Summary of Winning Algorithms for CLEAR Corpus

Overview

All algorithms follow the basic pattern of using a training set (60% of the dataset) to develop a model that was used to predict the ease of readability scores for the test set. The major differences between the winning algorithms revolved around:

1. How did they approach the training data? (i.e., Did they find additional data to supplement the training data to make the training set larger, or did they use the training set as-is?)
2. What and how many models were used for training? (i.e., Did they use many smaller models, or did they use larger models? Which base models did they choose to use?)
3. How were the models trained and fine-tuned to improve the model performance? (Any explanation of this would be full of terminology)

Below, we provide some simple definitions of key terms and address the differences between the algorithms while also providing an accessible, brief explanation of how each winner's specific algorithm differed from the other winners. It is important to note that all of the algorithms had similar accuracy metrics, so none of winning algorithms are really "better" than others. All of the winning algorithms could be used in a production environment, and algorithm selection should be based on practical feasibility.

Definitions

Models: A model represents what was learned in a machine learning algorithm. It can also be thought of as a file that is trained to search through large amounts of data to find patterns and make predictions. There may be multiple models used in a single machine learning algorithm.

Algorithm: A procedure that is performed on data to create a model (e.g., each of the winning *algorithms* from the Kaggle challenge). This can get confusing because algorithms can contain models. One way to think about models versus algorithms is that models are concrete entities, while algorithms can be changed. For example, one might want to improve their *algorithm* to create a more accurate *model*.

Training/test set: Developing machine learning algorithms typically involves “training” and “testing” the algorithm with a “training” and “test” dataset. The training and test datasets are each a subset of the total dataset, and the training dataset should be much larger than the test dataset. For these algorithms, 70% of the full CLEAR corpus was used as the training set, and the remaining 30% of the dataset was used as the test set.

Training: The “training” stage in the process of developing a machine learning algorithm involves providing the algorithm with examples of inputs and the corresponding outputs expected if the algorithm is given those inputs. This set of inputs and outputs is the training set (also defined above), and the model uses this training set to learn how the outputs can be predicted based off the inputs. In this Kaggle challenge, the algorithms were “trained” on the text samples (i.e., inputs) and their corresponding Bradley-Terry ease of readability scores (i.e., outputs).

Testing: The “testing” stage is done after the training stage, and it is for evaluating the performance of the fully trained model. In this stage, new input data the model has never seen before (i.e., the test set) is run through the fully trained model to get the new predictions. The predictions made from this stage are tested for accuracy using various measures (e.g., Root Mean

Squared Error). For these algorithms, the input is the text samples from the test set, and the output is their *predicted* Bradley-Terry ease of readability scores.

Fine-tune: Fine-tuning refers to making small adjustments in a process to obtain more accurate results. A common use for fine-tuning is to fine-tune an existing pre-trained model (e.g., BERT) which was downloaded in order to customize it to a specific dataset for better performance.

k-fold Cross-validation: Cross-validation is a statistical method to evaluate the effectiveness of a machine learning model. An example of cross-validation is using the train/test data sets listed above. Specifically, *k-fold cross-validation* is a type of cross-validation which was used in many of the winning algorithms because it is easy to use and works well with small datasets. This involves shuffling the training data, breaking the training data into *k* number of folds, using one of the folds as the test set and the remaining folds as the training set, running and evaluating the model with those train/test folds, doing it again until each fold has been the test set once, and then averaging their evaluation scores to get the final performance of the model. It may help to remember the *k*-fold term is one which already exists in English (e.g., splitting something "three-fold").

Dropout. Dropout is a method used in machine learning to prevent overfitting. Neural networks consist of multiple layers and nodes; dropout simply means that certain layers or nodes are randomly ignored during training.

Ensemble approaches. Ensemble models (EM) combine the predictions of multiple different models to derive a final prediction that is improved in terms of variance and accuracy. The output of each model contributes differently to the final outcome of the EM according to their weight, which can be assigned using different methods. The prediction (or output) of each model within an EM are given weights, which means that different outputs have a different level of impact on the final outcome. For example, if an EM has two models, and Model A (with a weight of 0.5) predicted a BT coefficient of 0.5 and Model B (weight: 0.8) predicted a BT coefficient of 1.0, the final prediction of the EM would be 1.05 (= 0.5*0.5 + 1.0*0.8).
There are many different methods that can be used to derive the weights of each model. The 2nd place model uses Nelder-Mead (a gradient-free optimization method), and the 4th place model uses ridge regression (which is a regression model that results in less overfitting.

Transformers. Transformers are natural language processing (NLP) models that are used in a wide variety of tasks. In transformer models, each word in a sentence is represented using a sequence of numbers (= word embeddings). These word embeddings are not created at random – computers are assigned specific tasks. In the case of Transformer models, these tasks are 1: removing words at random and making the computer guess the word, and 2: asking the computer whether two sentences are consecutive or not. The transformer model self-corrects by tweaking the numbers until it gets the optimal outcome.
Word embeddings created by Transformers are different from word embeddings created by other models in that they contain more contextualized information by using what is called attention. Attention, simply put, is another set of vectors attributed to each word. These vectors keep track of the relationship between words by marking the most relevant word within the same sequence. The image below is an illustrated example of how irrelevant words are assigned less weight (= grayed out) in an attention vector for a particular word.

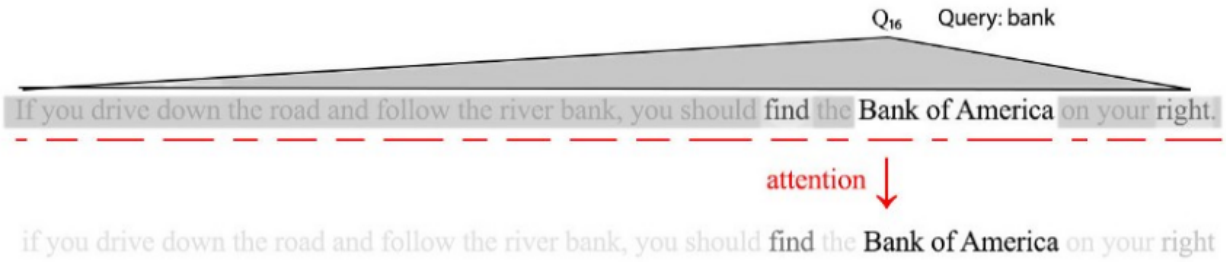

Figure 1. An illustration of how attention for the word "Bank" is represented in an attention vector by Jonathan Hui source: https://jonathan-hui.medium.com/nlp-bert-transformer-7f0ac397f524

Attention vectors are used to create the final word embedding output of Transformers, and the final word embeddings of individual texts are pooled to make a prediction of the respective BT coefficient score. There are many Transformer models that come pre-trained (= they are given a huge number of texts to perform tasks such as those mentioned above), but they need to be trained further (also called fine-tuning) to perform better on specific tasks.
The common Transformer models used in the competition were BERT (Bidirectional Encoder Representations from Transformers) and its variants such as RoBERTa, DeBERTa, DistilBERT, and XLNet. The core concept of attention and Transformers are intact in all models. The differences in the final models produced by participants come from other aspects such as the size of the corpus that is used for pre-training or their specific training methods.
Runtime: The length of time the algorithm takes to run from start to finish.
Storage space: The amount of disk space the algorithm and its corresponding models take up on a storage device (e.g., a computer, a network server for websites, etc.).

Model Overview

1st place (Mathis). A 6-fold cross validation method was used to evaluate the models in this algorithm. Additionally, this algorithm supplemented the training set with a large amount of external data, which was retrieved from a variety of sources such as: Wikipedia, HuggingFace (a large machine learning library), and various children's books text. This algorithm was unique in its large amount of external data added. The external data was added because it was relatively similar to data in the training set, and the additional external data made the training set much larger, which is a strategy that can help increase the prediction accuracy of the final model. A Roberta-base transformers model was then trained to pseudo-label the external data (i.e., use the labeled data from the original training set to process and label the rest of the external data). Then, multiple transformers models were trained and two ridge-regression ensemble models were made. The final algorithm uses and trains the following models: albert-xxlarge, deberta-large, roberta-large, electra-large, roberta-base and two ensemble models.

2nd place (Takoi). The 2nd place model was an ensemble model (EM). The individual models that constitute the EMs in the 2nd place solutions are different Transformer models. First, five separate folds were created for cross validation (which used randomization Sturge's rule to decide the number of bins in each fold). These exact same folds were also used in the 4th place solution. Second, dropout was set to 0 for the majority of models in the 2nd place solution (17 out of 19 models) and all models in the 4th place solution. The results from the two models that did use dropout was supplemented with other regression methods such as support vector regression and ridge regression.

Takoi's model includes post-processing – the output of the EM was binned and weighted using Nelder-Mead and then manually tweaked to get better results. This means that the output of the EM was multiplied by different coefficients depending on its value. For example, if the output of the EM (= pred) was 0.2 (which would make it fall under the bin of 0.3 > pred >= 0), the output was multiplied by 1.2 to get the final BT prediction of 0.24. Takoi's model also includes negative weights from some of its models, which means, as the 3rd place winner noted in their comments, that the prediction by some models were "so bad, taking a negative of its prediction improves score." As an example, the output from the Electra base model had a negative weight of -0.170, meaning it predicted the exact opposite of the individual model's prediction to a certain extent. Multiplying the prediction by a negative weight of -0.170 improved the performance of the final EM.

3rd place (hxcc). This algorithm supplemented the original training set with some pseudo-labeled external data and used a 5-fold cross validation method. An explanation of these pseudo-labeling and external data concepts can be found in the 1st place algorithm overview. After pseudo-labeling, deberta-large and roberta-large models were trained on the original and pseudo-labeled training sets, and these models were used to test the final data. This algorithm differed from others in that it trained a couple of large training models, rather than multiple smaller models. Other algorithms likely did not go this route because a powerful machine (e.g., computer, laptop, GPU) is needed to train a model of this size. The final algorithm used and trained the following models: deberta-large and roberta-large.

4th place (Datasaurus). This solution was similar to 2nd place, but with small differences as noted below. First, Datasaurus used a Python package called Textstat to derive Flesch Reading Ease and SMOG scores to include in the model. These are traditional measures of text difficulty developed more than half a century ago that use the number of polysyllabic words (for SMOG) and the average length of sentence and average number of syllables per word (for Flesch Reading Ease) to predict text difficulty. Datasaurus reports that the inclusion of these features brought "a tiny but repeatable improvement" to his model's performance. Second, each model was trained using five different seeds and each of these trained models was treated as a separate model, meaning that certain models were included multiple times in the ensemble (= the same baseline model but trained using a different seed). For example, DeBERTa Large (DeBERTA, meaning that it is a variant of the BERT model, and Large meaning that it has been pre-trained on a larger dataset compared to the base version – DeBERTa Base) was included in the EM five times. These models, although they started from the same, pre-trained baseline model, give slightly different predictions because they were trained using different seeds, and are therefore assigned different weights. Third, Datasaurus included all of his models in the EM and removed models one by one (as opposed to starting from scratch and adding models to the EM) until the model ran within the time limit. Fourth, Datasaurus used a specific attention block that uses Kullback–Leibler loss instead of mean-squared error (MSE; which Takoi used for his models) as the loss function (= the equation you use to calculate how far off the current prediction is from the target value) for certain models.

5th place (ruby). This algorithm used 5-fold cross-validation as well as multiple fine-tuning steps which re-trained previously trained models multiple times to achieve higher accuracy scores. A ridge-regression ensemble model was also developed during the training steps and incorporated into the final solution. This algorithm contains 11 notebooks/code files which are connected and require moderate re-working to reduce it to one file/script (if desired). Additionally, individual local files are not stored for each model (e.g., roberta-base). This algorithm is also the only one which will error (run out of memory) on larger datasets.

6th place (ofnt). Similar to algorithms #1 and #3, this algorithm also supplemented the training set with some external data. 5-fold cross validation was used on the training set and 9 total transformers models were fine-tuned using various strategies. The models used were: roberta-large, deberta-large, xlnet-large, longformer-large, bart-large, and electra-large. Additionally, this algorithm differed from the others in that it trained a Gaussian Process Regression (GPR) model, which is a model frequently used to increase the prediction accuracy for smaller datasets and can also provide the levels of uncertainty in its predictions. Specifically, this algorithm used the GPR model to correct overconfident predictions and improve the accuracy of their single transformer models.

Evaluation Information

Evaluation Metrics by Algorithm

	1st Place		2nd Place		3rd Place		4th Place		5th Place		6th Place	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
Test Set	**0.446**	**0.814**	**0.446**	**0.816**	**0.445**	**0.816**	**0.449**	**0.812**	**0.450**	**0.811**	**0.448**	**0.812**
Train Set	0.250	0.941	0.178	0.970	0.176	0.971	0.132	0.984	0.216	0.956	0.171	0.972
Full Set	0.342	0.890	0.313	0.908	0.313	0.908	0.301	0.915	0.330	0.898	0.313	0.908

General Specs

	1st Place	2nd Place	3rd Place	4th Place	5th Place	6th Place
Storage Space	65GB	105GB	42GB	200GB+	<1GB*	78GB
Runtime (7 samples)	15mins	20+mins	7mins	20+mins	11mins	25min
Runtime (full 5k dataset)	4hrs 15mins	4+hrs	45mins	4+hrs	6hrs	2hrs 45mins

**This algorithm downloads transformers models as it goes, rather than storing the files for each individual model, which is why it takes up less than 1GB of storage.*

Kaggle Notebook Specs

All models were run on the Kaggle Notebook. This machine is single NVIDIA Tesla P100, 2 CPU cores, 13GB RAM. The above time frames are based on using a P100 GPU (as in the Kaggle notebooks), but the algorithms would run faster if you had access to a better GPU. The runtimes are broken up into how long it took each algorithm to run 7 text samples versus the full ~5,000 sample dataset.