally, we also have other tags that we are not considering for the preprocessing of the dataset.

SSFAPI.py is a Python script to preprocess the dataset.
Classes Mentioned in the SSFAPI file

- Node class is used to represent a linguistic parse tree
- Chunk Node is used to represent an individual node in the linguistic parse tree
- Document is a collection of sentences, and there is another class named sentence, which indicates an individual sentence

FUNCTIONS used in the SSFAPI file

• GetChunkFeats and TokenFeats are two feature-extracting functions used for two different components that are Chunk Features and Token Features, and these are extracted using the Chunk node and the Token node

• FindSentences is another function that is used to find the next sentence with the tag <sentence>

• FolderWalk function helps us grab all the files in that folder. Using this function, we find all the .pb files in the particular destination folder, store them in a list, and iterate each file in the list to get the headwords and other attributes required

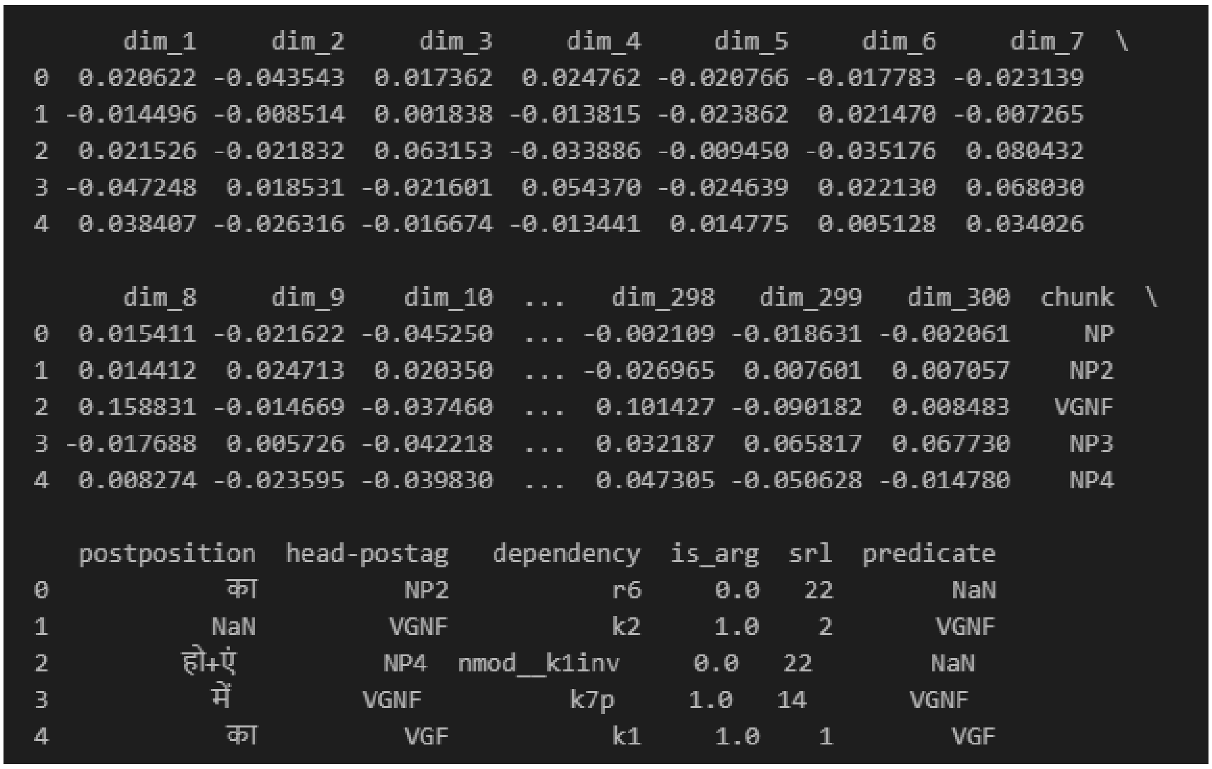
- We read the input files and construct the Document Objects for every .pb file.
- Then, we parse the sentences into linguistic elements using the sentence objects.
- Then, we process each node in the parse tree, and from it, we extract the features.
- There are many features we have extracted and stored into the .txt file, and then we converted the .txt file into the .csv file.

Final Dataset File (dataset.csv)

- After storing the .txt file obtained from the above step, we have converted the above file into a .csv file with the column names as mentioned: “word, chunk, postposition, head-postag, dependency, is_arg, srl, predicate.”

- Using the fasttext embeddings for the Hindi language, we have taken the head column for the words and then searched up the fasttext embeddings for that particular word and got the embeddings for that word.
- Now, we have stored the 300 dimension embeddings into the 300 columns where each column and retaining the remaining columns.
- The total CSV file has 308 columns.

These embeddings in the final dataset.csv file are used in the models.
The Data representation of the dataset.csv file



```
      dim_1     dim_2     dim_3     dim_4     dim_5     dim_6     dim_7  \
0  0.020622 -0.043543  0.017362  0.024762 -0.020766 -0.017783 -0.023139
1 -0.014496 -0.008514  0.001838 -0.013815 -0.023862  0.021470 -0.007265
2  0.021526 -0.021832  0.063153 -0.033886 -0.009450 -0.035176  0.080432
3 -0.047248  0.018531 -0.021601  0.054370 -0.024639  0.022130  0.068030
4  0.038407 -0.026316 -0.016674 -0.013441  0.014775  0.005128  0.034026

      dim_8     dim_9    dim_10  ...   dim_298   dim_299   dim_300  chunk  \
0  0.015411 -0.021622 -0.045250  ... -0.002109 -0.018631 -0.002061     NP
1  0.014412  0.024713  0.020350  ... -0.026965  0.007601  0.007057    NP2
2  0.158831 -0.014669 -0.037460  ...  0.101427 -0.090182  0.008483   VGNF
3 -0.017688  0.005726 -0.042218  ...  0.032187  0.065817  0.067730    NP3
4  0.008274 -0.023595 -0.039830  ...  0.047305 -0.050628 -0.014780    NP4

   postposition  head-postag   dependency  is_arg  srl  predicate
0            का          NP2           r6     0.0   22        NaN
1           NaN         VGNF           k2     1.0    2       VGNF
2         हो+एं          NP4  nmod__k1inv     0.0   22        NaN
3           में         VGNF          k7p     1.0   14       VGNF
4            का          VGF           k1     1.0    1        VGF
```

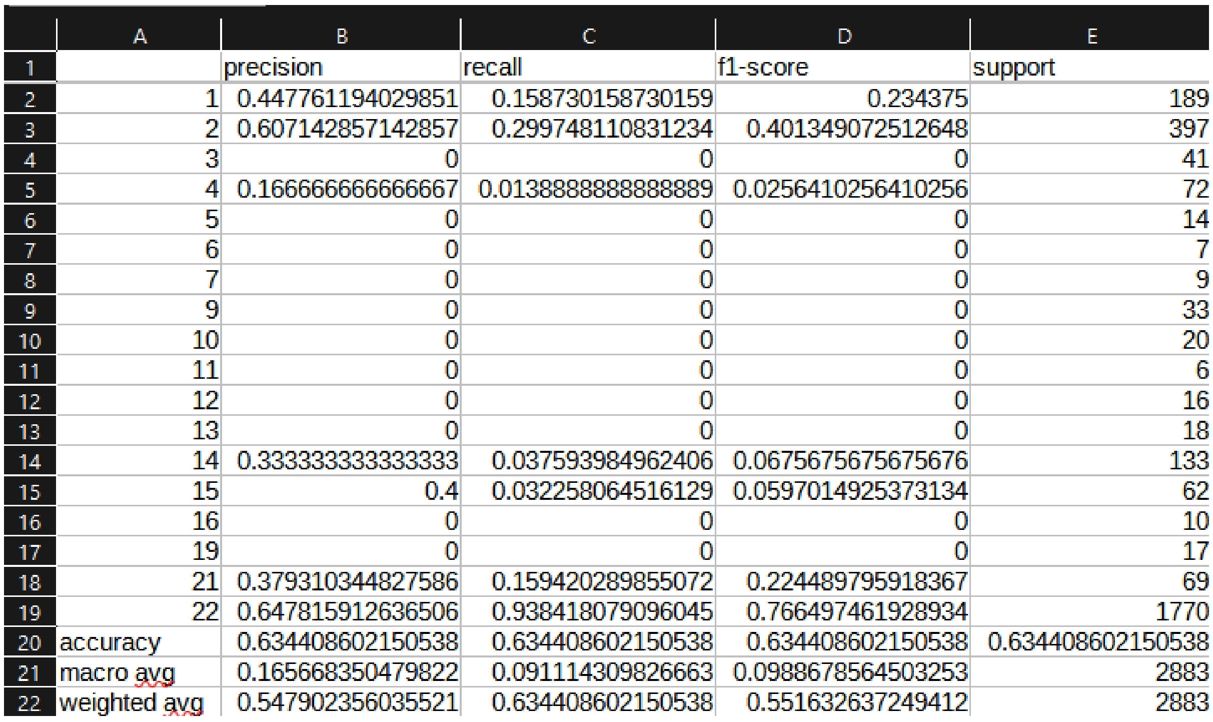
Argument Classification:

Models

We trained 4 models on a train-test split of 85% and 15% on the available dataset.

1. Logistic Regression

We trained a Logistic Regression model for labeling, and the results are shown below.



	A	B	C	D	E
1		precision	recall	f1-score	support
2	1	0.447761194029851	0.158730158730159	0.234375	189
3	2	0.607142857142857	0.299748110831234	0.401349072512648	397
4	3	0	0	0	41
5	4	0.166666666666667	0.0138888888888889	0.0256410256410256	72
6	5	0	0	0	14
7	6	0	0	0	7
8	7	0	0	0	9
9	9	0	0	0	33
10	10	0	0	0	20
11	11	0	0	0	6
12	12	0	0	0	16
13	13	0	0	0	18
14	14	0.333333333333333	0.037593984962406	0.0675675675675676	133
15	15	0.4	0.032258064516129	0.0597014925373134	62
16	16	0	0	0	10
17	19	0	0	0	17
18	21	0.379310344827586	0.159420289855072	0.224489795918367	69
19	22	0.647815912636506	0.938418079096045	0.766497461928934	1770
20	accuracy	0.634408602150538	0.634408602150538	0.634408602150538	0.634408602150538
21	macro avg	0.165668350479822	0.091114309826663	0.0988678564503253	2883
22	weighted avg	0.547902356035521	0.634408602150538	0.551632637249412	2883

Methodology

We employed a variety of features including:

- Chunk tags
- Headword of the chunk
- POS-tag of the headword
- Dependency labels
- Postpositions

We analyzed the performance of different combinations of these features to determine their effectiveness in the tasks at hand.

Feature Combinations

We experimented with various combinations of features and evaluated their performance. The combinations included:

- Feature set 1: Chunk tags, Dependency labels

- Feature set 2: POS-tag of headword, Dependency labels
- Feature set 3: Predicate, Dependency labels
- Feature set 4: Postpositions, Pos-tag of headword

Results

Model	Accuracy
Feature set 1	65
Feature set 2	71
Feature set 3	85
Feature set 4	66

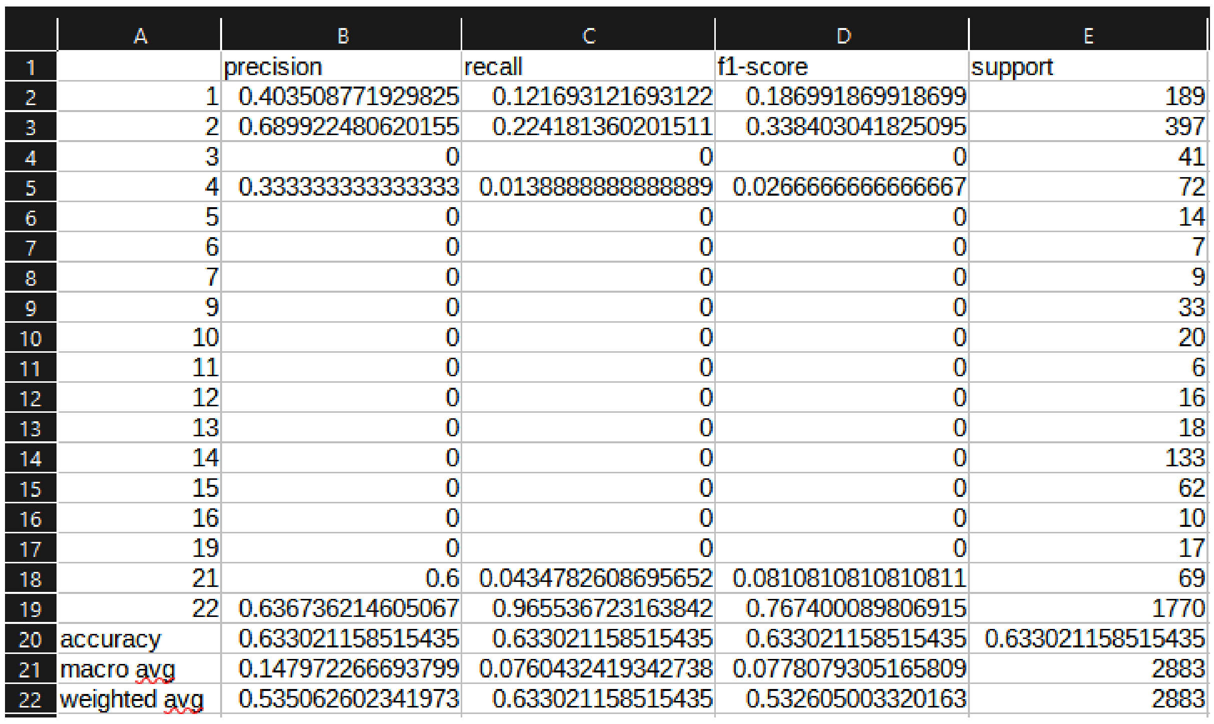
After thorough analysis, we found that the combination of features including the ”predicate + Dependency labels” consistently produced the best results for both argument identification and argument classification tasks.

Conclusion

In conclusion, our feature analysis reveals that the combination of features (predicate, Dependency labels) is the most effective for the tasks of argument identification and classification. These features provide valuable linguistic information that significantly improves the performance of our models.

2. Support Vector Machine (SVM)

We trained an SVM classifier for the labeling, and the results are shown below.



	A	B	C	D	E
1		precision	recall	f1-score	support
2	1	0.403508771929825	0.121693121693122	0.186991869918699	189
3	2	0.689922480620155	0.224181360201511	0.338403041825095	397
4	3	0	0	0	41
5	4	0.333333333333333	0.0138888888888889	0.0266666666666667	72
6	5	0	0	0	14
7	6	0	0	0	7
8	7	0	0	0	9
9	9	0	0	0	33
10	10	0	0	0	20
11	11	0	0	0	6
12	12	0	0	0	16
13	13	0	0	0	18
14	14	0	0	0	133
15	15	0	0	0	62
16	16	0	0	0	10
17	19	0	0	0	17
18	21	0.6	0.0434782608695652	0.0810810810810811	69
19	22	0.636736214605067	0.965536723163842	0.767400089806915	1770
20	accuracy	0.633021158515435	0.633021158515435	0.633021158515435	0.633021158515435
21	macro avg	0.147972266693799	0.0760432419342738	0.0778079305165809	2883
22	weighted avg	0.535062602341973	0.633021158515435	0.532605003320163	2883

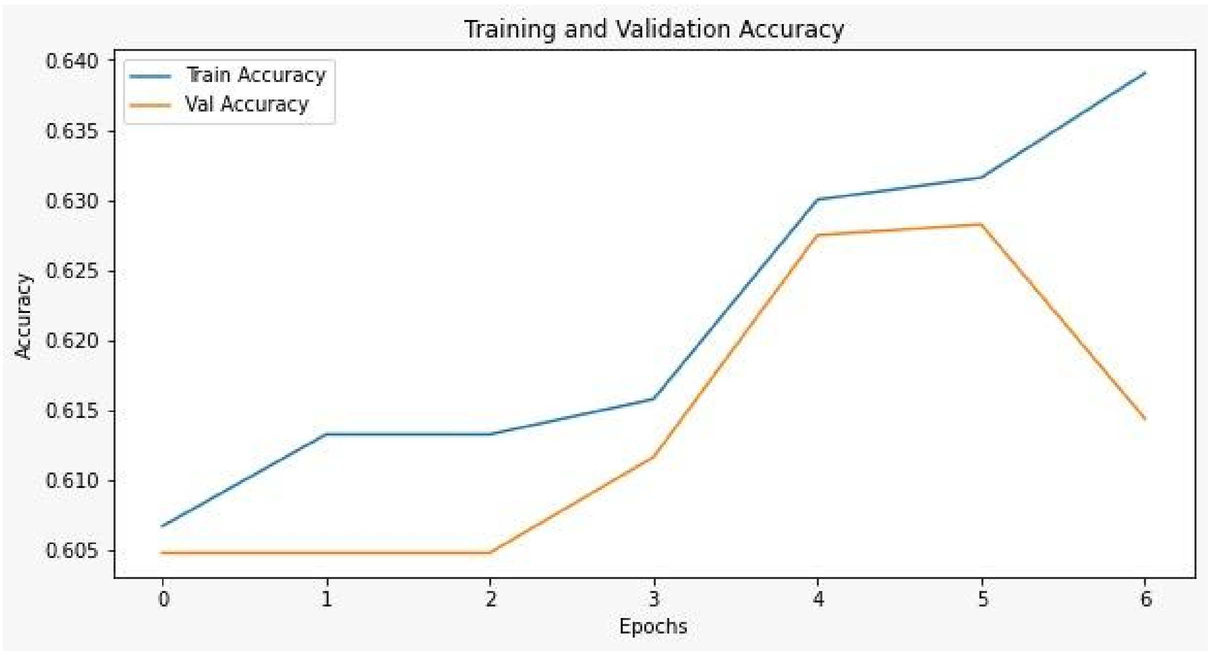
3. Recurrent Neural Network (RNN)

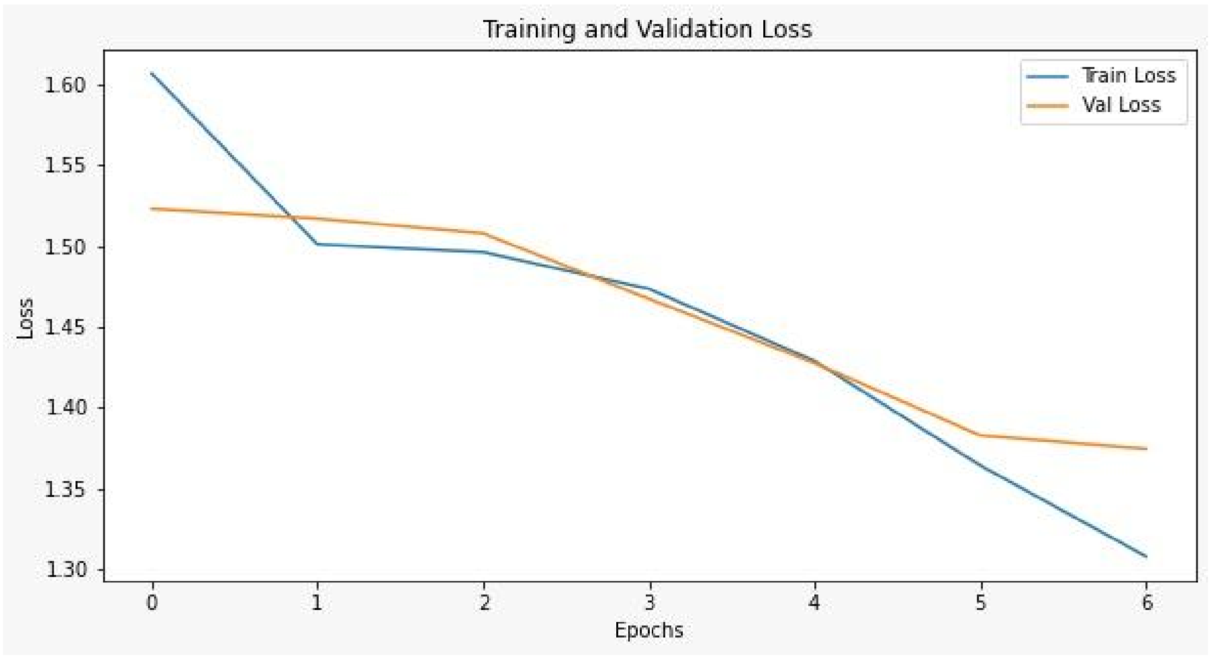
We trained an RNN model with the hyperparameters that follow:

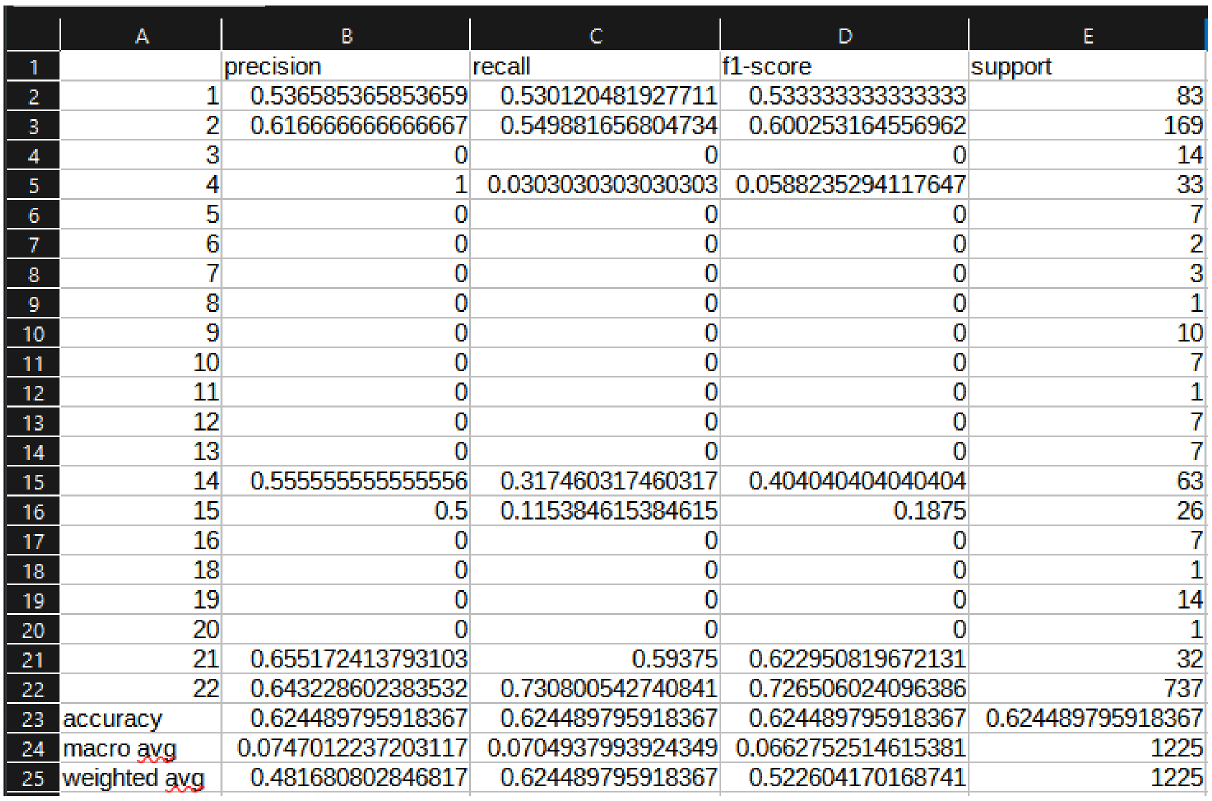
EMBEDDING_DIMENSION= 300
HIDDEN_NODES = 200
BATCH_SIZE = 128
Epochs = 6
learning_rate = 0.0005

LOSS = CROSSENTROPYLOSS
OPTIMIZER = ADAM

The results can be seen below.


Training and Validation Accuracy
Train Accuracy
Val Accuracy
Accuracy
Epochs
0.640
0.635
0.630
0.625
0.620
0.615
0.610
0.605
0
1
2
3
4
5
6


Training and Validation Loss
Train Loss
Val Loss
Loss
Epochs
1.60
1.55
1.50
1.45
1.40
1.35
1.30
0
1
2
3
4
5
6



	A	B	C	D	E
1		precision	recall	f1-score	support
2	1	0.536585365853659	0.530120481927711	0.533333333333333	83
3	2	0.616666666666667	0.549881656804734	0.600253164556962	169
4	3	0	0	0	14
5	4	1	0.0303030303030303	0.0588235294117647	33
6	5	0	0	0	7
7	6	0	0	0	2
8	7	0	0	0	3
9	8	0	0	0	1
10	9	0	0	0	10
11	10	0	0	0	7
12	11	0	0	0	1
13	12	0	0	0	7
14	13	0	0	0	7
15	14	0.555555555555556	0.317460317460317	0.404040404040404	63
16	15	0.5	0.115384615384615	0.1875	26
17	16	0	0	0	7
18	18	0	0	0	1
19	19	0	0	0	14
20	20	0	0	0	1
21	21	0.655172413793103	0.59375	0.622950819672131	32
22	22	0.643228602383532	0.730800542740841	0.726506024096386	737
23	accuracy	0.624489795918367	0.624489795918367	0.624489795918367	0.624489795918367
24	macro avg	0.0747012237203117	0.0704937993924349	0.0662752514615381	1225
25	weighted avg	0.481680802846817	0.624489795918367	0.522604170168741	1225

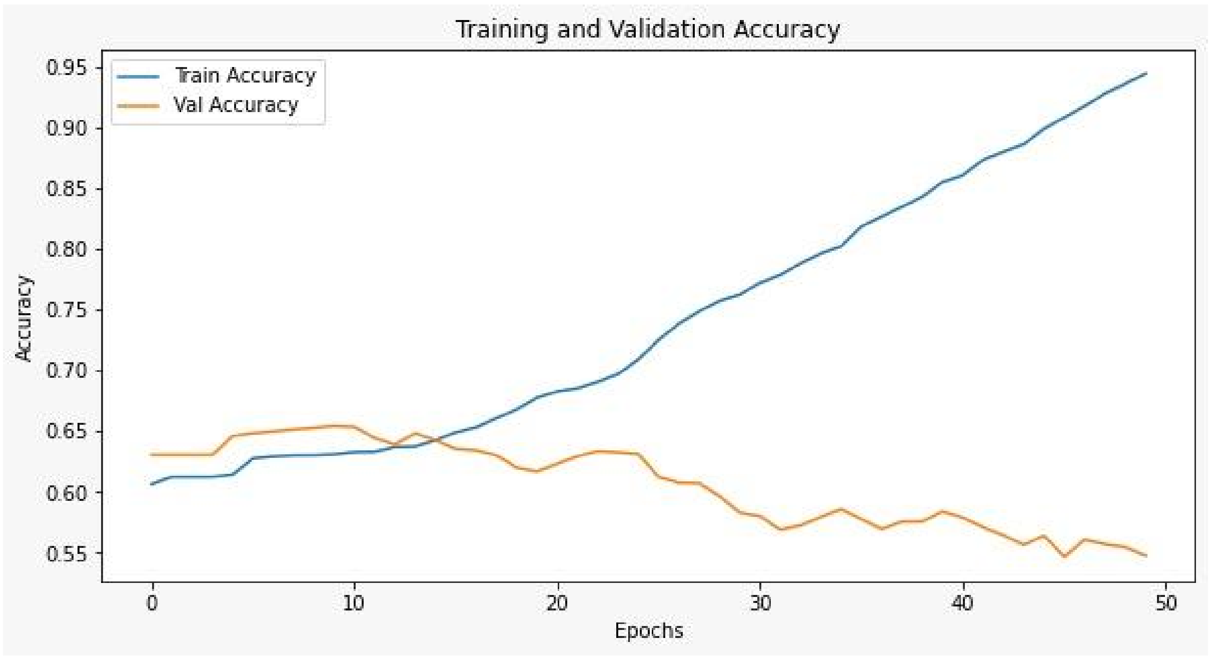
4. LSTM (Long Short-Term Memory)

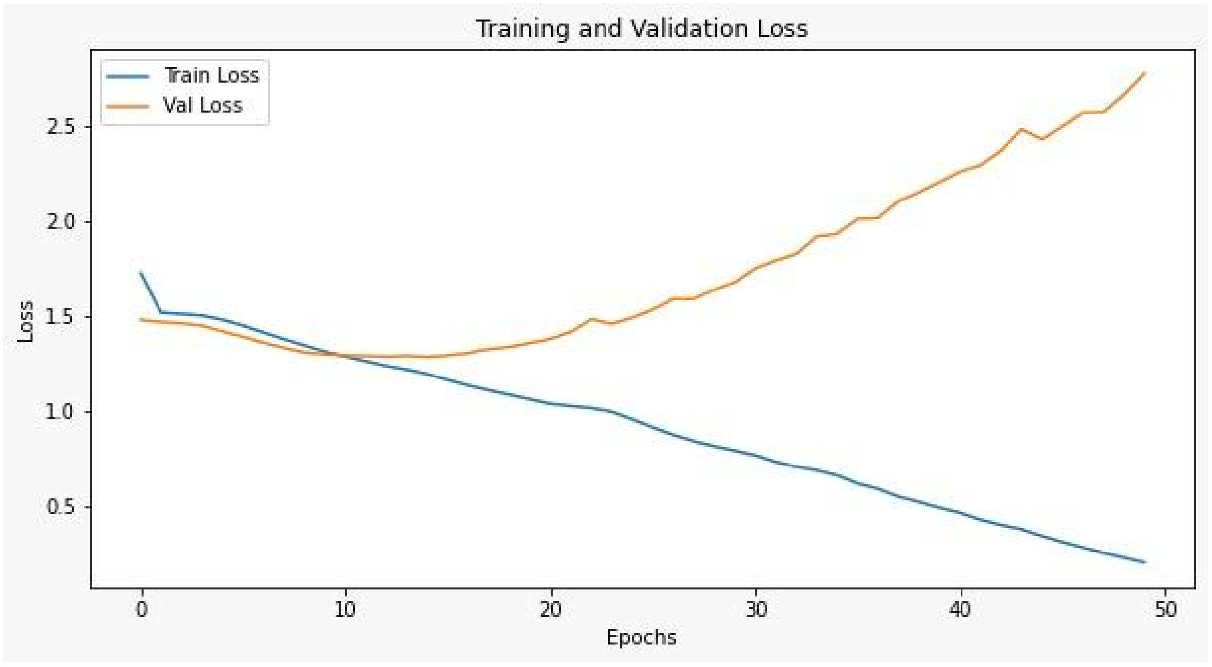
We trained an LSTM model with the hyperparameters that follow:

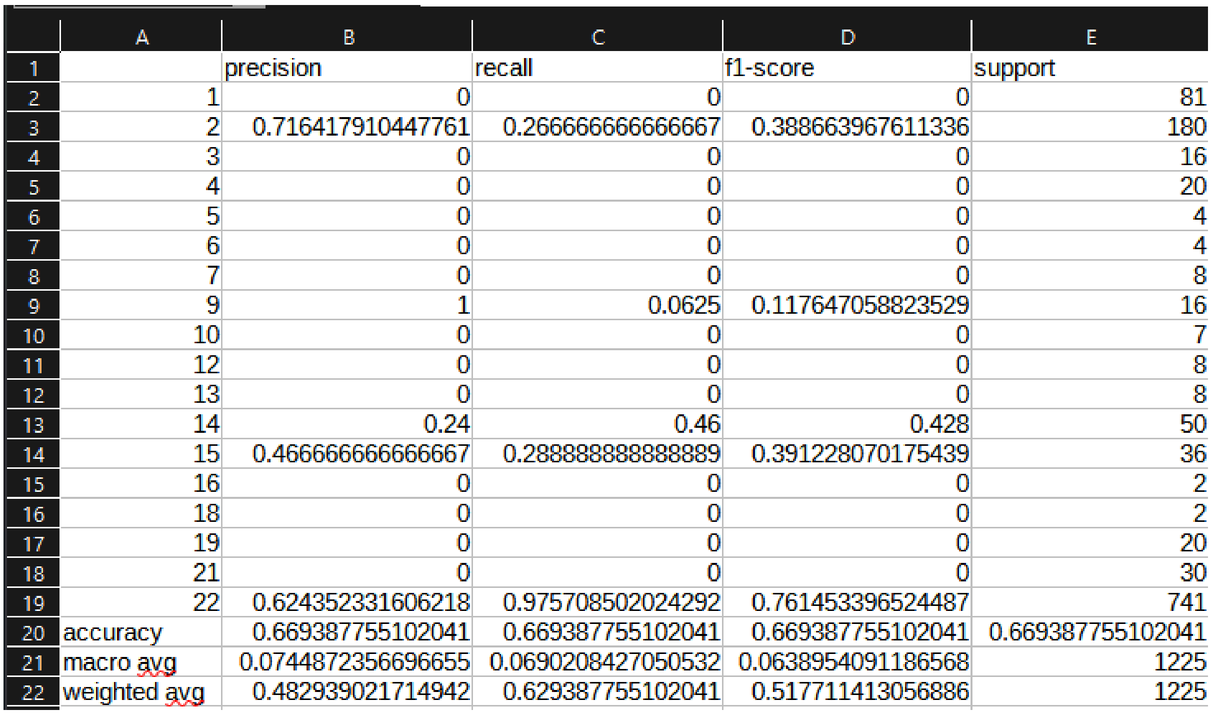
EMBEDDING_DIMENSION= 300
HIDDEN_NODES = 200
BATCH_SIZE = 128
Epochs = 50
learning_rate = 0.0005

LOSS = CROSSENTROPYLOSS
OPTIMIZER = ADAM

The results can be seen below.


Training and Validation Accuracy
Train Accuracy
Val Accuracy
Accuracy
Epochs
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
10
20
30
40
50


Training and Validation Loss
Train Loss
Val Loss
Loss
Epochs
2.5
2.0
1.5
1.0
0.5
0
10
20
30
40
50



	A	B	C	D	E
1		precision	recall	f1-score	support
2	1	0	0	0	81
3	2	0.716417910447761	0.266666666666667	0.388663967611336	180
4	3	0	0	0	16
5	4	0	0	0	20
6	5	0	0	0	4
7	6	0	0	0	4
8	7	0	0	0	8
9	9	1	0.0625	0.117647058823529	16
10	10	0	0	0	7
11	12	0	0	0	8
12	13	0	0	0	8
13	14	0.24	0.46	0.428	50
14	15	0.466666666666667	0.288888888888889	0.391228070175439	36
15	16	0	0	0	2
16	18	0	0	0	2
17	19	0	0	0	20
18	21	0	0	0	30
19	22	0.624352331606218	0.975708502024292	0.761453396524487	741
20	accuracy	0.669387755102041	0.669387755102041	0.669387755102041	0.669387755102041
21	macro avg	0.0744872356696655	0.0690208427050532	0.0638954091186568	1225
22	weighted avg	0.482939021714942	0.629387755102041	0.517711413056886	1225

Conclusion

The results from the classification report indicate that the performance of the linear classifier improves significantly when incorporating additional features alongside the embeddings as the input data (X), with the SRL label serving as the target variable for every model. This enhancement in performance can be attributed to the relationship between the SRL tag, dependency tag, and predicate, which results in higher accuracies for this feature set.

Furthermore, we conducted multi-class classification using neural networks, specifically utilizing RNN and LSTM architectures. While both models yielded similar accuracies, the LSTM model with bidirectionality outperformed the RNN. This suggests that the LSTM's ability to capture bidirectional dependencies within the input sequences contributed to its superior performance compared to the unidirectional RNN

Code and References

Code : https://github.com/Hrishikesh0511/Logical-parsers-INLP-Project-SRL-

References :

https://www.aclweb.org/anthology/L16-1727.pdf

http://lrec-conf.org/workshops/lrec2018/W29/pdf/28_W29.pdf