POS tagging for Bi-dialected Bengali text (Standard Indian and "Barendri" dialect) and a brief comparison of performance of different POS tagging methods on the basis of the corresponding output.

Who?

Saswata Bose
Roll No. - 2023114004

What?

Given a text written in two varieties (dialects) of the same language (here, Bengali), we aim to achieve POS tagging on that corpus using different models and compare their performance on the basis of the output generated, using some commonly used evaluation metrics.

Planned weightages:

- Dataset Creation: 30%
- Modelling: 30%
- Analysis: 30%
- Viva and Report Submission: 10%

Why?

Primary Goal:
To analyse what happens to POS Tagging models when the input data has increased entropy (unpredictability, or randomness)

Possible Enhancement:
To analyse what might happen if the “bijectivity” of a training set is destroyed, i.e, if the same idea/reference has two different realisations in the same corpus (two words in two dialects referring the same thing/concept), how does it affect model performance in various other tasks.

How?

1) Collecting or Developing dataset by annotation on Standard Bengali and “Barendri” Bengali to be used as Gold Dataset
2) Developing models, training and testing them on the dataset. If required, fine tuning may be done on these models.
3) Analysing accuracy of the outputs from each model

When?

The expected timeline, as of now, is as below:

Till 22nd April (Done) – Annotation of Gold dataset, Configuring HMM and CRF
Till 29th April – Configuring and Developing LSTM using PyTorch
Till 6th May – Obtaining Results

Outline

Detailed discussion on the overall procedure applied

Problem Scoping

Bi-dialected code mixed data, to be POS tagged by a single model.

Dataset features ensured:

1) The test set has unknown words with respect to the system, so that the scalability is measured

2) Equal number of tokens from both dialects have been taken.

Data Acquisition

1) Native speakers of the dialect were contacted through personal connections
2) Each one of them were asked to translate a story (about 500 tokens each) from standard Bengali to “Barendri” Bengali
3) Inter-translator Agreement was established
4) Standard Bengali dataset was obtained from previously provided data

Data Exploration

1) Data was manually converted to TSV format from oral and written means given by the translators
2) The POS Tagging was done manually in accordance to the BIS Tagset
3) The Data was converted to CoNLL-U format.

Assumptions:
A word has been uniformly given a single tag based on its maximum occurrence. However, there exist certain words which have completely different meaning, but same spelling and pronunciation across the dialects
Eg:
“আলো” (Alo):
Light (In Standard Bengali)
To come (In Barendri Bengali, derived from “আসলো” (Ashlo) in Standard Bengali)

Modelling

3 POS Tagging approaches were used and accordingly results were obtained

1) Hidden Markov Model (HMM)
2) Conditional Random Fields (CRF)
3) Long Short Term Memory (LSTM)

Evaluation

The following metrics were used to evaluate the models:

1) Accuracy (and Recall, which calculate to be the same value for a POS tagger)
2) Precision
3) F1 Score

Assumptions, Bottlenecks, Problems, And Issues

1) The program executing LSTM requires time for getting a decent output. Therefore multiple random samples were used for testing. However, the program is still executing the tagging using LSTM on the complete dataset whose results, if obtained in reasonable amount of time would be duly submitted in the report.

2) CRF++ toolkit has been used for tagging using CRF algorithm. However, it has issues in rendering Bengali alphabets. As a result, the issue was ignored and the tags were used.

3) Keeping in mind the humongous training time required in training these algorithms for POS tagging, out of 3154 total tokens acquired for “Barendri” Bengali, only 727 tokens were used for training and 440 tokens were used for testing. Therefore, it was tried to have data of high variance to try to cover all possible cases. This has also made the testing data include lots of unknown words.

4) FastText pre-trained word vectors have been used for LSTM implementation, done using PyTorch

Results and Analysis

Null Hypothesis:

1) The normal performance of POS taggers will deteriorate due to increased number of features

2) The performance by LSTM will be the best, followed by CRF - HMM will perform the worst.

3) As is a persistent problem with HMM, a single tag will get repeated a lot of times which would result in decrement of its accuracy.

HMM

```
Accuracy: 7.60%
Recall: 7.60%
Precision: 93.92%
F1 Score: 0.0378377463951235
```

Trained on code-mixed data

```
Accuracy: 0.41505791505791506
Precision: 0.8249284209998496
Recall: 0.41505791505791506
F1 Score: 0.49229975328681025
```

Trained on standard data

HMM has failed drastically, below are the possible reasons for the same, along with some analytical insights:

1) HMM tends to give the same tag to all words with high frequency. As the test data had unknown words, they were wrongly tagged. Also, HMM depends on the Transition Probability, so a wrong tag results in a Domino Effect of continuous wrong tags
2) However, HMM boasts a ridiculously high Precision, which means the model is confident about the few positive instances it predicts but misses a substantial number of actual positives because of the faulty predictions made earlier.

CRF

```
Accuracy: 46.59%
Recall: 46.59%
Precision: 57.51%
F1 Score: 0.4855956533271409
```

Trained on code-mixed data

CRF performs much better than HMM. Also, it can recognise the unknown words to some extent

```
Accuracy: 0.6196911196911197
Precision: 0.6890402808014305
Recall: 0.6196911196911197
F1 Score: 0.5893495350384276
```

Trained on standard data

Variation: MIRA POS Tagging Algorithm

MIRA POS Tagging can be achieved using `--algorithm=MIRA` option in `crf_learn` command. The results are analysed further

MIRA

```
Accuracy: 47.04%
Recall: 47.04%
Precision: 58.02%
F1 Score: 0.4957105285738713
```

We observe that MIRA performs slightly better than CRF in this particular problem, with about 1% improvement in Accuracy, Precision and Recall and 0.01 increment in F1 score as a consequence.

LSTM

As mentioned earlier, the LSTM implementation requires a lot of time to complete execution.
As per the execution status till now:
100 steps of prediction consume about 30 secs of time. For a reliable result, 50000 steps has been considered to be the threshold. This makes the complete execution time of about 4 hours.

As a result sampling has been used.

Sampling Strategy

1) 100 random tokens were selected from the larger training set, to be used as a smaller training set for the model.
2) 10 random tokens were selected from the larger testing set, to be used as a smaller testing set for the model.
3) The model is run and results are achieved
4) This is repeated 5 times
5) All achieved tagged results are computed for the evaluation metrics altogether, considering them to be the result of a single run

Merits and Demerits of Sampling

Merits

Can obtain an average result in lesser time (Obtaining one sample result takes about 25 mins of time, so 5 samples are obtained in about 2 hours)

Demerits

The metric values may vary, and therefore the result is much less reliable as are the results for the other two approaches. Also, the training dataset and testing dataset are much reduced, So, obtained results may show variations from actual results. (As we are using only 5 samples, maximum number of unique data tokens is 500 (about 69% of actual size), similarly only 11% of testing tokens have been used. In this proportion, the resulting data is not reliable, but because of time constraints the actual result can't be provided as of now

LSTM

```
Accuracy: 28.00%
Recall: 28.00%
Precision: 61.83%
F1 Score: 0.2557011494252874
```

Combination of all 5 samples

```
Accuracy: 0.00%
Recall: 0.00%
Precision: 90.00%
F1 Score: 0.0
```

```
Accuracy: 30.00%
Recall: 30.00%
Precision: 75.00%
F1 Score: 0.27142857142857146
```

```
Accuracy: 30.00%
Recall: 30.00%
Precision: 76.00%
F1 Score: 0.2777777777777778
```

```
Accuracy: 30.00%
Recall: 30.00%
Precision: 100.00%
F1 Score: 0.36
```

```
Accuracy: 50.00%
Recall: 50.00%
Precision: 67.00%
F1 Score: 0.41666666666666663
```

Runtime Analysis of the LSTM execution

During the runtime of the LSTM function analysing the complete dataset, we can make the following deductions:
1) The training loss per 100 steps is in the order of 0.001.
2) The change in training loss is in every 100 steps is between the order of 0.001 and 0.0001.

It is expected that LSTM will outperform CRF, but the same can’t be tested until the LSTM execution is completed.

Variation: LSTM POS Tagging Algorithm

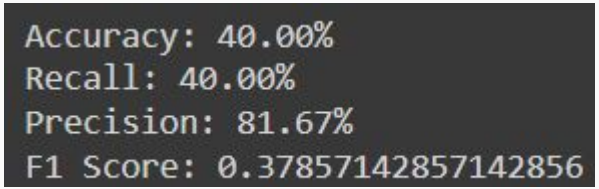
Optimiser Variation

We compare 3 different Optimisers:

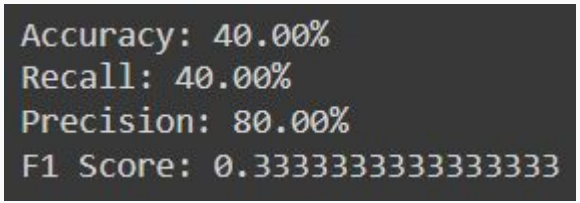
1) Adagrad
2) Adadelta
3) Adam

Optimiser Comparison

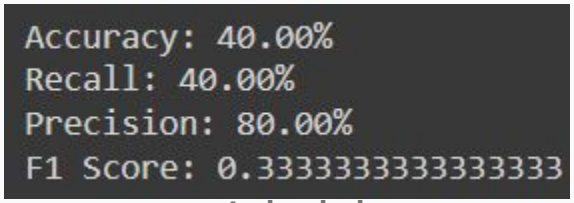
Optimiser	Result (F1 Score)
Adam	0.3786
Adadelta	0.3333
Adagrad	0.3333



Adam



Adagrad



Adadelta

Null Hypothesis Verification

1) The scores are relatively lower than the natural scores of the POS taggers.
2) CRF has indeed performed better than HMM, however the decision on LSTM is subject to the code output.
3) A single tag was repeatedly tagged by HMM in certain occasions, decreasing its accuracy.

THANK YOU