CS 229 Project Report: Text Complexity (Natural Language)

Harry Sha (harry2), Tyler Yep (tyep)

December 18, 2018

1 Introduction

The goal of our project is to explore text complexity in the context of machine learning. More specifically, we will answer the following questions:

1. What features of the text are most relevant to this classification?
2. To what extent can machine learning methods be used to classify the complexity of a document?
3. How can we build a model to generate or transform text into different levels of complexity?

This project's outcomes have the potential of enhancing education immensely. Complexity-classified documents allow students to find papers or conceptual explanations at understandable difficulty level. Generating or transforming text into simpler levels of complexity encourages more widespread knowledge, approachable from different fields and backgrounds. Students gain the power to understand big picture ideas and ramp up the difficulty level as they see fit, ultimately resulting in a more personalized educational experience.

2 Related Works

There has already been some success in using ML for text complexity classification. One paper from the University of Washington [4] used SVMs in order to assess reading level between texts from 2nd to 5th grade. They found the most success using features like average sentence length and word counts. Specifically for our dataset, papers published by the University of Wolverhampton [6] found success in using Random Forests on the Weebit corpus, which we intend to experiment with. However, they also pull a substantial amount of outside texts in order to supplement their training, which we do not have, so we may not find as much success using their algorithms. Finally, one study from a German university thesis [1] used SVMs on the Weebit corpus to varying degrees of success - opting for a simplified model to provide high-level insights. From these past works, we see a great opportunity in trying out newer ML algorithms, like AdaBoost or RNNs, and see how they compare to previously-used ones.

3 Data: Feature Extraction and Selection

We are using the Weebit Dataset [5], which has 2226 example texts separated into 3 different reading levels. Each text is roughly 1-5 paragraphs, and is already classified into one of the three reading levels. The input of our algorithm is a nonfiction text document of roughly 1-5 paragraphs, and the output is one of the three reading levels as a measure of its complexity. We will try several different machine learning algorithms such as Logistic Regression, AdaBoost, k-NNs, and variations of neural networks to predict our output.

Data Preprocessing To preprocess the data, we removed new line characters, set all words to be lowercase and removed any disclaimers. We also split the dataset into training, validation and test sets. For the following section, let x represent one example of a preprocessed text. Word count and Tf-Idf feature extraction were completed using sci-kit learn [3].

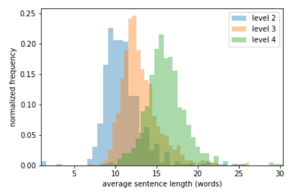
Word Count We used a binary and a regular word count feature extractor. We also experimented with changing min_{df}, and max_{df}, which represent the minimum or maximum document frequencies of a word in order to be included as a feature. Empirically, we found that model performance was not very sensitive to the min_{df} and max_{df} parameters. However, the binary word counts option substantially increased accuracy. In our analysis, we set $min_{df} = 5$, $max_{df} = 80\%$, and tried both binary and non-binary word counts.

Tf-Idf Tf-Idf extracts the word count weighted by a measure of inverse document frequency (Idf). This diminishes the importance of common words such as 'a', and 'the', and highlights the importance of uncommon words. However, we found that the Tf-Idf features gave worse performance than the word count feature extractor. One possible reason for this is that Tf-Idf creates feature vectors which are more similar in their topic/meaning than in their structure. In our task, the topic/meaning of the text may not be as important as the ordering and structure of the words.

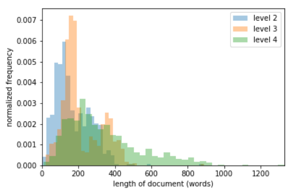
Natural Language Features We also added features for the counts of each part of speech using spaCy [2]. We also added average sentence length, number of sentences, and average word length to our features array. For simplicity, we decided not to include common readability metrics like Flesch-Kincaid score, because we are already using features used to calculate those scores. Empirically, in models like logistic regression, we found that the optimal features were a concatenation of word count with the parameters described above, and the natural language features.

3.1 Basic Analytics on Natural Language Features

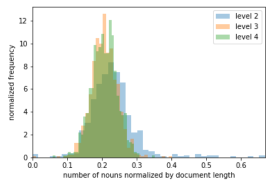
We first wanted to find which of the natural language features were the most promising candidates for complexity classification. Figure 1 shows the distributions of several features in each complexity level. We see that average sentence length and document length increased with the complexity classification. Furthermore, we see that document length explains much of the variance for other features as seen by comparing 1c and 1d. Though average sentence length is a valuable feature to use in classification, it is by no means a perfect indicator, as more than 50% of the data lies in the overlapping region between levels shown in Figure 1a.



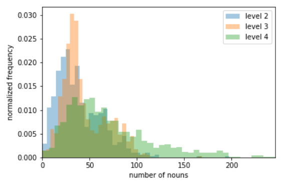
(a)



(b)



(c)



(d)

Figure 1: Natural Language Features

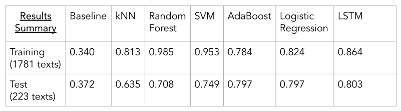
4 Model Selection

Baseline Our baseline model used a Dummy classifier to randomly predict results based on the probability of each complexity of each text appearing. In our dataset, 630/2226 examples were level 2, 789/2226 examples were level 3, and 807/2226 examples were level 4. Our baseline obtained 37.2% accuracy on the test set.

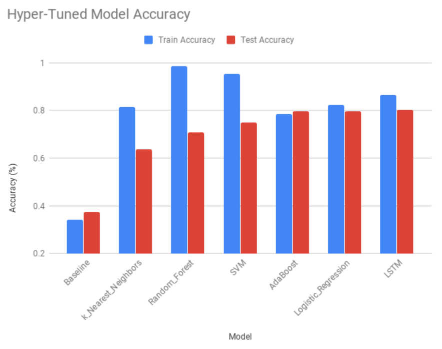
Logistic Regression Logistic Regression was very successful, with the highest accuracy on the validation set at 79.9%. The hyperparameters for Logistic Regression were type of regularization (L_1 or L_2), and the amount to regularize by, $1/C$. We conducted a grid search, trying C using powers of 10 between 0.001 and 100. We found that L_1 regularization generally performed better than L_2 regularization. This is likely because L_1 results in sparse weights, which is advantageous in reducing the effects of less-useful features.

AdaBoost Another successful classifier was AdaBoost. Given the relatively high results we obtained from using only average sentence length as a feature, we expected an ensemble of basic classifiers to perform much better. After tuning the number of classifiers and the learning rate, AdaBoost achieved 79.7% on the validation set. In Figure 2, we have a plot of AdaBoost test accuracy using different learning rates and estimators (using only word count + natural language features).

Other Other algorithms we tried, such as Naive Bayes or k-Nearest-Neighbors, performed better than our baseline, but did not have as much initial success as AdaBoost and Logistic Regression, and did not seem to fit our problem as well. For example, they made questionable assumptions of independence or modeled complex documents as clusters, which did not fit with our selected features.



Results Summary	Baseline	kNN	Random Forest	SVM	AdaBoost	Logistic Regression	LSTM
Training (1781 texts)	0.340	0.813	0.985	0.953	0.784	0.824	0.864
Test (223 texts)	0.372	0.635	0.708	0.749	0.797	0.797	0.803



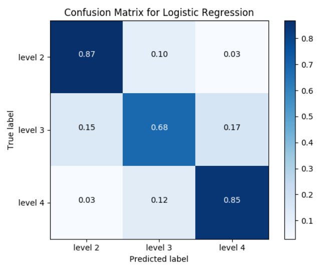


Figure 2: Algorithm Accuracy Results & Confusion Matrix

5 Recurrent Neural Networks

One key drawback in our current representation of the documents is that all sequential information is lost in the feature encoding. In other words, any permutation of a document's words results in the same feature vector. However, sequential relationships likely play a key role in determining the complexity of a text. We did not use sequential encodings originally because models such as Logistic Regression expect fixed-length inputs. Our documents were of different lengths, so using sequential encodings would require us to add padding to all of our documents. However, the high variance in document length made padding difficult.

To address this problem, we will use a Recurrent Neural Network. The architecture of this neural network allows for arbitrary length inputs, and have been successfully applied in NLP in the past. For this model, we encode each document as a sequence of POS tags instead of word embeddings, with hopes that this will both allow the model to fit the data better and also generalize to unseen texts. Since there are complex and simple texts of any given topic, we hypothesize that a more important factor in determining complexity is the grammatical structure of the sentence, rather than the content.

5.1 Model Architecture

1. **Encoding.** The encoding we chose was to represent texts as a sequence of POS tags. We also chose to replace the $PUNCT$ tag with the actual punctuation used in the sentence to help distinguish commas and periods, so the algorithm may can learn the difference between a series of simple sentences vs compound sentences. The final vocabulary consists of 46 POS tags and punctuation marks.

2. **Embedding Layer** The first layer of the LSTM is an embedding layer. This is inspired by NLP methods which typically use word2vec or trainable embedding layers to represent each item of the vocabulary. For our case, the embedding layer takes in a element of the vocabulary and maps it to a $EMBED\_DIM$ dimensional vector, which is trained using the optimization algorithm.

3. **LSTM** The embeddings are then put into a LSTM model. The tunable parameters of this step are:

    - N_LAYERS, the number of LSTM layers.
    - $HIDDEN\_DIM$, the dimension of each LSTM layer.
    - $DROPOUT$, the percentage of neurons that are deactivated in each LSTM layer.

The LSTM has an output for each value in the sequence. To get a fixed-length vector for the next linear layer's input, we experimented with either using the output at the final value in the sequence, or using the mean output across all values in the sequence. However, neither seemed to have any noticeable effect on our results.

4. **Linear** The output of the LSTM layer is then fed into a linear layer. The purpose of this layer is to transform the output dimension of *HIDDEN_DIM* to 3, as we are trying to predict 3 levels of difficulty.

5. **Softmax** Finally, the outputs of the linear layer are fed into a softmax layer, which normalizes the outputs so that they can be interpreted as the probabilities that the text was of each of the levels.

5.2 Results

We used the Adam optimizer to train our model and used a randomized grid search to tune our hyperparameters. Some high-level findings were that a learning rate of 0.01 converges much faster, and often does much better than a lower learning rate. We found that increasing hidden layers made the model strongly overfit the data, decreasing performance on the test set. While dropout helped with overfitting, adding more layers still appeared to lower the test set accuracy, even with dropout. Our optimal parameters were: *N_LAYERS* = 1, *BATCH_SIZE* = 16, *EMBED_DIM* = 64, and *HIDDEN_DIM* = 128 with LR = 0.01.

After hyperparameter tuning, we gained our best result pair of 80.3% on the test set and 86.4% on the train set using the LSTM. Overall, the LSTM did roughly 1% better than both AdaBoost or Logistic Regression on the test set. However, our LSTM notably only used natural language features and was still able to obtain better results than the previous classifiers using both natural language features and word count.

6 Text Generation

The final goal of this project was to generate texts of different complexity levels. In this project, we focused on grammar and sentence/document structure as the primary determinant of complexity. We employ a similar LSTM model to before, but for sequence prediction instead of classification. Given the sequence of POS encodings described in the previous section, the model learns $p(pos^{t+1} \mid pos^t, pos^{t-1}, \ldots, pos^1)$. We can sample from this probability distribution to generate sequences of POS tags.

6.1 Training

The hyperparameters we selected for the generation model were *N_HIDDEN* = 100, *N_LAYERS* = 3, *EMBED_DIM* = 64, *LR* = 0.0001. We found that this model was able to adequately learn in the different difficulty levels without being over-complex (taking very long to train), as the training loss curves are shown in Figure 3. Level 3 and 4 loss curves seem to end with a higher loss because there are more level 3 and 4 training examples, and the levels with more varied sentence structure have harder distributions to learn.

6.2 Sampling

Let G be the trained model, V be the vocabulary, $D = |V|$, and x be a document. We approximate G as:

$$G(x_1, x_2, \ldots, x_{t-1}) \approx \begin{bmatrix} p(x_t = V_1 | \{x_j \mid j < t\}) \\ p(x_t = V_2 | \{x_j \mid j < t\}) \\ \vdots \\ p(x_t = V_D | \{x_j \mid j < t\}) \end{bmatrix}$$

where x_t is the t^{th} POS tag in the document x. To sample a sequence of POS tags, we sample from G to get a vector of probabilities, normalize the vector to sum to 1 using our temperature parameter, and then choose tags from the resulting multinomial distribution. Our *temperature* parameter controls the rigidness of the sampling. Low values for temperature result in more grammatically correct sentences, and higher values encourage more diverse sentence structures.

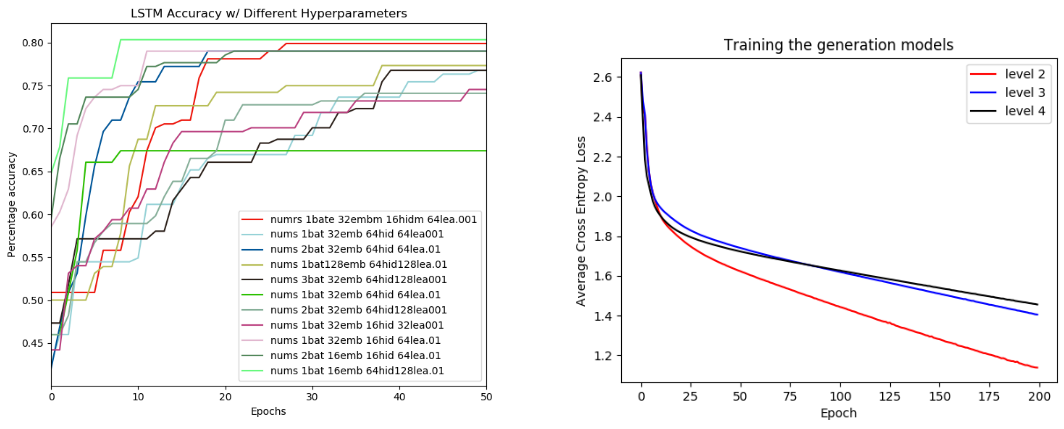


Figure 3: Training curves for classification LSTM (left) and generation LSTM (right)

Finally, we attempted to substitute word values back into our generated sequences. Since we did not have a methodical way to plug words back in, we opted to randomly fill them in with the given text, and then verify their class using our original Logistic Regression classifier (more time-efficient than our LSTM). Using 100 randomly generated POS structures of each level, we took 100 random texts and inserted words of each POS type to train, and then attempted to predict the reading level of 20 more unseen examples. We found that the classification accuracy dropped to 50.6% using our Logistic Regression model.

6.3 Example Generated Sentence

One generated level 2 sentence segment using our trained LSTM:

DET NOUN VERB DET NOUN ADP DET NOUN ADP DET NOUN...

Below is a sample Level 2 sequence of POS tags that we filled in using a level 4 text:

The Giants founded the team with the help of the shelter.

A real level 2 text excerpt, with matched POS tags underlined:

The Giants founded the dog team with the help of a local animal shelter.

7 Conclusion

Overall, our classification algorithms categorize the different levels of complexity in the Weebit corpus with 80.3% accuracy on an unseen dataset using an LSTM, only requiring structural features with POS tags, which is a significant improvement from our baseline or relying on average sentence length alone. Moving forward, we plan to use more types of texts (fiction, biographical, etc.) and add additional features like individual word complexity in order to better understand the content of a passage for classification.

Though we succeeded in generating text from learned examples with our second LSTM, our generation model had a major weakness in finding sensical ways of re-inserting words into our generated POS tag sequence. Furthermore, even after generating these nonsensical sentences, our original classification algorithm could not successfully classify the intended sequences with significant accuracy, implying that there is a deeper influence on the actual content of a passage when determining readability. In the future, we would look for better ways of substituting POS tags, or we may try fixing certain POS words to remove ambiguity in filling in parts of speech, thus improving readability.

8 Contributions, Code

Harry - Word count, Tf-Idf feature extraction, exploration of natural language features, experiments with Logistic Regression, SVM, Naive Bayes, organizational code. Implementation of LSTM models, and sampling method. Training of generation model.

Tyler - Natural language feature extraction, AdaBoost, Naive Bayes, baseline results. For the LSTM models: hyperparameter tuning, plotting loss graphs, sampling and substituting for text generation and creating comparison to original model.

Code can be found at: https://github.com/TylerYep/complex-text.

References

[1] Sowmya Vajjala Balakrishna. Analyzing text complexity and text simplification: Connecting linguistics, processing and educational applications. *Dissertation zur Erlangung des akademischen Grades Doktor der Philosophie in der Philosophischen Fakultät der Eberhard Karls Universität Tübingen*, 2015.

[2] Matthew Honnibal and Ines Montani. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. 2017.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] Sarah E. Petersen and Mari Ostendorf. A machine learning approach to reading level assessment. *University of Washington CSE Technical Report*, Jun 2006.

[5] S Vajjalla and D Meurers. Readability assessment for text simplification: From analysing documents to identifying sentential simplifications. *Recent Advances in Automatic Readability Assessment and Text Simplification ITL - International Journal of Applied Linguistics*, 165(2):194–222, 2015.

[6] Richard Evans Victoria Yaneva, Constantin Orasan and Omid Rohanian. Combining multiple corpora for readability assessment for people with cognitive disabilities. *Research Institute in Information and Language Processing, University of Wolverhampton, UK.*