Syntactic and semantic features for human like judgement in spoken CALL

Ahmed Magooda[1], *Diane Litman*[2]

[1]University of Pittsburgh, PA, USA
[2]University of Pittsburgh, PA, USA

amagooda@cs.pitt.edu, litman@cs.pitt.edu

Abstract

Educational applications of Natural Language Processing (NLP) and Automatic Speech Recognition (ASR) have included providing learners with helpful and accurate feedback. In this paper we present a system that takes a first step towards providing feedback during spoken Computer-Assisted Language Learning (spokenCALL). We propose a machine learning based approach that combines syntactic and semantic features in order to accept or reject a textual response given a provided prompt. Our approach was evaluated as part of the SpokenCALL shared task, ranking third place among the submitted systems and outperforming the provided baselines.

Index Terms: Natural Language Processing, Language modeling, Word embedding, Machine learning

1. Introduction

Any educational application has to deal with learner answer/response variability, and spoken Computer-Assisted Language Learning (spokenCALL) is no exception. For any system question or prompt, it is expected that learners will not all provide the exact same answer/response, and it is the system's responsibility to deal with this variability. To provide accurate feedback, a system should thus be able to tolerate different language aspects (e.g., word synonymy, paraphrasing) in learner responses. The process of giving response feedback can be done in multiple stages (e.g., accept/reject, highlight errors, propose more accurate answers). In this paper we focus on the first stage (accept/reject), since the first stage has to be the most accurate. It is meaningless for a system to highlight errors or propose corrections if the answer/response is correct to begin with, while it can be less severe to just reject a wrong answer/response without proposing any corrections.

In this context, this paper targets the task of providing accept/reject feedback for data collected from CALL-SLT [1], a speech-enabled Computer-Assisted Language Learning application that is based on prompts and associated responses. The application was developed to help Swiss German teens practice English conversation. A prompt is a piece of German text that is introduced to the learner, while the response is an English response recorded using an audio capturing device. The response is supposed to be an English sentence that follows the German prompt request. Based on data collected from users of the CALL-SLT tool, Baur et al. [2] proposed the SpokenCALL shared task[1]. Given a German prompt and an English response, the shared task is to accept linguistically correct and meaningful responses and to reject incorrect responses. The shared task consists of both a text track and a speech track. The speech track uses the recorded response as an input, and based on this input, a system submitted to this track should accept or reject the response. In contrast, the text track uses text responses generated from two state of the art speech recognition systems (Kaldi [3] and Nuance [4]). Systems participating in the text track can use the output text of either of these speech recognition systems, rather than the originally recorded audio, to accept or reject the response. In this paper we target the text track, where we are supposed to give accept or reject feedback for a text response, based on meaning and language quality.

This paper describes three systems we developed and incorporated in this shared task, where our best system achieved the 3^{rd} position in the final evaluation. Our classifiers were developed by using different machine learning techniques to combine syntactic and semantic features extracted from learner responses. The paper is organized as follows: Section 2 presents related work, Section 3 describes the dataset, Section 4 details our proposed approaches, Section 5 summarizes the experiments, Section 6 presents the results, and Section 7 concludes and suggests future work.

2. Related work

Accepting or rejecting a targeted response based on both language quality and meaning can be seen as a hybrid of two tasks: 1- error detection. 2- similarity measurement. It is not enough to just detect grammatical errors, since a meaningless sentence can be correct grammatically. Conversely, detecting similarity is also not enough, as some tasks require grammatically sound responses. We thus tackle the shared task by combining prior work in error detection and similarity measurement.

Multiple shared tasks have previously targeted the problem of error detection and correction (CONLL 2013 [5], CONLL 2014 [6], HOO 2011 [7] and HOO 2012 [8]). Sometimes the task is to only correct errors, while other times the task is divided into error detection and correction. A variety of approaches have been based on classification. Cahill et al. [9] used a classification model based on logistic regression to develop a system for error detection. Cahill et al. mined Wikipedia revisions to produce a large error-annotated training corpus, and evaluated the system using other publicly available datasets (HOO 2011 [7] and FCE [10]). Another system that used a classification-based approach for error detection and correction is the system proposed by Rozovskaya et al. [11, 12]. On the other hand, Gamon [13] argued that using language modeling outperforms classification-based models, and also proposed combining language modeling and classification for error detection and correction. As the task we are dealing with targets English learners, responses are expected to be very basic short English sentences and very limited in vocabulary. Due to these aspects, we decided to follow Gamon and adopt language modeling as the core approach for our error detection. Moreover, we adopted Cahill et al.'s idea of using external error resources.

With respect to detecting text similarity, different approaches have also been introduced. Šarić et al. [14] used a

[1]https://regulus.unige.ch/spokencallsharedtask/

combination of unigram, bigram and trigram overlap features in addition to other corpus and knowledge-based similarity measures to detect similarity. Magooda et al. [15] combined different word embedding models to measure sentence similarity, where sentences are represented using summation of word vectors and similarity is measured using cosine. In our work we use machine learning to combine features based on this prior work in measuring similarity with features based on the error detection work described above.

3. Dataset

The dataset we used during this work is the dataset introduced by the shared task competition committee, which consists of annotated training data as well as test data in which the annotations were hidden until the end of the competition. Table 1 shows the number of samples in the training and test sets. Three native English speakers independently annotated the data along two dimensions (Language and Meaning). Table 2 and Table 3 show the annotation distribution over both training and testing data, respectively. Each training sample in the training set consists of the following:

- Prompt.
- Response "result of speech recognition".
- Human transcript of audio response.
- Language Annotation.
- Meaning Annotation.

While a transcript is provided for the training samples, no transcript is provided for the test samples. Without transcription and with errors due to speech recognition, the text track was quite challenging. Another useful resource that was publicly available for use was a sample XML grammar file that has a set of possible correct responses for each of the prompts in either the training or test sets.

Table 1: *Training and testing samples*

Data	**Number of samples**
Training	5222
Test	996

Table 2: *Training set annotations*

		Language	
		Correct	Incorrect
Meaning	Correct	3880	802
	Incorrect	0	540

Table 3: *Test set annotations*

		Language	
		Correct	Incorrect
Meaning	Correct	716	159
	Incorrect	0	121

4. Proposed approaches

This section details the three systems we submitted to the competition. The systems are based mainly on machine learning techniques and combine a set of syntactic and semantic features. To introduce the systems in detail, we will divide our description into 3 parts: feature extraction, feature selection, and classification.

4.1. Feature extraction

During our system development we extracted a wide set of features, which can be divided into two general types:

- Features used to detect inconsistency on the language level (spelling mistakes, part of speech score, language modeling).
- Features used to detect inconsistency on the meaning level (syntactic relatedness, semantic relatedness).

The intuition behind this separation is mimicking the annotation process. Since samples are annotated based on two criteria, we decided to adopt the same criteria for feature engineering. We employed both the idea of using language modeling [13] combined with the idea of using external error resources [9]. We also adopted some features to detect similarity within sentences [14, 15].

4.1.1. Language-related features

To detect inconsistency on the language level, we use three types of features to differentiate between sound English responses, and responses that have grammatical mistakes or that make no sense regardless of the response's relatedness to the prompt.

Spelling mistakes (F1). To capture the presence of spelling mistakes, we use a binary feature (which we will refer to as F1) indicating if any spelling mistakes exist in the response or not. To check for spelling mistakes we use the NLTK English spell checker [16].

Part of speech score (F2). A score is given to each sentence by a part of speech (POS) tagger and represents how likely the sentence is to be a real sentence. For the tagging process we use the Stanford part of speech tagger [17].

Language modeling (F3-F10). To extract these features we use multiple language models (LMs). All language models are 5-gram language models trained using the SRILM language modeling toolkit [18]. Since the amount of data we have for this task is not much, it was easy to go beyond trigrams without any noticeable time complexity. We tried multiple values of n for training n-gram models and decided to use 5-grams as they achieved the lowest perplexity over the training data. We also found that going beyond 5-grams didn't yield any improvement in perplexity. We used interpolation and the unknown tag ("unk") to deal with out of vocabulary and sparsity issues. While it is not a best practice to evaluate models using training data, we did that due to the lack of data. To distinguish between linguistically sound and unsound responses at an abstract level, we then trained two types of 5-gram language models:

- Models trained using correct English sentences
- Models trained using incorrect English sentences

The intuition behind using these two different types of language models is that incorrect sentences are more likely to get higher probability from language models trained on incorrect

sentences compared to language models trained on correct sentences. The same is plausible for correct sentences, which are more likely to get higher probability from language models trained using correct sentences.

Specifically, within each of these two types, we train four different language models. Two of the four models are trained using words, while the other two are trained using part of speech tagged versions of the same data. For the language models based on correct sentences, word and POS models are trained from the following two sources of data:

- F3, F4: Speaker responses "Transcripts" annotated as correct on the language level.
- F5, F6: All sample responses from the grammar XML. Since these are proposed answers, they are guaranteed to be correct English sentences.

For the language models based on incorrect sentences, the word and POS models are instead trained on the following two data sources:

- F7, F8: Speaker responses "Transcripts" annotated as incorrect on the language level.
- F9, F10: Incorrect sentences collected from two previous error detection and correction shared tasks (HOO 2011 [7] and CONLL 2014 [6]). These shared tasks provided datasets of English sentences with linguistic errors, where participants were asked to detect and propose a correction for these errors. These prior shared tasks yielded a total of 2029 different sentences.

4.1.2. Meaning-related features

The other set of features we use are meant to capture relatedness between the prompt and the speaker response. These features aim to detect both syntactic and semantic relatedness and are extracted using word matching, language modeling, and word embeddings.

Syntactic relatedness (F11-F12). To capture syntactic relatedness between prompt and speaker response, we use the sample responses for each prompt that were provided in the grammar XML file and employ the concepts of n-gram matching and language modeling. We decided to use language models trained per prompt, as using such language models should assign high probabilities to similar responses. Moreover, we decided to count the number of n-gram matches as this should also capture a degree of relatedness. In more detail, the syntactic relatedness features we use are as follows;

- F11: Number of matching unigrams, bigrams and trigrams. For a prompt-response pair, we calculate the number of matching unigrams bigrams and trigrams between a speaker response and all the sample responses from the grammar XML file for the prompt of concern. The numbers are then averaged over the number of sample responses.
- F12: Language modeling. For a prompt-response pair, we train a language model using the sample responses from the grammar XML file for the prompt of concern. Once we have a language model trained using the sample responses, we can calculate the probability of the speaker response. The trained language model is expected to assign high probability to responses that are syntacticly similar to the sample responses.

Semantic relatedness (F13-F16). To capture semantic relatedness, we decided to use the concept of word embeddings. Word embedding is based on representing words with vectors in high dimensional space, where each dimension of the generated space can hold a semantic or a syntactic feature. This high dimensionality representation of words can be utilized to measure semantic relatedness between words or sentences, using a distance measure like cosine. Two very popular models of word embeddings (skip-gram, continuous bag of words) were developed by Mikolov et al. in [19]. These two models have a structure similar to neural networks while using log linear classifiers as the core of the model. The parameters of the trained log linear classifiers are used as word embeddings. Continuous bag of words and skip-gram models are trained differently. The first is trained to predict word given context while the second is trained the other way around to predict context given a pivot word. For a prompt-response pair, we train multiple word embedding models using both skip-gram and continuous bag of words (CBOW) algorithms [20, 19]. We train the models per prompt, and for each prompt we train multiple models over the sample responses for the prompt of concern. In particular, for each prompt we train the following models:

- F13: Skip-gram model (50 dimensions and negative sampling [20])
- F14: Skip-gram model (30 dimensions and negative sampling)
- F15: Continuous bag of words model (50 dimensions and negative sampling)
- F16: Continuous bag of words model (30 dimensions and negative sampling)

We decided to train our models instead of using any of the already trained models like the Google News skip-gram model [19]. Since the data we have uses only basic English vocabulary and short simple sentences, tailoring models only on these data can capture relatedness between words that are specific for this data, which may not be found in news data like the ones used for the Google News model. As the data we have is not much, we trained models with a small number of dimensions (30 and 50 dimensions) compared to the one trained on Google News (300 dimensions). To use each of the word embedding models, cosine similarity is measured between the speaker response vector representation and the vector representation of each of the sample responses. We used cosine to be consistent with the word embeddings training objective. Since these word embeddings are trained to maximize the cosine similarity between classification output and expected word, we think it is better to follow the same training objective. Additionally, following Mikolov et al. [20] in employing the additive compositionality property of the word embeddings, a sentence vector is formed using the summation of the constructing words' vectors. The final cosine similarity is the maximum of all the cosine values calculated between response and sample responses. In this context, we tried using average cosine, maximum and minimum; selecting maximum cosine got the best results during validation.

Sentence ratio (F17). This final feature is neither semantic or syntactic based, this feature is used to make sure responses have a reasonable length. Sentence ratio, is the ratio between the length of speaker response, and the average length of sample responses assigned to that specific prompt. As the prompts are asking for very basic English responses, it is safe to assume that responses are expected to have almost the same length. This feature can capture responses that are not reasonable, responses that are too short or too long.

4.2. Feature selection

After extracting the features we discussed in the previous section, we performed feature selection using PCA to select a subset of the features that can achieve the best accuracy. To select features using PCA, we generated two versions of the features, one version with the raw values, the other with a normalized version of the values to be between [-1, 1]. Tables 4 and 5 show the set of features selected using raw and normalized values, respectively.

Table 4: *Selected features using PCA with "raw values" (feature set 1)*

	Features
Language	Part of speech sentence score (F2)
	LM trained on correct responses (F3)
	LM trained on incorrect responses (F7)
	LM trained on incorrect sentences (F9)
	LM trained on incorrect sentences POS (F10)
Meaning	LM trained on sample responses (F12)
	Skip-gram 50 dimensions (F13)
	Sentence ratio (F17)

Table 5: *Selected features using PCA with "normalized values" (feature set 2)*

	Features
Language	LM trained on correct responses (F3)
	LM trained on incorrect responses (F7)
Meaning	Unigram, bigram, trigram matches (F11)
	Skip-gram 30 dimensions (F14)
	CBOW 50 dimensions (F15)
	CBOW 30 dimensions (F16)

4.3. Classification

For the classification, we tried two different widely used machine learning classifiers:

- K-nearest neighbor (KNN) [21].
- Support vector machines (SVM) [22].

We trained SVM models using the normalized set of features. SVM works best with normalized data because it avoids prior whitening for the dimensions, which in turn tries to avoid dimension domination. On the other hand we trained KNN models using the raw set of features. KNN depends on pure Euclidean distance between samples, so using raw values can produce a wider spectrum of distances between samples, while normalizing can end up having very small differences in distances due to the tightened values allowed. That is why we decided to use the raw feature values with KNN and normalized versions with SVM. The next section will describe the experiments and tuning carried out to use both classifiers.

5. Experiments

To select the best performing classifier and hyper parameters, we performed multiple experiments to train and validate both classifiers. To train and tune both classifiers, we used 10 fold cross validation over the training data provided by the shared task. However for the 10 folds we used 3 different data splitting paradigms.

- Fixed data splitting, where data is split into 10 parts based on the order of training samples in the file.
- Random data splitting, where data is randomly split into 10 parts. The accuracy of random splitting is calculated by averaging the score of performing 10 iterations, of random 10 fold cross validation.
- Per prompt data splitting, where we split the responses for each prompt into 10 parts. In this paradigm we are making sure that responses for the same prompt appear in training and validation folds.

For KNN we tuned the K (number of neighbors) value, and for SVM we varied the cost, gamma and the kernel. We selected the parameters that maximized the D-scores (equation 1 & 2) over the 3 splitting paradigms.

$$D = \frac{C_R(F_R + C_A)}{F_R(C_R + F_A)} \tag{1}$$

$$F_A = PF_A + k.GF_A \tag{2}$$

where:

- C_A = Correct Accept, the student's answer is correct, the system accepts.
- C_R = Correct Reject, the student's answer is incorrect, the system rejects.
- PF_A = Plain False Accept, the student's answer is correct in meaning but incorrect English, the system accepts.
- GF_A = Gross False Accept, the student's answer is incorrect in meaning, the system accepts.
- F_R = False Reject, the student's answer is correct, the system rejects.
- k = A weighting factor, default value is 3

Table 6: *Accuracy over training data using 10 fold cross validation*

Features		**D-Score**		
		Fixed	Random	Prompt
Set 1 : Raw	KNN	23.034	18.758	17.981
Set 2 : Normalized	SVM	13.101	18.206	16.491
Combined : Raw	KNN	26.354	18.683	17.707
Baseline	2.06			

Table 6 summarizes the results obtained by using 10 fold validation on the training data. Raw Features are the features selected by PCA using the raw feature values, while normalized are the features selected using normalized values. The combined features are the feature set selected from normalized values and the feature set selected from raw values, but all values then kept without normalization. We submitted three systems to the shared task competition, where each system corresponds to one of the entries in Table 6. First entry corresponds to using KNN as a classifier with features set 1 "raw values", entry

2 corresponds to using SVM as a classifier with features set 2 "normalized values", while finally entry 3 corresponds to using KNN as classifier with features set 1 and set 2 combined without normalization "raw values". The next section will present the results achieved for each of these submissions over the test set.

6. Results

Out of the submitted 20 entries made by 9 teams, one of the entries we submitted achieved the 3^{rd} position. Table 7 shows the results of the highest 10 submissions, sorted by D-score over the test data. The results are anonymous, however our submissions are underlined in the table. Besides showing the scores, the table shows which speech recognition system output each submission used for training and testing, where custom means that the team used their own developed speech recognition system and participated in the speech track. Thus, beside achieving 3^{rd} in both tracks combined, we achieved 1^{st} place in the text track.

Note that in contrast to the results we achieved on the training data, our submissions came in reverse order on the test data. That is, the system that got the lowest score on training data (Set 2 - Normalized - SVM) got the highest on test and vice versa, the system that got the highest score on training data (Combined - Raw - KNN) got the lowest on test data. One thing that comes to mind here is that KNN classifiers are more liable to overfit than SVM ones. Another finding that actually agrees with our intuition is as follows. Since D-score penalizes meaning errors more than language ones, we thought that using more features to capture meaning errors would get us higher results. This intuition actually proved to be right on test data, but didn't prove to be correct on training data, which in turn can weight the scale towards hypothesizing there was an overfitting issue for KNN classifiers.

It is also worth mentioning that all the systems that came in the first 10 positions which aren't using their own developed speech recognition systems are using the Kaldi system, although Nuance's baseline achieves a marginally higher score than Kaldi's baseline. Due to this contradiction, after the competition we tried our 3 set of features in further runs, once using Nuance system output, and the other time using output of the speech recognition system developed by the 1^{st} place team. As the 1^{st} team only released the output of their speech recognition system of test data, we are not able to train a new model using their speech recognition system output and instead used the same Kaldi model we trained. In addition, we trained a new model using the Nuance system output. Using these two models we evaluated two versions of the test data, one using the Nuance version of the test data, the other using the 1^{st} team's version. Table 8 summarizes the results. Comparing to Table 7 we see that using the Nuance system instead of Kaldi really got us lower scores, which is somehow consistent with scores from other teams. We also didn't get any score enhancement by using the 1^{st} team's version of test data. This degraded performance is somehow expected, because we are training and testing using two different speech recognition systems. This conclusion agrees with our intuition as we know that training using a specific system tailors the model to avoid the errors the system makes, so testing using another system with a different set of errors will result in degraded performance as in our case.

Table 7: *Best 10 submissions, sorted by accuracy over test data*

Rank	Speech system	D-score
1	Custom	4.766
2	Custom	4.710
Baseline Perfect Rec		4.512
3 (Set 2 : SVM : Norm.)	**Kaldi**	**4.468**
4	Custom	4.371
5	Kaldi	4.353
6	Kaldi	4.273
7	Kaldi	3.998
8	Kaldi	3.678
9	Kaldi	3.352
10 (Set 1 : KNN : Raw)	**Kaldi**	**3.335**
BaselineNuance	Nuance	2.358
BaselineKaldi	Kaldi	1.694

Table 8: *Accuracy of training and testing using systems other than Kaldi*

Training	Test	Features	D-score
Kaldi	1^{st} Team	Set 1: KNN	0.831
	1^{st} Team	Set 2: SVM	0.923
	1^{st} Team	Combined: KNN	0.807
	Nuance	Set 1: KNN	3.047
	Nuance	Set 2: SVM	2.219
	Nuance	Combined: KNN	2.301
Nuance	1^{st} Team	Set 1: KNN	0.901
	1^{st} Team	Set 2: SVM	0.751
	1^{st} Team	Combined: KNN	0.889
	Nuance	Set 1: KNN	2.281
	Nuance	Set 2: SVM	2.644
	Nuance	Combined: KNN	2.594

7. Conclusions and future work

In this paper we presented a system that uses natural language processing techniques over the output of a speech recognition system in order to provide feedback on spoken responses to a spoken CALL system. Our approach to system development used machine learning to combine syntactic and semantic features based primarily on language modeling and word embeddings, which makes it easy to develop. We provided three different configurations of the system with three different set of features; these systems were also part of the SpokenCALL shared task and our best configuration achieved the 3^{rd} position in both tracks and 1^{st} position in text track only. This result suggests that with a combination of simple and easy to develop features and a basic machine learning classification model, promis-

ing performance can be achieved. Using the data released by the shared task after the competition, we further evaluated our method by training using the output of both Kaldi and Nuance speech recognition systems. However, we hope that the first place team will make the training version of their speech recognition system available for use by others. We would like to try our same pipeline using this data as we think that better results can be achieved by enhancing the speech recognition accuracy. We would also like to further investigate the interplay of speech recognition output, natural language processing features, and machine learning algorithms.

8. Acknowledgements

We would like to thank A.Mahgoub and M.Zahran for their helpful feedback.

9. References

[1] E. Rayner, N. Tsourakis, C. Baur, P. Bouillon, and J. Gerlach, "Call-slt: A spoken call system based on grammar and speech recognition," *Linguistic Issues in Language Technology*, vol. 10, no. 2, 2014.

[2] C. Baur, J. Gerlach, E. Rayner, M. Russell, and H. Strik, "A shared task for spoken call?" 2016.

[3] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz *et al.*, "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*, no. EPFL-CONF-192584. IEEE Signal Processing Society, 2011.

[4] "https://developer.nuance.com/public/index.php?task=home."

[5] H. T. Ng, S. M. Wu, Y. Wu, C. Hadiwinoto, and J. Tetreault, "The conll-2013 shared task on grammatical error correction."

[6] H. T. Ng, S. M. Wu, T. Briscoe, C. Hadiwinoto, R. H. Susanto, and C. Bryant, "The conll-2014 shared task on grammatical error correction." in *CoNLL Shared Task*, 2014, pp. 1–14.

[7] R. Dale and A. Kilgarriff, "Helping our own: The hoo 2011 pilot shared task," in *Proceedings of the 13th European Workshop on Natural Language Generation*. Association for Computational Linguistics, 2011, pp. 242–249.

[8] R. Dale, I. Anisimoff, and G. Narroway, "Hoo 2012: A report on the preposition and determiner error correction shared task," in *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, 2012, pp. 54–62.

[9] A. Cahill, N. Madnani, J. R. Tetreault, and D. Napolitano, "Robust systems for preposition error correction using Wikipedia revisions." in *HLT-NAACL*. Citeseer, 2013, pp. 507–517.

[10] H. Yannakoudakis, T. Briscoe, and B. Medlock, "A new dataset and method for automatically grading esol texts," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 2011, pp. 180–189.

[11] A. Rozovskaya, K. W. Chang, M. Sammons, D. Roth, and N. Habash, "The illinois-columbia system in the conll-2014 shared task." in *CoNLL Shared Task*, 2014, pp. 34–42.

[12] A. Rozovskaya, K. W. Chang, M. Sammons, and D. Roth, "The university of illinois system in the conll-2013 shared task," *CoNLL-2013*, vol. 51, p. 13, 2013.

[13] M. Gamon, "Using mostly native data to correct errors in learners' writing: a meta-classifier approach," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 163–171.

[14] F. Šarić, G. Glavaš, M. Karan, J. Šnajder, and B. Dalbelo Bašić, "Takelab: Systems for measuring semantic text similarity," in *Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*. Montréal, Canada: Association for Computational Linguistics, 7-8 June 2012, pp. 441–448. [Online]. Available: http://www.aclweb.org/anthology/S12-1060

[15] A. E. Magooda, M. A. Zahran, M. Rashwan, H. M. Raafat, and M. B. Fayek, "Vector based techniques for short answer grading." in *FLAIRS Conference*, 2016, pp. 238–243.

[16] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[17] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics, 2003, pp. 173–180.

[18] A. Stolcke *et al.*, "Srilm-an extensible language modeling toolkit." in *Interspeech*, vol. 2002, 2002, p. 2002.

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

[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[22] C. C. Chang and C. J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.