Autograder: Classifying Documents to Grade School Level

Kiley Yeakel
kileyy10@stanford.edu

Stephanie Tzeng
stzeng@stanford.edu

Abstract

Natural Language Processing (NLP) has been used for language translation, topic modeling, sentiment analysis, and other tasks. We explore using similar techniques for detecting language complexity. The motivation stems from education, where educators often struggle to find reading passages of the appropriate level for their students. In this paper, we show that NLP works just as well for detecting complexity of language, in that probabilities of words or particular groupings of words act as viable predictors of text complexity. Using a set of articles translated into different grade levels as our dataset, we found that Naive Bayes with a logistic regression classifier was able to achieve a 0.076 error on the test set, for the case of detecting "easy" versus "hard" passages. Using language models as a classification task achieved 0.153 error.

1. Introduction

Individualized attention to students can be difficult to provide due to limited resources. Teachers are often overly consumed with lesson planning, with limited amounts of time to find up-to-date reading materials at the appropriate comprehension level for their students.

The education technology company Newsela aims to reduce the burden on teachers by curating daily content from prominent publications. Their content production team translates articles and relevant reading materials into 5 different reading comprehension levels (denoted as grade levels).

We aim to produce an automated reading level classifier using Newsela's data as our training set, which will determine the most appropriate grade level for a given article or text passage. Such a tool could be useful to Newsela's staff as well as to teachers and students directly, allowing teachers to find individualized content for their students or helping educators write content for their target readers.

Currently, a vast array of readability scores do exist. However, common readability scores are typically done by counting syllables, words, and sentences while ignoring the semantic elements of the text.

We attempt to classify complexity of language by using both traditional machine learning techniques (Naive Bayes, recurrent neural networks (RNN)) as well as inferring similarity using probability distributions via language models. We then compare their outcomes and assess which models are best for certain tasks, and compare our results to the current industry standard.

1.1. Data

We used a subset of Newsela's database, which contains 2,155 articles, each of which are decomposed into 5 different grade levels (from grade 2 to 12) for a total of 10,786 articles. Since these have been written by Newsela staff, they have been manually labeled into different grade levels. For example, the original article text, assumed to be 12th grade, is:

> The group pushing to replace President Andrew Jackson with a woman on the $20 bill has revealed its final four candidates after more than 256,000 votes were placed.

The 4th grade translation is:

> A group wants to take President Andrew Jackson's face off the $20 bill. It wants to put a woman on the bill instead. It has four final choices of women to go on the $20 bill.

We explored multi-class classification, where we attempted to classify each text segment according to its exact grade level, as well as binary classification where elementary grades were considered "easy", and middle through high school were considered "hard". The original articles in the dataset are assumed to be at grade 12, with no translations to 12th grade. However, this default labeling is inherently noisy, as not all original articles are truly of grade 12. To account for this, we analyzed the binary and multi-class classifications with and without the original set of articles. For the multi-class classification, including the original articles adds 12th grade to our dataset. In all cases, grades 10 and 11 were excluded from the dataset due to extreme class imbalance. Table 1 shows the distribution of articles per grade level.

Grade Level	Number of Articles
2	283
3	616
4	1730
5	1488
6	1171
7	1365
8	1152
9	862
12	2096

Table 1. The number of articles per grade level where all 12th grade articles are simply the original text.

1.2. Dataset Preprocessing

For our models, we split the article corpus into 80% training, 10% validation, and 10% test. Each article was tokenized and lower-cased using the NLTK package. The data was inspected for inconsistencies such as the presence of html links and html tagged elements (such as embedded pictures). Further preprocessing steps (if any) were dependent on the algorithm approach used and are discussed in the relevant model subsections within the methodology.

1.3. Features

Depending on the type of model, we explored doing predictions on a per sentence level, as groupings of 2 or 3 sentences, or the entire article. Each text segment for a given article would then be labeled as the grade level for its encompassing article. From the tokenized article corpus we then created n-grams of varying lengths. Depending on the model, the maximum n-gram length was varied from 1 to 5 words.

2. Methodology

We explored 4 different models for both binary classification and multi-class classification - Naive Bayes, Naive Bayes with Secondary Classifier, Language Models with Kneser-Ney Smoothing and RNNs.

2.1. Naive Bayes

A Naive Bayes (NB) classifier makes the "naive" assumption that for a given text passage with terms $t_1 \dots t_n$ and label y, that the terms t_i are conditionally independent for a given y. Therefore, we can model a given passage as a bag of words in which only the multiplicity of a given term in a document is preserved, but the order of words within a document is ignored. For a collection of documents, we have an associated dictionary corresponding to all of the words (or n-grams) present. If we have N terms in our dictionary, then we can represent the NB assumption as:

$$p(x_1 \dots, x_N|y) = \prod_{j=1}^{N} p(x_j|y)$$

In our particular dataset, the translation of articles to various grade levels results in several of the same words being classified to multiple different grade levels. Therefore, we explored looking at various n-gram lengths under the assumption that n-grams may preserve more of the sentence complexity. Overall, for the NB classifier we explored n-gram lengths varying from 1 to 3 words and text lengths ranging from 1 to 3 sentences as well as the entire article.

2.1.1 Preprocessing

A dictionary was formed for each of the n-gram ranges, with terms appearing in less than 5 documents or greater than 1000 documents ignored. Text passages were then tokenized and converted to count vectors for the corresponding dictionary, i.e., for a dictionary length of N a text passage would be converted to a 1 by N vector for index i corresponding to the multiplicity of dictionary term t_i appeared in the passage. For training the NB classifier, count vectors were then transformed with term-frequency inverse document-frequency weights (TF-IDF). The tf-idf weight for term t in document d is given according to the equation:

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t)$$

where $\text{tf}(t, d)$ is the number of times term t appears in document d and:

$$\text{idf}(t) = \log \frac{1 + N}{1 + \text{df}(t)} + 1$$

where N is the total number of documents in the corpus, and $\text{df}(t)$ is the number of documents in the corpus containing term t. We enabled smoothing (addition of 1 in the numerator and denominator of the idf function as well as to the entire term) to control for the case where a term appears in all documents within the training set and therefore has an $\text{idf}(t) = 0$. As indicated by the above equation, the more times a term appears in a given document, the higher its weight. However, the TF-IDF weight is also inversely proportional to the number of times the word appears in all the documents within a corpus. Therefore, TF-IDF weights decrease the weights of terms that are common throughout the corpus, preventing common terms from having more impact than rarer terms which may be better predictors of a particular class.

2.2. Naive Bayes with Secondary Classifier

For instances in which the NB classifier was predicting only a subset of an entire article, we then had utilized a secondary classifier to aggregate the predictions for all the text segments within an article into a single prediction for the entire article. The secondary classifiers considered included calculating the mean, calculating the median, linear regression and logistic regression.

2.2.1 Preprocessing

The features for the linear regression and logistic regression classifiers were the percentages of the article calculated in each of the various classes. For example, if an article contained 10 text segments, 4 of which were classified as easy and the remainder hard, our features for the article would then be [0.4, 0.6]. Thus, the number of features is dependent upon the number of classes being classifier. Secondary classifiers were utilized for both binary classification as well as multiclass classification.

2.3. Language Models with Kneser-Ney Smoothing

A common NLP model is a language model, which assigns probabilities to sequences of words. [6] A language model is a useful model to be used in a future text-alignment tool that would paraphrase one set of text from one complexity level to another. The language model estimates how probable text is and can also be used as a similarity metric between corpora. [3]

In order to do this, we first created 5-grams, conditional on the previous 4 words in a phrase. Because of data sparsity, we used interpolation, which mixes weighted probabilities from all the 5-gram, 4-gram, down to the unigram counts [6]. This helps to deal with unseen phrases, since the reappearance of 5-grams are much less likely than the reappearance of bigrams or unigrams.

There are a variety of smoothing techniques to deal with phrases that have not been seen in a particular context. One of the most commonly used N-gram smoothing methods is the interpolated Kneser-Ney algorithm [6].

2.3.1 Preprocessing

In order to create the language model, we used the KenLM package[1], which implements interpolated Kneser-Ney smoothing on N-grams. We trained on an interpolated 5-gram model.

In order to evaluate the LMs, we used the perplexity metric, a common evaluation technique for language models:

$$\mathrm{PP}(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

Because this is an inverse, the higher the conditional probability of a new word sequence, the lower the perplexity. Therefore, minimizing perplexity is equivalent to maximizing the test set probability according to the language model [6]. Each sentence is then compared to each of the language models.

We calculate the perplexity of our validation sentence against all of our grade-level language models and compare them to each other. Our hypothesis is that in comparing this sentence against each of the language models, correct language model would give us the lowest perplexity. We then categorize an entire article by a variety of methods similar to that of Naive Bayes.

We concentrate on binary classification between easy (grades 5 and lower) and hard (grades 6 and above). For binary classifiers, all original levels of articles are included, as they are typically a hard level of complexity.

Using Kneser-Ney, we tried 3 binary classifiers: 1. creating all grade level models, choosing best article using a mean of all sentence predictions using minimum perplexity, then classifying "easy" versus "hard", 2. creating all grade level language models, classifying each sentence, choosing best article using linear regression, and applying the same binary classification, and 3. creating 2 language models, one for easy and one for hard. An article is classified as easy if the majority of its sentences are classified as easy.

2.4. Recurrent Neural Networks

We explored using a recurrent neural network (RNN) as an alternative method of classifying entire sentences of text. We tried variations of a basic RNN architecture consisting of an embedding layer, an RNN layer of LSTM neurons, and a dense layer with softmax activation for the multi-class case or sigmoid activation for the binary case. For the embedding layer, we explored using GloVe [5] encodings of dimensions 50 or 100. For optimizers, we explored using Root Mean Square Propagation (RMSProp) versus Adaptive Momentum Optimization (Adam). For the number of neurons in a single layer, we explored 20, 100 and 200 neurons, and for the number of layers we explored 1, 2, and 3 layers of 100 LSTM neurons each. Additionally L_1 and L_2 regularization values up to 0.01 were applied to input weights, bias weights and recurrent weights to see if regularization would prevent overfitting.

2.4.1 Preprocessing

Given the structure of the RNN, which relies upon having a uniform input sequence length, sentences were tokenized into discrete arrays of integers using Keras' pre-built tokenizer function and zero-padded for consistent length.

3. Results

3.1. Naive Bayes

For the NB classifier, we find that the performance is far greater for binary classification than multi-class classi-

Type	Sent.	n-gram	Test Err.	Train Err.
Binary w/ orig	3	2	0.225	0.177
Binary w/o orig	Article	1	0.158	0.148
Multi w/ orig	3	2	0.704	0.592
Multi w/o orig	3	2	0.716	0.538

Table 2. Performance of the best NB classifier with and without the original articles for binary and multi-class classification.

fication, as expected. Interestingly, segment lengths of 3 sentences with max n-grams lengths of 2 words appears to provide the best model. Table 3 summarizes the findings.

3.2. Naive Bayes with Secondary Classifier

Adding a secondary classifier after a NB classifier, we see substantial improvement for both binary and multi-class classification. With this approach we see that shorter text segments leads to a better model. This is likely because shorter text segments provide more text segments, and thus better features, to the linear classifier. The results are summarized in Table 3.

Type	Sent.	n-gram	SC	Test Err.	Train Err.
Binary, w/ orig	2	3	Logistic	0.076	0.036
Binary w/o orig	1	3	Linear	0.118	0.045
Multi w/ orig	1	2	Median	0.591	0.372
Multi w/o orig	1	2	Logistic	0.625	0.257

Table 3. Performance of the best NB classifier and secondary classifier (SC) with and without the original articles, binary and multi-class.

3.3. Language Models Kneser-Ney Smoothing

Using the language model approach, the best model was to create a different model for each grade level, use all of the articles, then classify the grade of a test article using the mean of all of the sentence classifications. The results are shown in Table 4.

3.4. Recurrent Neural Networks

For consistency, the model was trained and evaluated using the same training and validation steps, respectively, and was trained over only 10 epochs. We did not anticipate that the model would be fully trained within 10 epochs. However, judging the initial performance over a short training cycle allowed us to quickly investigate the performance of different parameters. In all cases of varying hyper-parameters and regularizations, we found the RNN model to severely overfit. The training accuracy far exceeded the validation accuracy, with the validation accuracy remaining constant and validation loss increasing. The utilization of GloVe citeglove embeddings also did not improve the model performance.

Type	Test Err.	Train Err.
Grade LM w/ orig, mean	0.153	0.809
Grade LM w/o orig, mean	0.266	0.117
Grade LM w/ orig, lin reg	0.171	0.035
Binary LM	0.202	0.0003
Multi w/ orig, means	0.734	0.632
Multi w/o orig, means	0.757	0.642

Table 4. Performance of the Kneser-Ney Smoothing for binary and multi-class classification.

Type	Error
Flesch Reading Ease	0.676
Flesch Kincaid Grade	0.356
Automated Readability Index	0.450

Table 5. Performance of Current Industry Readability Scales, Binary Classification

3.5. Comparison with Industry Standard Models

As some baseline models, we tried some of the well known readability scores in the education space: the Flesch Reading Ease Scale, the Flesch Kincaid Grade Level, and the Automated Readability Index [2]. We evaluated their binary classifications of easy vs hard. They are heuristic based, using counts of words, sentences, and syllables. The flaw with these tests is that they dont take difficulty of words or semantic difficulty into account.

We have reported their error rates in Table 5.

4. Discussion

Despite its "naive" assumptions, the NB algorithm combined with a secondary classifier provided the best classification both in the binary and multi-class case. Figure 1 illustrates one example of the refinement provided to the NB classifier's predictions with a secondary linear classifier. In general, the NB classifier is able to identify the two clusters corresponding to "easy" and "hard" articles - thus its exceptional performance in the binary case. We see this reflected in Figure 1 by clusters of predictions around grades 4 and 7. The linear classifier then acts to "smooth" out these predictions. Though using a logistic regression algorithm tended to provide a better accuracy (where accuracy is narrowly defined as as predicting the exact grade level), linear regression was much more adept as predicting the range of grades for a corresponding article. In other words, the logistic regression algorithm was not as effective as linear regression at breaking down the NB "clusters" into ranges of possible grades for a given article.

The language models using Kneser-Ney smoothing also performed well, ultimately performing best using a mean

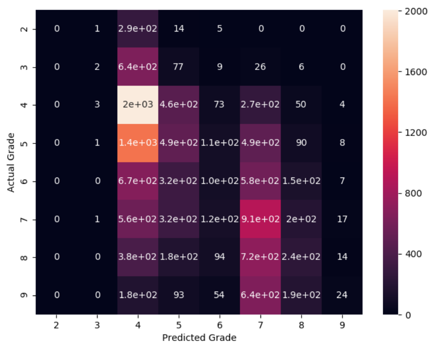
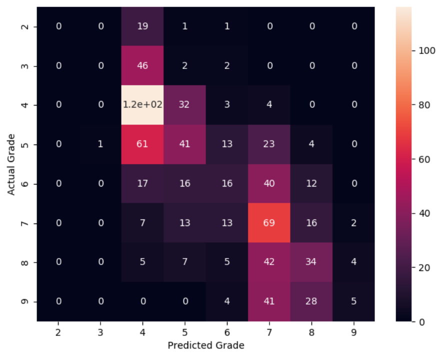
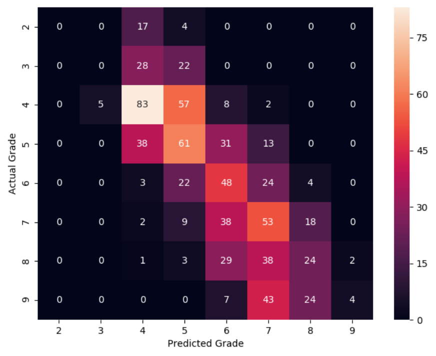
Figure 1. Comparison between the confusion matrices for the NB classifier (top), logistic regression (center), and linear regression (bottom). NB classifier has much higher counts since predictions are being made on individual text segments, whereas the linear and logistic regressions are for entire article.

of the entire article to classify a grade level, then choosing "easy" versus "hard" from these predictions. This also shows that the language models we created would be reasonable for future NLP tasks. Furthermore, these models did a good job in classifying within +/- 1 grade levels, achieving a 63.51 accuracy and picked up the trends of increasing difficulty (Figure 2).

5. Conclusions

Both methods of n-gram models performed well on detecting complexity of sentences. The appearance of more complex grammatical sequences as well as vocabulary was picked up fairly well by the structure of n-grams. Naive

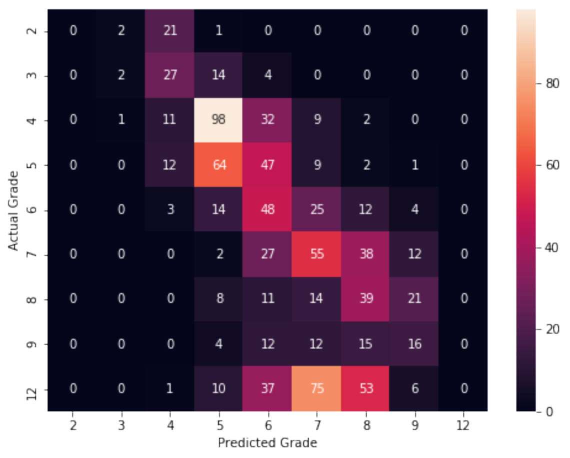
Figure 2. Kneser-Ney Smoothing Confusion Matrix

Bayes used the assumptions of generative algorithms as a classification task, whereas the language model created via Kneser-Ney smoothing gave us the ability to compute similar text to our grade level probability distributions.

Poor performance of the various RNN architectures explored, with immediate overfitting, implies that we did not have enough data to effectively train the model.

In the future, we would continue to gather more data to be sufficient for RNN architectures since it was cleaer the amoutn of data we had was insufficient. Furthermore, we would explore using BERT [4] embeddings for our text segments rather than GloVe.

6. Team Member Contributions

Kiley Yeakel worked on the Naive Bayes and RNN models, and Stephanie Tzeng worked on gathering the data sets, the Kneser-Ney smoothed n-gram approach, and industry standards. Kiley Yeakel made the final poster and both team mates contributed to the final report write-up.

Our code is at https://github.com/stephtzeng/autograder for reference.

7. Acknowledgements

We would like to acknowledge Haojun Li and Luke Orland for their advice and support in our research. We also thank Newsela for allowing us to use their corpus in our research.

References

[1] Kenlm github. https://github.com/kpu/kenlm. Accessed: 2019-06-10. 3

[2] Pypi textstat. https://pypi.org/project/textstat/. Accessed: 2019-06-10. 4

[3] K. Heafield. Efficient language modeling algorithms with applications to statistical machine translation. Accessed: 2019-06-10. 3

[4] K. L. K. T. Jacob Devlin, Ming-Wei Chang. Bert: Pre-training of deep bidirectional transformers for language understanding. arxiv:1810.04805, 2018. 5

[5] R. S. Jeffrey Pennington and C. D. Manning. *GloVe: Global Vectors for Word Representation*. 3

[6] D. Jurafsky and J. H. Martin. *Speech and Language Processing, year = 2018*. 3