Intro to NLP

Project : Text Fluency

Team Members
Shravan Sharma 2021201058
Vineet Agrawal 2021201049
Sankalp Thakur 2021201042

Introduction

Text fluency is an estimate of text's quality, readability, accuracy, and comprehension. When text is generated by machine it is prone to be erroneous and must be corrected by human intervention. To check if the text generated is of good quality fluency provides a measure.

In this project we try to classify the text data into 3 classes Fluent, neutral and non fluent .We do this using supervised algorithms. We use both traditional as wel as deep learning approaches for this purpose.

Dataset

- We used Microsoft Compression Dataset. This dataset contains sentences and short paragraphs with corresponding shorter versions.
- There are up to five compressions for each input text, together with quality judgements of their meaning preservation and grammaticality.
- The dataset contains around 23 thousand shorter version which are unique.
- Average meaning and grammar scores for every sentence range from 1 to 3. These scores are average of scores given by human judges.

Methodology

- We take on this problem as multi-class classification, with classes being FLUENT, NEUTRAL and NOT FLUENT.

Fluency Score	Represents	No. of Sample
0	Not-fluent	19402
1	Neutral	3352
2	Fluent	127

Methodology

- First we calculate metrics for every sentences.We use BLEU score , ROUGE-L , ROUGE-N, n-gram-overlap and find the correlation between different values of weighted sum of avg meaning and grammar.
- The weights that has highest amount of correlation are chosen to calculate fluency.
- We found highest positive correlation was with following weightage

$$Fluency = closest\_integer(0.2 * meaning_{score} + 0.8 * grammar_{score})$$

Methodology

```
averageMeaning weight : 0.0 , averageGrammar weight : 1.0  correlation to fluency : 0.08835025225926323
averageMeaning weight : 0.1 , averageGrammar weight : 0.9  correlation to fluency : 0.08983207992680046
averageMeaning weight : 0.2 , averageGrammar weight : 0.8  correlation to fluency : 0.09025994831472317
averageMeaning weight : 0.3 , averageGrammar weight : 0.7  correlation to fluency : 0.08918458219569154
averageMeaning weight : 0.4 , averageGrammar weight : 0.6  correlation to fluency : 0.08625491168806287
averageMeaning weight : 0.5 , averageGrammar weight : 0.5  correlation to fluency : 0.08138502484553721
averageMeaning weight : 0.6 , averageGrammar weight : 0.4  correlation to fluency : 0.07484737103806909
averageMeaning weight : 0.7 , averageGrammar weight : 0.3  correlation to fluency : 0.06720266385791064
averageMeaning weight : 0.8 , averageGrammar weight : 0.2  correlation to fluency : 0.05910413484254443
averageMeaning weight : 0.9 , averageGrammar weight : 0.1  correlation to fluency : 0.05110903357203279
averageMeaning weight : 1.0 , averageGrammar weight : 0.0  correlation to fluency : 0.04358919350254622
```

Methodology

- We did minimal preprocessing because fluency of sentence must be calculated as it is for the sentence. The scores were given by humans according to the output generated by the algorithms. Tampering with the punctuation or stop words would make the data unreliable.
- So we only converted the sentence to lower case and tokenized the sentences.
- We made the train test split of ratio 2:1 .

Methodology

- Since the data was Unbalanced we used 3 sampling techniques to balance the data.
- These techniques are:
  - SMOTE
  - SMOTE Borderline
  - ADASYN

SMOTE (Synthetic Minority Oversampling Technique)

SMOTE is an oversampling technique that generates synthetic samples from the minority class. It is used to obtain a synthetically class-balanced or nearly class-balanced training set, which is then used to train the classifier. The SMOTE samples are linear combinations of two similar samples from the minority class

Borderline SMOTE

This algorithm starts by classifying the minority class observations. It classifies any minority observation as a noise point if all the neighbors are the majority class and such an observation is ignored while creating synthetic data . Further, it classifies a few points as border points that have both majority and minority class as neighborhood and resample completely from these points

ADASYN

ADASYN is a more generic framework, for each of the minority observations it first finds the impurity of the neighborhood, by taking the ratio of majority observations in the neighborhood and k.this impurity ratio is converted into a probability distribution by making the sum as 1. Then higher the ratio more synthetic points are generated for that particular point.

Methodology

- We used 4 methods for classification.
  - Non-DL classifiers
  - Non-DL classifiers using POS Tagging
  - LSTM using GLove Embeddings
  - BERT Encoding

Non-DL classifiers

- Converted tokenized text into Sequence and padded the sequences to get inputs of equal lengths.
- Used 4 Classification algorithms:
    - Logistic regression
    - SVM
    - Decision Tree
    - XGBoost

Non-DL classifiers using POS - Tagging

- Converted the text into Part of speech tags. And then used the same algorithms used in Text2Sequence method to classify the text.
- Using Part of speech makes the data more generic and can potentially provide better results.
- Tried TF-IDF but took to long to train due to large vocab size

LSTM using Glove embeddings

- Created model using TensorFlow and added glove embedding to the model.
- Used embeddings of size 300
- Tried both LSTM as well as BI-LSTM.

BERT Encoding

- Used BERT to get encoding for every word and then took mean to get embedding of a sentence giving a fixed size representation for every sentence.
- Used FCNN to classify the data.

Analysis (Non-DL)

Logistic regression

	Precision	Recall	Accuracy	F1-score
None	0.48	0.35	0.85	0.34
Smote	0.35	0.37	0.57	0.32
Borderline	0.35	0.36	0.62	0.33
Adasyn	0.36	0.38	0.58	0.32

SVM

	Precision	Recall	Accuracy	F1-score
None	0.48	0.35	0.85	0.35
Smote	0.44	0.34	0.84	0.34
Borderline	0.37	0.36	0.81	0.34
Adasyn	0.40	0.35	0.83	0.34

Analysis (Non-DL)

Random Forest

	Precision	Recall	Accuracy	F1-score
None	0.49	0.37	0.86	0.37
Smote	0.41	0.38	0.83	0.39
Borderline	0.41	0.38	0.83	0.39
Adasyn	0.41	0.38	0.83	0.39

XGBOOST

	Precision	Recall	Accuracy	F1-score
None	0.47	0.37	0.85	0.38
Smote	0.38	0.38	0.79	0.38
Borderline	0.39	0.39	0.80	0.39
Adasyn	0.38	0.38	0.80	0.38

Analysis (Non-DL)

- Logistic regression and SVM perform poorly on data
- Tree based algo perform better.
- While the accuracy is comparable, both XGBoost and Random Forest have better metrics than logistic regression and svm
- Random Forest have better average precision while XGBOOST has better recall when oversampled using borderline

Analysis (Non-DL POS)

Logistic regression

	Precision	Recall	Accuracy	F1-score
None	0.28	0.33	0.85	0.31
Smote	0.34	0.33	0.42	0.26
Borderline	0.34	0.35	0.53	0.30
Adasyn	0.33	0.32	0.42	0.26

SVM

	Precision	Recall	Accuracy	F1-score
None	0.28	0.33	0.85	0.31
Smote	0.33	0.34	0.69	0.33
Borderline	0.34	0.35	0.58	0.32
Adasyn	0.34	0.35	0.55	0.31

Analysis (Non-DL POS)

Random Forest

	Precision	Recall	Accuracy	F1-score
None	0.49	0.37	0.86	0.38
Smote	0.46	0.39	0.86	0.40
Borderline	0.46	0.38	0.85	0.40
Adasyn	0.46	0.38	0.85	0.40

XGBOOST

	Precision	Recall	Accuracy	F1-score
None	0.48	0.39	0.86	0.39
Smote	0.42	0.39	0.83	0.40
Borderline	0.32	0.49	0.84	0.40
Adasyn	0.42	0.39	0.83	0.40

Analysis (Non-DL POS)

- Again we see similar trend where Tree based algorithms outperform other 2 algorithms.
- Using we get slightly better results when compared to previous method.

Analysis (GLove)

Glove

	Precision	Recall	Accuracy	F1-score
None	0.53	0.37	0.86	0.38
Smote	0.46	0.38	0.85	0.39
Borderline	0.46	0.38	0.85	0.39
Adasyn	0.48	0.39	0.85	0.40

Analysis (BERT)

BERT

	Precision	Recall	Accuracy	F1-score
None	0.34	0.33	0.85	0.31
Smote	0.35	0.38	0.45	0.29
Borderline	0.36	0.38	0.61	0.35
Adasyn	0.35	0.38	0.46	0.29

Analysis

- LSTM with GLove performed better, just like XGBoost and Random Forest.
- Best model was trained when using Adasyn sampling
- Precision - 0.48 Recall - 0.39 and F1 score - 0.40 is best combination for model.
- Bert Performed Poorly. Possible Reason - instead of taking individual word embeddings, mean of the word embeddings was assumed to be sentence embeddigs.

Conclusion

- The dataset was highly imbalance and could not be efficiently divided into classes.
- A better dataset required which has higher number of non fluent samples which are also distinguishable is required to achieve the task of fluency classification.
- Over sampling helped to an extent but was not very beneficial.