Project Report

Automated Essay Scoring

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. For example, writing tasks were suppressed from French PACES[1] competitive exam this year because teachers will not have enough time to grade them because of the coronavirus outbreak [2]. An automated grading system could help to solve such problems. 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

[1] "Première année commune des études de santé", the first year of health studies in France.

substance of an argument doesn't matter [for such algorithms], as long as it looks to the computer as if it's nicely argued" [3]. 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*[2] 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 [4–7], 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 [8], 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 [9]. After that, the field grows rapidly and different approaches are tested with the state-of-the-art statistical inference techniques known at the time [10].

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 [11], convolutional networks [12], or recurrent networks [13].

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

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

the text. The required dimension for the vectors depends on the richness of the vocabulary (both in quantity and lexical diversity).

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 [14] 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 [15]. 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

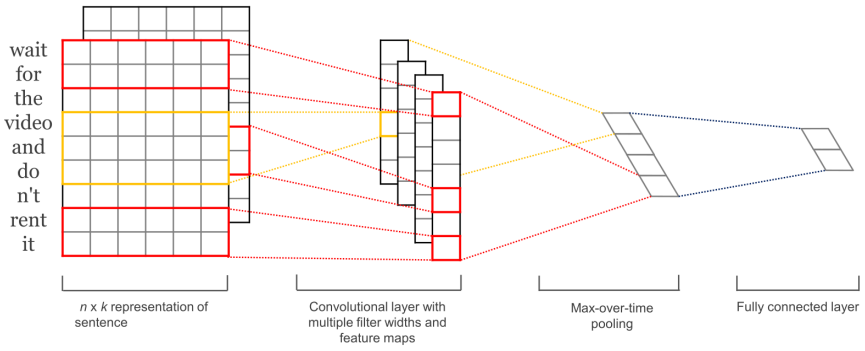


Figure 1: The architecture of the convolutional network [12]

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

Pursuing on our considerations on the strucutre of the text, we also propose a convolutional model. Convolutional architectures are very popular in image analysis tasks, because they can detect local patterns in the input data, and can adapt to different input sizes. Our architecture is inspired from the famous work of Kim et al. [12]. The embeddings are fed into three parallel 2D-convolution layers with kernels of different shapes. Our implementation chose kernels of size 3,4 and 5 along the time axis, and covering the whole embedding (see Figure 1). After that, we apply max pooling along the time axis for each channel, and feed the resulting nodes into a fully connected layer. This architecture has the advantage of being able to adapt to any essay length (thanks to the pooling), and should be able to detect syntactical richness thanks to the convolutional layers that shed light onto the local structure of the input.

3 Experiments

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

3.1 Dataset

The dataset we work on can be downloaded on the competition's website [1], along with explanations on the essays and the (human) grading methods.

Essay set	Type	Set size	Average length	Range 1	Range 2
1	Argumentative	1785	350	1-6	2-12
2	Argumentative	1800	350	1-6	1-4
3	Source based	1720	150	0-3	-
4	Source based	1772	150	0-3	-
5	Source based	1805	150	0-4	-
6	Source based	1800	150	0-4	-
7	Argumentative	1730	250	0-15	0-30
8	Argumentative	918	650	0-30	0-60

Table 1: Some statistics about the essay sets. A source essay is provided for all "Source-based" essay. The scoring methods are quite complex and are described more thoroughly in Section 3.2. The scoring ranges provided are used for detailed scores or global ones.

Essays description There are eight essay sets, each generated from a single prompt: each set was created by collecting essays from students who worked on a particular topic. Hence, topics are *coherent* among one set, and so is grading. The instructions across sets vary a lot : some essays are argumentative and should defend a point about something (e.g., "the effects on computers on people"), and others consist in text commentary. More details are given on each set in Table 1. All essays were written by Grade 7 to Grade 10 students, and hand graded and double-scored. What's more, essays are graded on different criteria and, depending on the set, a detailed score is given as well as how to compute the global score from the detailed one.

Dataset The dataset comes with the following fields:

- `essay_id` An essay identifier, unique for each essay
- `essay_set` 1-8, an identifier for each set of essays
- `essay` The text of the essay, encoded in ASCII
- Fields describing the detailed and global scores:
  - `rater1_domain1`, `rater2_domain1`, `rater3_domain1` Domain 1 score for each rater
  - `domain1_score` Resolved Domain 1 score
  - `rater1_domain2`, `rater2_domain2` Domain 2 score for each rater [3]

[3]Only for Set 2.

- `domain2_score` Resolved Domain 2[4]
- `rater1_trait1_score` - `rater3_trait6_score` Trait scores[5]

The scoring fields are missing in the validation and test sets. However, we absolutely need them in order to first train, and then evaluate our models. Hence, we only work on the competition's training set and we divide it into our own training, validation and test sets. What's more, we do not follow the formatting of the output as it is described on the competition's website, since it is not required for this project.

A note on anonymization The essays may mention names, dates, locations, organizations, etc. Such mentions were replaced before the publication of the dataset with easily identifiable keywords to anonymize the essay: we will keep that in mind when designing our algorithm.

Additional features In the raw dataset, the only features we can use for prediction are the set number and the raw ASCII text. As described in Section 2.5, we start by enriching the dataset with extra features we compute from the text itself. The enriched dataset is generally denoted with the suffix `_x` in our files, and is the one we use for prediction.

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).

[4]Only for Set 2.
[5]Only for Sets 7 and 8.

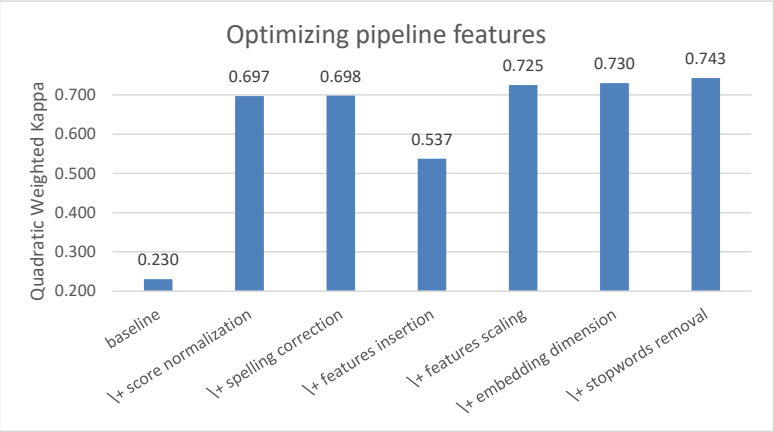
3.4 Results

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 at `data/raw_results.xlsx` in the Github repository coming alongside with this report.

3.4.1 Model selection

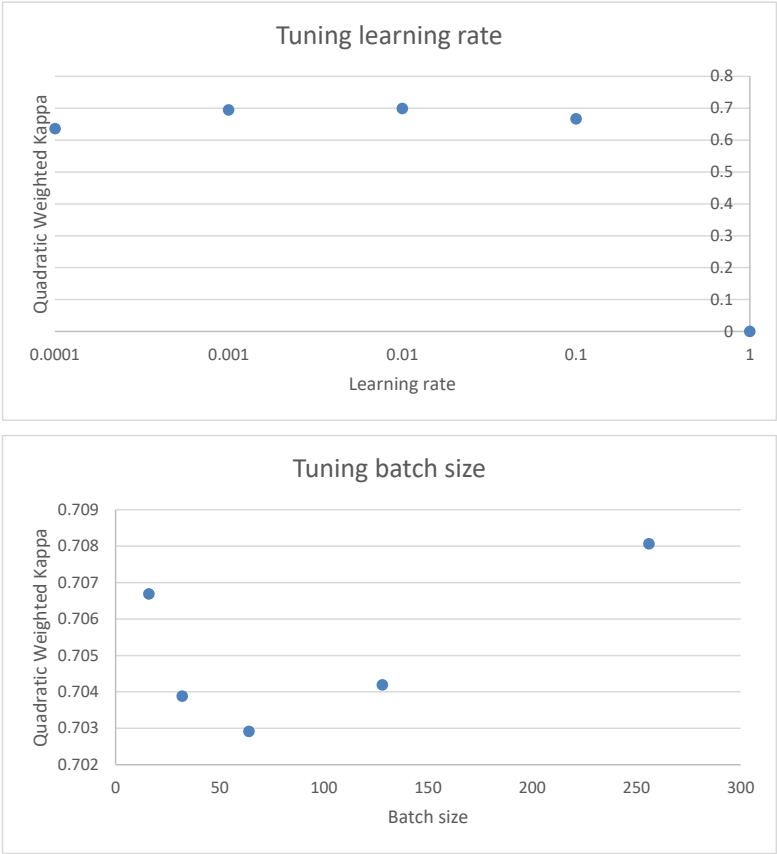
In this first part, we describe the process we followed to find our best-performing model.

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.

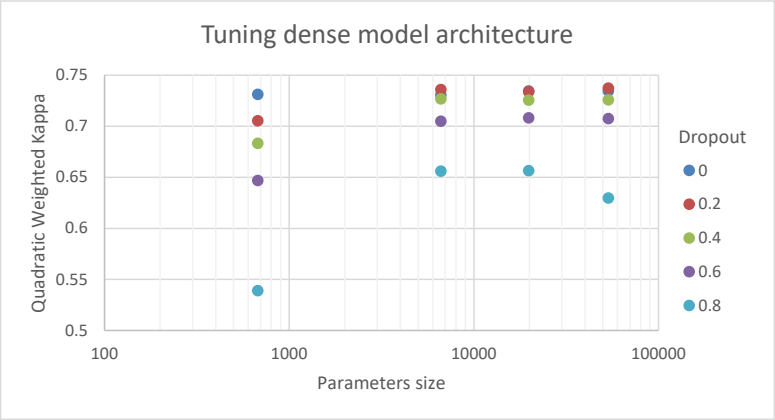


Hyperparameters tuning Then we focused on tuning the key learning hyperparameters. For the choice of the learning rate, two values (0.001 and 0.01) led to similar results. We selected 0.01 because it was slightly better. For the batch size, results indicated that we should select either a small batch

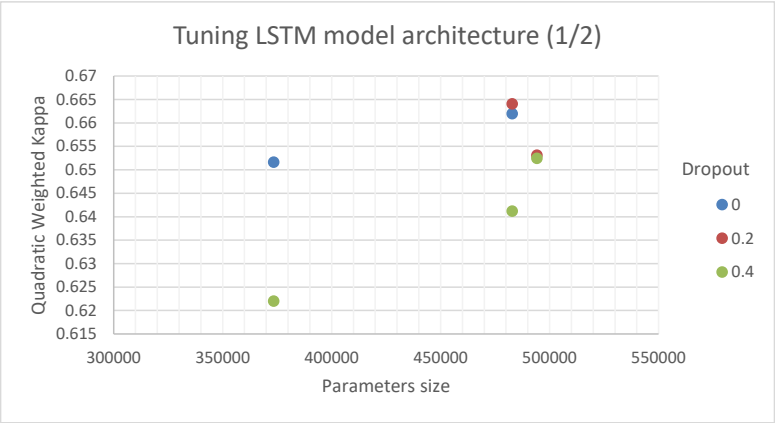
size or a very large one. As it speeds up the learning process we decided to go with a rather large batch size (256).

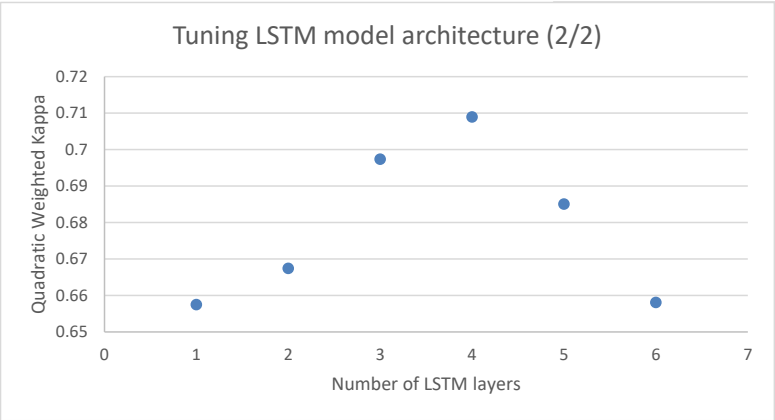


Dense model optimization We tried various architecture for the fully connected model. We show the results for different dropout values and number of parameters. The results shown correspond to learning from embedded essays as well as extra-features. We also tried learning solely from extra-features a 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.

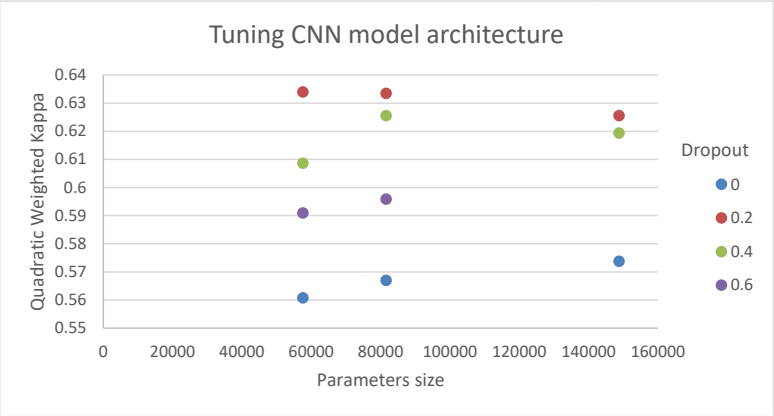


LSTM model optimization We conducted similar experiments for our LSTM model. The best LSTM model was obtained with hidden layer size of 100 for the recurrent units and 16 for the fully connected part and droupout value of 0.2. We then tried deeper model architectures by stacking recurrent units on top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

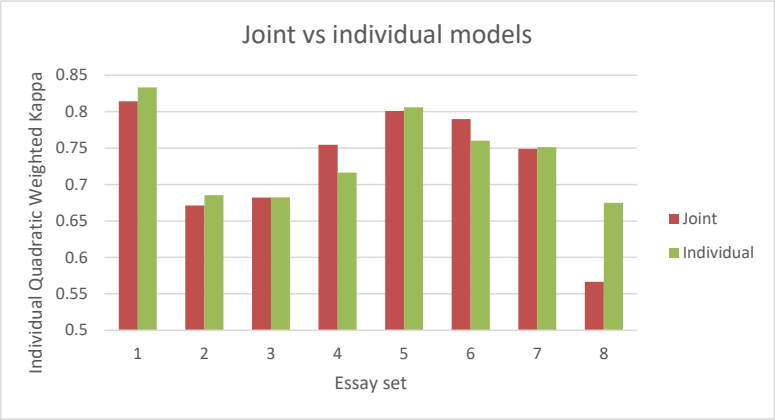




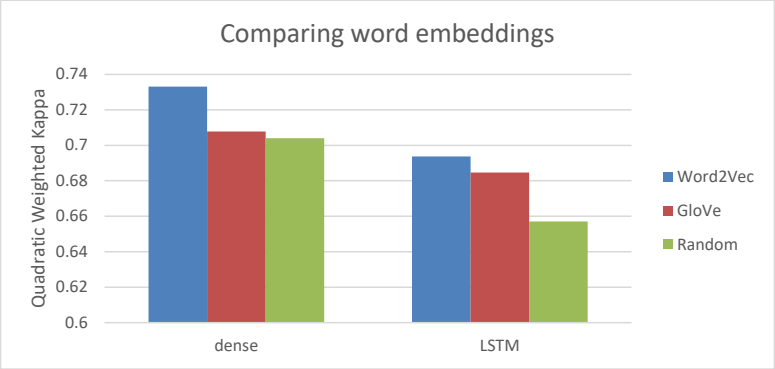
CNN model optimization We tested only one architecture for the CNN and did not try vary the weights or the number of layers. The main parameter we tuned was dropout. Indeed, a 0 dropout seemed to result in quick overfitting, and increasing dropout helped reduce it. The optimal value for dropout for our model is 0.2, but even with this, this model is the least performing: on the validation set, the weighted kappa tops at 0.62, whereas dense models can achieve 0.74. More particularly, the CNN is outperformed in all the essay sets by other architectures. The way we interpret this result is that local structure is not predominant in the grading of the essays, and a larger essay-wise view is very beneficial.



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 8^{th} 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 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.

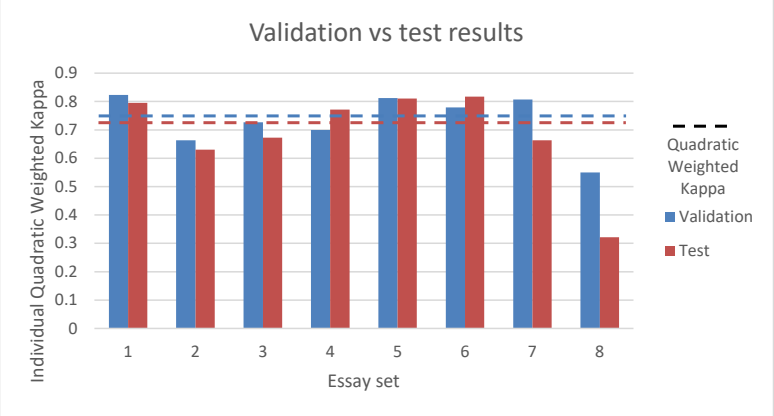


3.4.2 Best model performance analysis

The previous section shows that our best performing model has the dense architecture, a word2vec embedding and some dropout. Now, we evaluate the performance of this model on the test data - which has not been used until this point -, compare it with the competition's results, and provide a brief heuristical analysis about the keywords detected by our model.

Test results We show the performance of the best model on the test set compared to the performance on the validation set. The individual quadratic

weighted kappa are equivalent for all sets except the last one. The reason for this is that the score range is the largest for the 8$^{\text{th}}$ set and the kappa metric is expecting identical scores to count true positives. So even if the scores are close the kappa can be low. Moreover the 8$^{\text{th}}$ set contains less training samples so it is harder to generalize to unseen essays for this set. All in all, the overall performance is quite satisfying and comparable to other works dedicated to this Kaggle competition.



4 Conclusion

TODO

References

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

[2] L. Iribarnegaray, "« c'est le concours de notre vie, et ce n'est plus du tout égalitaire » : le défi des candidats aux études de santé," *Le Monde*, 2020.

[3] 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.

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

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

[6] 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.

[7] 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.

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

[9] 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.

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

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

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

[13] 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.

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

[15] 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.