The development of an artificial intelligence classifier to automate assessment in large class settings: Preliminary results

Prof. Euan Lindsay, Aalborg University

Euan Lindsay is Professor of PBL and Digitalisation in Engineering Education at Aalborg University. His focus is the use of technology to flexibly support providing authentic learning experiences for student engineers. He is best known for his work as Foundation Professor of Engineering at Charles Sturt University.

Mohammad Naser Sabet Jahromi, Visual Analysis of People Laboratory (VAP), Aalborg University

Mohammad Sabet earned his Ph.D. in Signal Processing in 2017. That same year, he joined the Visual Analysis and Perception (VAP) Laboratory at the Media Technology Section of Aalborg University as a Postdoctoral Fellow. In 2020, he transitioned to the role of Imaging Scientist at the Research and Technology Department of the Demant Group, located in Copenhagen, Denmark. Subsequently, in summer 2022, he returned to Aalborg University to serve as an Assistant Professor at the VAP Laboratory within the Media Technology Section. His current research interests encompass a range of topics, including Machine Learning, Deep Learning, Natural Language Processing (NLP), Explainable Artificial Intelligence (XAI), Medical Image Processing, and Computer Vision.

The development of an artificial intelligence classifier to automate assessment in large class settings: preliminary results

Abstract

This evidence based practice paper presents preliminary results in using an artificial intelligence classifier to mark student assignments in a large class setting. The assessment task consists of an approximately 2000 word reflective essay that is produced under examination conditions and submitted electronically. The marking is a simple pass/fail determination, and no explicit feedback beyond the pass/fail grade is provided to the students. Each year around 1500 students complete this assignment, which places a significant and time-constrained marking load upon the teaching faculty.

This paper presents a Natural Language Process (NLP) framework/tool for developing a machine learning based binary classifier for automated assessment of these assignments. The classifier allocates each assignment a score representing the probability that the assignment would receive a passing grade from a human marker. The effectiveness and performance of the classifier is measured by investigating the accuracy of those predictions.

Several iterations and statistical analyses were carried out to determine operational thresholds that balance the risks of false positives and false negatives with the required quantity of human marking to assess the assignment.

The resulting classifier was able to provide accuracy levels that are potentially feasible in an operational context, and the potential for significant overall reductions in the human marking load for this assignment.

Keywords: Automated Grading, Natural Language Processing, Reflection

Introduction

Aalborg University is well known for its Problem Based Learning (PBL) curriculum, with all enrolled students undertaking a semester long project course in every semester of their study. In this learning environment it is critically important that all students develop strong skills in PBL. To this end all first semester bachelor students enroll in a course on PBL skill development.

A key summative assessment task for this course is an assignment where students present reflections on their development of the competencies necessary to work in a PBL environment. In this way the development of interpersonal, structural, task planning etc skills becomes an explicit part of our curriculum, which is assessed separately to the project based courses in which these skills are applied.

The assessment task consists of an approximately 2000 word essay that is produced under examination conditions. It is submitted electronically through the learning management system Moodle. Each year around 1500 students complete this essay. The essay is co-marked by both PBL experts and department staff, with a total workload allocation of 20 minutes per essay. The assignment is marked with a simple pass/fail determination, and no explicit feedback beyond the pass/fail grade is provided to the students. All assignments marked as a fail (historically around 5-10% of submissions) are co-marked by a second assessor to confirm that the essay indeed warrants a fail. Students whose assignments are marked as a fail – ie those that cannot explicitly demonstrate their development of the necessary PBL core competencies - are not permitted to proceed to the presentation / viva voce stage of their project based courses in that semester.

Marking takes place inside a three-week time window, so that overall course grades can be finalised by the third week of December. A total of eight markers are used, with cross-marking to ensure consistency across all of the submissions. The overall process requires well over 500 hours of pass/fail summative assessment work within a very short timeframe – it is a task well suited to potentially being automated.

Addressing the challenge of such labour-intensive marking has received much attention in recent years with the significant advances in the field of Artificial Intelligence (AI) and Machine Learning (ML) [1]. In particular, Natural Language Processing (NLP) techniques are being developed to exploit textual information for a variety of interesting linguistic applications such as spam email filtering and sentiment analysis [2, 3]. Automated essay assessment model belongs to a class of NLP problems where the main task of the model is to learn features and relationships between classes of human-marked sample essays and produce their scores without further intervention of human markers. From an AI perspective, the process of assessing these essays is a conceptually simple task: to take 1500 essays and classify each of them as either a pass or a fail.

The classifier offers potentially greater consistency in the way that it marks. In particular, there will be no variability based on when an assignment is marked – the first and the last essay will be classified identically, without the impact of marker fatigue, moods etc. The classifier has the further advantage that this automated marking will be functionally instant; once the training data has been given to the classifier, the rest of the marking process proceeds at computing speeds.

Using an automated classifier potentially reduces the amount of human marking that is required in order to complete the overall marking of this assignment. A subset of the assessment submissions will need to be marked to provide a training set for the classifier, and assignments classified as fails by the model will still need to be remarked for operational reasons; but assignments classified as passing with an appropriate level of confidence can be passed without the need for any human marking of that assignment submission.

This paper explores the feasibility of building a classifier to fully replicate the human marking of these assignments, and investigates the tradeoffs between training set size, accuracy and potential human workload savings. The dataset for this investigation is a subset of the overall body of marking, representing a single year, single discipline subset of the overall cohort. The dataset contained a total of 243 student assignments, with 226 passes and 17 fails.

Building the classifier

The assessment task is a reflective exercise in which students are asked to respond to four questions about their PBL practice. In order to build a classifier, it is therefore necessary to pre-process the data in order to provide a standardized machine readable structure that can be used to build the classifier model. All of the text in the documents is converted to lowercase, and all of the punctuation is converted to whitespace. The preprocessing was implemented using regular expressions in the NLTK Python library [4].

The responses to each of the four questions were identified as a Region of Interest for the classifier; as such it was necessary to capture each of them separately from the files. While there is no standard structure required for submissions, there are emergent common practices. Standard headings such as “Opgave 1” (Task 1), “Kilde” (Source), “Bilag” (Appendix) are common throughout the submissions to identify the different sections. The pre-processor used python regular expressions to recognise these headings, and to partition the assignments into answers to each of the four questions. This also usually allowed for content within the assignments that offers less predictive power towards grades, such as names and cover sheets, and references, to be discarded from the text to be classified.

The remaining text in each sample was then tokenised using the Keras library [5]. The Keras library is an open-source neural network library that is designed to provide computing and preprocessing tools with deep neural network and NLP. Each word (or short sequence of words that commonly appear together) was converted to a token, and then represented in the model by the integer index from the library that matches that token. We begin by splitting the input text into words using the ‘text.split()’ function. This function splits the text at whitespace characters, resulting in a list of individual words. Next, we initialize the Keras Tokenizer and fit the tokenizer on the list of words obtained from the text splitting step. This process involves building a vocabulary from the words in the input text. The tokenizer assigns a unique integer value to each word in the vocabulary based on the word's frequency in the input text. Higher-frequency words receive lower integer values (starting from 1), while lower-frequency words receive higher integer values. After fitting the tokenizer, it can be used to' convert text to sequences of integer values corresponding to the words in the vocabulary.

The classifier is built as a Convolutional Neural Network [6]. The first layer, the embedding layer, takes our input assignment represented as a list of tokens, and replaces each of those tokens with a vector that captures the similarities between different tokens within the dataset. This more compact matrix representation of the overall submitted assignment allows for the relationships between ideas and words in the submitted assignment to be represented in a format that can be captured by our classifier.

The second layer, the convolution layer, then applies a series of filters to this matrix, with each filter looking for the presence of a particular feature in the matrix. These features are initially not known; it is through the process of training the classifier that weightings that represent meaningful features of the data emerge. The output of the convolution layer is a flattened vector that represents how these features manifest (or not) within the assignment being classified.

From this point the rest of the classifier is a traditional multilayer neural network. This network is made up of weighted connections between nodes, and by adjusting the weightings between the nodes we can train our classifier. The input layer size (128 in our case) corresponds to the size of the feature vector, and then using hidden layers of sizes 16 and 32 to represent the interactions between the features. This gives us 2592 different weights available for tuning, in addition to the parameters in the embedding and convolution layers.

The final output of this model is a probability value that this submission would be graded as a pass by a human marker. A threshold filter is then used to partition submissions with a probability below 50% as fails and those with 50% or higher as passes. A binary classification introduces potential issues around the threshold, where a small change in calculated probability (eg 49.99% to 50.01%) can lead to a complete change in the subsequent prediction (eg fail to pass); however, for the purposes of this feasibility study this simple threshold filtering is adequate for our purposes.

The overall classifier is represented in figure 1, which illustrates how a sentence like “The cat sat on the mat” would be processed by the classifier. The sample sentence is first tokenized and padded to account for varying sentence lengths. Each index is then mapped to an embedded vector generated through the embedding layer, where word embeddings are learned as 'fixed length' weights in a continuous vector space to capture semantic information. The embedded vectors pass through a one-dimensional convolution layer consisting of filters with learnable weights for local feature extraction and pattern recognition, and a rectified linear unit (ReLU) activation function is applied for introducing non-linearity. This is followed by a pooling operation to reduce the spatial dimensions. Subsequent Dense layers with 16 ReLU and 32 tanh nodes further refine the features before reaching the final Dense sigmoid layer with a single node, which outputs a probability prediction for the input's class – effectively the likelihood that the input would be assigned a pass grade by the human markers.

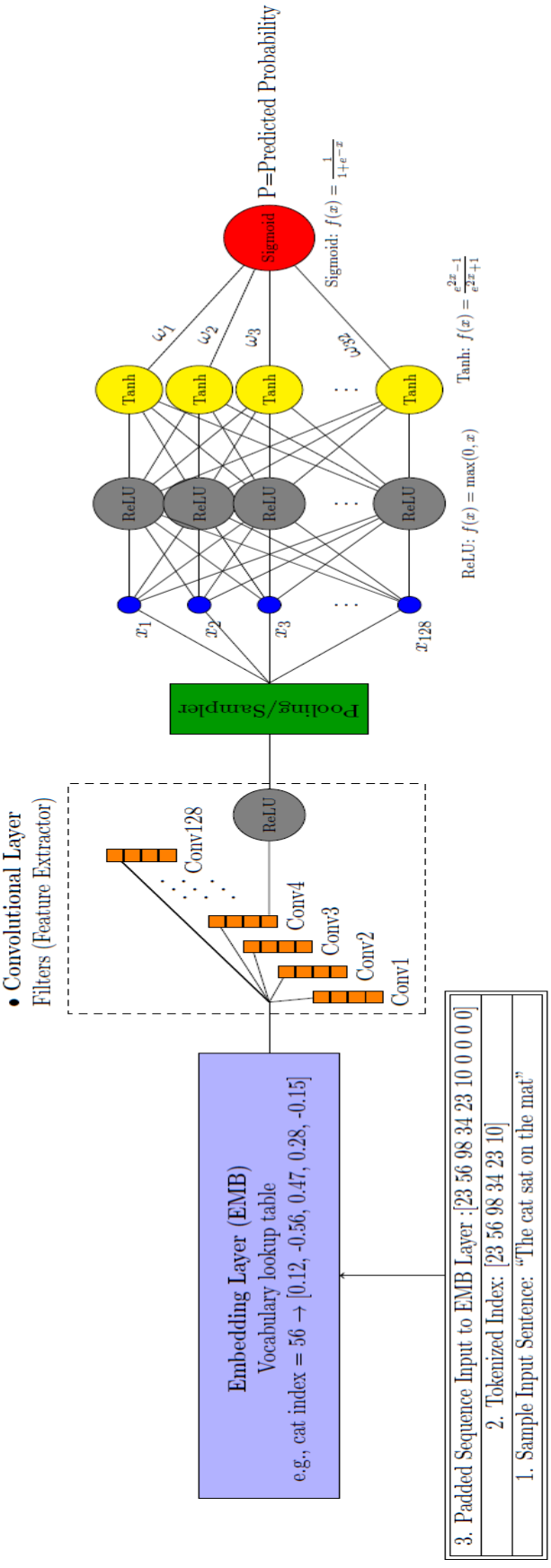

Figure 1: Representation of the classifer

Training the classifier

The classifier is trained by initialising the weighting parameters to arbitrary or random values, and then repeatedly exposing it to data from the training set, and adjusting the weightings to make its outputs closer to what was expected.

Considering the limited dataset size (n=243) and the potential for overfitting, we have incorporated L2 regularization in the network. This regularization technique helps to control the magnitudes of the weight values, discouraging the model from relying on a few large weights while minimizing the overall cost. As a result, it helps to prevent overfitting and encourages the learning of generalizable patterns in the data.

In light of the class imbalance in our dataset (226 passes vs 17 fails), we have incorporated class weights during the training process. This approach assigns different weights to each class, with higher weights assigned to underrepresented classes and lower weights to overrepresented classes. The class weights help the model to focus more on the minority class during training, improving its performance on imbalanced data. We opted for this technique over other methods such as sampling, SMOTE [7] or data augmentation, as they might not be as effective for text data and could lead to loss of valuable information or introduce errors. By using class weights, we can account for the imbalance while preserving the original essay content and context.

The classifier is exposed to inputs in batches of 16, and a resulting error (or "loss") function determines how accurately this version of the classifier predicts the already known outputs for that batch. The results of the error function are then backpropagated through the model, with each weight updated in the direction most likely to decrease the final error function. The classifier is then presented the next batch of training data and the error measurement and iteration is repeated.

The process of passing all of the training data through the classifier is known as an epoch. Each successive epoch should improve the accuracy of the classifier by exposing it again to the full training set, and in so doing improving the weightings and lowering the final loss function.

The training process continues until consecutive epochs no longer display sufficient improvement of accuracy (ie further epochs won't make the classifier more accurate) or when a pre-set maximum number of epochs (in our case 200) is reached.

Multiple versions of each classifier were built to explore the tradeoffs between the size of the training set and the classifier accuracy. Three sizes of training set were considered: 100 (small), 150 (medium) and 192 (large, representing the standard 80% of the data). Multiple instances of each classifier were trained; the average performance of each classifier is presented below.

Results and Discussions

The first classifier was built to the standard 80:20 model for classifiers: 80% of the data were used as the training set, and 20% used as test data. For this instance this means a 192 item

training set and a 50 item test set. Three instances of this classifier were built, and their aggregated performance is illustrated in figure 2:

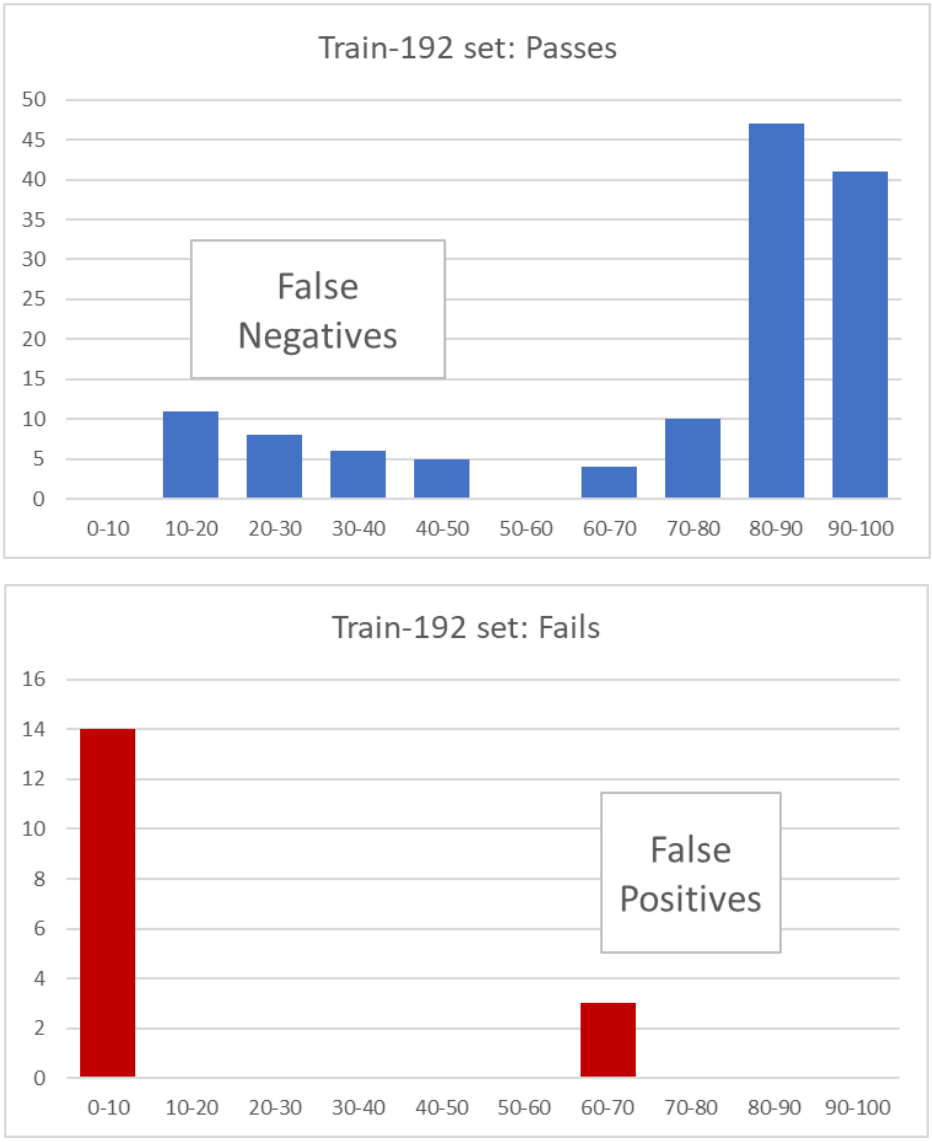

Figure 2: Histograms of aggregated performance for training-192 sets

Figure 2 shows a histogram of the probabilities allocated by the classifier to the 150 assignments in this dataset, with the assignments passed by the human markers (blue) and those failed by the human markers (red) presented separately. The horizontal axis represents the probability assigned by the classifier of each assignment being a pass.

Figure 2 shows that the classifier does reasonably well overall, with an average accuracy of 77.9%; however, the accuracy of specific instances varied from 69.4% to 84.0%. The classifier has the advantage that the errors in the accuracy are caused primarily by false negatives – indeed one of the three instances produced zero false positives in its classification.

We observe that the classifier for the most part does not classify ambiguously – there is in fact a window from 45% to 65% which contains only two outputs across the three instances. This means that the 50% threshold is a clear breakpoint for the purposes of dividing the test

data into passes and fails, and no instances in which a tiny variation in the output probability would result in a different prediction for that assignment. It also means that there is no "ambiguous" region that would require human intervention to resolve.

Three instances of the classifier were trained with a training set of 150 items, and tested with the remaining 93 items each time for a total of 279 test items. Reducing the size of the training set to 150 reduces the average accuracy of the classifier to 66.9% (Figure 3).

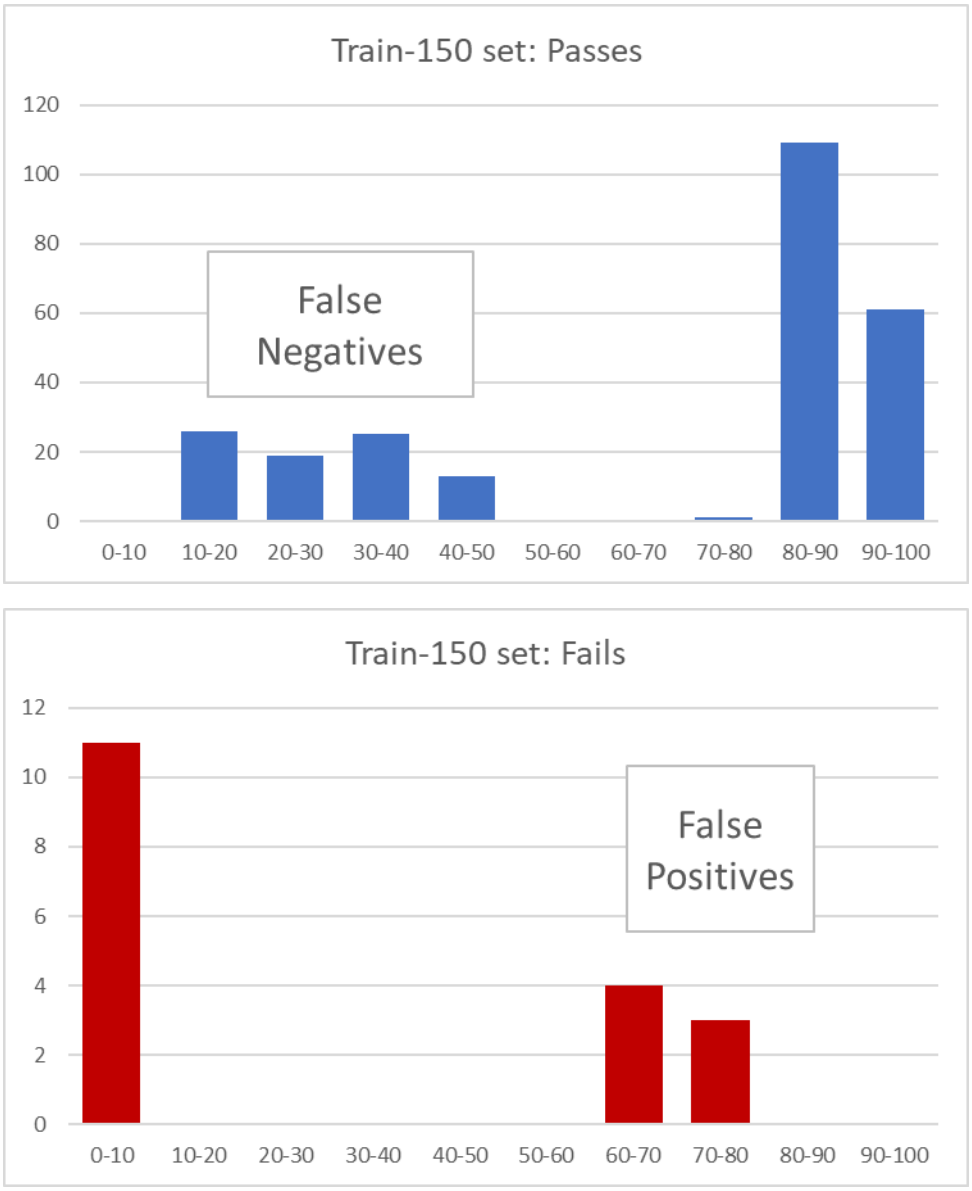

Figure 3: Histograms of aggregated performance for training-150 sets

Figure 3 shows that the training-150 set is less accurate than the training-192 set; it has an increased prevalence of both false positives and false negatives. It is worth noting that the instances were however much more consistent in their accuracy, ranging only from 65.6% to 67.8% in their accuracy.

The probability distribution retains the "valley" at around 50%, with only 8 assignments in total no assignments being classified with a probability between 45% and 65%. This means that the sharp cutoff of 50% is still viable as a way of allocating assignments to pass or fail.

Four instances of the classifier were implemented with a training set of 100 items, owing to a greater variation in output accuracy. Reducing the training set size further to 100 reduces the average accuracy of the classifier to 61.8% (figure 4) for 572 test data points:

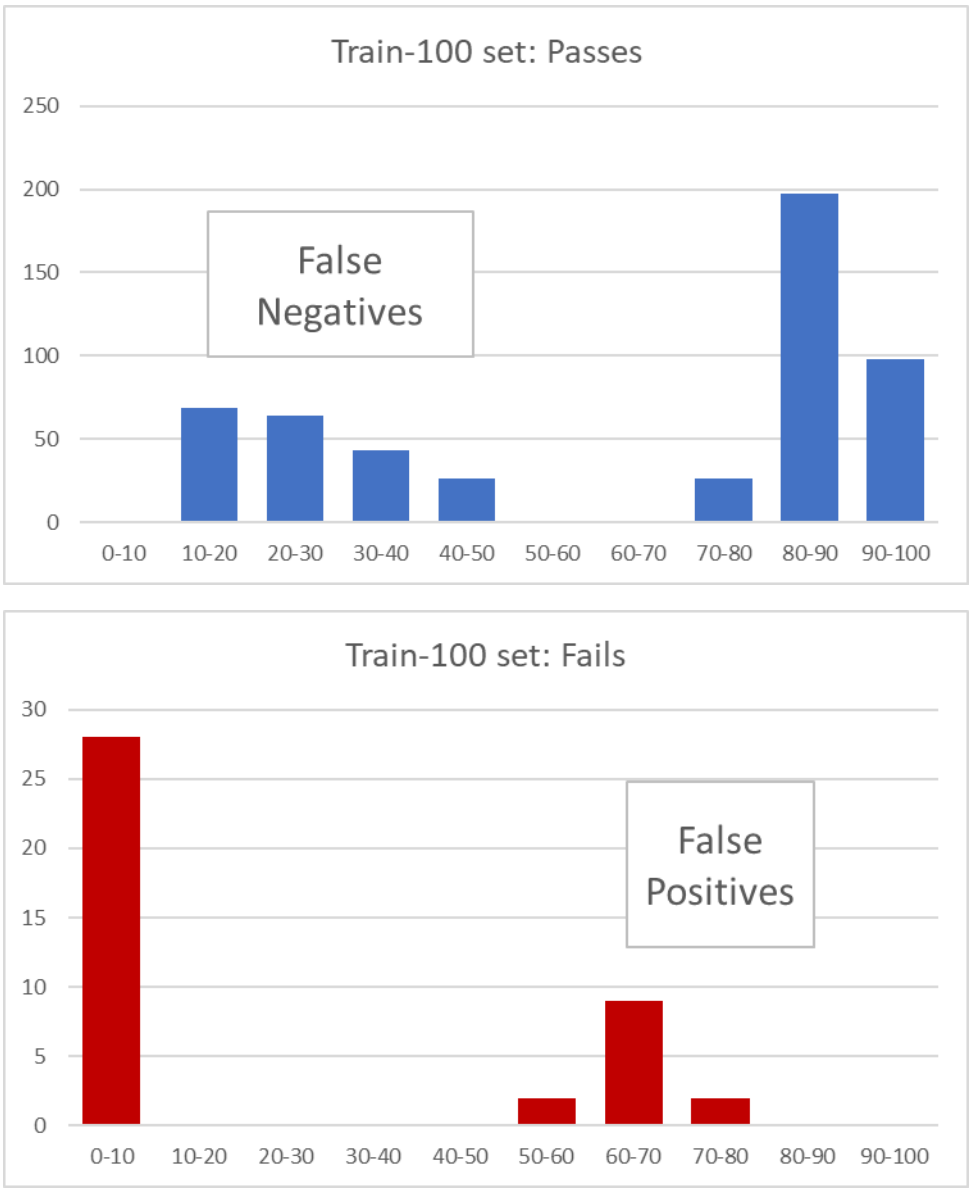

Figure 4: Histograms of aggregated performance for training-100 sets

Figure 4 shows that the reduction in accuracy is largely due to an increased prevalence of false negatives – the classifier is more likely to fail assignments that have been passed by the human markers. The "valley" at 50% is also present, with very few assignments being classified near the cut-off threshold. Performance of the training-100 classifiers is much more variable, with the output accuracy varying from 53.2% to 66.0%.

Lowering the training set even further to 50 led to a classifier that would not converge within the 200 epoch limit. There simply was insufficient information in a training-50 dataset in order to correctly train the classifier to mark these assignments.

Discussion

The effectiveness of the classifier is dependent upon the size of the training set; larger training sets produce more accurate classifiers (figure 5), but at the cost of requiring more manual assessment before the training can begin.

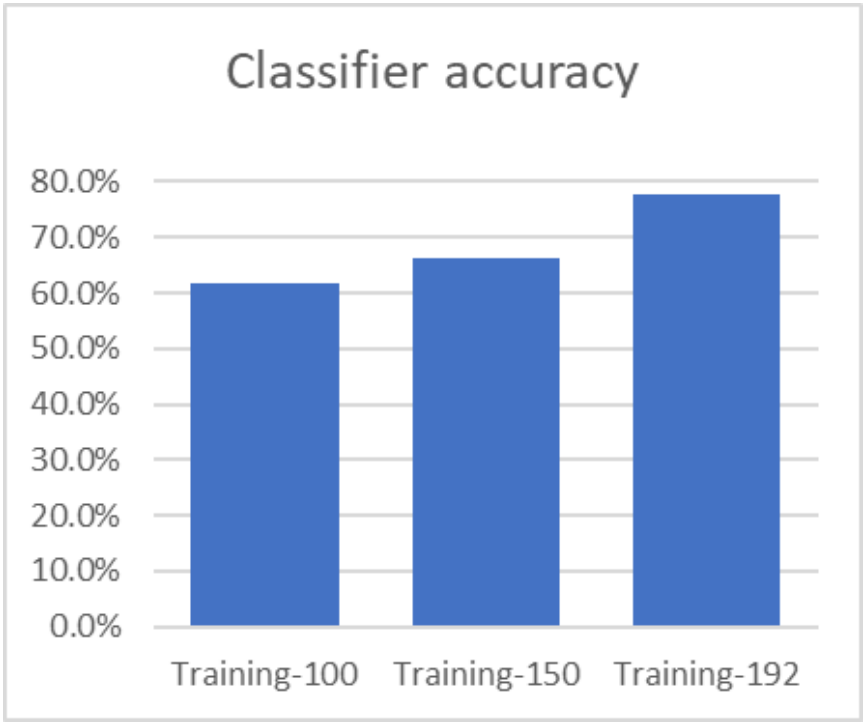

Figure 5: Classifier accuracy by training set

The worst case scenario for classifying was a training-100 set with an accuracy of 53.2%, which is functionally equivalent to a coin flip. The best case scenario was a training-192 scenario with 84.0%, which while below the desired level of accuracy is nonetheless potentially feasible given the operational context of the classifier.

The classifier is intended as a decision recommender system, not a decision-making system; there will still be human markers in the loop. In particular it is intended that humans would co-mark any assignment that is classified as a fail. This is consistent with the current operational practice of no assignment being failed without being marked by two different markers. It has the advantage in this context of converting false negatives from being errors into being lost savings – they become assignments that end up having to be marked anyway. The remarking set will contain proportionally more fails than the training set – it is drawn from the same population, but it has had a significant proportion of passing assignments removed. This phenomenon should ensure that the remarking process is not merely a rubber stamp of a recommendation from the classifier, but rather a true assessment of the assignments presented.

As such the overall efficiency of the classifier – the marking savings – is a balance between the size of the training set, the false negatives that need to be marked by a human anyway, and the cost of the false positives that go unquestioned through the system (figure 6):

Figure 6: Classifier efficiency by training set size

Figure 6 shows a clear overall trend: a larger training set leads to a higher accuracy classifier, but it does so at the cost of additional human marking to prepare that larger training set. Reducing the training set reduces the overall human effort required, even considering the remarking of false negatives; but it does so at the cost of the introduction of false positives as a consequence.

Conclusion

Overall it appears that it is possible to train a classifier to be an effective marker of these assignments; however it requires a substantial proportion of the dataset to be manually marked in order to train this classifier. The operational feasibility of the classifier is therefore dependent upon how large a saving of marking time is required, and how the necessary development effort of the classifier compares to that figure. The "price" of false positives will also need to be considered as part of this cost-benefit analysis.

This analysis has presented average accuracy figures as a way of considering feasibility. Operational use of the classifier will need to consider whether it is in fact the worst case figures that should be considered; after all the actual accuracy of the classifier will not be known at run time.

NLP classifiers appear to be a feasible alternative for supporting human markers in assessing large quantities of student assignments. As the marking pile grows larger the relative cost of

developing the classifier will become proportionally smaller and thus proportionally more attractive.

References

[1] Hussein MA, Hassan H, Nassef M. Automated language essay scoring systems: A literature review. PeerJ Computer Science. 2019 Aug 12;5:e208.

[2] Rogers A, Gardner M, Augenstein I. Qa dataset explosion: A taxonomy of nlp resources for question answering and reading comprehension. ACM Computing Surveys (CSUR). 2022.

[3] Salloum S, Gaber T, Vadera S, Sharan K. A systematic literature review on phishing email detection using natural language processing techniques. IEEE Access. 2022 Jun 14.

[4] Bird S. NLTK: the natural language toolkit. InProceedings of the COLING/ACL 2006 Interactive Presentation Sessions 2006 Jul (pp. 69-72).

[5] Chollet, F. & others, 2015. Keras. Available at: https://github.com/fchollet/keras

[6] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Communications of the ACM. 2017 May 24;60(6):84-90.

[7] Elor, Yotam; Averbuch-Elor, Hadar. To SMOTE, or not to SMOTE?. arXiv preprint arXiv:2201.08528, 2022.