Detecting Suggestions in Peer Assessments

Gabriel Zingle, Balaji Radhakrishnan, Yunkai Xiao, Edward Gehringer, Zhongcan Xiao, Ferry Pramudianto, Gauraang Khurana, and Ayush Arnav

Department of Computer Science
North Carolina State University
Raleigh, NC 27606
+1 919-515-2066
{gzingle, bradhak, yxiao28, efg, zxiao2, fferry, gkhuran, aarnav}@ncsu.edu

ABSTRACT

Peer assessment has proven to be a useful strategy for increasing the timeliness and quantity of formative feedback, as well as for promoting metacognitive thinking among students. Previous research has determined that reviews that contain suggestions can motivate students to revise and improve their work. This paper describes a method for automatically detecting suggestions in review text. The quantity of suggestions can be treated as a metric for the helpfulness of review text. Even before a review is submitted, the system can tell a reviewer when a review is lacking in suggestions and consequently advise that they be added. This paper presents several neural-network approaches for detecting suggestions and compares them against traditional natural language processing (NLP) methods such as rule-based techniques, as well as past machine-learning approaches.

Our network-based classifiers outperformed rule-based classifiers in every experiment. Our neural-network classifiers attained F1-scores in the low 90% range, outperforming the support vector machine (SVM) classifier whose F1-score was 88%. The naïve Bayes (NB) classifier had an F1-score of 84% and the rule-based classifier had an F1-score of 80%. As in other domains such as determining sentiment, we found that neural-network models perform better than the likes of naïve Bayes and support vector machines when classifying suggestions in text.

Keywords

Peer assessment, suggestion mining, classification techniques, text analytics, text mining.

1. INTRODUCTION

Peer assessment is known to have several advantages for student learning [1]. It provides students with prompt and rich feedback that helps them improve their performance and learning readiness [2]. It gives students an opportunity to learn from others by observing their approaches to solving a problem. Not only do students learn from the feedback they receive; they also benefit from reviewing others. In fact, they probably learn more from reviewing than they do from receiving feedback [3, 4, 5, 6, 7, 8]. Finally, reviews from other students may help instructors to assign more informed grades.

Gabriel Zingle, Balaji Radhakrishnan, Yunkai Xiao, Edward Gehringer, Zhongcan Xiao, Ferry Pramudianto, Gauraang Khurana and Ayush Arnav "Detecting suggestions in peer assessments" In: *Proceedings of The 12th International Conference on Educational Data Mining (EDM 2019),* Collin F. Lynch, Agathe Merceron, Michel Desmarais, & Roger Nkambou (eds.) 2019, pp. 474 - 479

However, these advantages can only be achieved with high-quality feedback. Nelson and Schunn [2] showed that high-quality feedback comprises several features, including suggestions on how to address problems in the work. Having concrete suggestions makes the feedback actionable for the reviewee and also trains the reviewer to solve problems [2], instead of just focusing on the existence of the problems.

So it is desirable for peer reviews to contain suggestions, but it is not easy for the instructor to give students credit for making them. The instructor would have to look through each review and keep a tally. If this could be done automatically, particularly before the reviewer submits a review [9, 10], it could help reviewers to improve the quality of their feedback, which in turn would help reviewees to improve their work.

2. LITERATURE REVIEW

Before digging in further, we must first decide what constitutes a suggestion. Negi and Buitelaar [11] state that, due to the variation in the definition of suggestion, previous results may be incompatible with each other. In this paper, we define suggestions as comments that constitute advice for making improvements [12].

Suggestions normally contain some or all of the components of specificity [2]: locating the problem, identifying the problem, offering a solution to the problem. Specificity has proven to have a direct correlation to understanding. Understanding is found to be the only significant mediator that directly contributes to the likelihood of implementing the feedback, thus improving the student's work.

Some effort has been made to detect suggestions through conventional natural language processing. These NLP approaches usually utilize rules that match the feedback to a set of predefined linguistic patterns, as well as part-of-speech (POS) tagging and a carefully crafted thesaurus relevant to each domain on which the approach is going to be implemented [13, 14, 15]. The main drawback of these approaches is that they require the knowledge of engineers to define the rules and patterns in the software logic. However, it is almost impossible to foresee all possible rules and patterns that indicate the existence of suggestions in a text document. Thus, as the number of patterns and rules grow, it becomes very difficult for engineers to maintain the software.

Another approach that has shown promising results for detecting patterns is based on machine-learning techniques. For instance, researchers have utilized machine-learning methods such as SVM with some degree of success to find patterns that indicate the sentiment polarity of a text [9].

Machine-learning algorithms are able to discover patterns and rules automatically based on training samples [16]. For software engineers, this is a game-changer since it allows them to keep the complexity of the software consistent and manageable. There are many machine-learning algorithms that can be used for classifying text as to whether it contains suggestions or not, such as decision trees (DT) and support vector machines [17]. Machine-learning approaches can usually outperform rule-based approaches once they are trained with sufficient and representative samples that have been labeled. However, obtaining labeled data is usually very expensive.

Recently, the neural network approach has stood out for its ability to solve classification problems in different domains, e.g., image and voice recognition, and text classification [11]. This work tries to apply neural networks to suggestion detection and compares the results with the rule-based and traditional machine-learning classifiers.

3. DATA

In this experiment, we used a balanced dataset of peer reviews with a total of 3878 peer reviews, each labeled as containing a suggestion (1) or as not containing a suggestion (0). It is extracted from the Expertiza [18, 19] platform, which we would describe later on in this section. The dataset has two main components: (i) the input text, and (ii) a label indicating whether suggestions were present in the text. The datasets for the neural network classifiers were broken down into an 80–10–10% split for training, validation, and test sets. For the naïve Bayes classifier and the support vector machine classifier, split for training and test sets was 90-10%. All the models were trained and tested on the same dataset.

The platform we mentioned in the previous paragraph, Expertiza, is a web-based peer-review system that allows students to exchange ideas and build shared knowledge. It facilitates anonymous reviews for students' submissions. In this paper, we use reviews submitted to Expertiza to test our approaches. Authors who received reviews manually labeled 15,067 review comments as to whether they contained a suggestion. The labeling of these reviews was incentivized by giving extra credit to students in a class who tagged review comments. A team made up of the instructor, TAs, and authors of this paper went through the labels and removed labels assigned by students whose labels were clearly not trustworthy. Untrustworthy labels included random labels unrelated to review content, all no's or all yes's on reviews with obvious variety of status of containing suggestions. Inter-rater reliability of the dataset was calculated between each team of students labeling peer reviews with a Krippendorff alpha [20] of 0.69 before removing unreliable labels and 0.74 afterwards. Students receiving these reviews were an ideal choice for annotating these reviews, since the reviews were of their own work. Thus, they were capable of judging whether they really contained suggestions. In the preprocessing stage, entries that are invalid to the experiment, such as blank entries, duplicated entries, entries with unrecognizable symbols and marks, as well as html tags, were striped from the dataset. After preprocessing, the dataset contained 5,842 entries without suggestions, and 1,939 entries with suggestions.

With an imbalanced dataset, the process of machine learning is greatly compromised. Classifiers such as naïve Bayes that have been trained on too great a prevalence of a single label type can create a classifier that will only predict that single label. Machine-learning algorithms tend to focus much more on a prevalent class and much less on rare cases, even if the rare cases are trustworthy [21].

Table 1: Sample review comments.

Review Comment	Contains a suggestion?
"Very well written and very obvious how much of an impact the refactoring had."	No
"Yes, all the functionality are covered in the design document."	No
"The test plan talks about automated tests, but then goes on and talks about 2 manual testing scenarios. Details for automation test cases are missing. If the team does not plan on writing automated test cases, then it should be mentioned in the documentation. Otherwise, the team should provide more details about the test plan for automated test cases."	Yes
"Code is good. Just in tests, the code could've been reduced in some cases where same object is being mocked multiple times. "	Yes

One way to deal with this problem is to apply selection techniques. A well-known method is to down-sample the dataset. One should always keep all rare positive samples that need to be focused on and only prune out negative samples [22].

We applied this technique on our dataset in order to minimize the impact of an imbalanced dataset. Down-sampling kept all 1,939 of the positive cases in our dataset and randomly selected 1,939 negative cases to form a new balanced dataset that we used for analysis. Table 1 shows sample review comments with their respective labels. The dataset is available upon request.

4. METHODOLOGY

This section describes the methodology for each classifier. Our input data for these Classifiers is document level review text instead of sentence level, frequently we would observe students describing their solutions in sentences or even paragraphs, and we believe keeping them in their original form would be beneficial for the study. Currently, we are rating the quality of a review based on the presence of a single suggestion. Later on, we can generalize our approach to count the number of sentences that contain suggestions.

4.1 Traditional Machine Learning and Rule-based Methods

Section 4.1 discusses the formation of the rule-based, naïve Bayes, and support vector machine classifiers. Text preprocessing in the form of stop-word removal and stemming was not used except for stemming the data for the naïve Bayes classifier. The reason behind this was due to decreased classifier performance by experimenting with these preprocessing techniques and then testing the classifiers.

4.1.1 Rule-based NLP Methods

Part-of-speech tagging was used to determine the word class of each processed word. The relevant tags used for this classifier included MD (modal auxiliary), VB (verb, base form), VBZ (verb, present tense, 3rd person singular), VBP (verb, present tense, not 3rd person singular), NNS (noun, common, plural), and NNP (noun, proper, singular).

The idea and rules for this rule-based classifier were derived from Gottipati et al. [17]. We used pattern-matching and part-of-speech tagging methods, based on the work of Brun and Hagège [14], Bird et al. [13], and Marcus et al. [15]. The patterns we used are as follows.

• The first rule utilized a pattern-matching technique that locates specific phrases and keywords that indicate the likely presence of a suggestion. The phrases used included “have more”, “suggestion”, “perhaps”, and “better if”.

• The second rule uses part of speech tagging through the Natural Language Toolkit (NLTK) [13] to find a sequence of verb pairs in a review. If a word was tagged as a modal (MD) and the word directly following this was tagged as either of the verb types VB, VBZ, or VBP, then the review would be classified as a suggestion. For example, in the sentence “You should have included a diagram to visualize your results.” the words “should have” would be tagged as MD and VB respectively. This sentence would be classified as a suggestion.

• Following Gottipati, the third rule also used POS tagging. Reviews were classified as a suggestion if a word tagged NNS or NNP was followed by a word of type VB, VBZ, or VBP. This is similar to rule 2 and it was used due to the fact that some words would be labeled with different tags depending on whether they were the first word in a sentence or were located within the sentence. An example of this rule would be the processing of a sentence such as “Could include more examples in your lab report.” where the words “Could include” are tagged as NNP and VBP respectively by the NLTK POS tagger. It is important to note that a word such as “Could” will be tagged as NNP or MD depending on whether it starts a sentence or not. This would classify the sentence as a suggestion. Rule three was later removed from the classifier due to decreased performance as discussed in the results section.

4.1.2 Naïve Bayes
The Naïve Bayes classifier was formed using a train-and-test split of 90-10%. The functionality for creating this classifier is provided by libraries from Scikit-learn [23]. A pipeline of transformers was used to facilitate the construction of the classifier. These transformers include a count vectorizer that converted a collection of text documents to a matrix of token counts for future use. Then the count matrix was transformed into a normalized tf-idf representation to form the linguistic features. These fractional counts from the tf-idf representation then enabled the naïve Bayes classifier to fit the model. As previously noted, stemming the data did result in a slight increase in classifier performance, therefore composing part of the preprocessing for the final model.

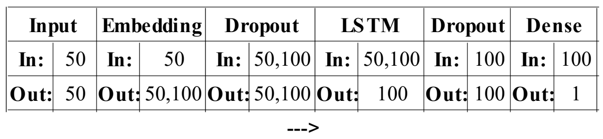
4.1.3 Support Vector Machine
The support vector machine classifier was formed using a similar approach to that of the naïve Bayes classifier. The train-and-test split was 90-10%. The same count vectorizer and tf-idf transformer were used to prepare the data as with the naïve Bayes model. Scikit-learn’s [23] stochastic gradient descent training for a linear model was then used to form the support vector machine classifier.

4.1.4 Tools: Traditional Machine Learning and Rule-based
The Python natural language toolkit [13] was used for tokenization and part-of-speech tagging to prepare the text for analysis by the rule-based classifier. Stop-word removal from NLTK was attempted in the rule-based and naïve Bayes classifier, though later abandoned due to the exclusion of vital words for finding patterns for suggestions, which decreased classifier performance. The Python library Pandas [24] was used to read the dataset into a data frame for reference within the program. Scikit-learn [23] was also used to calculate and present the resulting F1-score for the classifiers. Scikit-learn provided feature extraction and linguistic-feature processing for the naïve Bayes and support vector machine classifiers. The Scitkit-learn functionality utilized included a count-vectorizer and tf-idf transformer to prepare the data for classification by these two classifiers.

4.2 Artificial Neural-Network Methods
Neural networks are a state-of-the-art method when it comes to classification. They generally tend to perform well across domains as well as across different types of data such as images, text, and speech. The core idea behind this work is to convert the task of detecting suggestions into a text-classification task and using neural networks to perform this classification. Neural networks work well for this kind of text classification. This is especially true for any kind of recurrent model, such as RNN or LSTM. Recurrent models are widely used for processing any kind of sequential data such as text, speech, and patterns. This is due to their inherent ability to process data in a sequential manner, one step of the sequence at a time. Thus, they are an ideal choice for tasks involving text, such as text classification and text generation.

In this paper, we have focused on utilizing recurrent models to perform text classification. More specifically, we focus on LSTMs and bi-directional LSTMs to achieve the best performance. In addition to the aforementioned recurrent models, we also explore the feasibility of using convolutional neural networks (CNNs) to perform the same tasks. CNNs have traditionally been used to deal with images, but there has been a recent trend toward using them for text-classification tasks. CNNs are not as optimized as recurrent models are for inputs in the form of a sequence, but they offer other benefits such as significantly improved model training times. CNNs can be used either on their own or in conjunction with recurrent models such as LSTMs. We implemented both variants. As is the norm with applying CNN’s to text classification tasks, we used a 1-D CNN everywhere in this paper.

4.2.1 LSTM
LSTM networks are a type of RNNs (Recurrent Neural Networks). They improve upon RNNs by avoiding their single biggest pitfall, the inability to capture long-term dependencies. They incorporate memory into the network in the form of a cell state, thus allowing for relevant information to be retained for long periods of time. LSTMs are preceded by word embeddings, and these embeddings may be pre-trained or not. In this work, we used pre-trained word embeddings only with the Bi-LSTM model which will be discussed next. We do not use pre-trained word embeddings with the “vanilla” LSTM model. For the vanilla LSTM model, we use the embeddings trained as a part of the Keras [25] embedding layer. We use an LSTM of size 100 (100 hidden units). We use dropout as a regularization mechanism to try to combat overfitting. Finally, we use a sigmoid layer to make the predictions.



Input		Embedding		Dropout		LSTM		Dropout		Dense	
In:	50	**In:**	50	**In:**	50,100	**In:**	50,100	**In:**	100	**In:**	100
Out:	50	**Out:**	50,100	**Out:**	50,100	**Out:**	100	**Out:**	100	**Out:**	1

--->

Figure 1: LSTM Architecture

Figure 1 shows the LSTM architecture that we implemented with a detailed description of its various layers and their respective shapes in Keras.

4.2.2 Bidirectional LSTM

Bidirectional LSTMs are an extension of LSTMs. A drawback of Vanilla LSTMs is that they can learn representations only from the previous time steps. This means that when processing a given word, the model only has access to all the words that came before this word. This way, the model might lose out on a lot of valuable and relevant information that might be present after the word that is currently being processed. In order to combat this problem, we need to be able to look ahead of the current word. A Bidirectional LSTM has the ability to use words both preceding and following the word being processed. This should give it an edge over a vanilla LSTM. We use the pre-trained Glove embeddings [26] in tandem with the bidirectional LSTM. We chose a Glove embedding that generates a 300-dimensional vector embedding for every word. The size of this bidirectional LSTM is 150 (150 hidden units). As with the vanilla LSTM, we use dropout for regularization and sigmoid as the final classification layer.

Input		Embedding		Dropout		BiLSTM		Dropout		Dense	
In:	50	In:	50	In:	50,300	In:	50,300	In:	300	In:	300
Out:	50	Out:	50,300	Out:	50,300	Out:	300	Out:	300	Out:	1

--->

Figure 2: Bi-LSTM Architecture

Figure 2 shows the Bidirectional LSTM architecture that was implemented with a detailed description of its various layers and their respective shapes in Keras [25].

4.2.3 CNN

CNNs are a type of feed-forward deep neural network that is primarily applied to data in the form of images. They require minimal pre-processing and are shift invariant. They are used in the domains of image and video recognition, recommender systems and, more recently, NLP. They have already achieved human-level performance on image recognition and have recently made the transition to text classification, where they have been shown to work surprisingly well. CNNs work well with images because they preserve the 2D spatial orientation of the input image. In contrast, texts have a 1D orientation, wherein the sequence of the input words matter. So, we use a 1D CNN to be able to capture this orientation in text. Here we utilize a 1D CNN, followed by a dense layer of size 100. We continue to use sigmoid for the final classification.

Input		Embedding		Dropout		Conv1D		GlobalMax Pooling		Dense		Dropout		Dense	
In:	50	In:	50	In:	50,100	In:	50,100	In:	48,100	In:	100	In:	100	In:	100
Out:	50	Out:	50,100	Out:	50,100	Out:	48,100	Out:	100	Out:	100	Out:	100	Out:	1

--->

Figure 3: CNN Architecture

Input		Embedding		Dropout		Conv1D		MaxPooling		LSTM		Dropout		Dense	
In:	50	In:	50	In:	50,100	In:	50,100	In:	46,50	In:	11,50	In:	100	In:	100
Out:	50	Out:	50,100	Out:	50,100	Out:	46,50	Out:	11,50	Out:	100	Out:	100	Out:	1

--->

Figure 4: CNN+LSTM Architecture

Figure 3 shows the CNN architecture that is implemented in this paper with a detailed description of its various layers and their respective shapes in Keras [25].

4.2.4 CNN + LSTM

It is interesting to combine the CNN and the LSTM models. LSTM models are very well suited for text classification, but due to their inherent design and sequential processing, they take a long time to train. CNNs, on the other hand, are highly parallel in nature and process the entire data at once. This allows for very short training times, especially when compared to an LSTM. To avoid this shortcoming of LSTMs, we add a convolutional layer before the LSTM layer. This convolutional layer passes a filter over the input text and generates a high-level representation of the input. This high-level representation is then fed to the LSTM instead of the word embeddings. This has a significant positive effect on training times of the LSTM model. Thus we can achieve the best of both worlds by following a CNN with an LSTM, and reducing the training times of the LSTM while retaining its sequence-processing abilities. In this paper, we use a 1D CNN followed by an LSTM of size 100 (100 hidden units). As is the case with all other models, we stick with dropout for regularization and sigmoid for the final classification layer.

Figure 4 shows the CNN+LSTM architecture that is implemented in this paper with a detailed description of its various layers and their respective shapes in Keras [25].

4.2.5 Tools: Artificial Neural Network

Keras was the deep learning framework of choice that was used to implement the various neural network models. Scikit-learn's classification report was used to generate metrics including precision, recall, and F1-Score. The inbuilt tokenizer from Keras was used to tokenize the peer reviews before feeding them to the neural network models.

For each of these networks, a series of hyper-parameters are isolated and tuned, which include parameters such as input batch size, number of memory states, recurrent dropout rate, dropout rate between layers, padding rules, CNN window size and stride size, activation methods, number of epochs to train. Based on classification accuracy on the validation dataset, we have tuned the network into their respective final stages and have achieved results described in the next section.

5. RESULTS

Table 2 displays the total number of suggestions present in the peer review dataset used for this experiment, along with a summary of the weighted average F1-scores. Dataset reformatting was accomplished prior to this so that the number of suggestions to non-suggestions resulted in a mostly equal proportion within the dataset. Abbreviations for the tables are as follows: Rule (as itself), NB (Naïve Bayes), SVM (Support Vector Machine), and N1-N4 as the neural network classifiers (LSTM, BiLSTM, CNN, and CNN+LSTM respectively).

From the results obtained by testing on the Peer Review dataset, we found that the LSTM, CNN, and CNN+LSTM neural network architectures outperform the other classifiers used in this experiment. This relates to our discussion in the literature section where the neural network approaches have demonstrated the greatest potential

Table 2: Accuracy of classifiers on peer-review dataset

# of comments		3878
# containing suggestions.		1939
F1 score	Rule	0.80
	NB	0.84
	SVM	0.88
	N1	0.91
	N2	0.93
	N3	0.92
	N4	0.90

for text classification tasks like suggestion detection. The BiLSTM classifier was the second best at detecting suggestions, followed by the support vector machine, rule-based, and naïve Bayes classifiers. Tables 3 and 4 display the extended breakdown of the results of the classifiers.

These results show precision, recall, and F1-score as generated by the classification report function by Scikit-learn. The number of data points that are false and true is noted in the Support column. The rates of classification for both positives and negatives are included in the table as False and True, along with the support for each. Finally, the weighted average was used to demonstrate the classifier's overall effectiveness by combining the metrics for positives and negatives using a weighting based upon the quantity in Support.

The weighted average is calculated by taking the mean of the false and true predictions by their relevance in the dataset. In terms of these results tables, the weighted average is determined by multiplying the metric (precision, recall, or F1-score) of the false predictions by the percentage of these predictions over the whole dataset. The same calculation is made for the true predictions. Then the value calculated by the False row is added to the value calculated for the True row.

The classifiers performed similarly on overall accuracy, with some deviation in the performance for False and True observations. The F1-scores of the classifiers ranged from 80% up to 93%, where the neural network classifiers were close in performance. Rule 3 of the rule-based classifier was removed due to the tagging of some words as NNS or NNP that were not indicative of a suggestion. This resulted in numerous false positive classifications and therefore a net loss of overall performance. Rule 2 can also have result in a high false-positive classification rate since it does not consider the context of a tagged pair of words within a sentence, although it was the most effective rule at finding suggestions.

6. DISCUSSION AND FUTURE WORK

In this study, we have compared the performance of a few neural network classifiers against a rule-based classifier on suggestion detection towards multiple datasets. We found that neural network classifiers outperformed existing rule-based and traditional machine learning classifiers across the board.

Despite using a small dataset, we were able to obtain F1 scores in the low 90% range. This demonstrates the potential of the neural network classifiers. When large-scale datasets are obtained, classification scores in the upper 90% range will be probable, therefore reaching the high accuracy levels that some sentiment analysis models have obtained.

One of the biggest challenges we've encountered in this study is the insufficient amount of readily labeled data in hand. Most of the times we would need to manually label datasets that we acquired. Larger datasets would help with tuning and training the classifier. They would also give us the opportunity to train on more balanced sets of data, while not excluding too large of a portion of the overall dataset. Larger training samples would help alleviate the concern that the models may not be able to generalize to new input strings that aren't similar enough to the current training set. However, as noted in Section 2 we removed around 7000 duplicate observations from the dataset, indicating that there are commonly occurring review comments that the model would be tuned for. Additional pattern-matching phrases can be added to the rule-based classifier to improve its accuracy, although additional conditions would be required to consider the context of the phrases within a larger body of text. Also, the inclusion of more types of classifiers such as a decision tree can provide an extended comparison of what classifiers perform best for this text classification task.

As discussed in the introduction, the domain of primary interest revolved around peer reviews. Later on, we might find a way to let instructors use the classifier to grade the quality of peer reviews. We would seek feedback on the effectiveness of the classifier, and suggestions for its improvement. This system could also notify the reviewer that a review should have more

Table 3: Breakdown of accuracy of non-neural network classifiers on peer review

Peer review	Precision			Recall			F1 score			Support		
	Rule	NB	SVM	Rule	NB	SVM	Rule	NB	SVM	Rule	NB	SVM
False	0.76	0.87	0.87	0.88	0.81	0.90	0.82	0.84	0.89	1939	205	205
True	0.86	0.80	0.89	0.73	0.87	0.85	0.79	0.83	0.87	1939	183	183
Avg.	0.81	0.84	0.88	0.80	0.84	0.88	0.80	0.84	0.88	3878	388	388

Table 4: Breakdown of accuracy of neural-network classifiers on peer review

Peer review	Precision				Recall				F1 score				Support
	N1	N2	N3	N4	N1	N2	N3	N4	N1	N2	N3	N4	All
False	0.90	0.94	0.90	0.90	0.92	0.93	0.94	0.90	0.91	0.93	0.92	0.90	189
True	0.92	0.93	0.94	0.90	0.90	0.94	0.90	0.90	0.91	0.93	0.92	0.90	199
Avg.	0.91	0.93	0.92	0.90	0.91	0.93	0.92	0.90	0.91	0.93	0.92	0.90	388

suggestions. Furthermore, the reviewer could have the opportunity to identify any suggestions that were missed by the system. This would function as a useful method for generating labeled training data. These missed suggestions indicated by the reviewer could enable analysis in the form of determining what types of sentences are not being properly detected as suggestions. This system could be deployed as a supplement to instructor grading until enough data has been obtained to train a sufficiently accurate classifier for automatic suggestion extraction.

7. CONCLUSION

In this paper, we have demonstrated several models that parse and classify suggestions. The greater performance of neural network architectures over rule-based methods in this task demonstrates the advantage of classifiers that train on text in a certain domain, rather than following strict rules for classification. Furthermore, forming an extensive list of phrases and keywords that improve a rule-based classifier's performance in a domain is more time consuming than training a statistical classifier. While rule-based approaches have the advantage of being simpler to implement and not requiring data to train on, they are typically outperformed by statistical classifiers provided they have access to enough labeled data.

8. ACKNOWLEDGMENTS

The authors acknowledge the support of the National Science Foundation, through grant #1432347.

9. REFERENCES

[1] Topping, K.J. 2009. Peer Assessment. Theory into practice. 48, 1 (Jan. 2009), 20–27.

[2] Nelson, M.M. and Schunn, C.D. 2009. The nature of feedback: how different types of peer feedback affect writing performance. Instructional Science. 37, 4 (Jul. 2009), 375–401.

[3] Demiraslan Çevik, Y. et al. 2015. The effect of peer assessment on problem solving skills of prospective teachers supported by online learning activities. Studies in Educational Evaluation. 44, (Mar. 2015), 23–35.

[4] Liu, X. and Li, L. 2014. Assessment training effects on student assessment skills and task performance in a technology-facilitated peer assessment. Assessment & Evaluation in Higher Education. 39, 3 (Apr. 2014), 275–292.

[5] Lundstrom, K. and Baker, W. 2009. To give is better than to receive: The benefits of peer review to the reviewer's own writing. Journal of Second Language Writing. 18, 1 (Mar. 2009), 30–43.

[6] moocs peer to peer: https://docs.google.com/presentation/d/1wkJOFgH0DYihsfH_tddcYeI5DMNl9uP1qYgHMPRf_UY/edit. Accessed: 2018-10-01.

[7] Van Popta, E. et al. 2017. Exploring the value of peer feedback in online learning for the provider. Educational Research Review. 20, (Feb. 2017), 24–34.

[8] Tsivitanidou, O.E. and Constantinou, C.P. 2016. A study of students' heuristics and strategy patterns in web-based reciprocal peer assessment for science learning. The Internet and Higher Education. 29, (Apr. 2016), 12–22.

[9] Ramachandran, L. et al. 2017. Automated Assessment of the Quality of Peer Reviews using Natural Language Processing Techniques. International Journal of Artificial Intelligence in Education. 27, 3 (Sep. 2017), 534–581.

[10] Xiong, W., & Litman, D. 2011, Automatically predicting peer-review helpfulness. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2 (pp. 502-507). Association for Computational Linguistics.

[11] Negi, S. and Buitelaar, P. 2017. Chapter 8 - Suggestion Mining From Opinionated Text. Sentiment Analysis in Social Networks. F.A. Pozzi et al., eds. Morgan Kaufmann. 129–139.

[12] Austin, J. L. 1962. How to do things with words. Cambridge: Harvard University Press.

[13] Bird, S. et al. 2009. Natural Language Processing with Python. O'Reilly Media, Inc.

[14] Brun, C. and Hagège, C. 2013. Suggestion Mining: Detecting Suggestions for Improvement in Users' Comments. Research in Computing Science. 70, 79.7179 (2013), 5379–5362.

[15] Marcus, M.P. et al. 1993. Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics. (1993).

[16] Murphy, K.P. 2012. Machine Learning : A Probabilistic Perspective. MIT Press.

[17] Gottipati, S. et al. 2018. Text analytics approach to extract course improvement suggestions from students' feedback. Research and Practice in Technology Enhanced Learning. 13, 1 (Jun. 2018), 6.

[18] Gehringer, E. et al. 2007. "Reusable learning objects through peer review: The Expertiza approach," Innovate—Journal of Online Education 3:6

[19] Gehringer, E. et al. 2006. "Expertiza: Reusable learning objects and active learning for distance education," Proceedings of the UNC Teaching and Learning with Technology conference, Raleigh

[20] Krippendorff, Klaus. "Computing Krippendorff's alpha-reliability." (2011).

[21] Menardi, G. and Torelli, N. 2014. Training and assessing classification rules with imbalanced data. Data mining and knowledge discovery. 28, 1 (Jan. 2014), 92–122.

[22] Kubat, M. et al. 1997. Learning when negative examples abound. Machine Learning: ECML-97. M. Somerenand G. Widmer, eds. Springer Berlin Heidelberg. 146–153.

[23] Pedregosa, F. et al. 2011 "Scikit-learn: Machine learning in Python." Journal of machine learning research: 2825-2830.

[24] Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)

[25] François Chollet. Keras: Deep learning library for theano and tensorflow. https://github.com/keras-team/keras, 2015.

[26] Pennington, J. et al. 2014. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)