Estimating the CEFR level of a text with Machine Learning

Pablo Lizcano Sánchez-Cruzado

Motivation

Lingua Ignis's mission is to create a platform where students can learn a language with the materials they choose: books, films, blogs, etc. Since using materials which are too difficult is discouraging, it is necessary to help students select those materials fit for their level. For simplicity purposes, I have started with the creation of an algorithm which estimates the CEFR level of a text. This algorithm carry out the following tasks:

1) Feature extraction. The text is processed to extract features related to the complexity of text.
2) Level prediction with a ML model. A ML model trained with labelled texts will use those features to estimate the level.

ML algorithm selection

What a learner can do with the language (e.g. introducing themselves, answering questions during a job interview, understanding texts of a certain complexity) for information determines their CEFR level. A text will be easy to follow for beginners if it is composed of frequent words and simple sentences.

As far as I know (or I remember), recurrent neural networks use vector representations related to the meaning of the words. Two words with similar meanings would have similar vectors, even if their frequencies are very different. Since the most informative characteristics to determine the difficulty of a text are the complexity of the sentences and the rarity of the words, I have opted to build features which summarise these characteristics.

One of the most effective ML algorithms for tabular data is XGBoost. There are six CEFR levels, from A1 to C2, so this is, in principle, a classification problem, which requires an **XGBoost classifier**. However, there is a relation of order among these classes, which the XGBoost classifier could ignore. An **XGBoost regressor** would have the capacity to understand this order, but it would have to assume that the distance between A1 and A2 is the same as the distance between A2 and B1. The third option is to train **five XGBoost binary classifiers**, each of them trained to detect the border between two levels. We could get contradictory results, like having a model which estimates that the level of a text is lower than B1, and another one which says that the level of that text is higher than C1. I have explored these three options.

Datasets

The dataset used to train and test the models can be found in the following link: https://www.kaggle.com/datasets/amontgomerie/cefr-levelled-english-texts . It contains 1500 texts labelled with their CEFR level.

The training dataset is composed of 80 % of texts contained in the original dataset. The rest will be used to test the models.

Feature engineering and labelling

To fully understand the code, some knowledge domain is required. In this section, the steps to process the text will be explained with detail.

Feature engineering

The text is processed by a spacy model which allows obtaining the part-of-speech of each word and the limits between sentences. Here you have some links to understand the part-of-speech tagging Spacy carry on:

- https://spacy.io/usage/linguistic-features#pos-tagging
- https://universaldependencies.org/u/pos/

Features related to the rarity of the words
Features: VERB_0, VERB_1, ADJ_0, ADJ_1, ADV_0, ADV_1…

In every text, regardless of their difficulty, one can find frequent common words. Complex texts will also include rare words. Therefore, the difficulty of a text will be determined by the rarity (or frequency) of the rarest words which appear in it, and not by the rarity of all the words which appear. The rarest word of the text cannot determine the difficulty of the whole text on its own, because the rest of the vocabulary may be easy and the reader can make sense. A text will be difficult if it contains a certain number (or proportion) of rare words.

A difficult text will contain rarer prepositions, rarer nouns,... but since the total number of prepositions is very reduced compared to the number of nouns in any language, they will be considered rare at different frequencies.

The rarity has to be a number which decreases when frequency increases. It doesn't matter how, since XGBoost is based on decision trees. I have computed the rarity with the following formula:

rarity(word) = - ln(word_frequency(word))

where ln is the natural logarithm and word_frequency is a function of the Python library wordfreq (https://pypi.org/project/wordfreq/), which computes the frequency of a word in a corpus. Sadly, it doesn't differentiate two homograph words (words written the same, but with different classes or meanings). For example, the word "duck" may be more frequently used as a noun than as a verb. It has been assumed that words which are not part of that corpus, which have frequency zero and infinite rarity, are typos of frequent words. They have been assigned a rarity equal to zero.

The difficulty of the vocabulary in the text is represented by the **percentiles 60** () and **85 rarity** of the **words tagged** by the spacy model **with the same part-of-speech**. The features ending with "_0" correspond to the percentile 60, and the features ending with "_1" (VERB_1, NOUN_1, ADJ_1). Initially, I wanted to compute only one percentile, but I didn't know beforehand which would have been the most useful. Let's wait to analyse the results.

<u>Features related to the complexity of the sentences</u>
Feature list: LEN, VARIETY, VERB, CCONJ, SCONJ, AUX

In the same fashion even in a difficult text we can find simple sentences. The complexity of the text will be determined by the most complex sentences which appear on them. After leaving out the sentences shorter than 5 words, the 20 %-top long sentences have been selected. These six features have been computed for each of those sentences: **number of words** (LEN), number of **words tagged as CCONJ**, **SCONJ**, **AUX**, **VERB** and the **number of different tags** (VARIETY) which appear in the sentence.

<u>Features related to the length of the text</u>
Features: NR_SENT, NR_WORDS

The **total number of sentences** and the **total number of words** have been computed, because there seemed to be a correlation between the length of the text and its CEFR level in the dataset. Taking into account the use case, **the model should be blind to these two features**. However, **models which also use these features will be trained**, in order to measure the impact of ignoring them.

Labelling

The texts were originally labelled with a CEFR level. In order to train the **XGBoost regressor** and the single **XGBoost classifier**, these CEFR levels are mapped to an **integer number**: A1 -> 0, A2 -> 1, B1->2…

The **multiple XGBoost binary-classification model** requires **binary vectors** with five elements: A1 becomes [0,0,0,0,0], A2 becomes [1, 0, 0, 0, 0], B1 becomes [1, 1, 0, 0, 0]...

Hyperparameter selection and training

The function train_xgb in the code is clear and easy to understand for any ML practitioner, so only some key points will be highlighted:

- The BayesSearchCV function from the Python library skopt has been used to find the hyperparameters which optimise the scoring function "accuracy" for XGBoost classificators (the single XGBoost classifier and the multiple XGBoost binary-classification model), and the scoring function "neg_mean_squared_error" for the XGBoost regressor.
- **Only the training dataset has been** used during the hyperparameter search.

- Once the optimised hyperparameters have been determined, the model has been trained on the whole training dataset.

The model which can be put on production cannot use the length of the text as a variable. These three models are trained without the last two features defined (number of words and number of sentences):
- Model UC1: XGBoost regressor
- Model UC2: XGBoost classifier
- Model UC3: multiple XGBoost binary-classification model

Since The effect of including the dimensions of the text is being analysed, there are three models trained on all the features defined before.
- Model D1: XGBoost regressor
- Model D2: XGBoost classifier
- Model D3: multiple XGBoost binary-classification model

The feature importance for each model can be found in the notebook text_level_prediction_v1.pynb

Results

The levels of the texts in the testing dataset have been predicted with the models which have just been trained. The raw output of the XGBoost regressor and the multiple XGBoost binary-classification model must be corrected and driven to one of the six existing classes:
- The output of the XGBoost regressor has been approximated to the closest integer number. If it was lower than zero, it has been set to zero (A1), and if it was higher than five, it has been set to five (C2).
- The output of the XGBoost classifiers which comprise the multiple XGBoost binary-classification models have been summed. The result is a number between zero (A1) and 5 (C2). If the single models give incompatible results with a “one” after a “zero” (for example: [0, 1, 0, 0, 0], or [1, 0, 1, 0, 0]) are labelled as “INCOHERENT”.

The accuracy is the proportion of texts in the testing dataset correctly labelled by the model. The “relaxed accuracy” is the proportion of texts which have been correctly labelled or with a light error, i.e. labelled with a level immediately lower or higher than the real one.

	Accuracy	Relaxed accuracy	Incoherence
Model UC1	61.9 %	96.3 %	
Model UC2	64.5 %	95 %	
Model UC3	63.2 %	94.3 %	0 %
Model D1	58.5 %	96 %	
Model D2	63.9 %	96 %	
Model D3	66.9 %	94.6 %	0.3 %

The confusion matrix for each model can be found in the notebook text_level_prediction_v1.pynb

Conclusions

Leaving variables related to the length of the text (NR_WORDS and NR_SENT) out doesn't have an important impact on the performance of the model.

Classifiers seem to have a better performance in terms of accuracy. This can be due to the fact that the hyperparameters search of classifiers was aimed to optimise the accuracy, whereas the hyperparameters search of the regressor was aimed to optimise the mean square error. The multiple XGBoost binary-classification model doesn't seem to present significant advantages over the single XGBoost multiclass model, and the computation cost of optimising their hyperparameters is too high.

Regressors have a better performance in terms of "relaxed accuracy", thanks to the computation of the average level in each region the XGBoost regressor has defined, instead of the "voting system" of XGBoost classifiers.

For my application, **the performance of our models can be enough**. The user can be told when the level of the text they are reading is too high for them (2 levels above), according to the XGBoost regressor model.

These models could also be useful for publishing houses. The models could classify texts, and later an expert in that CEFR level could verify if the level is OK or reassess it.

Next steps

Error analysis should be performed to understand if it is possible to identify the cases where the model doubts between two levels. We may find that the model makes mistakes generally when it assigns the similar high probabilities to two consecutive levels.

Instead of using the classical accuracy as the scoring function for the hyperparameters search of the classifiers, it would be sensible to create a **scoring system** which takes into consideration the **distance** between the level predicted and the real level.