Predicting Sentence Complexity

By Christian Dallago and Chris Lynch. Report for Milestone II - September 2021
Our code for this project can be found at https://github.com/qiopta/MADSm2.git

Part A: Supervised Learning

Motivation

It has been shown that certain words, phrases and sentence structures can make a given body of text easier or harder to read. Text simplification is seen as a way to increase access to knowledge and therefore systems have been developed to automate this task. D'Amour et al., 2020 [1] showed that end to end simplification models become 'overfit' to the training data. In response to this, Garbacea et al., 2021 [2] demonstrated that the prior steps of first predicting sentence complexity then identifying which text needs to be simplified, improve the outputs and explainability of automated text simplification models against unseen data. Our supervised learning task in this report emulates that first step, that is to train a classifier to be able to accurately predict if a given sentence is complex, and would therefore benefit from simplification. This project therefore demonstrates a binary classification task.

Data Source

The data for this project has been provided on Kaggle [3] by Cristina Garbacea in the form of two CSV files, representing a labelled training set and an unlabelled test set. The training set comprises 416,768 sentences, with labels to identify 1: a sentence needs to be clarified or 0: the sentence does not need simplification. The training set was balanced, with each half of the data being labelled either 1 or 0. The test set is 119,092 unlabelled sentences. Both the train and test sets were originally extracted from Wikipedia articles. We used the Kaggle API in our scripts to download these data and also to make submissions of our test set results to the Kaggle leaderboard.

Methods and Evaluation

All of our analytics were conducted with Python, we used Google Colab to create and share the majority of our code. For the BERT classification task we sought a higher compute power and longer term storage of our work together with a stronger IDE experience. To achieve this we instantiated a VM on Microsoft Azure, using scripts to achieve a more manageable machine learning pipeline. Our work on the BERT model was based upon Abishek Thakur's IMDB sentiment classification repository on Github [4], which we fine tuned for the purposes of sentence complexity classification, running predictions and producing more detailed logs during training runs.

We initially set out a **baseline** for the supervised learning task using two different dummy classifiers (a random or ‘uniform’ classifier and a ‘most frequent’ one). Our approach to evaluation of these dummy classifiers performance was to use only a sample of the training data (10,000 rows) and to run a five-fold Cross Validation with a shuffle split. The five-fold validation showed little variation between each fold and returned a mean accuracy score of 49.999% . We took this to be the baseline against which all other models could be compared. At each stage in the process we made submissions of our classifiers predictions using the test set to the public and private kaggle leaderboards and found very little difference between our local estimated accuracy and that reported on the leaderboard.

The next step in our supervised learning experiments was to create and **compare different classifiers**, initially on the same 10,000 data sample as the baseline. We created the following classifiers with default hyperparameters to achieve the following results: Random Forest 55.524%, Logistic Regression 55.798%, Support Vector Machines SVM 56.708%, Gradient Boosted Decision Trees 56.888%. We achieved promising results considering we were training on only a 10K sample of the dataset, with the Gradient Boosted Decision Trees Classifier achieving the highest score.

We then began experimenting with **hyperparameter tuning** using scikit learn’s grid search cross validation and randomized search cross validation. We found that the randomized search was more efficient - taking about the same time to train as grid search but typically returning slightly stronger results. For our 10K sample with a logistic regression classifier, the baseline performance was 56.866%[1]. After tuning hyperparameters this increased marginally to 57.17%. The optimal parameters found in our search were:

```
(C=54.555947811685144, class_weight=None, dual=False,
fit_intercept=True, intercept_scaling=1, l1_ratio=None,
max_iter=100, multi_class='auto', n_jobs=-1, penalty='l2',
random_state=42, solver='liblinear', tol=0.0001, verbose=0,
warm_start=False)
```

Similarly we ran the same 10K sample through a Random Forest model with hyperparameter tuning using randomized grid search cv, this achieved a slightly better result of 57.48% with the following being determined as optimal parameters:

```
(bootstrap=True, ccp_alpha=0.0, class_weight=None,
criterion='gini', max_depth=9, max_features='auto',
max_leaf_nodes=None, max_samples=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=1300,
```

[1] Slight changes occurred in baselines classifier results due to differences in the random sample we drew from the training data set.

```
n_jobs=-1, oob_score=False, random_state=42, verbose=0,
warm_start=False)
```

We also experimented with the **parameters for the scikit learn tf-idf vectorizer** to look at the impact on our results. From this we discovered that our first choice of ngram ranges (1,2) was best. When we used (1,1) the results did not improve and as we increased the number of n-grams such as (1,3) or (2,3) or higher, our results became worse. This suggests that for the task of predicting sentence complexity unigrams and bigrams are the most useful ngram parameter for the given dataset.

Another tfidf parameter that we investigated was changing the min_df. As we lowered the threshold for min_df, it had the effect of increasing the number of features, and with the LR classifier this led to better results. Initially all our baseline models were set with min_df = 100. Increasing this did not improve scores but decreasing it did, optimal results were achieved when then min_df = 25 was used. At this level the vectorizer produces 691 features (compared to c. 80 features when min_df = 100) and this increase in the number of features resulted in a c.2% boost to the accuracy of the classifier (on CV testing).

Increasing the sample size improved the overall accuracy of our models. For our baseline Logistic Regression model increasing the sample size to 200K produced an accuracy score of 66.17%.

Before beginning our analysis we knew from the Garbacea et al., 2021[2] paper that fine tuning a **BERT model** was likely to produce the best results. This was proven with an initial result of 76.502% (Public Score) from the BERT model from its first EPOCH. The BERT model took 2.5 hours per epoch to complete and we set it to run for 10 epochs, which we stopped after 7 epochs as no performance improvements were found. **Our best result at this stage was 77.496% (Public Score)** if the leaderboard on Kaggle were live this would have been a top 5 result and is approaching the state of the art performance reported by Garbecea et al. of 81.45%. It is worth noting that our local test results were about 1% lower than the submissions to Kaggle which was a pleasant surprise, showing our model does well on out of sample data and that our local testing and development strategies were effective.

We also experimented with incorporating some of the included materials, notably the **Age of Acquisition dataset**, as we were interested in finding out whether the age at which a lemmatized word is learned could contribute to increasing the accuracy of the model. Splitting the training data based on difficulty, we calculated and averaged various statistics, including the **mean, median, minimum and maximum** age, to determine if there were any apparent differences. Testing on our 10K sample, the averages from both simple and difficult datasets appeared strikingly similar.

Despite these unpromising initial results, we created a more comprehensive experiment, calculating the same Age of Acquisition metrics for each entry in the randomized 10K sample, and added those to a new dataframe. Training a Logistic Regression model with default parameters on these stats alone, not including vectorized sparse data matrix of the text itself, we achieved an accuracy score of **59.730%**, scoring higher than our initial default classifiers described above. The next step was to combine these new metrics with the vectorized text data to further increase the accuracy.

After vectorizing the text data in the same manner as we had done previously, we then extracted the sparse matrix values of each individual sample as an array to add to the dataframe. However, due to the fact that using multiple vectors alongside the Age of Acquisition metrics would result in an error when trying to fit the classifier, we decided instead to take the average vector for each sample. We attempted using two separate vector averages, one including the 0.0 value that every set of vectors contained, and one where a slice of the vectors was taken to not include the 0.0 value, unless it would otherwise be empty. After compiling both of these averages into the dataset and training a Logistic Regression classifier on this final model, the resulting accuracy score reached **60.067%**. This was the highest score achieved on a 10K sample.

Exploring Machine Translation

Working with text data initially led to an interest in utilizing **machine translation** to compare results with the original English text, not as a means of improving the overall accuracy score of the model, but more out of general curiosity and a desire to experiment with current machine translation techniques. Using PyTorch and pre-trained open-source translation pipelines from HuggingFace [5], we were able to translate a subset of the original dataset into Italian, Russian and Simplified Chinese to represent linguistic diversity, and because they are languages Christian is familiar with. The language-specific models were also downloaded from HuggingFace [6].

Using the same 10K randomized dataset, each sample was fed into each of the three translation pipelines. CSV files of the translations have been included in our documentation. Both English and target language stopwords were dropped when vectorizing the text, and it should be noted that because NLTK does not natively include a list of Simplified Chinese stopwords, as it does for the other languages, one had to be found externally through the Chinese search engine Baidu.

Comparing both unigrams and bigrams with the number of features set at 25, we trained a Random Forest Classifier tuned to the default parameters. The original English data scored the highest at exactly 58.458%, with Russian in second with a score of 58.408%. Italian came in third with a score of 57.403%, while Chinese scored the lowest at 52.878%.

The results are interesting, though there are far too many factors to determine whether there is indeed linguistic bias in the pre-trained methods we utilized. In order to pursue this line of research, it would be beneficial to include a larger sample size, a different dataset and new languages to compare the results with those we discovered here, along with the effects of certain types of classifiers on different languages.

Failure Analysis

On the computing failure front, we initially found Microsoft Azure's pre-configured VMs to be quite difficult to work with as some images come with GPUs but without the drivers installed. On three VMs that we built we failed to get the GPU drivers installed, even with extensive support from Microsoft. Eventually we did overcome this by creating a VM within Azure's Machine Learning Studio environment that had both the GPU and the appropriate drivers to use them. Even so, we found that running an Azure VM via a local IDE was great for coding but unreliable for long training times. When our local machine eventually went to sleep the process terminated - this was resolved simply by running our scripts from the command line within a browser pointing to our VM/Azure environment inside Microsoft's Machine Learning Studio.

On the analytical side our initial attempts to improve supervised learning with a prior step of unsupervised learning showed no improvement and therefore could be considered a failure. One effort to make our clusters more 'human understandable' was to remove the left and right bracket notation within the text. This proved helpful in creating clusters that were more readily understood by their most significant terms. However, simply removing brackets actually weakens the classifier, as it is likely that they have a significant weighting in terms of sentence complexity.

In our initial experiments the pd.sample method produced a different 10,000 rows of data for testing upon each investigation. This gave us slightly fluctuating results. The advantage of this was that we found our results and methods were largely robust to the data sample. The disadvantage was that the repeatability and explainability of our results was impacted. Overall we found the advantages to outweigh disadvantages in our early exploratory work, and as we progressed to building the final pipeline we were no-longer sampling (using the entire data set to train our classifiers) so the disadvantages went away.

Part B: Unsupervised Learning

Motivation

We set out to explore if unsupervised learning tasks could help us improve our supervised learning. Ideally we could introduce some aspect of unsupervised learning into our machine learning pipeline to build on any pre-processing that feeds the data into our classifiers. The main tasks we explored were scaling, dimensionality reduction, and clustering.

By observing some of the results of these experiments we also hoped to increase the explainability of our classification results, such as getting a stronger feel for the data (words, phrases and sentences) that occurred in the data with a label of 1 compared to those labelled 0. That is to say, we aimed to investigate whether some types of data - found in clusters - were more likely to be found in simple or complex sentences.

Data Source

The data used in this task is exactly the same as that described in Part A above, namely "UMich SIADS 695 Fall20: Predicting text difficulty" [3].

Methods

Our initial exploration of the data with unsupervised methods used the 10K-row sample from before, and a logistic regression classifier with the same parameters as used in our data exploration for Part A. This gave comparability of results in an efficient way before any effort was made to 'scale up' the results into our best classifier's pipeline.

Scaling

We ran a for loop to test the impact of the following scalers on classification accuracy: MaxAbsScaler, StandardScaler (with_mean = False) and RobustScaler(with_centering = False). These scales and parameters were chosen specifically for use with the sparse representations of the data produced by our tf-idf vectorizer. Again the tf-idf vectorized data input was the same as used in our baseline classification efforts to ensure consistency. Those scaled inputs were then passed to our cross validation with a shuffle split, as previously used. The mean CV score for each method gave a c.0% - 2% boost in accuracy, depending on our sample, from those in our baseline supervised classification efforts. Typical results were: baseline unscaled LR CLF produced 55.798%, MaxAbsScaler 57.796% StandardScaler 57.564% and RobustScaler 55.516%. Overall MaxAbsScaler produced better results, whilst RobustScaler would sometimes perform even worse than our baseline with unscaled data.

Dimensionality Reduction - Latent Semantic Analysis

Given that our pre-processing created a sparse matrix (from the tf-idf vectorizer) we initially used TruncatedSVD from scikit learn to reduce the dimensions of our vectorized and scaled 10K sample. We used this to reduce the number of features from c.80 (depending on the sample, in a range from 76 to 84 in our experiments) to 8. Simply feeding these clusters with reduced features eroded the performance of our baseline linear regression classifier from 55.798% to 54.294% (without the pre-processing step of scaling the performance is even lower at 53.534%).

K-Means Clustering and Cluster Labeling

Initially we decided to utilize K-means clustering as a way of gaining insight into the nature of the data. Using the same 10K dataset, and partitioning it based on the difficulty label, we calculated both the Calinski-Harabasz and Davies-Bouldin scores to determine the optimal number of clusters. As both methods resulted in different scores for each dataset, we experimented with both to see how the results compared.

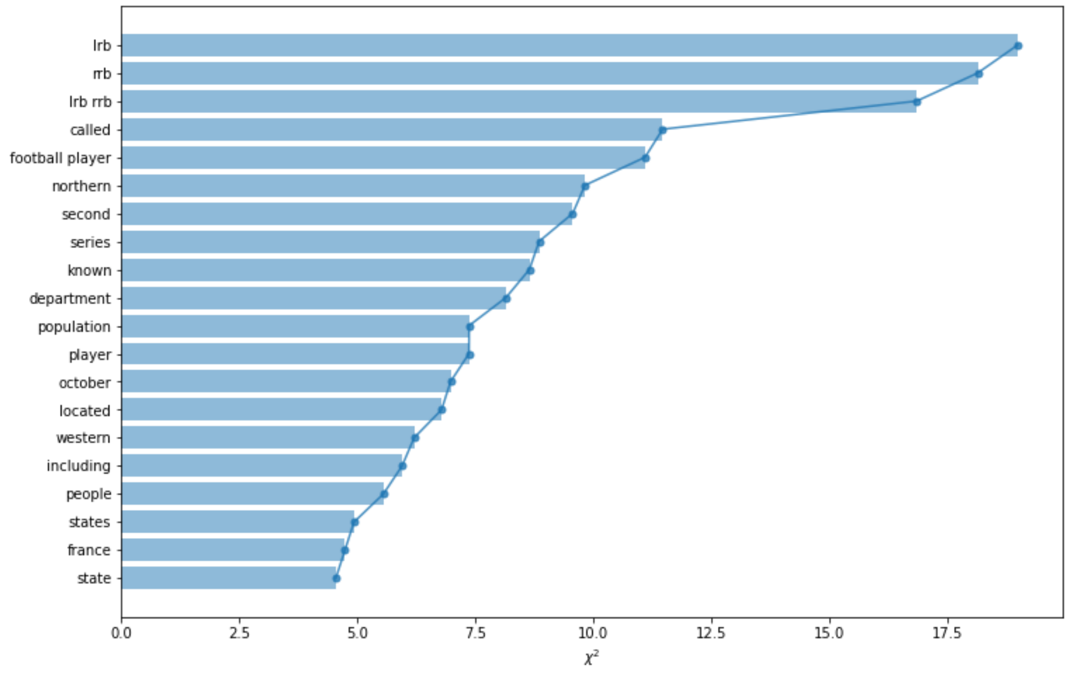
The results were not particularly enlightening as terms tended to be repetitive across datasets and overwhelmingly included the bracket placeholder terms "lrb" and "rrb". We decided to take it a step further and use cluster labeling techniques to better understand the differences between datasets. We used information gain as a means of dimensionality reduction to determine unique labels in each dataset, again comparing each of the previously computed ideal number of clusters. This resulted in a series of cluster labels that were more distinct than those from K-means clustering, and also removed the repetitive bracket terms.

Yet even cluster labeling did not provide sufficient insight into the text complexity of the dataset, and the labels were ultimately quite random. We attempted to use the cluster labels to improve the accuracy of the supervised learning model by including them in the training of the classifiers, vectorizing a list of the top 5 label terms for each cluster then taking the mean of the vectors. However, given that there were only two possibilities for each vector cluster, depending on whether they were simple or difficult text samples, this inflated the accuracy to over 90%, and wouldn't have any real utility when predicting labels in the test set.

Unsupervised Evaluation

The unsupervised scaling methods we used had utility limitations, we found no practical utility from scaling the data. We used Max Abs Scaler, Standard Scaler and Robust Scaler - feeding the processed data to our baseline logistic regression classifier. In each case we found accuracy slightly reduced by this scaling step.

We investigated the Tf-idf feature importances, using Chi2 test to see their impact on the clustering aspects of unsupervised learning. Our approach to this was inspired by Susan Li's article [7]. The chart below shows that the bracket notations were the most significant factors, and assume this lends itself to an indication of sentence complexity.



For our K-means clustering tasks, we experimented with different cluster scoring methods, such as the Calinski-Harabasz and Davies-Bouldin methods to determine the most suitable number of K clusters for both simple and difficult datasets. Both datasets scored differently, with the simple dataset resulting in K = 2 using the Calinski-Harabasz scoring method, and K = 8 using the Davies-Bouldin method, and while the difficult dataset resulted in in K = 4 using the Calinski-Harabasz scoring method, and K = 2 using the Davies-Bouldin method. There is certainly a tradeoff between choosing one over the other, yet we found that, perhaps given the smaller sample size used in our unsupervised learning tasks, as well as a clear idea how to apply the results to our supervised learning portion, there wasn't a real benefit to using one over the other. Given more time and resources, the effects of choosing a particular scoring method would likely be more consequential.

Discussion

In Part A we learned many things we already knew intuitively, but experiencing them came as a surprise, such as how having more data in the training dataset would improve the classifier results. Whilst we had read so many exceptions to this rule, it was refreshing/surprising to see it again in this context.

Similarly we knew that training models on large datasets would take time, but we had gotten used to using the curated data within Coursera as part of the MADS course - an environment in which a ten minute delay means you are probably wrong. Returning to a more "real world" environment such as the one for this project meant significantly longer training periods, from hours on some classifiers through to results measured in days for our BERT model (even with GPUs and high powered VM configurations). We noticed that our peers were experiencing the same thing from conversation in slack, and there was an element of novelty in this.

We were not surprised that the transformer (BERT) model performed best, as we had read the Garbacea et al. [2] paper and were expecting this. However we expected gradient boosted decision trees to be significantly better to simpler linear classifiers on the supervised learning task. Whilst they did perform better, the accuracy improvement as measured in five fold cross validation with a shuffle split, was often within a single percentage point range on our experimental 10K data sample. With more time we would have wished to test this differential classifier performance on larger data sets.

With more time and resources we would want to explore similar impacts of feature engineering for the BERT model as we did for the other classifiers, such as an exploration of the BERT tokenizer and how this might be adjusted for differential impacts on the classification task. In a similar vein, we would have liked to explore hyperparameter tuning to improve the performance of this model, but decided that a 'top 5' result was sufficient, and therefore turned our attention to Part B.

The unsupervised learning tasks were much more exploratory in nature compared with the supervised ones in Part A, and it wasn't always clear whether the results would be of any use.

Since we were working with text data, the initial reaction was to use unsupervised learning methods to better gain insight into the nature of the data, more specifically what differentiated the simple text from the difficult.

The results were surprising in that our cluster labels more or less seemed to be related to geographic locations or national football/soccer teams, and yet there didn't seem to be any real differences between the simple cluster labels and difficult cluster labels. We were also surprised that the results were not directly applicable in terms of helping with the prediction task. Reintroducing those labels as features in the supervised learning task was problematic and led to an inflated accuracy.

Given more time and resources, it would be intriguing to attempt to use unsupervised learning tasks to better understand how the data was sourced from Wikipedia, for example if the samples were completely random, or if there was in fact some underlying method to their collection. Because we were limited to working with only a small sample of the data due to processing restrictions, this wasn't feasible for us. However in theory, this manner of unsupervised learning could prove useful in uncovering any **data leakage**, as this could certainly give a competitive advantage in Kaggle competitions.

As an aside, working with other languages was also quite surprising and has sparked some curiosity into further research, particularly in regard to **linguistic bias** or preferential treatment in text vectorization and classification. It would be fascinating to create a new experiment to determine whether the decreased accuracy in other languages was a result of the translation, that the classifiers natively have better support for English, or some other reason. Given more time and resources, it might be useful to utilize the unsupervised learning tasks such as cluster labeling to see how labels compare across languages as well.

The immediate task of predicting sentence complexity didn't seem to prompt direct **ethical issues** for either Part A or Part B. There may however be ethical issues arising from related upstream (data collection) or, more likely, downstream tasks (in a machine learning pipeline).

> The upstream issues will only have an effect in the application of the tools and models which have been produced. These issues could arise from the way in which the data was sourced, scraping from Wikipedia, and prepared for this task. We were not advised if the data had had much pre-processing before publication to Kaggle for our use. However if the data was simply scraped in a random manner it could reflect a number of biases from the original contributions and contributors to Wikipedia, be they race, ethnicity, nationality or political views. These biases would then "infect" the models which are trained upon them which could have a downstream effect on the sentence simplification task - for example, by assuming minority group grammar is more complex - and thereby eliminating a potentially important perspective.
>
> The downstream issues will depend on the application of the sentence complexity prediction in a wider machine learning pipeline. From Garbacea [2] we know that this step was intended to improve the "auto simplification" of text inputs by eliminating inputs that were already simple and which would therefore lose meaning with further

simplification. It is possible to conceive of downstream ethical implications where groups of people might have their intended meaning obfuscated by unhelpful simplification which could flow from biases in the modelled classifier. Beyond a loss of meaning, a trend towards generalized simplification could lead to a “loss of language” and a concomitant loss of culture.

Statement of work

Both of us worked on initial, but different baseline classifiers for the supervised learning part. We also co-developed this project report, the backlog for the work supported on a Trello board and produced ipynb’s in Colab. Christian did all the exploratory work on machine translation, and was our main lead for unsupervised learning cluster labeling. Chris created the pytorch code for the BERT model, based on [4] which was designed for sentiment analysis, and repurposed it for the supervised learning aspects of predicting sentence complexity. Chris also investigated cloud computing options to support the BERT effort which required more compute power using Visual Studio Code and Microsoft Azure’s environment.

References

[1] D’Amour et al., 2020
[2] Garbacea et al.,2021
[3] Kaggle https://www.kaggle.com/c/umich-siads-695-predicting-text-difficulty
[4] Abishek Thakur’s repo https://github.com/abhishekkrthakur/bert-sentiment
[5] Translation pipelines https://huggingface.co/transformers/main_classes/pipelines.html
[6] Language models https://huggingface.co/models?filter=translation
[7] *https://towardsdatascience.com/latent-semantic-analysis-sentiment-classification-with-python-5f657346f6a3*