Project Report
Automated Essay Scoring

PSL UNIVERSITÉ PARIS

IASD Master

Pierre-François Massiani
Guillaume Le Moing

work completed during the
Natural Language Processing class

April 25th , 2020

Contents

1 Introduction

This is the final project for the « Natural Language Processing » class of Master IASD (Artificial Intelligence, Systems, Data), a joint PSL University Master program hosted by Paris-Dauphine, École normale supérieure, and MINES ParisTech.

In this project we propose and compare different neural network architectures as well as learning strategies for the automated essay scoring task. We test our algorithms using the data from the ASAP competition on Kaggle [1] sponsored by The Hewlett Foundation.

Our project was implemented in Python. The code is available on a Github repository at this address:

`https://github.com/16lemoing/automated-essay-scoring/`

Instructions are included in the repository to run the code on your machine.

1.1 The automated essay scoring task

The task we are trying to solve here is called *automated essay scoring* (AES). It consists in automatically assigning a grade to an essay written on a given topic. This is of particular interest in an educational setting, where incentives for developing unified and objective grading methods can be easily understood. Such a method could be used for instance to grade large scale exams where teachers need to evaluate hundreds of essays in a short amount of time, leading to exhaustion, unwanted bias introduced by how focused the teacher is when reading, and discrepancies in the grades caused by the teachers' own evaluation criteria. Using an automated grading system can then help to standardize this process. Other factors also account for the growing interest in this task, such as cost reduction.

In spite of these incentives pushing for the development of automatic essay scoring algorithms, the attempts at using them received quite a backlash. The arguments against these algorithms were that they did not understand the meaning of the essays they were grading and were relying only on "surface features of responses". To prove their point, protestors created essays exploiting the biases discovered in the algorithm to create nonsensical high-graded essays. According to MIT Director of Writing Les Perelman, "the substance of an argument doesn't matter [for such algorithms], as long as it looks to the computer as if it's nicely argued" [2]. Indeed, high grades were given to essays containing assertions that were simply not true, such as stating that the War of 1812 started in 1945.

The Hewlett Foundation challenge In 2012, *The Hewlett Foundation*[1] sponsored a competition on Kaggle [1] intended at demonstrating how AES algorithms - and more specifically neural networks - could be as reliable as humans in rating thousands of essays. Although this competition was very successful [3–6], it is still controversial whether the initial claim is backed up by the competition's results. In this project, we propose an algorithm for the Hewlett Foundation's challenge.

1.2 Related work

The interest for the AES task sparks around 1970 with the works of Ellis Batten Page [7], but it quickly faces the limitations imposed by the computational power available at the time. This practical limitation is lifted in the 1990s, and 1999 sees the first commercial automatic essay grader [8]. After that, the field grows rapidly and different approaches are tested with the state-of-the-art statistical inference techniques known at the time [9].

Today, the AES task is typically tackled using modern and state-of-the-art natural language processing (NLP) tools. Indeed, AES is a subtask of the more general text classification problem, which is a vibrant field in the NLP community. As such, classical NLP techniques can be used to achieve great results such as dense neural networks [10], convolutional networks [11], or recurrent networks [12].

2 A Neural Network based approach

We choose to rely on neural networks for this task. We describe in this section the whole pipeline from raw essays to the prediction of the scores. For each part, we present different options that are included in this project.

2.1 Words embedding

Essays have to be turned into vectors of numbers before being fed to a neural network. One popular strategy is to assign a vector to every single word in the text. The required dimension for the vectors depends on the richness of the vocabulary (both in quantity and lexical diversity).

[1]The Hewlett Foundation is a private foundation that grants awards in a variety of liberal and progressive causes such as education, climate, health, journalism...

2.1.1 Random

This is the most basic embedding. We assign to each word of the vocabulary a vector of random samples from a normal (Gaussian) distribution.

2.1.2 Word2Vec

We propose to train a Word2Vec model [13] on the sentences extracted from the training essays. This model learns meaningful embeddings by trying to guess a word from its context (a few words before and after in the sentence). To do this each word of the context is turned into a vector from which the prediction is made. Those vectors are what we use as embeddings once the model is trained. Each set of essays deals with a different topic. We hope to capture topic-related knowledge from the corpora by learning the embedding directly from the set of training essays.

2.1.3 GloVe

We also propose a different strategy for getting word embeddings using GloVe [14]. The particularity of GloVe embedding is that there are linear substructures between words sharing semantic aspects. For this embedding, we decide not to train the GloVe model from scratch but use a pre-trained model on large-scale corpora such as Wikipedia instead.

2.2 Essay preprocessing

We present here a few preprocessing methods that can help the learning process.

2.2.1 Spelling errors correction

Reading a few essays, we realised that there were lots of misspelled words in the essays. This can impair the prediction performance because we cannot provide a meaningful embedding to misspelled words and we have a much larger vocabulary. It would have come in handy to have a python wrapper for a correction tool such as LanguageTool which handles both syntaxic and grammatical errors. Instead we used pyspellchecker package which gives a list of the most plausible correction candidates for each of the misspelled words (but do not consider the sentence as a whole).

2.2.2 Stopwords removal

There are some very common words, called stopwords that usually do not add much meaning to the sentence. A common preprocessing step in natural language processing consists in removing these words that can pollute the learning process. However, in our case it is unclear whether we should remove these words or not. For example we need to take them into account when evaluating the gramatical correctness of a sentence. That being said, it would be surprising that our model learns what is a grammaticaly correct sentence from such a small corpus.

2.3 Essay processing

We describe here how the essays are transformed so that they can be understood by a neural network.

2.3.1 Word tokenization

The first step is to transform essays into tokens (isolated words). To do that we transform every special character into a space symbol and then split the essay at every occurence of the space symbol.

2.3.2 Essay padded encoding

Then we assign to each word its index in the vocabulary. This index will be used as a key when retrieving the word embedding. To enable batch-learning we pad every essay so that it matches the length of the longest essay.

2.4 Multitask model

The dataset contains multiple essay tasks which are scored on different scales. This makes it difficult to learn all tasks jointly. However this can be addressed by normalizing the scores.

2.4.1 Score normalization

We rescale all the scores so that they fall into $[0, 1]$, by applying this simple linear transformation:

$$s_{norm} = (s - s_{min})/(s_{max} - s_{min}) \tag{1}$$

2.4.2 Score recovering

For the evaluation we need to recover the score in its original scale. To do this we apply the inverse transformation and then round the obtained value to the nearest score value in the corresponding set.

2.5 Extra-features computation

When predicting a score for an essay it is possible to include higher-level features to give some insights about characteristics that are hard to grasp for the neural network. First, during the spelling correction, we can compute the number of misspelled words for each essay. Doing this, the model can be fed the corrected essays and have a better understanding of the meaning while still being able to judge on this aspect. We also extract part of speech indicators from the essays (number of nouns, adjectives, verbs...), usage of ponctuation (number of question marks, exclamation marks, commas...), semantic diversity (number of semantic roots that were used to build the words in the essay), quotations (counting quotation marks, references to organizations, locations, people...).

2.6 Models architecture

We propose highly customizable neural networks to optimize there architecture based on the validation results.

2.6.1 Dense

First, we propose a four-layer dense neural network. The first layer is the embedding layer. We then take the mean of all the embedding vectors and feed it to a series of dense layers. We use ReLu activation functions except for the output layer which has a Sigmoid activation when the scores are rescaled to $[0, 1]$. There is the possibility to add some dropouts between layers to prevent overfitting. The dense model can be fed encoded essays, encoded essays + extra-features, extra-features alone. This way we can evaluate the predictive power of each individual part.

2.6.2 LSTM

The dense neural network merges all word embeddings into a single vector. It does not take into account the order in which the words appear in the essay. We introduce an LSTM model so that we can work directly with sequenced data. Our LSTM model is made of a custom number of layers and can include dropout. We also add a few fully connected layers so that we can make use of the extra-features if they are provided.

2.6.3 CNN

TODO description du CNN

3 Experiments

We conduct a thorough analysis to compare and discuss all the proposed learning strategies.

3.1 Dataset

TODO description détaillée du dataset

3.2 Validation metrics

TODO présentation de la loss MSE TODO présentation du quadratic weighted kappa

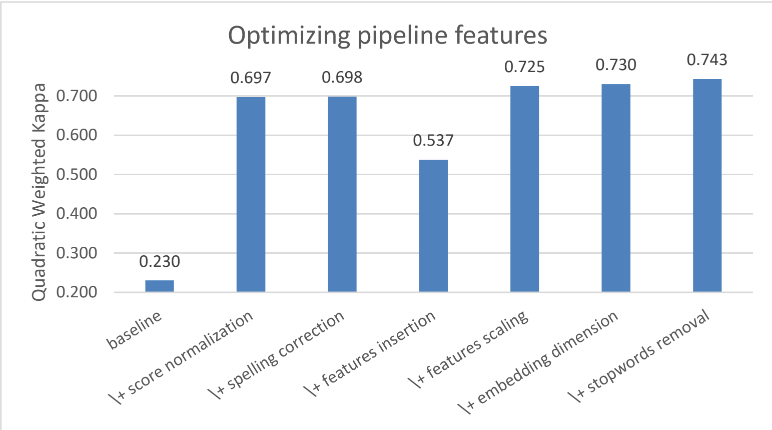
3.3 Experimental protocol

To compare all the configurations we use 5-fold cross-validation on the training data (for each fold the training data is split into subtraining and subvalidation sets) The best epoch is found by looking at the lowest loss value on the subvalidation set. We save the average validation metrics across all folds corresponding to the best epoch for each fold. When all configurations have been cross-validated we test the best configuration on the test data (which remained unseen up to this point).

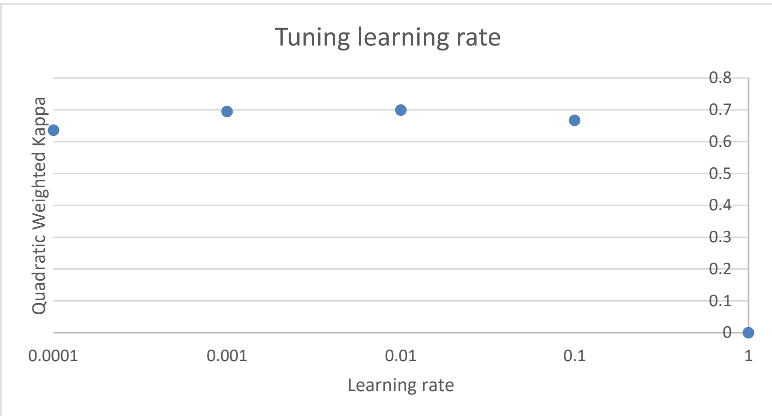
3.4 Results

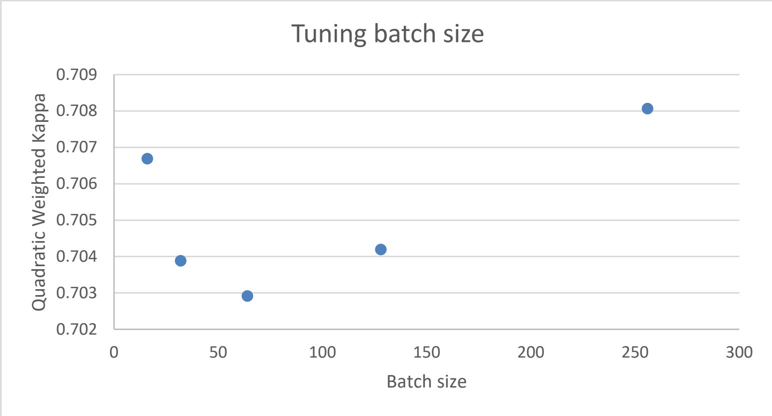
All the results for the experiments are saved as things progress in an excel spreadsheet. The raw file containing all these results can be found in the Github repository alongside this report.

Pipeline optimization First we show incremental results of pipeline features optimization compared to a baseline method. The baseline method corresponds to the set of default arguments (Word2Vec embedding of dim 50, shallow fully-connected model without dropout, no extra features, no special data preprocessing, learning from all essay sets at once). We found that the baseline method was rather unstable (sometimes giving decent results, sometimes not converging at all). Normalizing score so that they fit in $[0, 1]$ enabled a great improvement of the quadratic weighted kappa metric compared to the baseline method. Spelling correction led to maginally better scores. Extra-features insertion led to more instabilities but adding this together with feature scaling solved this issue and further improved the results. Changing the embedding dimension from 50 to 300 and removing stopwords helped improving the validation metrics even more.

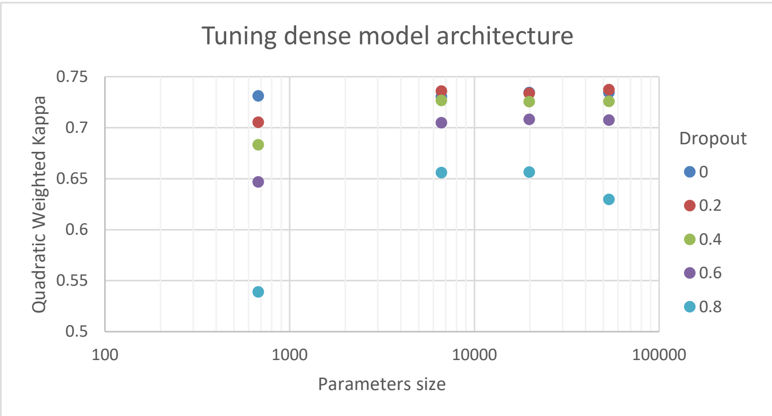


Hyperparameters tuning Then we focused on tuning the key learning hyperparameters. For the choice of the learning rate, two values (0.001 and 0.01) led to similar results. We selected 0.01 because it was slightly better. For the batch size, results indicated that we should select either a small batch size or a very large one. As it speeds up the learning process we decided to go with a rather large batch size (256).

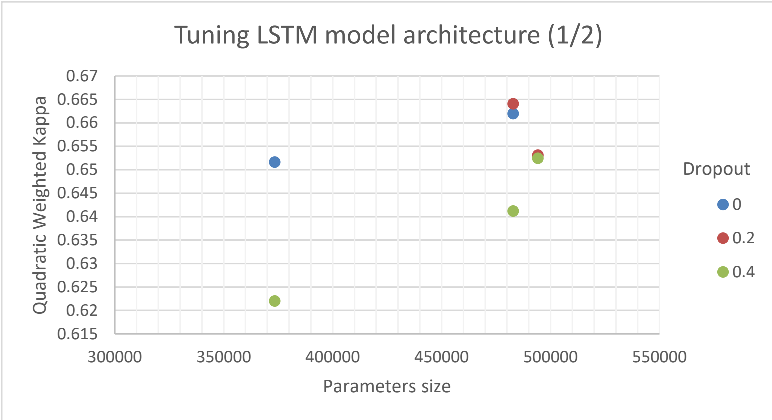


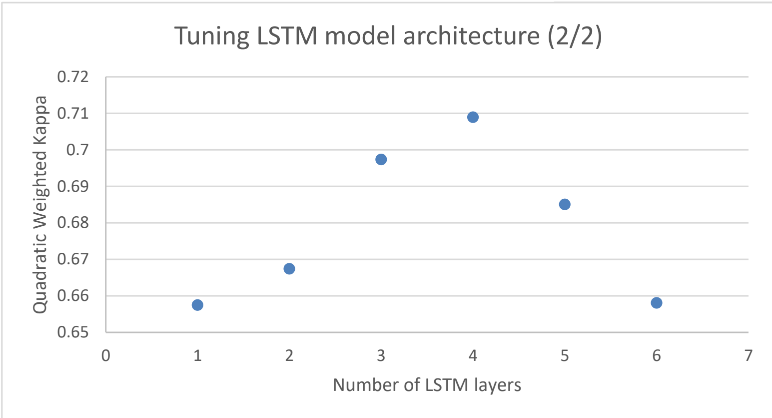


Dense model optimization We tried various architecture for the fully connected model. We show the results for different dropout values and number of parameters. The results shown correspond to learning from embedded essays as well as extra-features. We also tried learning solely from extra-features a reached a quadratic weighted kappa value of 0.678 which is far from the best results we can get when combining embedded essays and extra-features. The best configuration for the dense model is obtained with hidden layers of size 300 and 128 and dropout of 0.2.

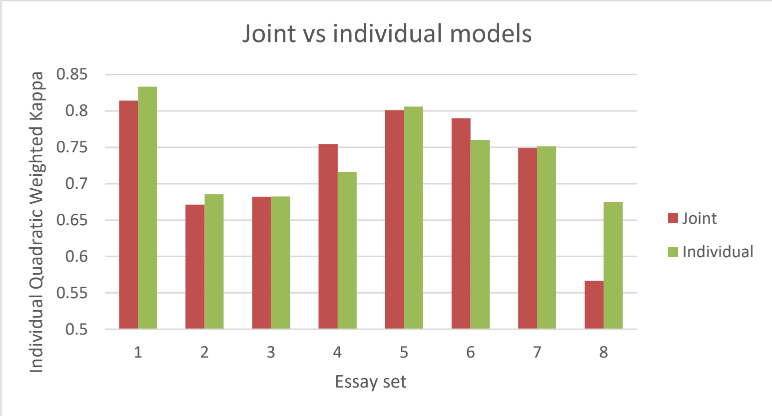


LSTM model optimization We conducted similar experiments for our LSTM model. The best LSTM model was obtained with hidden layer size of 100 for the recurrent units and 16 for the fully connected part and droupout value of 0.2. We then tried deeper model architectures by stacking recurrent units on top of each other. Results show that a depth of 4 is optimal in our case.

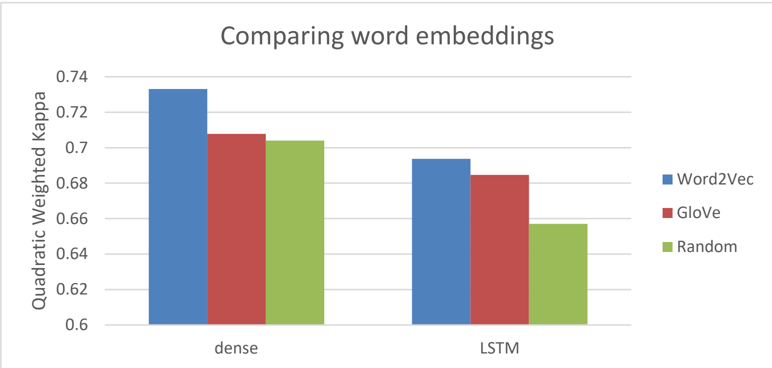




Joint vs individual models Multitask models are increasingly popular because achieving a good performance level for one task can help tackling other tasks. However, when we compare the results for our joint model compared to the results of individual models we achieve similar performance. Indeed, the quadratic weighted kappa for the joint model is 0.741 and 0.744 when compiling results together for individual models. The small difference is due to the joint model not being very good on the 8th essay set due to size imbalance between sets. Nevertheless, we prefer to keep the joint model as it is much faster to train and less likely to overfit.



Word embeddings comparison Finally, we compare all the proposed word embedding strategies. It turns out that the most successful word embedding strategy is the Word2Vec embedding trained on the essay sentences. We think this is due to specific words and contexts being good indicators for the value of an essay. This strategy leverages the whole corpus thus having task specific knowledge for each of the essay tasks.



TODO courbes et commentaires pour CNN

4 Conclusion

TODO

References

[1] "The hewlett foundation: Automated essay scoring." `https://www.kaggle.com/c/asap-aes/`. Accessed on April 30th.

[2] M. Winerip, "Facing a robo-grader? just keep obfuscating mellifluously." `https://www.nytimes.com/2012/04/23/education/robo-readers-used-to-grade-test-essays.html`, The New York Times, 2012. Last visit on May 2nd, 2020.

[3] H. Nguyen and L. Dery, "Neural networks for automated essay grading," 2018.

[4] S. Song and J. Zhao, "Automated essay scoring using machine learning," *Stanford University*, 2013.

[5] J. Nicolich, "Automatic student essay assessment." `https://github.com/Turanga1/Automated-Essay-Scoring/blob/master/Automatic%20Student%20Essay%20Assessment.pdf`. Last visited on May 2nd, 2020.

[6] A. A. Manvi Mahana, Mishel Johns, "Automatic essay grading using machine learning." `https://github.com/m-chanakya/AutoEssayGrading/blob/master/papers/paper1.pdf`. Last visited on May 2nd, 2020.

[7] E. B. Page, "The imminence of... grading essays by computer," *The Phi Delta Kappan*, vol. 47, no. 5, pp. 238–243, 1966.

[8] Y. Attali and J. Burstein, "Automated essay scoring with e-rater® v. 2.0," *ETS Research Report Series*, vol. 2004, no. 2, pp. i–21, 2004.

[9] L. M. Rudner and T. Liang, "Automated essay scoring using bayes' theorem," *The Journal of Technology, Learning and Assessment*, vol. 1, no. 2, 2002.

[10] K. W. Murray and N. Orii, "Automatic essay scoring," *IEICE Transactions on Information and Systems*, 2012.

[11] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[12] K. Taghipour and H. T. Ng, "A neural approach to automated essay scoring," in *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 1882–1891, 2016.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[14] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1532–1543, Association for Computational Linguistics, Oct. 2014.