Team 27: Measure Text Fluency

Thota Gokul Vamsi, 2019111009

Sai Akarsh C, 2019111017

Abstract

This report consists of the description of work done so far, findings and inferences of implementations which were tried out, further work plan of the project.

1. Introduction

The project we are working on is 'Measuring Text Fluency'. The importance and relevance of addressing this problem has increased to a significant extent, due to ever-growing amount of textual content in the world.

Text fluency refers to an estimate which corresponds to the quality, readability, accuracy and comprehensibility of a piece of text. It is an obvious fact that any such textual information generated by an automated (non-human) source is prone to errors of language understanding, and requires manual rectification. Measuring text fluency is an important task to address this issue, as it describes if the required information is conveyed in a desirable format, from a given piece of text. This requires taking into account various criteria such as grammar, style, choice of words etc. and hence is a relatively challenging and non-trivial task.

We address this problem as a multi-class classification problem, where a piece of text is classified either as Not Fluent, Neutral or Fluent. We also use some methods which take manual references and annotations into account, for measuring fluency. So far, we have worked on understanding different approaches which were used to address this challenge, by conducting a thorough literature survey, implemented the baseline and performed various experiments using well-known supervised machine learning algorithms, and performed a detailed analysis of the observations.

2. Dataset

We worked with the Microsoft Compression Dataset which contains about 23k data samples, where each sample contains a piece of text (1-2 sentences or a paragraph) and a manually added reference, to verify the fluency score of the corresponding text. Each sample was further manually annotated on a scale of 1 to 3, where 3 meant very fluent and 1 meant not fluent - and this fluency score was computed for both meaning and grammar of the sentence.

We have considered a weighted average of the both annotations, by assigning a weightage of 0.7 to grammar and 0.3 to meaning of the text, thus achieving a single score for every data sample. This score was further rounded off to nearest integer, which acts as the label of the sample. If this value was 1, it was considered to be 'non-fluent', a score 2 indicated 'neutral' and 3 indicated that it was 'fluent'. This has hence been approached as a 3-class classification problem, for validating various ideas and experiments.

3. Methodologies

For the baseline, we have implemented an n-gram language model (n = 3) which is trained on the train set, and is used to obtain probability scores for all n-grams in a sentence, and uses few highest and least probability values as features as done in this paper. Logistic Regression is used to perform the 3-class classification based on these features. Further, different scoring functions used in the work done by Liu et al. are also utilized to compute the correlations of necessary features.

Various experiments which were performed are enumerated below:

- Attempted different values for parameters such as number of features considered, n in n-grams, probability thresholds used in scoring functions, i.e, considering only those probability features for a piece of text which are greater than / less than a certain threshold - for fluency prediction.

- Extracted textual features alone (using Tf-idf vectorizer) to understand how these features play a role in classification. Previously mentioned classifiers were

employed here too.

- Implemented addition of POS (part of speech) n-gram features, and performed classification based on it using machine learning approaches.

After obtaining data from the required dataset, we analyzed it and applied appropriate pre-processing as necessary. One challenge with the dataset is that it is highly skewed with respect to the labels, where very few samples correspond to non-fluent texts and majority of samples correspond to fluent texts. For tackling class imbalance, we have tried two different over-sampling approaches which balance the class distribution - **Random over sampling** and **SMOTE (Synthetic Minority Over-sampling Technique)**. They are described briefly below:

- **Random Oversampling**: Random oversampling involves randomly duplicating examples from the minority class and adding them to the training dataset. Examples from the training dataset are selected randomly with replacement. This process can be repeated until the desired class distribution is achieved (in this case, there should eventually be equal number of examples corresponding to each class).
- **Synthetic Minority Over-sampling Technique (SMOTE)**: This technique aims to balance class distribution by randomly increasing minority class examples by replicating them. SMOTE synthesizes new minority instances between existing minority instances. It generates new training samples by linear interpolation for the minority class. In other words, these synthetic training records are generated by randomly selecting one or more of the k-nearest neighbors for each example in the minority class, and applying linear operations in the desired way, thereby increasing minority samples.

Data was initially shuffled, and split in a 70-15-15 ratio for training, validation and testing respectively. Different metrics like accuracy, precision, recall, f1-score, confusion matrix etc. were logged for analysis of a model's / scoring function's performance under different scenarios with respect to data sampling.

4. Analysis

The stats and their corresponding analysis, for each idea / experiment implemented, are explained below.

4.1. Baseline

As previously explained, the baseline approach used for this paper, primarily relies on a trained n-gram language model and n-gram probabilities of a sentence, where n is a pre-determined value. To fix the number of features, it was implemented such that only K highest and K least values of probabilities would be used as features, where K is chosen based on analysis of the data and lengths for each piece of text / sample in the data. The values of n and K are chosen to be 3 and 12 respectively, based on observations from the data. These probability features along with perplexity scores are used in the classification process. The results achieved on using different sampling techniques (without any oversampling, Random Oversampling, SMOTE) can be viewed below. This classification is highly affected by the bias in the dataset, leading to majority class / single class prediction (which explains low macro accuracy metrics).

Type	Accuracy	Recall	Precision	F1-score
None	0.871	0.333	0.290	0.310
Random Over	0.871	0.333	0.290	0.310
SMOTE	0.126	0.333	0.041	0.075

Further, there are 2 different scoring functions which were implemented to observe their correlation with fluency. The first scoring function simply considers the sum of log probabilities for all n-grams in the text, divided by number of tokens in the text (to normalize by text length). The Pearson correlation score of this function was found to be 0.0114 approximately, which indicates a positive correlation value between probabilities of an n-grams' occurrence and fluency.

The second scoring function considers only those probability values, if they either represent a rarely occurring n-gram (probability below a threshold) or a relatively frequently occurring n-gram (probability higher than a different threshold). The Pearson correlation score of this function was found to be -0.002 which indicates a negative weaker correlation, due to fact that the score is actually scaled by probability values based on the thresholds they satisfy.

4.2. Different parameters in N-gram approach

Sampling Type	n	K	Accuracy	Recall	Precision	F1-score
None	3	12	0.871	0.29	0.333	0.31
None	3	16	0.871	0.29	0.333	0.31
None	4	12	0.871	0.29	0.333	0.31
None	4	16	0.871	0.29	0.333	0.31
Random Over	3	12	0.871	0.29	0.333	0.31
Random Over	3	16	0.871	0.29	0.333	0.31
Random Over	4	12	0.871	0.29	0.333	0.31
Random Over	4	16	0.871	0.29	0.333	0.31
SMOTE	3	12	0.126	0.042	0.333	0.075
SMOTE	3	16	0.126	0.042	0.333	0.075
SMOTE	4	12	0.126	0.042	0.333	0.075
SMOTE	4	16	0.126	0.042	0.333	0.075

Here n is the N-gram size used, K is used in calculating the feature vector. The feature vector is made of k most and least frequently occurring N-gram probabilities.

From the results obtained above, it can be observed that there is not much change on varying n and K parameters. This indicates that the model used for feature extraction is not able to capture the details properly. We can see a significant reduction in performance when we use SMOTE as the sampler.

4.3. Supervised methods on textual features

Different supervised machine learning algorithms such as Decision Trees, Random Forest, Support Vector Machine, K-nearest neighbours were implemented by taking simply the textual features related to word counts and scores into account. Tf-idf vectorizer was utilized to obtain vectors for each data sample, where each dimension corresponds to a word in the vocabulary. The vocabulary size was fixed to be 6000 due to computational constraints. The results achieved on using different sampling techniques (without any oversampling, Random Oversampling, SMOTE) for each algorithm can be viewed below.

K-nearest neighbours

Type	Accuracy	Recall	Precision	F1-score
Train (None)	0.875	0.333	0.292	0.311
Test (None)	0.872	0.333	0.291	0.31
Train (Random Over)	0.645	0.645	0.651	0.592
Test (Random Over)	0.233	0.325	0.335	0.167
Train (SMOTE)	0.664	0.664	0.494	0.552
Test (SMOTE)	0.124	0.344	0.046	0.081

Decision Tree

Type	Accuracy	Recall	Precision	F1-score
Train (None)	0.893	0.564	0.667	0.604
Test (None)	0.789	0.361	0.358	0.359
Train (Random Over)	0.913	0.913	0.927	0.912
Test (Random Over)	0.685	0.358	0.345	0.339
Train (SMOTE)	0.958	0.958	0.959	0.959
Test (SMOTE)	0.786	0.362	0.356	0.358

Random Forest

Type	Accuracy	Recall	Precision	F1-score
Train (None)	0.893	0.437	0.772	0.483
Test (None)	0.834	0.355	0.393	0.36
Train (Random Over)	0.913	0.913	0.927	0.912
Test (Random Over)	0.693	0.357	0.345	0.34
Train (SMOTE)	0.959	0.959	0.959	0.959
Test (SMOTE)	0.8	0.36	0.362	0.361

Support Vector Machine

Type	Accuracy	Recall	Precision	F1-score
Train (None)	0.883	0.354	0.609	0.352
Test (None)	0.866	0.336	0.361	0.32
Train (Random Over)	0.97	0.97	0.972	0.969
Test (Random Over)	0.809	0.359	0.361	0.359
Train (SMOTE)	0.973	0.973	0.973	0.973
Test (SMOTE)	0.825	0.357	0.362	0.355

Based on the above results, we can see that extracting bag of words features from a particular piece of text could also be effective in evaluating its fluency. Using oversampling techniques has boosted training performance significantly (especially SMOTE), although test performance has deteriorated (significantly in K-nearest neighbours). We can see that test sets tend to have low macro value of precision, recall and f1-score, which highlights the skew in the data.

We can observe that K-nearest neighbours performs baldy in comparison with other approaches tried (especially when oversampling is done), while Support Vector Machine tends to achieve decent test accuracies.

4.4. POS n-gram features

For every piece of text in the data, individual sentences in the text are considered, and POS-tagging is performed such that more grammatical information corresponding to a word is available. This tagging was performed every word in the data, such that the data is now transformed such that every word is viewed as its corresponding POS tag, such as NNP (proper noun, singular), PRP (personal pronoun) etc. Further, an n-gram language model was trained on this transformed data, and was utilized to compute n-gram probability features for every textual sample as done in the baseline (considering K highest and K lowest values as features). These POS features were augmented with the actual n-gram probability features, and various algorithms such as Decision Trees, Random Forest, Support Vector Machine, K-nearest neighbours are used for the classification process. The results achieved on using different sampling techniques (without any oversampling, Random Oversampling, SMOTE) for each algorithm can be viewed below.

K-nearest neighbours

Type	Accuracy	Recall	Precision	F1-score
Train (None)	0.875	0.333	0.292	0.311
Test (None)	0.871	0.333	0.29	0.31
Train (Random Over)	0.582	0.582	0.571	0.551
Test (Random Over)	0.354	0.33	0.339	0.235
Train (SMOTE)	0.571	0.571	0.579	0.528
Test (SMOTE)	0.261	0.328	0.337	0.193

Decision Tree

Type	Accuracy	Recall	Precision	F1-score
Train (None)	0.893	0.57	0.672	0.61
Test (None)	0.778	0.36	0.354	0.357
Train (Random Over)	0.914	0.914	0.928	0.912
Test (Random Over)	0.69	0.355	0.343	0.338
Train (SMOTE)	0.958	0.958	0.959	0.958
Test (SMOTE)	0.779	0.359	0.352	0.354

Random Forest

Type	Accuracy	Recall	Precision	F1-score
Train (None)	0.893	0.456	0.725	0.508
Test (None)	0.833	0.356	0.384	0.359
Train (Random Over)	0.914	0.914	0.927	0.912
Test (Random Over)	0.694	0.355	0.343	0.339
Train (SMOTE)	0.958	0.958	0.959	0.958
Test (SMOTE)	0.8	0.358	0.357	0.357

Support Vector Machine

Type	Accuracy	Recall	Precision	F1-score
Train (None)	0.875	0.333	0.292	0.311
Test (None)	0.871	0.333	0.29	0.31
Train (Random Over)	0.343	0.343	0.41	0.237
Test (Random Over)	0.746	0.325	0.337	0.304
Train (SMOTE)	0.347	0.347	0.44	0.24
Test (SMOTE)	0.767	0.333	0.337	0.308

It can be said that the addition of POS n-gram features was useful, as there is an improvement in performance with respect to baseline and related experiments. We can see that the performance in all algorithms was better in cases where no oversampling was done, due to the bias in the data. Although, oversampling has improved training performance significantly (note that the test set performance deteriorated). It also worth observing that the precision and recall in all test sets are very poor, as the macro averages were considered and the results were mostly dominated by a single label.

We can observe that Decision Tree and Random Forest approaches seem to achieve better results in comparison with K-nearest neighbours and SVM.

5. Code

The link to the github repository with the complete implementation can be found here.

6. Progress with respect to tentative timeline

With respect to this phase, all the tasks planned have been completely in sync with the initial timeline proposed in the project outline document.

- Baseline model implementation (n-gram, LR), experiments with scoring system and parameters of work

done by Liu et al. in this paper, correlation score computation

- Experimentation with supervised approaches - SVM, DT, RF, KNN via textual features
- Experimentation by adding multiple features - POS n-grams, frequency of rare/frequent n-grams in sentence.

7. Further plan

For the next phase, we intend to work on the following list of tasks.

- Implementing key metrics for text fluency such as ROUGE-S, ROUGE-L, SLOR, n-gram overlap for RNN and LSTM language models.
- Implementing above metrics for transformer language model.
- Implementing application utilizing best approach / ensemble for fluency prediction.
- Detailed analysis of all experiments and results - final report.

Hence, the final deliverables we intend to submit are enumerated below

- Final report
- Entire code for baseline, all experimentations, implementation of metrics and their functioning with various LMs - with README
- Relevant model checkpoints
- Application for real-time fluency prediction