Complexity of literary passages - Kaggle competition.

Project Report - Udacity -Machine Learning Engineer Nanodegree.

Index

Abstract

Tackling the reading ease of a excerpt is not an easy task to automate, people that are use to reading may detect some characteristics on the text that may hint how difficult it will be to read for children and fellow readers, but there is no rule of thumb, there are several complexity indexes but none of them fully captures all the possible rules that give a text its complexity, so Comonlit decided to give it a try from a different angle, let's try to solve this problem with machine learning!

In the following pages you will find my thought process to a solution for this problem, being my primary goal to beat the most common benchmark index for complexity in the test set: the Flesch-Kincaid test (**FRE test from now on**).

TLDR:

The final model got an RMSE of 0.7739 in the test set, better than the benchmark model score 0.966, this implies almost a 25% improvement.

Problem Statement

“Currently, most educational texts are matched to readers using traditional readability methods or commercially available formulas. However, each has its issues. Tools like Flesch-Kincaid Grade Level are based on weak proxies of text decoding (i.e., characters or syllables per word) and syntactic complexity (i.e., number or words per sentence). As a result, they lack construct and theoretical validity. At the same time, commercially available formulas, such as Lexile, can be cost-prohibitive, lack suitable validation studies, and suffer from transparency issues when the formula’s features aren’t publicly available.”

- Quoting Kaggle CommonLit Readability competition.

I see a lot of potential with the idea, imagine yourself for a moment, if you could know beforehand buying a book the complexity of the text that lurks inside, that may encourage you to find books that fit your comfort level. I could easily imagine every book in a shelve with an stamp like the “best seller” one but stating the complexity. Then buying books for kids will be easier as well as for foreign people to pick books written in other languages also for older people with cognitive problems, the possibilities are endless.

Since the current indexes for text complexity aren’t perfect, nor accurate with real people opinions, there may be a way to improve current result with the help of NLP and ML. Finally making a humble contribution to cause greater than myself as finding a free, open-source complexity index is a reward on itself.

Datasets and Inputs

The dataset have 6 columns, and 2834 rows. The columns are: id, url_legal, license, excerpt, target and standard_error. The id is just the excerpt identifier, both url_legal and license contains info about the excerpt license of use. The excerpt is the actual text you want to find out the complexity, the target columns is the average reading ease according to a panel of teachers, and the standard_error is the error of such average. Please notice the target variable is a continuous number, this means this is regression problem.

They also include a test dataset, but this ones doesn’t have neither the target nor the standard_error. So, we must train the model solely using the excerpt text.

In addition to the above dataset I made some research and found an open source repository which contains some info regarding the frequency of English words, specifically for the top 50.000 more frequent words. My idea is to use this auxiliar dataset as well to determine the “rarity” of the words in the excerpt. The link to this repo is here: https://github.com/hermitdave/FrequencyWords.

Benchmark Model

I will use two models as benchmark, first a basic linear regression between the Flesch-Kincaid test index and the target variable, this is the most popular complexity index so performing better that it, its an accomplishment in on itself. This model will give us an idea of how good are classic indicators at estimating the real complexity of a excerpt.

Secondly I'll use other Kaggle contestant RMSE to compare against mine.

Evaluation Metrics

The benchmark for the model will be the root mean square error (RMSE) using the predictions of the model and the real values as inputs to the formula.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Exploratory data analysis

The exploratory_analysis.ipynb file contains all the details, here you can find some of the most important insights about the data.

Train, validation and Test set.

The whole dataset contains 2833 rows, very limited for the kind of problem we want to solve, so I needed to use as many records as possible for training, otherwise it would have been very difficult for any model to learn. For the kaggle comptetion there is a test set that is not available to the main public, so the participants doesn't need to evaluate how good the model was for themselves. Since I don't have access to that dataset, I needed to divide the given data into different test to drive the development of my model forward. In summary:

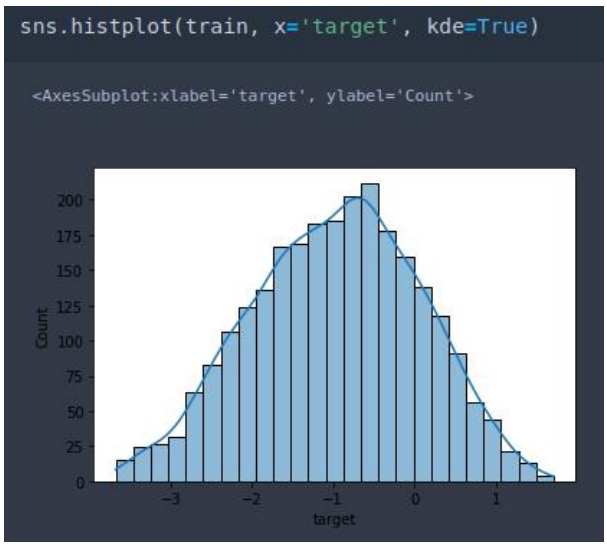
Train set (90%) have 2550 rows.
Validation set (5%) have 142 rows.
Test set (5%) have 141 rows.

Just after dividing the datasets, I proceeded to explore the data, some of the main results are:

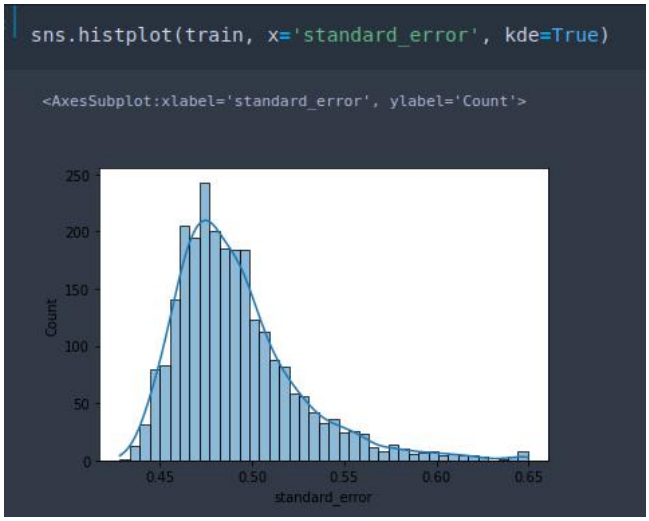
- Target Variable Distribution

There is some obscurity from Comonlit about how to interpret the target variable, as far a some people suggest in Kaggle forums, the target is (after normalization) the average complexity score that several professors gave to the given excerpt. That also explains why the data have a standar_deviation variable, which is (according to this assumption) the standard deviation of such average.

The target variable have a bell-shaped distribution. With a mean very close to -1 and a standard deviation of 1.03, quite high for a variable that ranges from -3,67 and 1,71.



- Standard deviation variable distribution

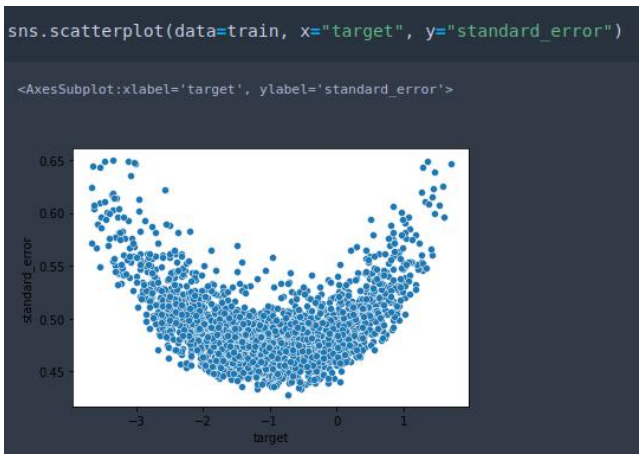


The standard deviation variable for a given excerpt (that shall only be used for the train set, since it is not present in the kaggle test set) have a skewed distribution, whit most of the values around 0.49, again quite high value considering that the target variable ranges from -3,67 and 1,71. This number can be interpreted as, even for expert teachers, they can heavily disagree when trying to determine the complexity of a given text.

This means that whatever model we try to train using this data will struggle a little estimating the target compleity variable, as it is an average among multiple people with opinions that varied a lot. We should take this into consideration when trying to measure the performance on the final model.

- Relation between target and its standard deviation.

If we plot the target variable (average complexity for a given excerpt) against the standard deviation of such average we can spot a clear pattern:



The scatter plot tell us that the easier and hardest excerpt were the ones were the teachers disagreed the most (higher standard deviation) when punctuating the excerpt difficulty. I would have expected rather the opposite, but it is not like that, the more 'average' looking excerpts were the ones were the teachers agree on the score the most. I can foresee the machine learning model will struggle more to identify the hardest and easiest texts because of this.

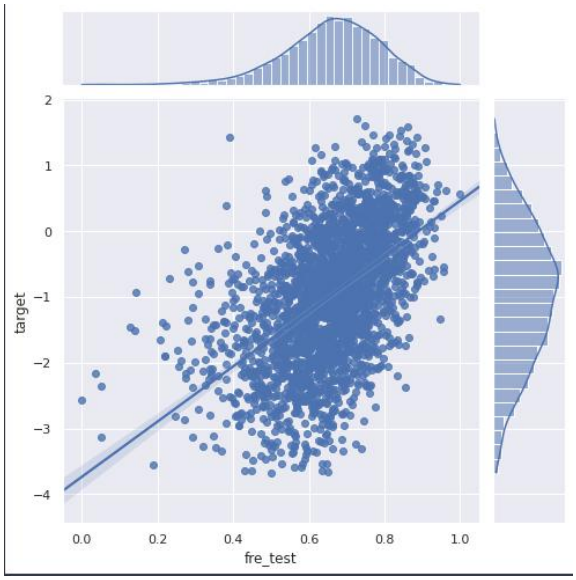
- Benchmark Model Score

Remember the benchmark model was a linear regression between the most recognized complexity index, the FRE test and the target variable. The linear model was trained on the train set and them applied on the test set directly, obtaining the following results:

```
test set R2: 0.188189
test set mae: 0.767187
test set rmse: 0.966072
```

We can see that the RMSE was 0.96, which is very high error considering the target variable varies between -3,67 and 1,71. This are the metrics we want to beat with our custom model.

Here you can see in a visual matter how good the benchmark model is:



It's easy to see that the line is not fitting very well the cloud of points.

First model

My first thoughts on how to tackle the problem was to combine the FRE test with other commonly used reading ease test and some custom functions I wrote to find a better model that the benchmark. You can find in the tools.py script the custom function I wrote to draw down the complexity:

1. **A rarity index**, this function can receive any excerpt as input at it will output the logarithm of the 'average' frequency of the words in it. In order to know how rare a word is, I used the https://github.com/hermitdave/FrequencyWords repository. This contains a list of the most common 50,000 english words, and it frequency after checking thousands of books. I.e:

The sentence: 'I love you' will have a rarity index calculated as:
Rarity = np.log(freq('I') + freq('love') + freq('you'))

Where the freq() function is just the frequency that the FrequencyWords repository gives to each word. If a word appears in the excerpt but not in the top 50,000 words repository, it's frequency is considered to be 0.

The logic behind this custom index is that, the rarer the words in a text are, the more comple the excerpt may be.

2. **num_unique_words**: This custom feature as the name suggest, simply counts how many unique words an excerpt have. I excepted that the more hard the excerpt is, the higher the number of words it will have. For instance.:

'The cat was angry, the human give her no food' contains 9 unique words since 'the' appears more than once.

3. **num_punct_marks**: as the function above, I'm simply counting how many punctuation marks there are in the text, and normalizing the result against the total number of words. Example:

'The cat was angry, the human give her no food' have 1 punctuation mark and a total of 10 words, so the function will return 1/10.

The idea is that, the more punctuation marks there are, the shorter are the ideas the excerpts is trying to explain, this the complexity should be lower.

4. **Average_word_len**: This function simply return the average longitude of the words for a given excerpt. The idea is that the bigger the words are on average, the harder is the text to understand, for example:

'The kid eats ice cream ' have an average word longitude of (3+3+4+3+5)/5 = 3.6

I applied all those custom features to my train dataset, also the FRE test and 5 more complexity indexes that were freely available:

- flesch_reading_ease
- gunning_fog
- smog_index
- dale_chall_readability_score

The details about this models can be found inside the jupyter notebook.

First model Score

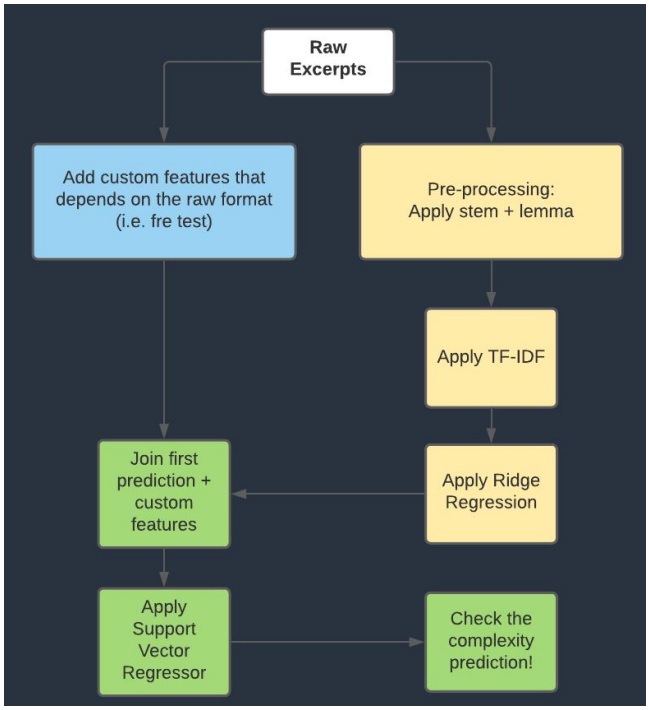
After some feature selection process I decided to use only the rarity index, the avg_word_len, FRE_test and DCRS_test as inputs for the first model. Also I tried with several regression models, being the Multi-Layer Perceptron regressor the one that score the best in the validation set:

```
MLPRegressor vs target... R2: 0.374
MLPRegressor  vs target... MAE: 0.631
MLPRegressor  vs target... RMSE: 0.805
```

Having and RMSE of 0.805, this was considerably better than the 0,96 RMSE the benchmark model presented. Please look at the notebook for further details.

Final model architecture

After a lot of iterations, weeks of reading, researching and discussing on Kaggle I came to the best model using this process:



You can consider it a 3 step process, first you need to stem and lemmatize all the words in the excerpt, this process reduces the complexity of the text since it transform every word to its 'root' thus reducing unnecessary complexity from it, then the model can learn better.

After applying this cleaning process you can transform the text into a tf-idf sparce matrix, this effectively transform the cleaned words into a matrix of numbers, were every number represents the importance that each word carry in relation to the whole text from the text. Using this transformation was the key to feed the model relevant information. You can check the notebook for the details but I tried several ML models in search for a good performer, the best one result being a Ridge Regression with default parameters. The output of this model is what I consider to be the 'baseline prediction'. Please notice that the search for the best model was done always comparing the metrics results in the VALIDATION set, thus avoiding to be overfiting the training set.

The following steps adjust the 'baseline prediction' to be closer to the actual target, in order to do so, I added the custom functions I wrote for the first model and trained a second model that effectively adjust the baseline prediction to a better one. The model that performed the best was a Support Vector Regressor with a regularization parameter C equal to 5.

```
SVR_C5 vs target... R2: 0.571
SVR_C5  vs target... MAE: 0.552
SVR_C5  vs target... RMSE: 0.666
```

Final results

When I was confident enough that I had created a decent model I decided to try it with new data in the test set. Remember this were the results of the benchmark model:

```
test set R2: 0.188189
test set mae: 0.767187
test set rmse: 0.966072
```

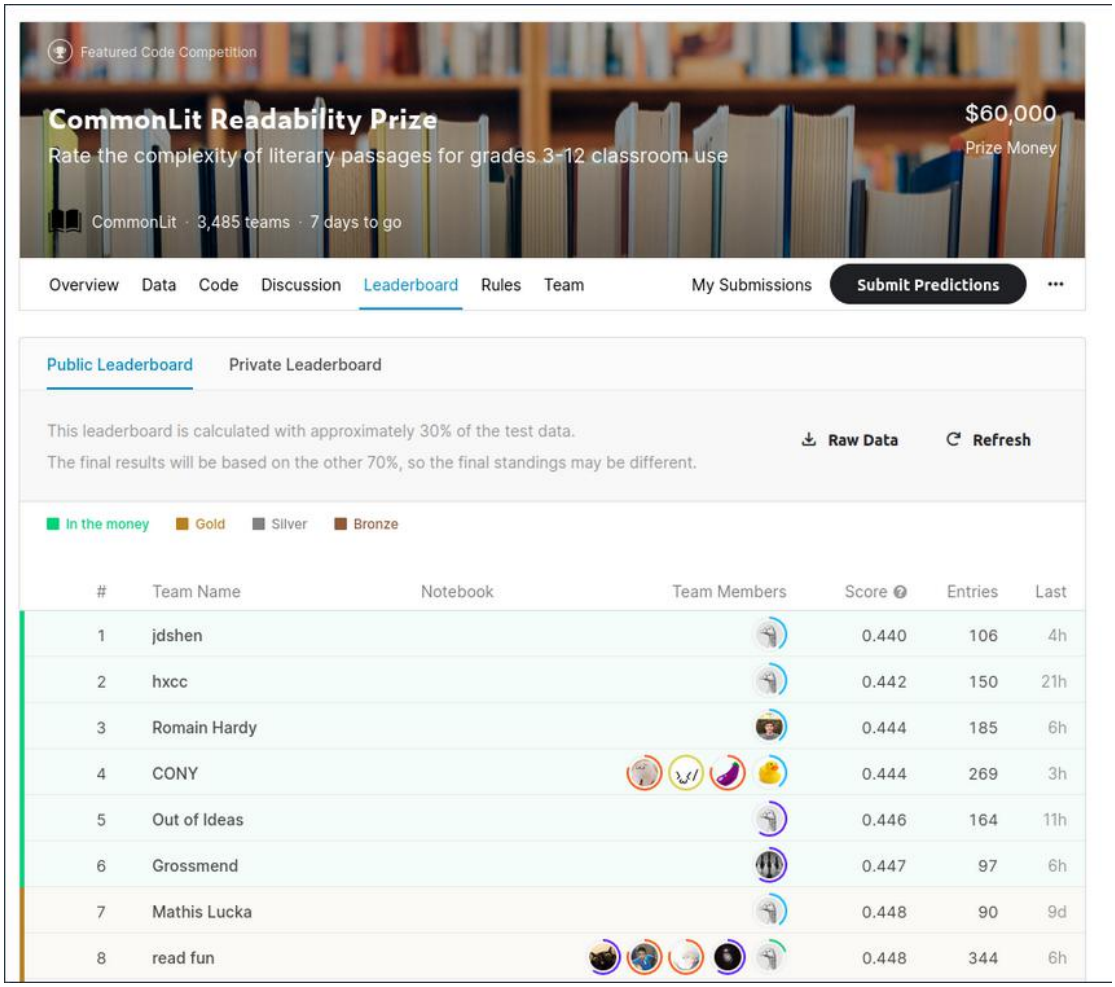
So, I applied the whole process to the test set obtaining the following metrics:

```
SVR_C5 vs target... R2: 0.478906
SVR_C5  vs target... MAE: 0.615625
SVR_C5  vs target... RMSE: 0.773999
```

Yeah! This means the custom process I design have a better accuracy than the benchmark model! In fact, the RMSE is 24% lower that the benchmark's RMSE.

Kaggle results

Comparison with the Best Kaggle submition so far:



The best performer got an RMSE of 0.440, well above my result but obviosly he is spending more time on the project that I can possible can.

Bear in mind:

- I trained my data on just a portion of the whole training set.
- The test set that kaggle uses is private and different than the one I used, so is not fair to compare RMSE unless we use the same test set.
- The person on the top of this list may be overfitting the testing set (he/she have tried a total of 106 times against the same test set)
- I used the test set just once to avoid overfiting it.