Understanding Feedback: Extracting Suggestions From Student Evaluations

Felix Zhang

***Abstract*—Suggestion extraction is a domain within text classification in which models identify suggestions in texts. Research has been dedicated to applying various rule-based, classical and deep learning models to this task. Within the field of education, however, there remains a lack of research on the applicability of deep learning models to suggestion extraction.**

In our work, we address this gap by comparing the efficacy of various classification models at identifying suggestions in student feedback. We use a custom dataset of 7632 labelled sentences taken from SALG responses from a large introductory physics course. Overall, we found that deep learning classifiers outperformed classical models. The highest F_1 score of 0.823 was achieved by a BERT model while the SVM and naïve Bayes models both had a score near 0. The exceedingly low scores of the classical models are surprising as in previous research, they achieve F_1 scores of 0.6 on average. This raises an interesting avenue for future work investigating why the classical models performed significantly worse than expected as well as to develop a suggestion classifier that is not limited to feedback from physics-based courses.

***Index Terms*—Education, Natural Language Processing (NLP), Suggestion Extraction, Student Evaluations of Teaching (SET), Machine learning (ML), Deep Learning, Text Classification**

I. Introduction

Student evaluations of teaching (SET) are a valuable resource for instructors to obtain feedback on their courses. They typically consist of a combination of both Likert-scale questions and open-ended free-response questions. For large courses, however, it can be unreasonable for instructors to read and interpret all the student feedback [13]. Even when it is possible, it is difficult for them to summarize it all into a few actionable recommendations. To alleviate this, classical methods such as correlations and linear regression analysis [15] have been developed to summarize the quantitative feedback provided by Likert-scales. Likewise, for the free-response questions, much work has been done to develop models that identify the sentiments expressed by students in their evaluations [2], [21], [24] and ones that categorize student feedback into separate topics [6], [15]. These two branches of research have been combined to produce a program that automatically separates student feedback into various categories and then determines the overall student sentiment towards that subject [13]. This enables instructors to easily identify aspects of a course that students enjoy and those they find lacking.

I would like to thank Prof. Carolyn Sealfon for her constant aid and guidance throughout the research process. I would also like offer my gratitude to Hongzip Kim, Janice Qin, and Haoxiang Zhang for their invaluable aid with manually coding the data. This research was enabled in part by support provided by Compute Ontario and the Digital Research Alliance of Canada.

The code used to implement and train the models discussed in this paper can be accessed at *https://github.com/ff-zhang/extracting-suggestions*. All the diagrams used in this paper were created using LaTeX and the PGF/TikZ package.

In the free-response sections of SETs, students also offer concrete suggestions to the instructors as to what they believe would improve the course. This aspect of student feedback is not captured by the methods mentioned above as they simply summarize the opinions expressed in student comments without highlighting the specific improvements suggested by various students. Thus, it is an important piece of information that can be missed unless the instructor manually reads comments which, as outlined above, can be unreasonable. This problem, know as suggestion extraction, has been investigated with regards to online reviews for products [16], [23], [26]. There remains, however, a lack of research into its solutions with regards to identifying student suggestions with none examining the effectiveness of deep learning models at tackling this problem. This forces instructors of large courses into relying upon rule-based tools to summarize student feedback, which results in a loss of information due to their inaccuracy [2], or manually analyzing student responses, a tedious and time-consuming process. Improving the tools that instructors have in this regard could also enable them to ask for and receive student feedback throughout the semester. This allows them to implement course recommendations during the semester which helps students who are currently taking the class, not just those who will take it in the future. We hope to address this lack of research into suggestion extraction with regards to SET in our research.

II. Background

Within the field of text classification, there are broadly three types of models used when tackling problems. The most primitive are rule-based classifiers that use manually created syntactic rules to classify inputs. There are also classical models which use training data to generate features indicative of a class [22]. Finally, there are deep-learning classifiers that, generally, build a representation of each class and use them to sort inputs [11].

Suggestion extraction is a relatively new field of inquiry with text classification [26], having been chosen as one of the problems domains for the SemEval-2019 competition. As such, many machine learning models have been used to tackle this problem. Negi et al. [23] compare the performance of support vector machine (SVM), long short-term memory (LSTM), convolutional neural network (CNN) classifiers at identifying suggestions in tweets. They found that in a majority of cases the LSTM classifier has the highest F_1 score [23]. Reddy et al.

TABLE I
CLASSIFICATION OF DIFFERENT MACHINE LEARNING MODELS

Rule-Based	Classical	Deep-Learning
Text	Naïve-Bayes	LSTM
POS	SVM	GRU
		BERT

[26] then experimented with both a gated recurrent unit (GRU) and an attention-based LSTM classifier while Jayasekara et al. [16] used a bidirectional encoder representations from transformers (BERT) model to perform suggestion mining on a dataset of online reviews. As can be seen in Table I, models can be classified as either classical or deep-learning models.

Within the field of education, however, research into suggestions classification has been solely focused on rule-based and classical methods. Gottipati et al. [12] examined the effectiveness of parts-of-speech (POS) and classical classifiers. They found that, overall, classical machine learning models outperformed ones that rely on POS tagging [12]. Likewise, Almatrafi and Johri [2] proposed syntactic rules for identifying suggestions which showed similar performance. If we consider research into sentiment analysis of student feedback, a related field of text classification, we see that other models have also been used. Linear discriminant analysis was used by Cunningham-Nelson et al. [6] and Hujala et al. [15] to identify student satisfaction with regards to various aspects of a course. El-Halees [9] tested k-nearest neighbour(s), naïve Bayes and SVM classifiers. Onan [24] compared the efficacy of various combinations of word embedding schemes and models at determining the sentiment of student feedback. Of interest to us, they tested naïve Bayes, SVM, LSTM, and GRU classifiers and they showed that deep learning models outperformed their classical counterparts [24].

In the rest of this section, we will briefly review some of the most popular models mentioned above and how they function.

A. Naïve Bayes Classifiers

A naïve Bayes classifier is a machine learning model that applies the naïve Bayes theorem to probabilistically classify inputs [22]. This is done with a conditional probability model which, given a certain set of input features, determines the likelihood that the input is a member of the class y. Features, in this case, are simply quantitatively measurable properties of an object or event [11]. So, for an output class y, a naïve Bayes classifier determines its probability as

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} \tag{1}$$

given a set of input features $\mathbf{x}$ [22]. As the name suggests, it relies on the assumption that the input parameters are independent of one another and, as such, contribute equally to the probability of input being a member of a class. Using this assumption, a naïve Bayes classifier applies the equation

$$f(\mathbf{x}) = \underset{y}{\operatorname{argmax}}\, P(\mathbf{x}|y)P(y) \tag{2}$$

$$= \underset{y}{\operatorname{argmax}} \prod_{\alpha}^{d} P(x_\alpha|y)P(y) \tag{3}$$

to determine the class of a particular input $\mathbf{x}$ where x_α is the value of the feature α.

The assumption of independence on which this classifier is unfounded with regards to text as, if we consider a particular word in a sentence, it is clear that the choice was impacted by the rest of the sentence. Despite this, naïve Bayes classifiers have shown excellent performance in various (text) classification tasks [8].

B. Support Vector Machine Classifiers

SVM models are a type of non-probabilistic classifier that aim to find a decision function separating input data into two classes [27]. Due to the way an input is represented as a vector of n features, we can represent the range of possible inputs as a n-dimensional vector space. This means that each data point can be described as a point within this feature space. So, the goal of SVM classifiers is to produce a function that separates data points in different classes. This is equivalent to creating a boundary within the feature space such that all data points of the two classes lie on either side of it.

The simplest type of SVM classifier are linear. As the name suggests, they attempt to find the optimal hyperplane, the n-dimensional equivalent of straight line in a 2-dimensional space, to separate the data. These hyperplanes are described by the following linear function

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \tag{4}$$

where b is the bias term, and $\mathbf{w}$ is the weight vector of the hyperplane [27]. This is done by finding the function where the data points are far as possible from the boundary while ensuring that they are on the correct side. This is expressed using the equation

$$\min_{\mathbf{w},b} \mathbf{w}^T\mathbf{w} \quad \text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \tag{5}$$

where x_i is a data point and y_i is an output class [22].

If the data is not linearly separable, however, this style of SVM classifier will not be generalizable due to the non-linear nature of the decision plane. This issue is handled by allowing the decision function to be non-linear. Some popular alternatives used are radial bases, polynomial, and sigmoid functions. Mathematically, this is done by mapping the input data onto a higher dimensional Hilbert space where a linear SVM classifier can then be trained to differentiate between different points [27]. The input data in the Hilbert space can then be represented by $\phi(\mathbf{x})$ with the associated hyperplane defined as

$$\mathbf{w}^T\phi(\mathbf{x}) + b \tag{6}$$

Through a mathematical process known as the kernel trick, this gives rise to the following prediction function

$$f(\mathbf{x}) = \sum_{n=1}^{N} y_n a_n \mathcal{K}(\mathbf{x}, x_i) + b \tag{7}$$

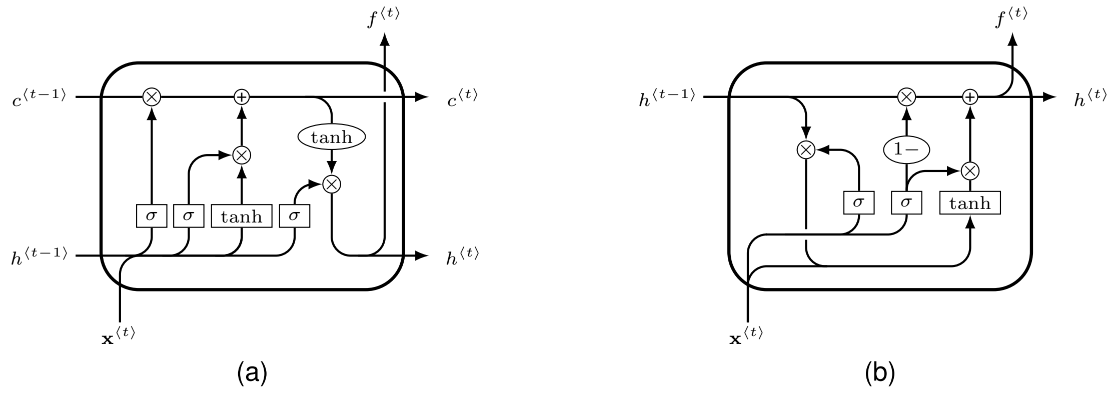


Fig. 1. Structure of a (a) LSTM memory cell, and (b) GRU memory cell. Note that σ denotes the sigmoid function, $+$ the element-wise sum, and $\times$ the element-wise product with arrows representing how the functions are composed. By following the arrow from a particular input, we can see which functions are applied to it in a time step and how it impacts each output value.

where x_i is an input feature and $\mathcal{K}$ is a Mercer kernel function [22]. As alluded to earlier, some popular types of kernels include radial basis, polynomial, and sigmoid functions.

C. Long Short-Term Memory Classifiers

Recurrent neural networks (RNNs) refer to neural networks that iteratively update their hidden state by repeatedly inputting data into the same neurons. Vanilla RNNs are the simplest form of these as they use their neurons to repeatedly apply the same weight matrices to an input sequence. It is these weight matrices that determine how an input feature affects the output of a neuron. This, however, results in the vanishing gradient problem which occurs when, in training, the gradient used to update the values of neurons becomes vanishingly small [22]. This effectively prevents further training from occurring from then onwards.

LSTM classifiers are a form of recurrent neural network (RNN) that were developed by Sepp Hochreiter and Jürgen Schmidhuber [14] to address the vanishing gradient problem [17]. This is done by replacing the neurons used in vanilla RNNs with LSTM memory cells. Each LSTM cell is comprised of an input gate, forget gate, and output gate in addition to the self-connected linear unit [10] found in vanilla RNN neurons. They operate in discrete time steps, with the output of a cell being fed as an input to itself. At timestep t, we denote the input vector as $\mathbf{x}^{\langle t \rangle}$, the cell state as $c^{\langle t \rangle}$, the hidden value as $h^{\langle t \rangle}$, and its output vector as $f^{\langle t \rangle}$. While slight variations exist, the LSTM cell architecture used in this paper is defined as

$$f_t = c^{\langle t-1 \rangle} \times \sigma(\mathbf{W}_f \mathbf{x}^{\langle t \rangle} + \mathbf{U}_f h^{\langle t-1 \rangle} + b_f) \tag{8}$$
$$i_t = \sigma(\mathbf{W}_i \mathbf{x}^{\langle t \rangle} + \mathbf{U}_i h^{\langle t-1 \rangle}) \tag{9}$$
$$\ell_t = i_t \times \tanh(\mathbf{W}_\ell \mathbf{x}^{\langle t \rangle} + \mathbf{U}_\ell h^{\langle t-1 \rangle}) \tag{10}$$
$$o_t = \sigma(\mathbf{W}_o \mathbf{x}^{\langle t \rangle} + \mathbf{U}_o h^{\langle t-1 \rangle}) \tag{11}$$
$$c^{\langle t \rangle} = i_t + \ell_t \tag{12}$$
$$h^{\langle t \rangle} = \tanh(c^{\langle t \rangle}) + o_t \tag{13}$$

where b is a bias vector and $\mathbf{W}, \mathbf{U}$ are trainable weight matrices that determine the impact of the associated vector on the output value. Refer to Fig. 1a for a depiction of this structure.

From the definition, we can see that LSTM classifiers pass a hidden state and a cell state in each iteration. Similarly to vanilla RNNs, the hidden state enables the LSTM classifier to make decisions over a short period of time. The addition of the cell state enables it to retain information for longer, helping to fight the vanishing gradient problem.

D. Gated Recurrent Unit Classifiers

Another architecture developed to address the vanishing gradient problem is the GRU classifier [5]. It was motivated by the LSTM cell and replaces the recurrent neurons in vanilla RNNs with GRU memory cells. These are composed of a reset gate and an update gate which are defined as

$$r_t = \sigma(\mathbf{W}_r \mathbf{x} + \mathbf{U}_r h^{\langle t-1 \rangle} + b_r) \tag{14}$$
$$u_t = \sigma(\mathbf{W}_u \mathbf{x} + \mathbf{U}_u h^{\langle t-1 \rangle} + b_u) \tag{15}$$

respectively. The cell itself is then defined by

$$\hat{h}_t = \tanh(\mathbf{W}_h \mathbf{x} + \mathbf{U}_h (r \times h^{\langle t-1 \rangle}) + b_h) \tag{16}$$
$$h^{\langle t \rangle} = z_t \times \hat{h}_t + (1 - z_t) \times h^{\langle t-1 \rangle} \tag{17}$$

where $h^{\langle t \rangle}$ is the output for timestep t. Fig. 1b provides a depiction of the structure of a GRU cell. Its structure allows for the cell to "adaptively control how much [it] remembers" [5] at each time step. This enables it to include both cells that catch short-term dependencies and long-term memory cells in its architecture [5] without the need for the two hidden states found in the LSTM cell. The ability to include long-term memory cells is what allows the GRU architecture to tackle the vanishing gradient problem.

E. Bidirectional Encoder Representations from Transformers Classifiers

For most deep learning classifiers such as the LSTM and GRU models described earlier, an sentence must be preprocessed before being input into the model. This preprocessing includes an embedding step in which semantically similar

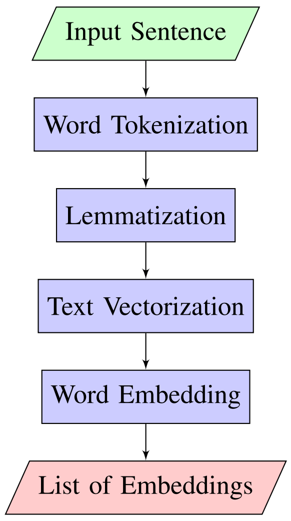


Fig. 2. Sentence Preprocessing Pipeline

words are represented with nearby vectors. This is done with a pretrained embedding model that has learned a vector for each word in its vocabulary. These models, however, only develop a static association between a vector and a word which means that, for words with multiple meanings, they have only one representation for them. The concept of self-attention was introduced to tackle this problem at it allows contextual information to be included when embedding sequences. Self-attention mechanisms work by comparing every word in an input sequence to every other input word [22]. An encoder can then use a self-attention mechanism to generate a contextual embedding of an input sentence as a vector of features [19].

BERT [7] is a language model based on these encoders equipped with a self-attention mechanism [29]. The base BERT model contains 12 encoders, each with 12 self-attention heads, and a hidden vector with 768 parameters [7]. The multiple self-attention heads enables an encoder to focus on different parts of a input sequence when determining the context of an input word. It is pretrained on 800 million words from Books Corpus and another 2.5 billion words from English Wikipedia [7]. The output of the first encoder is fed sequentially to the other 11 to ultimately produce a hidden state which can then be used tackle various tasks.

In order to accomplish a certain job, a task-specific layer is combined with the output of the BERT model described above [4]. This combined model is then jointly fine-tuned on a task-specific dataset to produce the final trained model [4]. In the case of text classification, a simple BERT classifier adds a dense layer equipped with a sigmoid activation function that takes the hidden state produced by the encoders as an input.

III. Data and Methods

To investigate the efficacy of various machine learning models at suggestion extraction, we used anonymized student responses from the Student Assessment of their Learning Gains (SALG) surveys [28] given to PHY132 undergraduate students at the University of Toronto at the end of the semester. These surveys consist of Likert scale and open-ended question(s) organized into categories such as “Your understanding of class content”, “Class impact on your attitudes”, and “Class Resources”. At the end of each category, there are free-response questions ask for feedback regarding that specific aspect of the course. Additionally, at the end of the SALG, it has an optional section that explicitly asks for possible improvements regarding any aspect of the course. In reviewing a sample of the responses provided, however, we found that many students do not limit their concrete suggestions to this optional final category and include recommendations for course improvements in their response to the other free-response questions.

The specific offerings of this course for which survey data is available are 2017, 2018, and 2019 summer and 2019, 2020, and 2021 winter. We only used the responses from the summer 2017 and winter 2021 offerings of PHY132, a large introductory physics course, due to time constraints. First, these responses were separated into individual sentences using the NLTK sentence tokenizer [3]. These sentences were then manually labelled by one individual as either containing a direct suggestion or not. This allowed us to create a dataset with a total of 7632 labelled sentences with a total of 869 sentences classified as suggestions. This dataset was then split into a training, validation, and testing dataset each consisting of 70%, 15%, and 15% of the entries respectively. Stratified sampling was used to ensure that there is the same percentage of suggestions to non-suggestions across the three datasets.

Before being input into a model, a sentence is processed with a text preprocessing pipeline consisting of tokenization, lemmatization, text vectorization, and word embedding. For the word tokenization and lemmatization, we used the NLTK library [3] due to its prevalence in the research on this topic. The resulting lemmas where then vectorized by frequency using TensorFlow [1]. The original word2vec model [20] was then used to embed each word in the vectorized sentence. The word2vec model has a vocabulary of three million words, which was limited to one million due to computing constraints, and an output vector length of 300 features. These embedded vectors were then fed into all the models except the BERT classifier during the training and evaluation process. Fig. 2 depicts this preprocessing pipeline. Note that inputs for the BERT classifier were not preprocessed as it contains the BERT language model which embeds input sentences.

A. *Classical Classifiers*

The SVM and naïve Bayes classifiers were both implemented using the scikit-learn library [25]. The hyperparameters for both the models were then determined using a Bayesian hyperparameter optimizer from the scikit-optimize library. After testing linear, radial basis, polynomial, and sigmoid kernels for the SVM, we found that the best performing kernel in training was a linear function with a regularization parameter of 23694. Similarly, the naïve Bayes classifier was trained with a smoothing parameter of 0.345 after it was determined to be the most effective. Tab. III of the Appendix shows the full range of parameters tested in the hyperparameter optimization process.

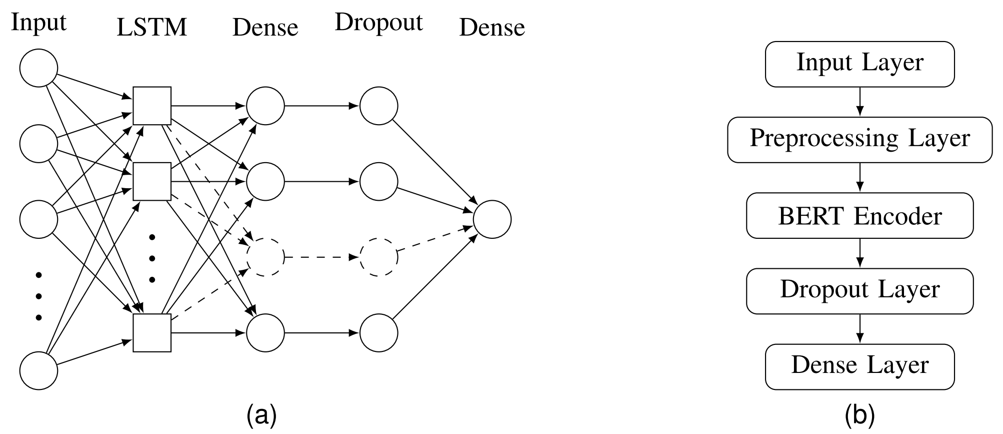


Fig. 3. Architecture of the (a) LSTM classifier, and (b) BERT classifier. Each circle in the LSTM classifier represents a neuron and each square a LSTM memory cell. The arrows represent where the output of a layer or neuron is used as an input. The dashed circle denotes a hypothetical neuron whose output has been set to 0 in training. This is done to show that its output does not impact the output value. Likewise, the dashed arrows denote outputs that have no affect on the final prediction.

B. Deep Learning Classifiers

The LSTM, GRU, and BERT models were implemented in TensorFlow [1] which is a machine learning framework developed by Google. The hyperparameters for these models were determined through a grid-search hyperparameter optimizer. The list of options considered by the grid-search optimizer can be found in Tab. IV of the Appendix.

After testing, we found the optimal LSTM model architecture to be as follows. The input layer is directly passed to a hidden layer composed of 64 LSTM memory cells. This is then fed into fully-connected layer comprised of 4 neurons and a dropout layer. In a fully-connected layer, the input for each neuron consists of every output from the previous layer which enables it to find relationships in the features identified by the previous layer [11]. So, in this case, the input to a fully-connected neuron is the output of the 64 memory cells. The dropout layer then randomly, during training, sets neurons' output to 0 to help prevent overfitting [11]. Finally, the output from the dropout layer is sent to one fully-connected neuron. This neuron applies the sigmoid function to determine the probability the input was a suggestion. The complete architecture of the LSTM classifier is shown in Fig 3a with one dropped neuron. The GRU classifier has the same architecture except the LSTM layer is replaced with a hidden layer consisting of 8 GRU cells.

Fig 3b depicts the architecture of the BERT classifier used where an input layer feeds, successively, into a preprocessing, BERT encoding, dropout, and dense layers. The architecture of the BERT layer used is the one described earlier with 12 encoders, 12 self-attention heads, and a hidden vector with 768 parameters. Furthermore, the final dense layer consists of a single neuron equipped with a sigmoid activation function. It performs the same functionality as the final dense layer in the LSTM and GRU classifiers in determine the probability an input sequence is a suggestion.

All three models were trained with a batch size of 32 and a learning rate of 1×10^{-4}. The LSTM classifier was trained with an $\hat{\epsilon}$ value of 1×10^{-6} for the Adam optimizer [18] and a dropout rate of 0.2 over 10 epochs. The hyperparameters used in the training of the GRU classifier were similar with an $\hat{\epsilon}$ value of 1×10^{-8} and dropout rate of 0.1 over 150 epochs. Finally, the BERT classifier was trained with a dropout rate of 0.5 over 4 epochs. The number of epochs were chosen to limit the amount of overfitting in the training process. This was done by identifying the point in training at which the F_1 score of a model on the training dataset increased while it was decreasing on the validation dataset. The rest of the hyperparameters were determined with a grid-search optimizer.

IV. Analysis

Usually, when training and testing classifiers, accuracy is used to measure their performance. When models are trained to minimize errors on imbalanced datasets, however, they tend to predict the majority class more frequently [16]. Moreover, in the case of an imbalanced dataset achieving a high accuracy score is misleading. For example, a model which always predicted a sentence to not be a suggestion would have had an accuracy of 0.889 on our testing dataset making it difficult differentiate between the performance of the models. Due to these reasons, the classification performance of all the models was evaluated using the F_1 score for the positive class. The F_1 score, which is the harmonic mean of precision and recall, is a popular metric for imbalanced classification [16]. It is defined as

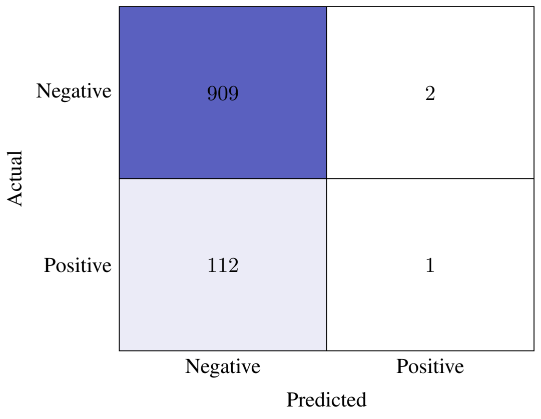
$$p = \frac{P_T}{P_T + P_F} \tag{18}$$

$$r = \frac{P_T}{P_T + N_T} \tag{19}$$

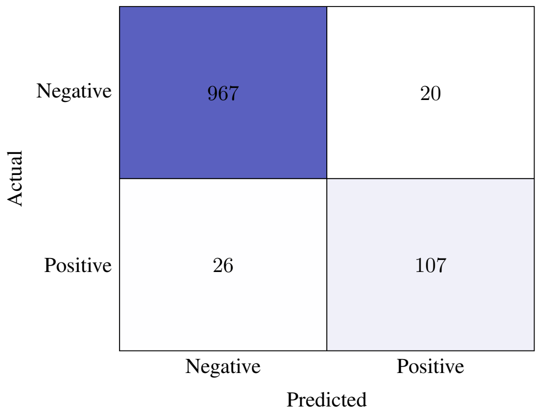
$$F_1 = 2 \times \frac{p \times r}{p + r} \tag{20}$$

where P_T is the number of true positive classifications, P_F the number of false positives, N_T the number of true negatives, and N_F the number of false negatives for a model.

From Table II, we can see that the deep learning classifiers significantly outperformed the classical models with regards to this metric. The highest F_1 score of 0.816 was achieved by the BERT classifier. The performance of the deep learning



(a)



(b)

Fig. 4. Confusion matrix of the (a) naïve Bayes and (b) BERT classifiers on our testing dataset

TABLE II
PERFORMANCE IN SUGGESTION EXTRACTION

Classifier	Precision	Recall	F_1 Score
SVM	0.000	0.000	0.000
naïve Bayes	0.333	0.009	0.017
LSTM	0.670	0.606	0.631
GRU	**0.871**	0.583	0.698
BERT	0.805	**0.842**	**0.823**

models on this task were within the range of scores that were on similar text classification problems [23], [24], [26]. If we examine the BERT classifier's performance as depicted in Fig 4b, out of the 1120 tested sentences, 46 of them were incorrectly classified with a total of 107 true positive predictions.

To place the performance of the BERT classifier in context, we compared its performance with that of another manual coder. This was done having a separate group of 3 individuals manual code a subset of the dataset. The labels they assigned to each sentence were then compared with original ones from the dataset. There was a total of 28 false negatives where the group of 3 coders labelled a sentence as a non-suggestion while the dataset labelled it as a suggestion and 43 false positives. Thus, out the 1268 sentences that were compared, there was agreement between 94.4% of the labels assigned by the two groups of manual coders. With 116 true positive labellings, the manual coders have a F_1 score of 0.766 which is lower than that of the BERT classifier. While the low F_1 score of the manual coders suggests that there is an indistinct boundary between non-suggestions and suggestions, this shows that the BERT classifier achieved human-like performance on this task.

On the other hand, the performance of the two classical classifiers is unexpected. SVM and naïve Bayes models typically achieve a F_1 score around 0.6 in other forms of text classification on student feedback [2], [9], [12]. Examining the confusion matrix of the naïve Bayes classifier in Fig. 4a, we see that the model predicted that almost none of the inputs were suggestions. This suggests that both classical models struggled to determine features within the training dataset that differentiate between a suggestion and a non-suggestion and those it did identify were not useful during testing. This raises the question as to why as, in previous research, they had been able to identify suggestions within student feedback but not within the dataset we developed.

V. CONCLUSION

In this paper, we examine the usefulness of classical and deep-learning classification models at extracting suggestions from student feedback. We contribute a new dataset of labelled SALG responses from an introductory physics course. Furthermore, we tested both models used in previous work as well as deep learning architectures that have not yet been applied to the problem of suggestion extraction from student evaluations. We found that the all the deep-learning classifiers significantly outperformed the two classical architectures within this problem domain. The highest F_1 score of 0.832 was achieved by a BERT classifier which surpassed the F_1 score of manual human coders on a subset of the dataset.

While we only worked with data from introductory undergraduate physics courses, "suggestions tend to exhibit similar linguistic nature, irrespective of topics" [23]. Furthermore, while there will be course and field specific suggestions given, many feedback topics such as homework and course pacing are common across disciplines. Hence, we expect that our results to be applicable with regards to the effectiveness of deep-learning models at analyzing student feedback in undergraduate courses across various disciplines. This highlights an interesting domain for research to explore in future with regards to creating a general suggestion classifier that is not limited to identifying suggestions from physics-based courses. Moreover, we also found an unexplained anomaly as to the poor performance of the classical classifiers on this dataset that suggest a future area of research. It seems pertinent to identify the reason behind this as well as developing methods for tacking this problem. This could be done by comparing the datasets on which the different classical models were trained. Furthermore, research could be done on the cross-applicability of a classical model by training it on one dataset and testing it on another.

Appendix

TABLE III
Hyperparameters Tested for Classical Models

Model	Parameter	Values/Range Tested
SVM	Kernel	Linear, Polynomial, RBF, Sigmoid
	C	$1e-6-1e+18$
	γ	$1e-18-1e+6$
Naïve Bayes	α	$1-2$

TABLE IV
Hyperparameters Tested for Deep Learning Models

Model	Parameter	Values Tested
LSTM	Number of LSTM Cells	$8, 16, 32, 64$
	Number of Dense Neurons	$4, 8, 16, 32$
	Dropout Value	$0.1, 0.2, 0.3, 0.4, 0.5$
	Learning Rate	$1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8$
	$\hat{\epsilon}$	$1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8$
GRU	Number of GRU Cells	$8, 16, 32, 64$
	Number of Dense Neurons	$4, 8, 16, 32$
	Dropout Value	$0.1, 0.2, 0.3, 0.4, 0.5$
	Learning Rate	$1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8$
	$\hat{\epsilon}$	$1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8$
BERT	Dropout Value	$0.1, 0.2, 0.3, 0.4, 0.5$
	Initial Learning Rate	$1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7$

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," Mar. 2016.

[2] O. Almatrafi and A. Johri, "Improving MOOCs Using Information From Discussion Forums: An Opinion Summarization and Suggestion Mining Approach," *IEEE Access*, vol. 10, pp. 15 565–15 573, 2022.

[3] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media Inc, 2009.

[4] I. Chalkidis, E. Fergadiotis, P. Malakasiotis, and I. Androutsopoulos, "Large-Scale Multi-Label Text Classification on EU Legislation," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 6314–6322.

[5] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734.

[6] S. Cunningham-Nelson, M. Baktashmotlagh, and W. Boles, "Visualizing Student Opinion Through Text Analysis," *IEEE Transactions on Education*, vol. 62, no. 4, pp. 305–311, Nov. 2019.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186.

[8] P. Domingos and M. Pazzani, "On the Optimality of the Simple Bayesian Classifier under Zero-One Loss," *Machine Learning*, vol. 29, no. 2, pp. 103–130, Nov. 1997.

[9] A. El-Halees, "Mining Opinions in User-Generated Contents to Improve Course Evaluation," in *Software Engineering and Computer Systems*, ser. Communications in Computer and Information Science, J. M. Zain, W. M. bt Wan Mohd, and E. El-Qawasmeh, Eds. Berlin, Heidelberg: Springer, 2011, pp. 107–115.

[10] F. Gers, N. Schraudolph, and J. Schmidhuber, "Learning Precise Timing with LSTM Recurrent Networks," *Journal of Machine Learning Research*, vol. 3, pp. 115–143, Jan. 2002.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ser. Adaptive Computation and Machine Learning. Cambridge, Massachusetts: The MIT Press, 2016.

[12] S. Gottipati, V. Shankararaman, and J. R. Lin, "Text analytics approach to extract course improvement suggestions from students' feedback," *Research and Practice in Technology Enhanced Learning*, vol. 13, no. 1, p. 6, Dec. 2018.

[13] N. Gronberg, A. Knutas, T. Hynninen, and M. Hujala, "Palaute: An Online Text Mining Tool for Analyzing Written Student Course Feedback," *IEEE Access*, vol. 9, pp. 134 518–134 529, 2021.

[14] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997.

[15] M. Hujala, A. Knutas, T. Hynninen, and H. Arminen, "Improving the quality of teaching by utilising written student feedback: A streamlined process," *Computers & Education*, vol. 157, p. 103965, Nov. 2020.

[16] T. Jayasekara, R. Weerasinghe, and V. Welgama, "Aspect Oriented Suggestion Extraction from Online Reviews," in *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2021, pp. 1561–1568.

[17] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An Empirical Exploration of Recurrent Network Architectures," p. 9.

[18] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Jan. 2017.

[19] Z. Lu, P. Du, and J.-Y. Nie, "VGCN-BERT: Augmenting BERT with Graph Embedding for Text Classification," in *Advances in Information Retrieval*, ser. Lecture Notes in Computer Science, J. M. Jose, E. Yilmaz, J. Magalhães, P. Castells, N. Ferro, M. J. Silva, and F. Martins, Eds. Cham: Springer International Publishing, 2020, pp. 369–382.

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *Proceedings of Workshop at ICLR*, vol. 2013, Jan. 2013.

[21] M. Misuraca, G. Scepi, and M. Spano, "Using Opinion Mining as an educational analytic: An integrated strategy for the analysis of students' feedback," *Studies in Educational Evaluation*, vol. 68, p. 100979, Mar. 2021.

[22] K. P. Murphy, *Probabilistic Machine Learning: An Introduction*, ser. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: The MIT Press, 2022.

[23] S. Negi, K. Asooja, S. Mehrotra, and P. Buitelaar, "A Study of Suggestions in Opinionated Texts and their Automatic Detection," in *Proceedings of the Fifth Joint Conference on Lexical and Computational Semantics*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 170–178.

[24] A. Onan, "Mining opinions from instructor evaluation reviews: A deep learning approach," *Computer Applications in Engineering Education*, vol. 28, no. 1, pp. 117–138, 2020.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011.

[26] T. R. Reddy, P. V. Reddy, T. M. Mohan, and R. Dara, "An approach for suggestion mining based on deep learning techniques," *IOP Conference Series: Materials Science and Engineering*, vol. 1074, no. 1, p. 12021, Feb. 2021.

[27] P. Samui, S. S. Roy, and V. E. Balas, Eds., *Handbook of Neural Computation*. London ; San Diego, CA: Academic Press, an imprint of Elsevier, 2017.

[28] E. Seymour, D. Wiese, A.-B. Hunter, and S. Daffinrud, "Creating a Better Mousetrap: On-line Student Assessment of Their Learning Gains," Jan. 2000.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., Dec. 2017, pp. 6000–6010.