INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Introduction to NLP (CS7.401)

Project Interim Report

Submitted by:

Team No.: 26

Ayush Lakshakar 2022201051

Aakash Tripathi 2022201053

Bhanuj Gandhi 2022201068

Contents

1 Project Objective

The objective of project is to understand the task of Natural Language Inference in Natural Language Processing. Natural Language Inference deals with understanding the relationship between 2 given sentences. It tries to identify whether a given sentence, called as ***"Hypothesis"*** can be derived/inferred/deduced from another given sentence, called ***"Premise"*** or not. The relationship can be classified into 3 different classes/labels

- **Contradiction** refers to a situation when both, premise and hypothesis, cannot be true at the same time.
- **Entailment** - refers to a situation where Hypotheses can be derived/inferred from given premise.
- **Neutral** refers to a situation where there is not enough information in premise to infer or derive the given hypothesis.

2 About Datasets

For this task, we have explored the 2 famous available datasets SNLI and Multi NLI.

2.1 SNLI

SNLI stands for Stanford Natural Language Inference. It is a benchmark dataset for natural language inference tasks.

- All the premises are the image captions from Flicker30 dataset and hence it makes SNLI somewhat genre restricted.
- All the hypotheses are written by Crowd-workers i.e., corresponding to a premise, crowd workers will write 3 sentences one for each class.
- Total of 550152 train examples with 10,000 as dev samples and 10,000 as test samples, balanced equally across 3 classes.
- Mean token length in SNLI dataset i.e., the average number of words per sentence
  - For premise - 12.9
  - For hypothesis - 7.4
- Approximately 56, 951 examples are validated by 4 additional annotators with 91.2% of gold labels matched with authors labels.
- For SNLI dataset the overall Fleiss' kappa is 0.70, which is defined as the degree of agreement between the annotators, based both categorical labels and similarity matrix.
- Progress on SNLI dataset for Natural Language Inference task
  - Red line indicates the human performance on SNLI dataset

The main fundamental logic in SNLI is relating to image dataset, if premise and hypothesis probably describe a different photo, then the label is a contradiction.

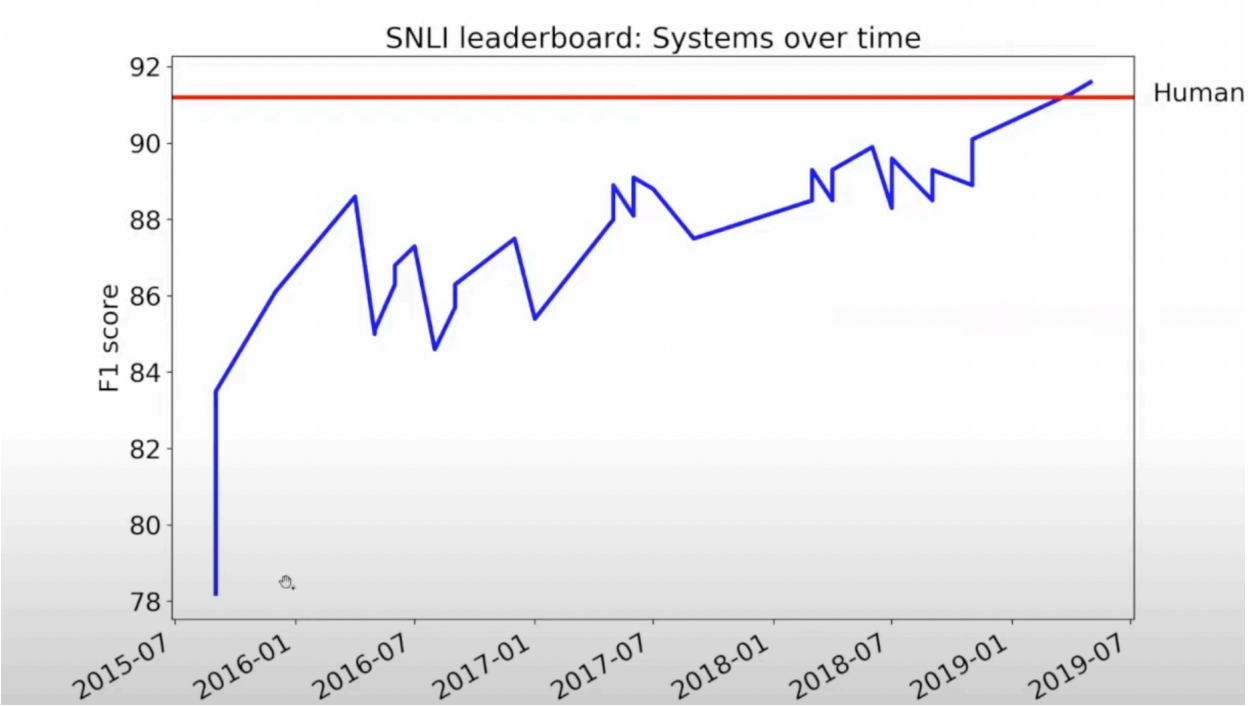

Figure 1: SNLI Leaderboard [1]

2.2 MultiNLI

MultiNLI stands for Multi-Genre Natural Language Inference. It is also another benchmark dataset for natural language inference tasks. It is an extension of the SNLI dataset, with a more diverse set of genres and sources and hence making it a more challenging dataset to train and evaluate the NLI models.

- Train Premises in MultiNLI are contributed from 5 genres
  - Fiction: works from 1912 2010 : spanned across many genres.
  - Govt. information available public reports, letters, speeches, Govt. websites etc.
  - The Slate website
  - The Switchboard corpus Telephonic conversation
  - Bertlitz travel guide
- In addition to above genres, premises from some other genres are also present in dev and test datasets like
  - From 9/11 reports
  - From fundraising letters
  - Nonfiction from Oxford University Press
  - Verbatim articles about linguistics.
- Total of approx. 3,92,702 train examples with 20,000 as dev samples and 20,000 as test samples.
- Approx. 19,647 samples are validated by 4 additional annotators with 92.6% of gold labels matched with authors labels.

- Test dataset of MultiNLI is only available on Kaggle and in the form of competition.
- Progress on SNLI dataset for Natural Language Inference task
  - Red line indicates the human performance on Multi NLI dataset

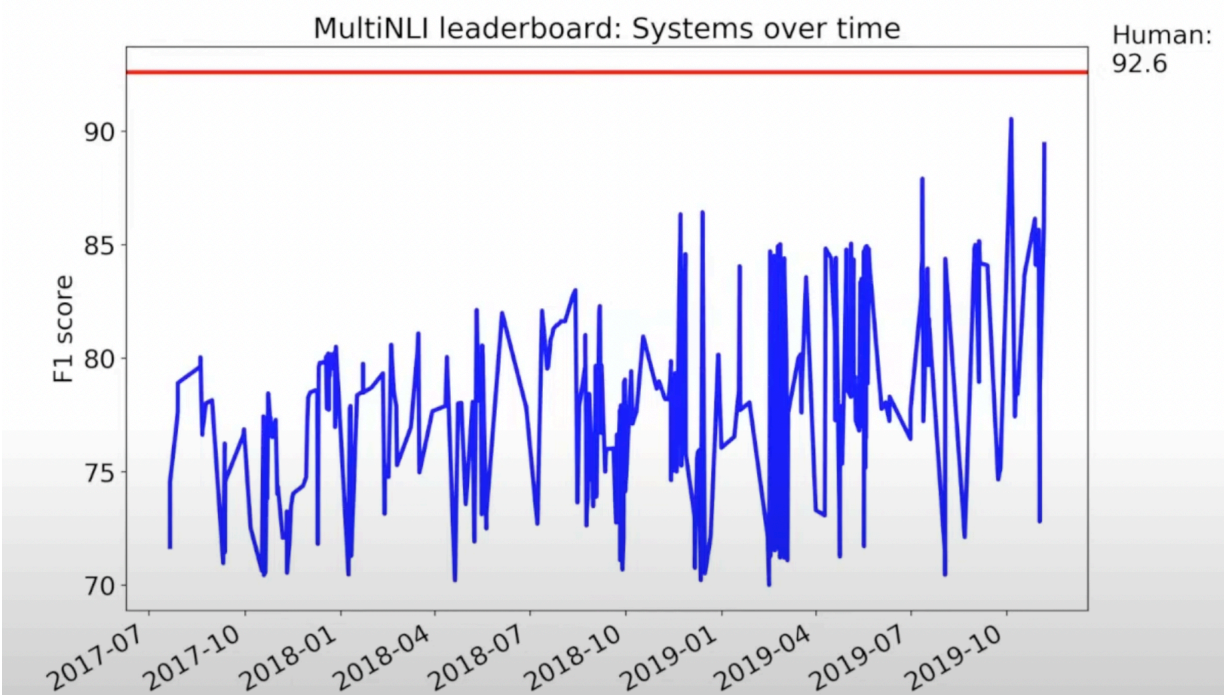

Figure 2: MultiNLI Leaderboard [1]

There is so much variance in MultiNLI progress as compared to SNLI dataset because MultiNLI dataset is available on Kaggle and so there are too much data points/model to compare. But in SNLI, the progress is reported only from the implemented research papers only and not some from public domain, which is the case in MultiNLI.

The MultiNLI dataset is filled with distributed annotations that help to perform out of the box error analysis.

For our further work, we have chosen the SNLI dataset because for SNLI, the test dataset is available and can be easily downloaded. In case of MultiNLI, test data is available only on Kaggle and in the form of competition and hence it cannot be downloaded. Thus after training the models we will be trying them out on the MultiNLI dataset as well.

3 Work done so far

Work done so far in the task of Natural Language Inference can be divided as -

1. Dataset Exploration
2. Basic EDA and data pre-processing of selected dataset
3. Experiments by fitting some basic ML/Language Modelling techniques.

3.1 Dataset Exploration

As described above, we have explored 2 datasets i.e., SNLI and MultiNLI datasets for NLI tasks. We will be continuing with the SNLI dataset as for SNLI the test dataset is available which is not the case with MultiNLI.

3.2 Basic EDA and Data Pre-processing

Basic EDA and data pre-processing has been done on SNLI dataset. This includes

1. Identified the columns that contained only NULL values and removed them.
2. Visualized the distribution of each gold label (Contradiction, Neutral, Entailment) using histogram and removed any samples that contained irrelevant label(s).

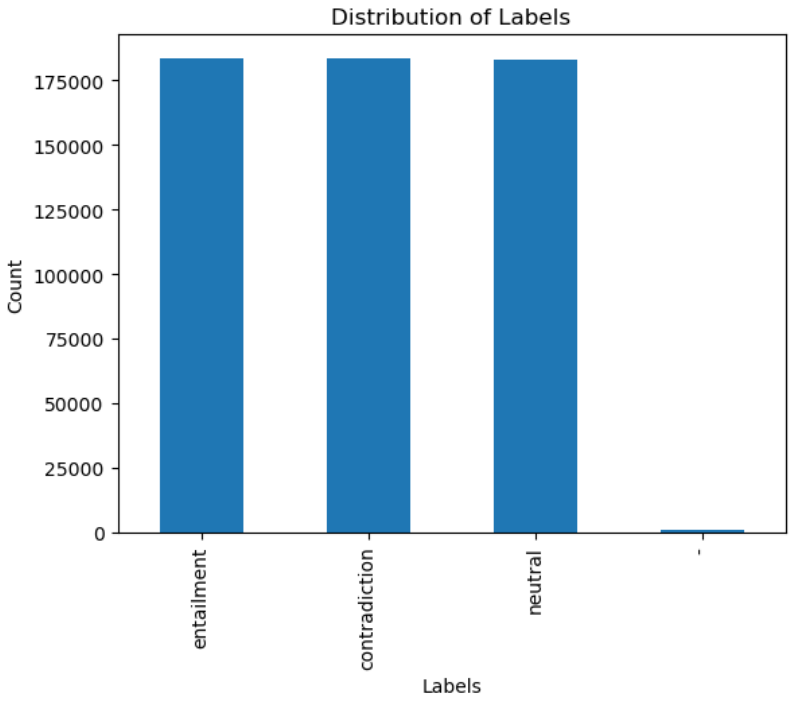

Figure 3: Label Distribution of the SNLI Dataset

- Identified any NULL premises or hypothesis and removed them from the dataset.
- Converted all premise and hypothesis to lowercase to ensure consistency in the dataset.
- Analysed the statistical parameters of premise and hypothesis that helped to better understand SNLI dataset.
- Encoded the gold labels to numerical categories.
- Identified the most frequently occurring 20 words in premise sentences.

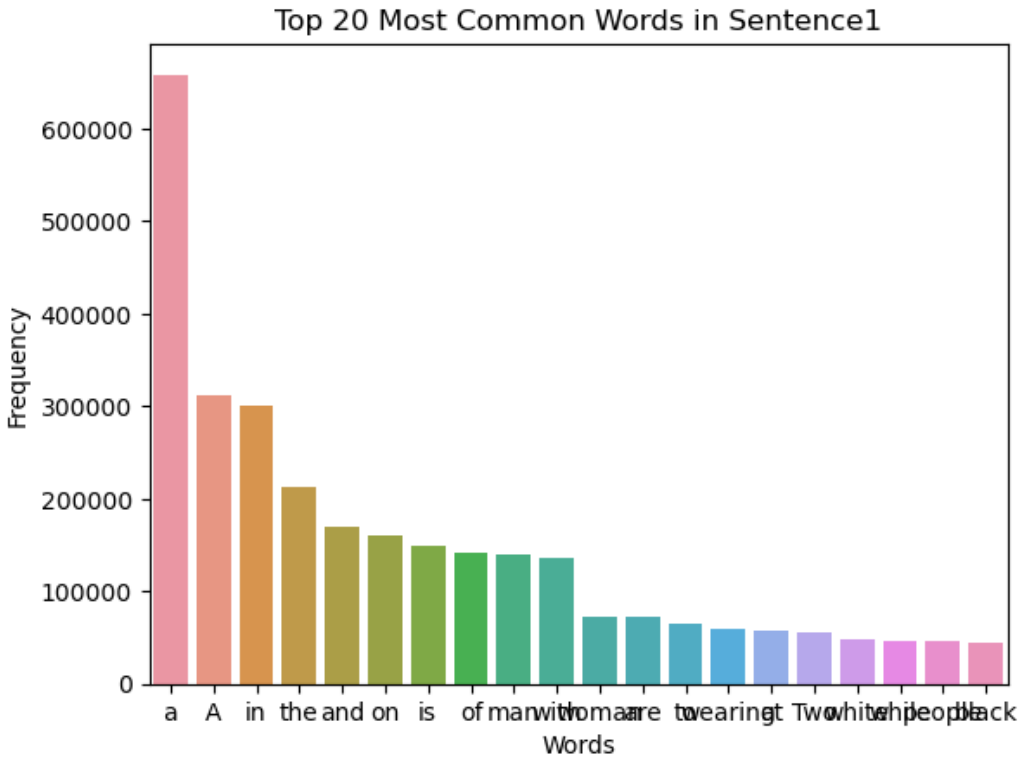

Figure 4: Word Distribution in `sentence1`

3.3 Experiments with ML/Language Modelling techniques

Experiments have been performed by fitting some ML /Language Modelling Techniques and finding the accuracy in each of the applied methods.

3.3.1 Logistic Regression

To perform logistic regression on a dataset containing premises and hypotheses, the premises and hypotheses are first concatenated. The concatenated data is then transformed using TfidfVectorizer, which includes stop word removal. The resulting data is used to fit a logistic regression model with ***max_iter = 10000***.

The trained logistic regression model is then used to predict the premises and hypotheses of a separate test dataset. The premises and hypotheses of the test dataset are transformed using

the same TfidfVectorizer used during training before being fed into the logistic regression model for prediction.

Classification report and accuracy score is calculated over test data as shown below

```
                precision    recall  f1-score   support

            -       0.00      0.00      0.00       176
contradiction       0.54      0.56      0.55      3237
   entailment       0.55      0.59      0.57      3368
      neutral       0.55      0.51      0.53      3219

     accuracy                           0.54     10000
    macro avg       0.41      0.41      0.41     10000
 weighted avg       0.53      0.54      0.54     10000

Accuracy: 0.5436
```

Figure 5: F1 Score using Logistic Regression

Accuracy on Test Data on applying Logistic regression is 54.36%.

3.3.2 Logistic Regression with Hyperparameter Tuning

We will be continuing with train and test data obtained after processing from TfidfVectorizer, implemented in the Logistic Regression model above.

GridSearchCV [2] is used to tune the hyperparameter and then predict the test data on the best hyperparameters found using grid search.

```
# Hyperparameter tuning
param_grid = {'C': [0.1, 1, 10]}
grid_search = GridSearchCV(lr, param_grid, cv=5, verbose=2, n_jobs=-1)
grid_search.fit(X_train, label_list)

y_pred = grid_search.predict(X_test)
```

The hyperparameter that is being tuned here is the regularization parameter ***'C'*** with 3 different values: [0.1, 1, 10].

For hyperparameter tuning, here we have used the ***GridSearchCV*** from scikit learn library. The parameter ***'cv'*** specifies the number of cross validation folds, ***'n_jobs'*** specifies the number of CPU cores to be used in parallel, and ***'verbose'*** controls the output that will be printed on screen.

The fit method on grid_search object is called on train data and labels to train a Logistic regression model with different hyperparameters using cross validation. The ***'predict'*** method on test data using the best hyperparameters found during gris search.

Classification report and accuracy score is calculated over test data as shown below

```
{'C': 10}
                precision    recall  f1-score   support

            -       0.00      0.00      0.00       176
contradiction       0.54      0.57      0.55      3237
   entailment       0.55      0.58      0.57      3368
      neutral       0.55      0.50      0.52      3219

     accuracy                           0.54     10000
    macro avg       0.41      0.42      0.41     10000
 weighted avg       0.53      0.54      0.54     10000

Accuracy: 0.5441
```

Figure 6: F1 Score using Logistic Regression and Hyper-parameter tuning

Out of the provided 3 values for regularization factor C; the best it can do is at C = 10 and generates an accuracy of 54.41%.

3.3.3 Bi-directional LSTM

To perform the Natural Language Inference task, a Bidirectional LSTM model is utilized. For this, the very first step is to use ***nltk Tokenizer*** maximum vocabulary size as 5000 and out of vocabulary word as ***"OOV"***. Concatenated train premise and hypothesis are then processed to generate word indices from the nltk tokenizer. The generated word indices are used to convert each sentence (concatenated premise and hypothesis) of train, validation and test to their equivalent numerical vector representation.

Before fitting the Neural Language Model, each sentence is padded with zeroes to make each sentence of equal length. Label lists for validation and test data are converted into their equivalent one-hot vector representation prior to being passed to the layers of NLM.

Here we have used ***Keras*** deep learning framework.

The layers that are added to build the NLM for NLI task are -

- The first layer is an Embedding layer that takes the sentences where each sentence is of

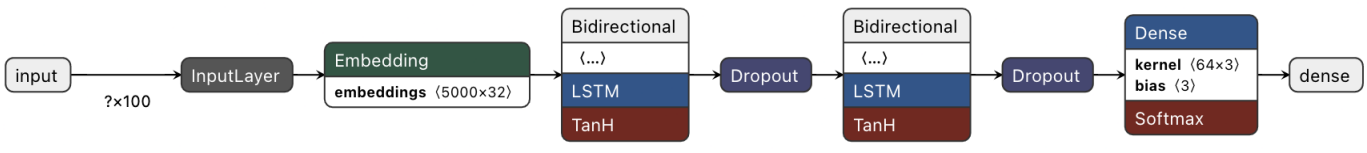

Figure 7: BiLSTM Model Visualisation

size max_len (== 100) and converts each word into a dense vector representation of size 32 (embedding dimension).

- The output from Embedding layer is then forwarded to Bidirectional LSTM layer with hidden dimension size 64.
- The third layer is a dropout layer. Dropout is a regularization technique that randomly drops out a proportion of the units (here 50%) in the layer during training and helps to prevent overfitting.
- The fourth layer is another bidirectional LSTM layer, with 32 units as hidden dimension.
- The fifth layer is another dropout layer, which helps to prevent overfitting.
- The final layer is a dense layer with num_classes units and a softmax activation function. The num_classes parameter specifies the number of classes in the classification problem, (here num_classes = 3) and the softmax activation function outputs the predicted class probabilities.
- The class for which probability is maximum will be selected as the label for premise-hypothesis pair.

The optimizer used is Adam Optimizer along with the Categorical Cross Entropy loss function with accuracy metric. The goal is to minimize the categorical cross-entropy loss and maximize the accuracy metric.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
    accuracy'], run_eagerly=True)
```

The model is trained with applied checkpoints that will save the best model based on validation data accuracy. Also early stopping criteria is applied to prevent overfitting. The batch size taken for training the model is 128.

```

# Train the model
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=3)
mc = ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max',
    verbose=1, save_best_only=True)


history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=20,
     batch_size=128, callbacks=[mc, es])
```

The labels for test premise and hypothesis pair are predicted using the best saved model.

Classification report and accuracy score is calculated over test data as shown below-

	precision	recall	f1-score	support
0	0.54	0.61	0.57	2853
1	0.68	0.61	0.64	3750
2	0.60	0.60	0.60	3212
accuracy			0.61	9815
macro avg	0.61	0.61	0.61	9815
weighted avg	0.61	0.61	0.61	9815

Figure 8: F1 Score using BiLSTM

4 Conclusion of work done so far

Three models were evaluated for the Natural Language Inference task, which involves determining whether a given sentence (hypothesis) can be inferred from another sentence (premise) or not. The models included Logistic Regression, Logistic Regression with hyperparameter tuning, and Bi-directional LSTM. The Results are as follows -

- Accuracy with Logistic Regression comes out to be ***54.36%***
- Accuracy with Logistic Regression with hyperparameter tuning comes out to be ***54.41%***
- Accuracy with Bi-directional LSTM comes out to be ***61%***

5 Comparison against Timelines

In our project outline, we set specific timelines to complete certain tasks, such as conducting Exploratory Data Analysis (EDA) and evaluating the performance of different models.

We promised to complete the EDA by March 8th, and we were able to achieve that goal. We analyzed the SNLI dataset in depth and provided insights into its characteristics, including data visualization. We also discussed the preprocessing requirements for the SNLI dataset and provided code to preprocess the dataset.

Moreover, we implemented Logistic Regression, Bidirectional LSTM, etc. models for the SNLI dataset and compared the performance with each other. By accomplishing these tasks according to the project timeline, we were able to stay on track and ensure timely delivery of project objectives.

6 Work plan till final submission

The further plan is to

- Research different NLI datasets and select a few for analysis. Preprocess and clean the selected datasets by removing stop words, stemming/lemmatization, and removing any unnecessary columns. Conduct exploratory data analysis (EDA) on the cleaned datasets to identify any patterns, trends, and anomalies.
- Evaluate the performance of the baseline model (Logistic Regression) and compare it with other models after experimenting with different models like Gated recurrent units (GRUs), different layers with LSTMs.
- Explore different embeddings like Word2Vec, GloVe, and FastText. Implement these embeddings in the BiLSTM model and evaluate their performance.
- Research and study the state-of-the-art models like BERT and RoBERTa. Implement BERT on the selected dataset and fine-tune its hyperparameters to improve its performance.
- Compare the performance of BERT with the previously selected models.

References

[1] SNLI, MultiNLI, and Adversarial NLI — Stanford CS224U Natural Language Understanding — Spring 2021

[2] Tuning the hyper-parameters of an estimator - Scikit Learn