

Final Project - Classifying Reading Level Using Language Models

Abstract

The problem we address in this paper is building a reading level classifier using natural language processing techniques. Prior work on this task includes some standard and widely-used readability formulas such as Flesch-Kincaid, along with some work using language models for classification. We set out to use statistical language models along with multinomial logistic regression models and features to try improve upon existing techniques.

Dataset

For our dataset for training and test data, we needed large passages of written prose that was labeled at specific reading levels. We decided to focus on novels that are suggested on reading lists at different grade levels, since this gave us the large amount of text we needed for building statistical language models, and additionally, labeled data was readily available.

A few caveats about our training data: First, due to copyright issues, we were only able to pull fulltext for books whose copyright had expired, so our dataset consists only of the more "classic" books. This turned out to be an advantage, since we limit our type of books to classic English novels, and avoid the problem of our classifier learning a different problem, for instance, a topical classifier. Additionally, since we pulled these books and manually cross-referenced them against reading lists, we weren't able to obtain a large number of books for use in our data set (although the documents do each provide a large amount of text to model on). Finally, the labels for our data is also quite noisy - While looking at different instances of reading lists online, we noticed a lot of variation in the recommended reading level of several of our books. In general, we tried to pick books that were representative of each class, and in the case of discrepancies we chose the class the book most commonly appeared in.

In total we drew 48 books, distributed as follows:
5th-7th Grade: 15 books
8th-10th Grade: 16 books
11th-12th Grade: 17 books

The actual list of books is included in the Appendix at the end of this report.

Framework and Organization

Our main module, ReadabilityTester, reads in all the necessary data files, builds the models, and test and evaluates the overall performance of the system. Our organization/program flow is as follows:

First, we read in the labeled training data one document at a time, and add the sentence data from each document to a language model corresponding to each class. That is, in our case, we maintain three different language models - one for each of our reading level classes. When we encounter a new document A which is labeled reading level "5-7", we add

all the sentences from document A to language model for "5-7", and similar for documents for other classes.

Then, during testing time, we classify each unseen document by computing the perplexity of the document according to each of the language models and choosing the class whose language model had the lowest perplexity. This, in effect, chooses the model that would have most likely produced this document. That is, we choose the model that has the highest probability of generating this document.

K-fold Cross-validation

Due to our limited dataset and our desire to both train and test on as many books as possible, we used standard techniques to do K-fold cross-validation, enabling us to train and test on all of our data (over the course of several trials). By default, we used 10-fold cross-validation, but our tester had a flag that allowed us to change this dynamically. Essentially, the K-fold cross validation method splits the data into K subsamples, and then during each trial holds out one of these subsamples and trains on the rest. This held out subsample is then used for evaluation for that trial. This is repeated K times, holding each subsample (and consequently each document) out exactly once and therefore using each evaluation exactly once. The overall cross-validated accuracy is then computed as the average of all the trial accuracies (weighted by the number of documents in that trial, since one subsample may be of different size).

Evaluation Methods

Our primary metric for model evaluation was simply the accuracy, which we defined as the overall fraction of books that were predicted correctly. That is,

$$\text{Accuracy} = (\# \text{ of books predicted correctly}) / (\text{total} \# \text{ of books})$$

Since cross-validation allowed us to use every book in our data set for evaluation exactly once our accuracy was essentially a fraction with a denominator of 48. In the results below, we report this accuracy as a decimal.

In order to get a finer granularity in our accuracy measurements, and additionally to reward the system from being "not too far off", we also created a weighted accuracy metric which gave partial credit to guesses whose true class was adjacent to the guessed class. These made sense for the context of our problem since our classes were ordered. The formula for weighted accuracy is,

$$\text{Weighted Accuracy} = ((\# \text{ predicted correctly}) + 0.5 * (\# \text{ off by one})) / (\text{total} \# \text{ of books})$$

Language Models

As a starting point, we began with one of our simplest models from PA1 - The Smoothed Unigram language model. We generated a Smoothed Unigram model for each class based on the training data, and then used these models to do classification as described above. The overall performance of the system is shown in the table below:

Model Name	10-Fold Cross-Validated Accuracy
Smoothed Unigram	0.3958

Based on our observations during error analysis and while data-mining our books for signals, we came up with several modifications to our base model which ultimately resulted in better overall model performance. Some of the modifications we made along with justifications are listed below.

Using the histogram of sentence lengths as a feature

While mining our documents for various signals, we noticed that the distribution of sentence lengths seemed to vary between different classes. As a result, we thought that capturing this information would help us differentiate between different classes during testing. As a simple way to take sentence lengths into account during classification, we added the sentence lengths right into our existing language model. That is, we simply emitted an extra "token" at the end of each sentence that denoted the sentence length, and then when we were computing the overall sentence probability, we made sure to include the length token in the calculation. As shown below, this indeed improved our model performance.

Reducing the weight given to the unknown token

In our base model, we were giving a large amount of weight to the unknown token based on the discounting techniques used in PA1. For actually modeling languages, this is an important feature of a language model since unknown words show up a fair amount and it is not desired to give a sentence a very low probability due to the presence of an unknown word. In our problem, however, the actual probability of sentence is not important, but rather the relative difference between models of different classes (there is more discussion of this below). For this reason we found that we had more differentiating power if we gave less weight to unknown words, and focussed on the vocabulary we had actually seen. This also gave huge improvements in overall performance.

Sharing vocabulary from other classes

One interesting observation we had during error analysis was that at an individual sentence level, the weight could shift from one class to another drastically if there was a word that appeared in one language model but not the other. Although smoothing probability to the unknown character should help with this problem, an alternative approach we came up with was to "share" the vocabulary from other models, but naturally giving them less weight. This was to decrease the effect mentioned above. Although this seemed promising, in practice borrowing vocabulary from other classes did not seem to help.

Dealing with different unknown probabilities

A very interesting observation was that each of our language models was trained on a different number of words (and a different vocabulary for that matter) and as a result, the probability assigned to the unknown character differed in each of these models. When we considered this, we realized models with a high unknown character probability had a significant bias towards them, so we tried fixing the unknown probability across all models to remove this bias. This turned out to help our model performance substantially.

With these changes to our base language model, we were able to get an overall system with much better performance. The following table summarizes our overall language model results:

Model Name	Accuracy	Weighted Accuracy
Smoothed Unigram (base model)	0.3958	0.5417
Unigram with Decreased Smoothing (UDS)	0.5417	0.7396
UDS with Sentence Length Distribution (SLD)	0.5417	0.7396
UDS with Fixed Unknown Weight (FUW)	0.5833	0.7812
UDS with SLD and FUW	0.6042	0.7917
UDS with Borrow Vocab All Classes (BVAC) - weight .5	0.5000	0.7083
UDS with BVAC - weight .3	0.5208	0.7396
UDS with Borrow Vocab Adjacent Classes (BVJC) - weight 0.5	0.5208	0.7396
UDS with BVJC - weight 0.3	0.5208	0.7396

As the evaluation data above shows, our best performing model was the Unigram with Decreased Smoothing, Sentence Length Distribution, and a Fixed Unknown Weight. This model performed about 50% better than our original base model.

Difference Between Modeling Languages and Classification

It is interesting to note that our task differs quite substantially from the task of modeling a language. When modeling a language, our primary goal is to give high probability to sentences that are likely to appear while giving low probability to sentences that are unlikely to appear. Language models are typically used in order to determine, given a set of sentences, which sentence is most likely to have been generated by the language, and other similar type of tasks. In our problem, we're using language models in a pretty different context. We have a fixed sentence (actually a fixed document), and we'd like to extract features from this document that are characteristic of the appropriate class, and more importantly, that differentiate this document from other classes. This idea of thinking of our language models *relative to other models* was primary focus in our error analysis, and was a driving factor in the modifications we made to our language models.

Overfitting

One observation we had during error analysis was that our models would all perform very well on training data - much better than they did on the test data. Specifically, many of our models, when trained on all our data, gave 100% accuracy when tested on that same data. Although it is common for models to perform better on training data than test data, when there is this large of a discrepancy in accuracy between the two, it often alludes to over-parameterization of your model and over-fitting to your training data. Since our models were language models with thousands of parameters (to capture the counts of all the words seen in various texts), it is understandable that such over-fitting could occur. Several of our model changes were directed at addressing this issue (such as some of the smoothing techniques mentioned above) but there is still room for exploration in finding better ways to allow our models to generalize to new data.

Visualizing the Clusters of Data

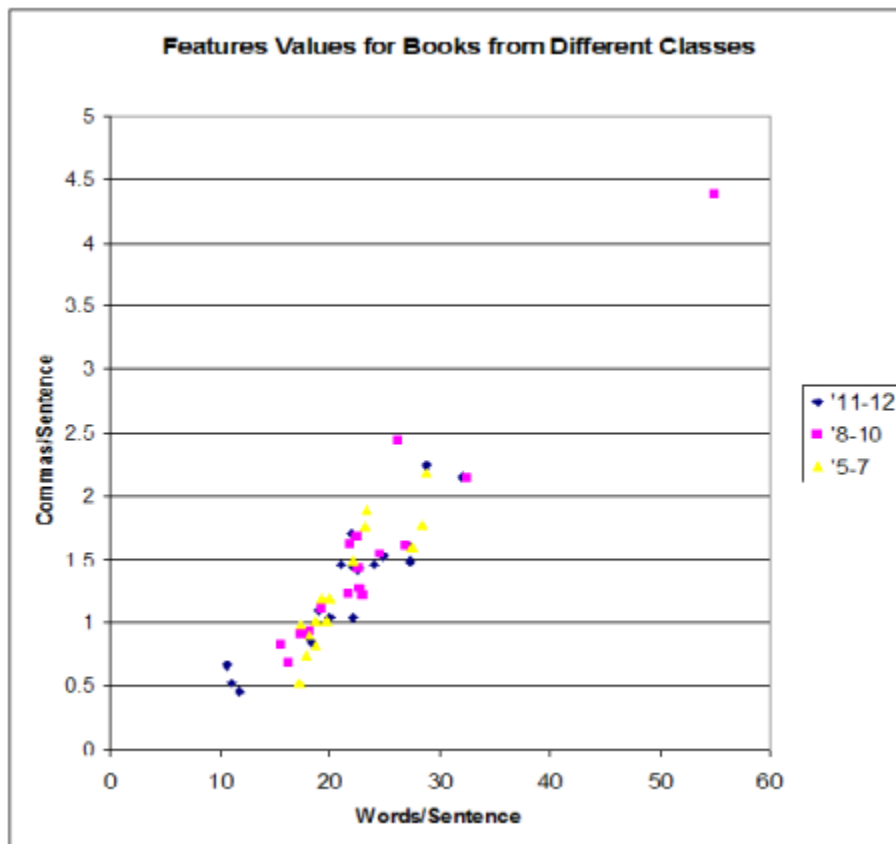
To help get an understanding of what kinds of signals would differentiate the books, we spent a lot of time data-mining and coming up with different types of features to see what they evaluated to for each class. Our hope was to find features that had similar values within a class and different values between classes, so we could use these features to help train a classifier.

In addition to looking at features at a class level, we decided to look at features at the individual book level for two main reasons:

1) Looking at the values a feature for various books from the same class would give us a visual sense of the variance of that mean class feature value, to help determine how noisy that measure was and how much trust we could place in the mean value.

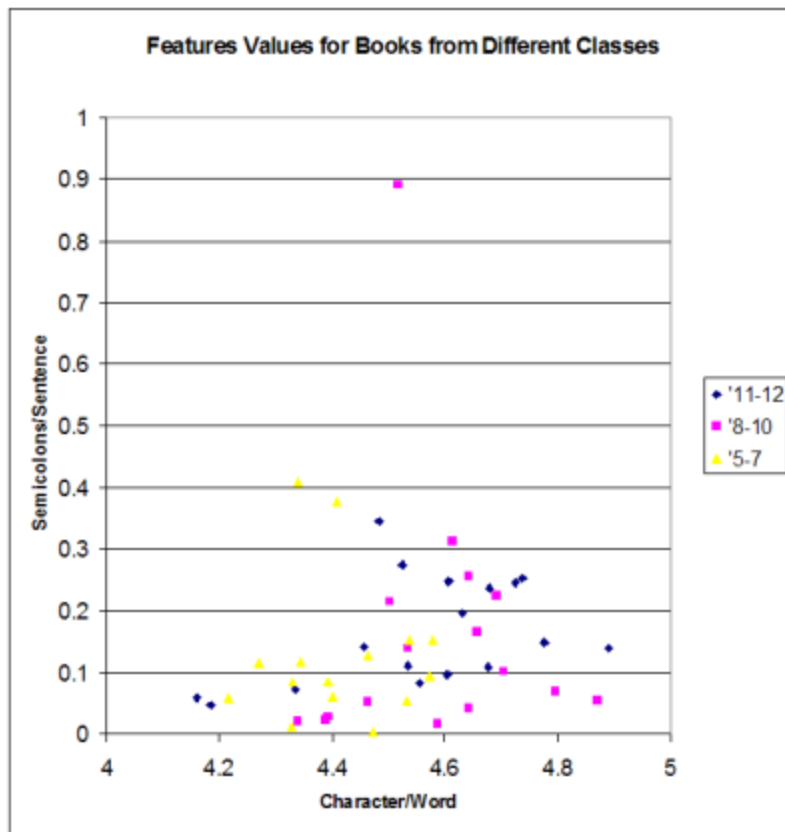
2) Visualizing the data from all the classes would give us a sense of how "separable" the books of different classes were according to this feature. Ideally, we'd get nice separated clusters that each contained all the books from a particular class. These were the features we were searching for.

A few scatter plots showing pairs of features are included below. The different shapes/ colors represent the true classes of the books.



A few observations from the above graph:

- 1) These two measure correlate highly, which is natural since longer sentence (more words/sentence) will also tend to have more commas
- 2) There is very little separation between the three classes here, and neither of these features seem to differentiate between classes very well. These are therefore, both, not very good features for classification
- 3) Although these metrics don't serve the purpose of separating classes, they do still pick up strange anomalies in books. The big outlier in the upper right is Robinson Crusoe, which has quite peculiar language and exceptionally long sentences (which have a commas, almost 5 per sentence on average!)



Flesch-Kinkaid Baseline

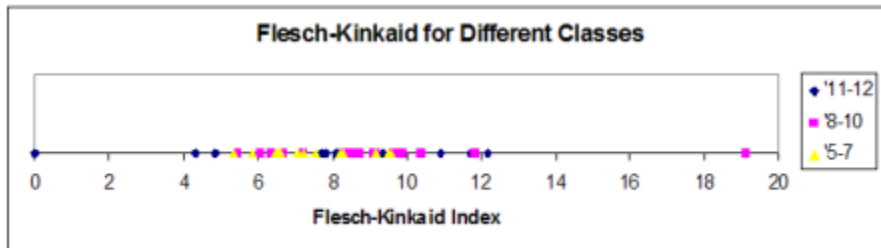
To get a general sense of how difficult the classification task we were trying to do was, and get an idea of how well a classifier based solely on the Flesch-Kinkaid Index would perform,

we built a basic classifier that simply drew boundaries at particular Flesch-Kinkaid levels in order to optimize the overall accuracy of the system. A important thing to note in this classifier is that the classifier **utilized all the test data** to find optimal boundaries. That is, it trained on the exact same data that it tested on. In this sense, we weren't really measuring how a true classifier based only on Flesch-Kinkaid index would perform on real test data, but rather exploring the upper-bound of the accuracy of such a classifier. The results are shown below:

Metric	Value
Optimal Boundary Between 5-7 and 8-10	8.3
Optimal Bounday Between 8-10 and 11-12	10.0
Overall Test/Train Set Accuracy	.5000

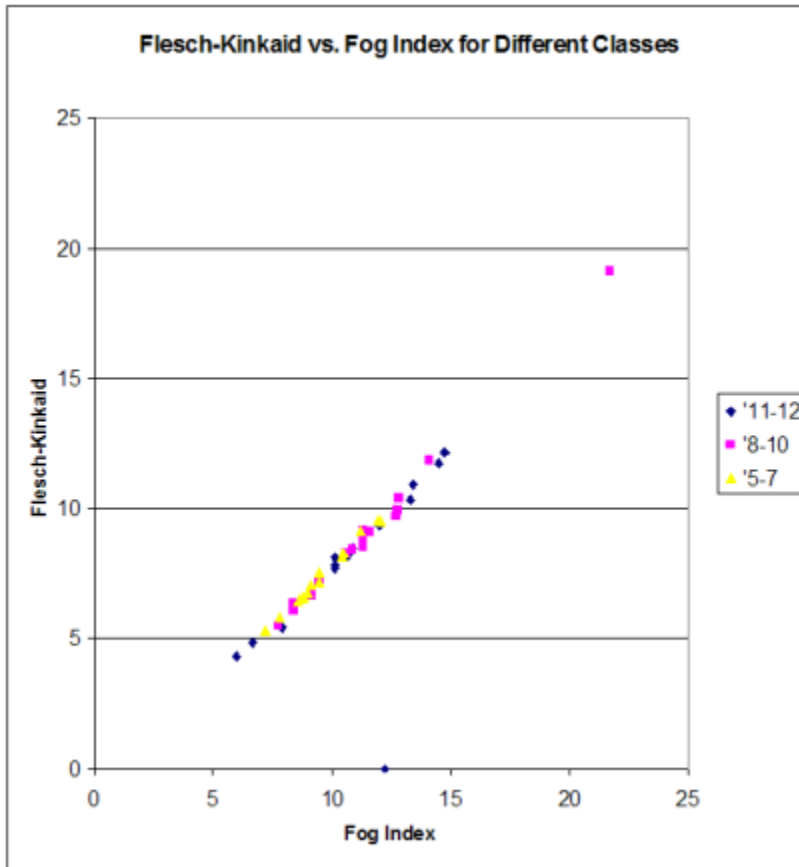
Therefore, this Flesch-Kinkaid model, despite having seen all the data when choosing optimal boundaries, still only scored at 50% accuracy. Our models that we presented above, on the other hand, performed better than this model even on unseen test data, as shown by our overall cross-validated accuracy numbers.

To get a sense of how much Flesch-Kinkaid was actually separating our data, we produced the graphs below:



The main observation from the graph above is that the Flesch-Kinkaid Index, in general, does not do a very good job of separating our classes. Overall, the points from all three classes are spread across the range of Flesch-Kinkaid values. Additionally, there are some major outliers according to this measure, many which are in the wrong direction (see three leftmost data points which are all of the 11-12 class, and the rightmost point which is class 8-10). This, however, may be due to data issues, as we discuss futher below.

The following graph also shows the Flesch-Kinkaid Index, but this time plotted against another common reading level measure, the Fog Index. The additional dimension makes it a little easier to see distribution of the data. Naturally, these two empirical measures correclate very highly.



Where the right most term is the regularization weight.
Taking the gradient of the objective function, we get:

$$\nabla_{w_c} = \sum_{i:c_i=c} \left(-f(c_i, d_i) + \frac{\exp(w_{c_i}^T f(c_i, d_i))}{\sum_{c' \in C} \exp(w_{c'}^T f(c', d))} f(c_i, d_i) \right) + \sum_{i:c_i \neq c} \left(\frac{\exp(w_{c_i}^T f(c_i, d_i))}{\sum_{c' \in C} \exp(w_{c'}^T f(c', d))} f(c_i, d_i) \right) + \lambda w$$

At each iteration step t , we update the weight vector,

$$w_c \leftarrow w_c - \nabla_{w_c} \eta_t$$

where we use

$$\eta_t \propto \frac{1}{\sqrt{t}}$$

so that we take smaller steps when we approach convergence.
We do this iteratively until the gradient is close enough to zero, at which point we've converged.

Feature Engineering

To start off, we tried a few simple features like the number of characters per word and the number of words per sentence. By briefly inspecting the data, we came up with some conceptually reasonable thresholds, as a first pass.

Naively trying each of these features individually, we get the following results:

Feature	Accuracy (10-fold validation)	Results
Characters Per Word * class == 11-12 * char / word > 8	0.3542	Good Classified per class: 14 True: 5-7 -> Guess: 5-7 3 True: 11-12 -> Guess: 11-12 Misclassified per class: 16 True: 8-10 -> Guess: 5-7 14 True: 11-12 -> Guess: 5-7 1 True: 5-7 -> Guess: 11-12
Words Per Sentence (Feature 1) * class == 5-7 * words / sent < 16 (Feature 2) * class == 8-10 * words / sent < 22 (Feature 3) * class == 11-12 * words / sent >= 22	0.3125	Good Classified per class: 12 True: 5-7 -> Guess: 5-7 3 True: 11-12 -> Guess: 11-12 Misclassified per class: 16 True: 8-10 -> Guess: 5-7 13 True: 11-12 -> Guess: 5-7 2 True: 5-7 -> Guess: 8-10 1 True: 11-12 -> Guess: 8-10 1 True: 5-7 -> Guess: 11-12

Of course, the accuracy is not very good, but they seem to be able to correctly classify the books in the Grades 5-7 pretty well.

We then tried to find features that can help classify the other classes better. We looked into more complicated features, such as the number of syllables.

Using Syllables

For calculating the number of syllables, we used a very simple common scheme: We count each continuous section of vowels as one syllable, except when the vowel 'e' appears at the end of the word. This approximate performs quite well, though would miscount some examples such as "lifhouse". The code is opensource provided by "java Fathom".

We mainly tried 3 features that used syllables. The obvious one is the Flesch-Kincaid index, the others are number of syllables per word and the ratio of the number of complex words, where a complex word is defined as a word with more than 3 syllables.

Using them individually, we get the results:

Feature	Accuracy (10-fold validation)	Results
<p>Flesch-Kincaid Index</p> <p>(Feature 1) * class == 5-7 * complex words / all words < 8.3</p> <p>(Feature 2) * class == 8-10 * complex words / all words < 10.0</p> <p>(Feature 3) * class == 11-12 * complex words / all words >= 10.0</p>	0.3333	<p>Good Classified per class: 13 True: 11-12 -> Guess: 11-12 3 True: 5-7 -> Guess: 5-7</p> <p>Misclassified per class: 12 True: 5-7 -> Guess: 11-12 8 True: 8-10 -> Guess: 11-12 8 True: 8-10 -> Guess: 5-7 4 True: 11-12 -> Guess: 5-7</p>
<p>Num Syllables per word</p> <p>(Feature 1) * class == 5-7 * complex words / all words < 1.43</p> <p>(Feature 2) * class == 8-10 * complex words / all words < 1.45</p> <p>(Feature 3) * class == 11-12 * complex words / all words >= 1.45</p>	0.3333	<p>Good Classified per class: 16 True: 11-12 -> Guess: 11-12</p> <p>Misclassified per class: 15 True: 8-10 -> Guess: 11-12 15 True: 5-7 -> Guess: 11-12 1 True: 8-10 -> Guess: 5-7 1 True: 11-12 -> Guess: 5-7</p>
<p>Ratio Num_Complex_Words : Num_Words</p> <p>(Feature 1) * class == 5-7 * complex words / all words < 7.8%</p> <p>(Feature 2) * class == 8-10 * complex words / all words < 9.4%</p> <p>(Feature 3) * class == 11-12 * complex words / all words >= 9.4%</p>	0.3125	<p>Good Classified per class: 15 True: 11-12 -> Guess: 11-12</p> <p>Misclassified per class: 15 True: 8-10 -> Guess: 11-12 15 True: 5-7 -> Guess: 11-12 2 True: 11-12 -> Guess: 5-7 1 True: 8-10 -> Guess: 5-7</p>

We see that these features tend to get the 11-12 class correct most of the time, but not the others. Since the first 2 features we tried got the class 5-7 correct most of the time, the intuition was to combine these features and see how it performs, hoping that the important features will be combined and weighted properly by the classifier. We didn't use the number of syllables per word feature in this combined trial, since that is already captured by the Flesch-Kincaid index. The results we got were:

Feature	Accuracy (10-fold validation)	Results
<ul style="list-style-type: none"> * Flesch-Kincaid * Ratio Num_Complex_Words : Num_Words * Words Per Sentence * Characters Per Word 	0.4375	Good Classified per class: 14 True: 5-7 -> Guess: 5-7 7 True: 8-10 -> Guess: 8-10 Misclassified per class: 13 True: 11-12 -> Guess: 8-10 9 True: 8-10 -> Guess: 5-7 4 True: 11-12 -> Guess: 5-7 1 True: 5-7 -> Guess: 8-10

Which is significantly better than the individual features on their own.

Using Language Model as a Feature

Finally we added the language model as a feature, and ran it through the classifier together with these four other features. The results were:

Feature	Accuracy (10-fold validation)	Results
<ul style="list-style-type: none"> * Flesch-Kincaid * RatioNum_Complex_Words : Num_Words * Words Per Sentence * Characters Per Word * Language Model 	0.5208	Good Classified per class: 15 True: 5-7 -> Guess: 5-7 10 True: 8-10 -> Guess: 8-10 Misclassified per class: 13 True: 11-12 -> Guess: 8-10 6 True: 8-10 -> Guess: 5-7 4 True: 11-12 -> Guess: 5-7

It is better than only the previous 4 features on their own, but still not as good as the language model itself. This may be caused by noise from some bad features, which would make it hard for the classifier to make good training iterations.

We spent a lot of time tweaking this to see if we can get better results. After performing the analysis documented above regarding the usefulness of the number of words per sentence and the number of characters per word features, we tried a run without these two features. This turned out to give the highest accuracy among all the different parameterizations we tried. Here is the result:

Feature	Accuracy (10-fold validation)	Results
* Flesch-Kincaid * RatioNum_Complex_Words : Num_Words * Language Model	0.6667	Good Classified per class: 14 True: 5-7 -> Guess: 5-7 11 True: 11-12 -> Guess: 11-12 7 True: 8-10 -> Guess: 8-10 Misclassified per class: 6 True: 8-10 -> Guess: 11-12 4 True: 11-12 -> Guess: 8-10 3 True: 8-10 -> Guess: 5-7 2 True: 11-12 -> Guess: 5-7 1 True: 5-7 -> Guess: 11-12

As we've seen from above, the number of words per sentence does not give very indicative signals for class separation. Thus, not including it as a feature benefited the model because it might have added noise. Similarly, from another graph from above, we see that the number of characters per word isn't really a good feature to use either.

The Flesch-Kincaid index and the ratio of the number of complex words to the total number of words proved to be useful features. Although we mentioned above that the Flesch-Kincaid index is, by itself, not terribly accurate, it does still provide some signal for our logistic regression classifier, so that when combined with the language model, this performed 10% better than solely using the language model itself.

Vocabulary Differences Between Classes

We also looked at trying to find specific words that vary significantly between models. If we find something, we might be able to add it as a feature into our multinomial logistic regression classifier. One feature we hypothesized could be interesting to consider was that more difficult books may tend to use more semicolons. Below is a sorted list of words that have a large difference in its rate of appearance between any two grade levels. The differences are percentage units.

Word	Difference Reason	Difference	5-7	8-10	11-12
"	8-10 >> 11-12	1.28	2.51788449269503	3.01012634306875	1.73351499019473
of	8-10 >> 5-7	0.9	1.87716931994456	2.779246955229	2.77271564130645
the	8-10 >> 5-7	0.83	4.79019585035566	5.61519621857549	5.22254756869062
,	5-7 >> 8-10	0.78	1.01426298471171	0.22933481981548	0.65728818072522
and	5-7 >> 8-10	0.77	3.95279062560768	3.18616940171918	3.48429257274067

she	5-7 >> 8-10	0.61	1.13566035675185	0.52330512512627	0.79892861879113
,	8-10 >> 5-7	0.6	5.99600497643134	6.59819542460416	6.10414103566434
his	8-10 >> 5-7	0.44	0.67154020939126	1.10831570320392	0.9238857367707
her	5-7 >> 8-10	0.36	1.02822102667603	0.66482265042789	0.96261185903442
;	11-12 >> 5-7	0.34	0.49757232571692	0.75865667722810	0.83322238383588

As can be seen, the semi-colon appears tenth on this list, and the other words that are on this list look rather suspicious, since it's not obvious that they would conceivably provide any indication of reading level difficulty. We included the semi-colon and the double-quote as features for our classifier in a few trial runs, but the results weren't promising. We therefore, disregarded this list, and considered it mostly noise for the rest of our experiment.

Related Work

There are several papers on different reading level classification techniques, and many widely-used reading level measures such Flesch-Kinkaid and Fog Index (although these are known not to perform too well). There is only one paper that we know of, however, that uses statistical language models to approach this problem, and this paper (listed in references) was an important source for this project. This paper talks about using a unigram model with some specific smoothing techniques to do the classification task, but has different features and smoothing techniques than the model we derived. Additionally, that paper focussed on classification of web pages and very short passages, while our task was classifying classic English novels. We're not aware of any existing paper that has tried to combine statistical language models with logistic regression (using the language models as features) and it was very promising that this combination worked so well on our dataset.

Future Work

While exploring this interesting classification problems, we encountered several different areas that warrant future exploration. First, as mentioned above, based on our relative test and training accuracies, it became apparent that we're likely over-fitting to our training data. While we considered this when making changes to our language models, we never fully understood how this over-fitting is occur and what we can do to combat it. In general, overfitting causes lower test accuracy and it would be interesting to try to come up with a more "coarse" model (with potentially fewer parameters) to see if this generalizes better to our test data. As a step towards this, our logistic regression model which only uses the highly-parameterized language models as single feature and has a set of much more coarse features did perform better on our test data.

It would also be interesting to explore some higher order language models and see how they perform in a classification task. We tested our Bigram_Katz model from PA1 and although it took an extremely long time to run (due to cross-validation, etc.) it eventually performed just as well as our best unigram model. If we could substantially improve the

performance of this model (or add some parallelism to our testing code) then it would be interesting to iterate on and explore the properties of these higher order models. An open question for us is whether bigrams and trigrams, etc. carry a lot of additional value in terms of indicating reading level, or whether just vocabulary (unigrams) is a primary driving factor. Past literature has focussed only on unigrams, but we would like to explore this area more.

Conclusion

In this paper, we combined a language model with a multinomial logistic regression model to the classification problem of readability levels. After many iterations of tweaking the smoothing on our unigram language model, we were able to substantially increase the performance of the language model. The best performing model used decreased smoothing, sentence length distribution, and fixed unknown weight techniques for smoothing. In feature engineering for our logistic regression classifier, we performed in depth analysis of various statistics from our book data. Having the wrong combination of features will decrease the classifier performance, but in the end, we found a good mix of features that allowed us to break through the performance of only the language model itself. The classifier gave the best results when we provide it with these three features: the Flesch-Kincaid index, the number of complex words per total number of words, and the language model.

Contributions

Johnson and Sameer worked together on most aspects of this project. They came up with the original idea jointly, and did a lot of brainstorming together of techniques to do reading level classification. Sameer set up the initial testing/classification framework, and Johnson got all of the data, formatted it, and set up a framework for mining the books for various signals. Sameer worked on iterating through many different language models to try to improve classification, while Johnson worked on building the logistic regression classifier which used these language models as feature, as well as finding other signals from mining the data. Johnson and Sameer both worked on the final writeup together.

References

Collins-Thompson, Kevyn and Jamie Callan. "A Language Modeling Approach to Predicting Reading Difficulty." Carnegie Mellon University.

Natural Language Processing CS 224n notes and code. Stanford University.

"Java Fathom" (Java package for syllable counting), <http://www.representqueens.com/fathom/>, June 2008.

Appendix

List of books we used in this paper.

Grade 5-7 Books

Alcott - Little Women
Barrie - Peter Pan
Burnett - A Little Princess
Burnett - The Secret Garden
Carroll - Alice in Wonderland
Grahame - The Wind in the Willows
Kipling - Puck of Pooks Hill
MacDonald - The Princess and the Goblin
Marryat - Masterman Ready
Montgomery - Anne of Green Gables
Nesbit - The Story of the Treasure Seekers
Sewell - Black Beauty
Spyri - Heidi
Twain - Tom Sawyer
Webster - Daddy Long Legs

Grade 8-10 Books

Austen - Emma
Cather - O Pioneers
Crane - The Red Badge of Courage
Defoe - Robinson Crusoe
Dickens - A Tale of Two Cities
Doyle - Sherlock Holmes
Dumas - The Count of Monte Cristo
Eliot - Silas Marner
George Orwell - Animal Farm traing
Hardy - Tess of the dUrbervilles
Hesse - Siddhartha
London - The Call of the Wild
Rand - Anthem
Twain - The Prince and the Pauper
Wells - The War of the Worlds
Wilde - The Picture of Dorian Gray

Grade 11-12 Books

Austen - Pride and Prejudice
Barrie - The Admirable Crichton
Bronte - Jane Eyre
Bronte - Wuthering Heights
Chopin - The Awakening
Conrad - Heart of Darkness
Hard - Times
Flaubert - Madame Bovary
Goldsmith - She Stoops to Conquer
Hardy - The Return of the Native
Hawthorne - The Scarlet Letter
Shaw - Major Barbara
Sinclair - The Jungle
Thoreau - Walden
Twain - Roughing It
Voltaire - Candide
Wharton - Ethan Frome