Using Stanford CoreNLP and SVM^{rank} in a Supervised Approach to Medical Abstract Readability

By Dylan Mann and Pia Kochar
University of Pennsylvania CIS 530 Final Project

Introduction

We developed a Learning to Rank system for predicting medical abstract text difficulty based on supervised learning of rankings of abstracts from 1-10. Our system is based on the idea that word difficulty and sentence length is a major indicator of passage difficulty, and texts will fit language models generated from texts of a similar difficulty. We therefore used many features that relate to that, such as syllables per word, average words per sentence, and a language model to predict the frequency of words. We used these feature vectors to represent each sample in the training and testing sets. Since we had difficulty ratings for the training set, we were able to train a learning to rank classifier on what values for features correlated with certain levels of difficulty and use these to find a difficulty score for each excerpt in the test set. We also included features for the proportion of each text that was comprised of each part of speech, under the assumption that passages with similar difficulties may be predicted by similar language models.

To predict readability, we fit an SVM classifier to the feature vectors generated from the training data. We chose to use an SVM classifier because theoretically and practically it lends itself well to predicting on text data (Joachims). Furthermore, Curto, Mamede and Baptista found that with five readability levels, SVM classifiers resulted in 70.04% accuracy, second only to a boosting algorithm that had 75.11% accuracy, so this seems to be a good system (Curto). We chose to run all of our tests with a linear kernel because “most text categorization problems are linearly separable” (Joachims). This fact, combined with the relative speed of running an SVM classifier with a linear kernel as compared to using a more complex kernel, made this the clear choice for our experiments.

Method

To begin this assignment, we considered methods from previous assignments, and read research papers on predicting reading difficulty. We decided to represent the data using feature vectors and experiment with the features used in these to optimize our system. This method made the most sense because feature vectors allow us to easily compare multiple metrics about the training and testing data to find similarities between training and testing data along multiple axes.

First, we lemmatized and tokenized the training and testing data using Stanford CoreNLP. This process also gave us syntactic information that we used to develop some of the features we used. Using this data, we computed various different feature vectors. For each experiment we conducted, we changed the feature vector to find the optimal set of features to predict readability for the corpus we were given.

Each time we generated a new set of feature vectors, we ran SVM^{rank} on them. SVM^{rank} is a program for ranking SVMs and is publically released for scientific use. We felt it would be perfect for our purposes, as we wanted a discrete ranking of all the training data. This program allowed us to easily tune the parameters for training the model. We used the model fit to the training data to generate readability scores for the test samples.

To begin, we ran a baseline with 19 features, including the following features: number of syllables per word, proportion of words with fewer than 5 syllables, median word length, average word length, average sentence length, number of sentences, Type-Token ratio, and binary features for each POS in the universal Google tagset (the universal tagset consists of punctuation, adjective, adposition, adverb, conjunction, determiner, noun, numeral, pronoun, particle, verb, other).

We ran a second baseline with 21 features, including all of the features from the previous paragraph, as well as the number of dependencies and the number of named entity tags. One of the papers we read found that with five readability levels, number of words, number of different words and number of dependencies were the top three features that contributed to the classification (Cuerto). In this trial, we include similar features (number of words, type-token ratio and number of dependencies).

Next, we trained a model using a feature vector with 23 features. For this model, we retained all 21 features from the second baseline we ran, and we added some more advanced scores that would be helpful in understanding readability. We added Flesch-Kincaid grade level. Additionally, we modeled the data with a unigram model and included a feature for each sample with the normalized probability of the sample. We found that the model did not converge when we added these two features. Through further trials, we found that omitting the unigram model probability made the model converge very quickly. We suspect that this is the case because not all sentences that were unlikely in the training data was necessarily difficult. Even though we normalized for passage length, it is possible that the medical abstract realm is so diverse that it was impossible for our language model to get a good fit on the data from just under 500 samples. After we tried using the unigram language model as some of our features, we trained a classification model using 23 features, in which we replaced the unigram probability with the number of nodes in the syntax tree generated for a sample. We thought this would be a good feature because it was the fourth most predictive feature for text classification with five readability levels (Cuerto).

We also experimented with a very large feature vector (2700 features) that included more complex features such as named entity tags, brown clusters, dependencies and syntax tree parsing. This feature vector consisted of several concatenated binary feature vectors, which provided a lot of data. However, we found that using such a large vector was impractical because it would take far too long to train. The major contributor to the size was results from syntax tree parsing, so we first experimented with removing these features altogether. This resulted in a feature vector with 211 features. In addition to the 19 features from the baseline, we included POS tags from the Stanford NLP parse, number of named entity tags, number of dependency features and Brown clusters. However, this trial did not appear to be converging (after running for more than 2 hours), so we weren't able to get a model for this.

Since running the classifier with 211 features also took a very long time, we tested a vector with 110 features that included all the same features as the previous trial except for Brown clusters. This did not do nearly as well as our baseline, so we concluded that one of the additional features (POS tags from the Stanford NLP parse, number of named entity tags, or number of dependency features) was poorly fit to the data.

Final system

First we broke the excerpt file into an individual file for each excerpt. Then, we ran the Stanford NLP tool on our data, to lemmatize, tokenize, extract POS information, extract named entities, extract dependency information, and create a syntax tree. Then, we generated feature vectors for each training sample and test sample. We then mapped POS information to the google universal tagset in order to reduce the size of the tag set and reduce the amount of overfitting. In our final iteration, we included the following 23 features: number of syllables per word, proportion of words with fewer than 5 syllables, median word length, average word length, average sentence length, number of sentences, Type-Token ratio, number of dependencies, number of named entity tags, Flesch-Kincaid rating, number of syntax tree nodes per word, and the proportion of each POS tag in the excerpt using the universal Google tagset (the universal tagset consists of punctuation, adjective, adposition, adverb, conjunction, determiner, noun, numeral, pronoun, particle, verb, other).

After generating the feature vectors, we used SVM^{rank}, found here, to fit a model to the training data, using a linear kernel and a C value of 10. Then, we used this model to generate predicted readability scores for each sample in the testing data.

Experiments

For our first submission, we classified the data using 19 element feature vectors consisting of average sentence length, number of sentences, Type-Token ratio, median word length, average word length, and binary features for each universal POS. The rankings based on this system had a Spearman Correlation of 0.390358355029 when we submitted to the leaderboard.

For our second submission, we classified the data using 110 features. These included all of the ones from the first submission, as well as POS tags from the Stanford NLP parse, proportion of named entity tags, and proportion of dependency features. This trial gave us a

Spearman Correlation of 0.259569556313. Since the correlation was so much worse than that for the first trial, we concluded that many of our additional features were not predictive, and our data was being fit too closely to get an accurate ranking.. Therefore, if two abstracts had similar values for whichever feature or features were not predictive, considering them to be similar resulted in a bad prediction.

Our third submission used feature vectors with all the same features as the first submission as well as the number of named entities and the number of dependencies (21 features total). The correlation we got for this is 0.393261547009. This is slightly better than the first trial that we submitted, so we can conclude that either one or both of the number of named entities and number of dependencies made our predictions better. This is what we expected, since these are descriptive of sentence structure. Additionally, one of the papers we read indicated that number of dependencies was among their top three features for classification (Cuerto).

The fourth submission we made used feature vectors with all the same features as the previous submission, with the addition of the number of nodes in the syntax tree and the Flesch-Kincaid grade level for the sample. This gave us a correlation of 0.384551971068. This is slightly worse than the first and third submissions, so we can conclude that these features didn't really help us predict on this data. Although it's slightly smaller, this is very similar to the correlations we got for the first and third submissions, so we infer that adding and removing a few features from the vector doesn't make a huge amount of difference to the prediction.

For our last submission, we changed the C parameter to the SVM model from 10 to 50. We decided to try increasing the C parameter, since a higher C value allows more overfitting to the training data. More overfitting makes sense for this system since our training and testing data are from the same corpus, and so we thought that perhaps fitting the data more closely with relevant features would result in a better ranking. We ran this trial using the feature vector we used for the third trial, since that got us the highest correlation so far. However, the correlation we got with a C of 50 was 0.392922841278, which is essentially the same as the correlation we got for the same feature vector with a C of 10 (0.393261547009).

<u>Discussion and Analysis.</u>

We found that adding and removing small numbers of features didn't have a great impact on the resulting correlation we got upon submitting to the leaderboard. Some features seemed to make it harder for the SVM model to converge, dramatically increasing training time. One such feature was the unigram probability of an entire sample occurring based on a unigram language model for the corpus. Our hypothesis for why this might be the case was that this feature did not correlate with difficulty the same way other features did, and therefore created noise that made convergence to a small epsilon value harder.

Another possibility for why that is the case is that many words are extremely uncommon in medical corpi, and therefore just because a passage is unlikely according to our bag of words language model, that doesn't necessarily mean it is any harder to read than a more likely passage according to the Bayes model.

The only submission we made that did significantly worse than the others was the one that included POS tags from the Stanford NLP parse in addition to the universal tags. We concluded from this fact that these features might have been so severely overfit to our training data that the model produced by them was not rooted in any linguistic basis, but instead in random distribution of our training data. To solve that problem we could either have come up with a larger corpus (which was impossible for this assignment) or lowered our C value to prevent overfitting. Instead of attempting to solve this problem in either of those ways, we instead returned to the feature vector that was so successful on our first attempt and tweaked that.

Stanford CoreNLP is an industrial class, extremely robust system that made working with our corpus extremely easy. Once we ran the tool on the data once, we never needed to again, and it gave us access to the passages in a form that was tokenized, lemmatized, syntax tree parsed, dependency extracted, and named entity extracted. This made feature creation far simpler than it would have been had we not used the library. We also determined that using SVM^{rank} for text difficulty ranking was a difficult and unwieldy task. It is an outdated tool without much reference documentation, and it does not provide any means for automatically cross validating or testing. It did perform extremely well on the task out of the box, however tweaking the parameters was difficult both because of the length the classifier took to run, and the archaic nature of the tool. Nevertheless, if the researcher can afford to wait for the classifier and wants to manually roll their own cross validation tool, it could be an extremely effective tool. Finally, we found that using the research of others makes doing your own research related work far easier. If many different paths have already been explored, it makes it possible to narrow your own view and focus on the directions that are most likely to prove successful for you.

Bibliography

Collins-Thompson, K., Callan, J. (2005) A Language Modeling Approach to Predicting Reading Difficulty. https://www.cs.cmu.edu/~callan/Papers/hlt04-kct.pdf.

Curto, P., Mamede, N., Baptista, J. (2015) Automatic Text Difficulty Classifier - Assisting the Selection Of Adequate Reading Materials For European Portuguese Teaching. *CSEDU 2015 - 7th International Conference on Computer Supported Education.* http://www.inesc-id.pt/pt/indicadores/Ficheiros/11043.pdf.

Joachims, T. (1998) Text Categorization with Support Vector Machines: Learning with Many Relevant Features. https://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf.