CS229 Final Report - Grammatical Error Classification for Non-native English Writers

Zihuan Diao, Junjie Dong, Jiaxing Geng
Department of Computer Science and Department of Electrical Engineering
Stanford University
{diaozh, junjied, jg755} @ stanford.edu

***Abstract*—In this project, we built a machine learning system to detect and classify grammatical errors made by non-native English writers. We engineered language features based on word2vec, part-of-speech, bigram, and dependency tree. And we experimented with various machine learning models such as logistic regression, random forest, softmax regression, support vector machine, and neural network. The final model achieves a 0.36 F1-score for error detection task, and 0.71 average F1-score for error type classification task.**

***Keywords*—Machine Learning, Logistic regression, Support vector machine, random forest, neural network, Grammatical error classification, Natural language processing**

I. Introduction

For most of the English speakers today, English is not their native language. In order to help people learn English and practice their writing, a method of detecting grammatical error and giving feedback to non-native English writers is needed.

Over the past years, researchers have put in enormous efforts trying to build a good grammatical error classification and correction system. Most of the past efforts fall into three categories: rule-based methods [1], machine learning classifier based methods [2], and statistical machine translation methods [3, 4]. For the latter two approaches, deep learning especially with recurrent neural network architecture has proven to give very good performance.

In this project, we design a grammatical error detection and classification system. The system performs two tasks: detect whether there is grammatical error associated with each word in a sentence, and classify the specific error types related to each error word. Due to project complexity and dataset size consideration, we will only classify seven most common types of errors: article or determiner, noun number, verb tense, preposition, verb form, word form, and subject-verb agreement. We use the CoNLL-2014 Shared Task dataset [5] for training and evaluation, and the goal is to achieve a high F1-score for both tasks.

To further illustrate the purpose of this project, an example input-output behavior is as follows:

Input: "Moreover, more efforts are need if they want to commercialize the product from their research."
Task 1: Classify whether there is grammatical error associated with each of the 15 words separately
Output 1: 1 for word "need", 0 for all other words
Task 2: Classify the error type associated with word "need"
Output 2: 1 for class "Vt", 0 for all other six classes

The rest of the report is organized as follows: Section II describes the CoNLL-2014 dataset, data preprocessing, and feature engineering; Section III gives an overview of the machine learning methods that we experimented with; Section IV discusses our experiment process, model performance, as well as both quantitative and qualitative error analysis; finally, Section V concludes the report and sheds light on potential future work.

II. Dataset and Features

This section gives an overview of the CoNLL-2014 dataset that we used, and describes the features that we engineered.

A. Dataset Description

In this project, we utilize the CoNLL-2014 Shared Task dataset which includes 1400 English essays written by students at the the National University of Singapore. The dataset contains 60K sentences with 45K annotated grammatical errors. We reserved 10% of the data for test set, and used the rest 90% for training and validation.

B. Data Preprocessing

To preprocess the data, we first split the essays into sentences using nltk punkt. We then remove all the incomplete sentences such as essay titles. Sentences that meet any of the following conditions are also removed: include grammatical errors that are not considered in this project, consists of only hyperlinks and numbers, consists of only references and numbers. We then tokenize the sentences using the nltk tokenizer.

Because our approach to the problem requires classification for each single word, we associate each grammatical error annotated in the original dataset to a single word. For example, if phrase "I have done" is corrected to "I did", we associate the error to either "have" or "done". Based on experiment results, this degrades the model performance quite a lot, and should be addressed in future work.

C. *Word2vec Features*

Word2vec is an algorithm that maps a word to a high-dimensional dense vector representation using either continuous bag-of-words model or continuous skim-gram model [6]. It is shown that word2vec is able to preserve both semantic and syntactic relations between words. For example, the nearest token to "Paris - France + Italy" is "Rome", and the nearest token to "bigger - big + cold" is "colder". However, since there's only limited syntactic information in word2vec representations, including too many word2vec features could very likely lead to overfitting. In this project, we use pre-trained 50-dimensional GloVe word vector of the current word as an input feature [7].

D. *Part-of-speech Features*

Part-of-speech tagging is the process of marking up a word in a text as corresponding to a particular part-of-speech such as noun, verb, adverb, etc. Pos tags are essential in determining whether there is grammatical error in a sentence. Part-of-speech taggers are very likely to output weird patterns when the input sentences are incorrect ("garbage in, garbage out") [8]. Furthermore, low emission probability from the tagger and pos mismatch between different taggers are both indicators of grammatical errors. In this project, we perform pos tagging using both the Stanford NLP parser [9] and Apache OpenNLP tagger [10], where OpenNLP tagger also outputs the emission probability of the tags. We use pos tags (from both parsers) and emission probability of current word, neighboring words, and parent words as features. There is also an additional feature indicating whether the pos output by the two taggers mismatch.

E. *Part-of-speech Bigram Features*

We use all the error-free sentences to train a part-of-speech bigram model. We further process the model to output conditional probability. For example, for the phrase "I drank", we can obtain the conditional probability of the current word being "VBD" (past tense verb) conditioned on the previous word being "PRP" (personal pronoun). We reference the dependency tree of the sentences, and include several such features for each word.

F. *Dependency Tree Features*

In dependency parsing, a sentence is parsed by choosing for each word what other word it is a dependent of. We reference the dependency structure obtained from the Stanford NLP parser [9] to generate some of the part-of-speech and bigram features (eg. pos of parent word). The dependency relations between the current word and its parent/children words are also included as one-hot features.

III. MACHINE LEARNING METHODS

This section introduces the machine learning methods that we used in this project, including logistic regression, random forest, softmax regression, support vector machine, and neural network.

A. *Logistic Regression*

Logistic regression is a binary classification model based on the assumption that the data points are drawn independently from a Bernoulli distribution. It fits a linear decision boundary to the training examples, and outputs a probability between 0 and 1:

$$\hat{y} = h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

The classification result is then determined by thresholding the model output $\hat{y}$.

B. *Random Forest*

The underlying principle of random forest is that several weak learners can be combined to form a strong learner. Random forest builds several decision trees on bootstrapped (sample with replacement) samples of the original dataset, and makes a prediction by letting all the decision trees do a majority vote. In each tree, each time we split, only a subset of the predictors are considered as candidates. This makes the decision trees less similar and therefore further reduces the variance of the classifier. Random forest can model very non-linear decision boundaries, has low bias, and generalizes well to unseen data thanks to the bootstrap training and majority vote.

C. *Softmax Regression*

The softmax regression model is a multi-class classification model. It assumes that the data points are drawn independently from a multinomial distribution. It is a generalization of logistic regression, and the model outputs:

$$h_\theta(x) = \left[\frac{exp(\theta_1{}^T x)}{\sum_{j=1}^{k} exp(\theta_j{}^T x)} , \cdots , \frac{exp(\theta_{k-1}{}^T x)}{\sum_{j=1}^{k} exp(\theta_j{}^T x)} \right]$$

During training, the model fits a weight vector θ_i to every output class i. The simplified output decision rule is then:

$$\hat{y} = argmax_i \, (\theta_i^T x)$$

In practice, a class weight vector can be combined with the categorical cross entropy loss during training to reduce the impact of imbalanced dataset.

D. Support Vector Machine

Support vector machine is a binary classifier that aims to maximize the geometric margin between the two output classes. A simple binary SVM minimizes the hinge loss with L2-regularization:

$$\frac{1}{m} \sum_{i=1}^{m} max(0,\ 1 - y^{(i)}(w \cdot x^{(i)} + b)) + \lambda \|w\|_2^2$$

where $y^{(i)}$ is the true label, and λ is the regularization parameter. In practice, SVM allows for efficient learning in very high-dimensional feature space using the kernel trick. In multi-class classification setting, a typical approach is to train a one-against-all classifier for each output class.

E. Neural Network

Neural network is a powerful network-structured model with one input layer, several hidden layers, and one output layer, while each layer consists of a number of neurons. Each neuron takes the output of the previous layer as input, computes a linear combination of the input, and applies a non-linearity to the linear combination. A typical choice of non-linearity function is the ReLU (rectified linear unit) function:

$$ReLU(x) \ = \ max(w \cdot x + b,\ 0)$$

Neural network is very versatile in that it can perform regression, binary classification, and multi-class classification. There are also powerful techniques such as L2 regularization and dropout regularization that helps prevent overfitting when training a neural network.

IV. Experiments, results, and discussion

In this section we present detailed experiment results and discuss the model performance both quantitatively and qualitatively.

A. Evaluation Metric

Because the dataset is highly skewed, accuracy is not a good measure of model performance. For both error detection and error classification tasks, we use precision, recall and F1-score as the evaluation metric:

$$P \ = \ True\ Positive \ / \ (True\ Positive + False\ Positive)$$

$$R \ = \ True\ Positive \ / \ (True\ Positive + False\ Negative)$$

$$F1 \ = \ 2PR \ / \ (P + R)$$

For the second task, the average F1-score is computed as the weighted average of the F1-score of each class.

B. Error Detection Task

The goal of the error detection task is to classify whether there is grammatical error associated with each word. For each word, we used all the features described in Section II, and experimented with three different models including logistic regression, neural network, and random forest. We did not use support vector machine since its training time is incredibly long for a large dataset with ~1M examples.

Since the dataset is extremely imbalanced (error : no error ~ 50:1), the model would classify almost all the words to be error-free if we do not make modifications to the algorithms. We experimented with oversampling, undersampling, and synthetic example generation using SMOTE [11]; however, the most effective method turns out to be simply modifying the class weight of the loss function, and then manually adjusting the decision thresholds.

The logistic regression baseline underfits the training set no matter how we adjust the hyperparameters such as class weight. The reason is that the logistic model can only fit a linear decision boundary, but almost all the grammatical errors have to be decided based on a combination of many features.

For the neural network model, we first scale the features to have zero mean and unit variance. We then tune the training optimizer to minimize the weighted binary cross entropy loss as much as possible. Experiment results show that the default "Adam" optimizer with batch size 128 performs the best. The neural network overfits the training set significantly without regularization. To mitigate this, we include a dropout layer after the input layer and each hidden layer, and the dropout probability is determined using cross validation. We tune the network hyperparameters using hold-out cross-validation with a train:dev ratio of 80:20. Table I shows our hyperparameter grid search space:

TABLE I
NEURAL NETWORK HYPERPARAMETER SEARCH SPACE

# of Hidden Layers	3, 4, 5
Dropout Probability p	0, 0.1, 0.15, 0.2
Class Weight	10:1, 20:1, 30:1

Based on experiments, the best performing network has four hidden layers; there are 300, 200, 200, and 100 neurons in the four hidden layers, respectively.

The model that performs the best in the error detection task is random forest. We use fully grown decision trees, while controlling the number of estimators, the number of feature candidates at each split, and the class weight using cross validation. The variance of the random forest model reduces as we increase the number of decision trees. On the other hand, computation at prediction time increases linearly with the number of estimators. Figure 1 shows the validation F1-score of the model with different number of estimators. We adjust the output threshold to trade off between precision and recall and thus optimize the overall performance. We choose to use 200 estimators since the performance saturates at that point.

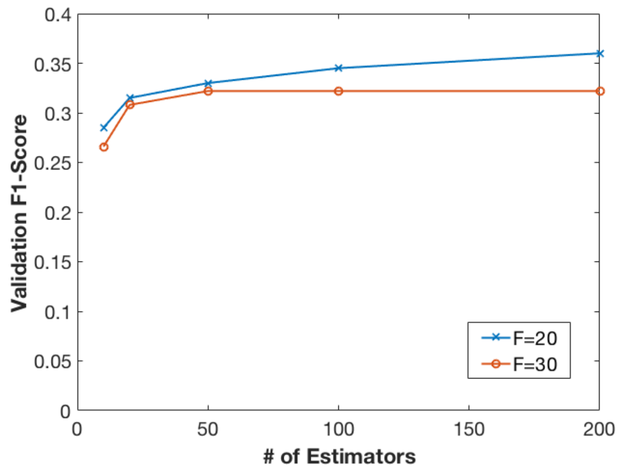

Fig. 1 Random Forest Dev Average F1-Score vs. # of Estimators (F: number of feature candidates at each split)

We tune the random forest model by grid searching hyperparameters with 5-fold cross validation (number of features candidates at each split $\in \{15, 20, 25, 30\}$, class weight $\in \{10:1, 20:1, 30:1\}$). The output threshold of the model is adjusted for each set of hyperparameters so that maximum F1-score is achieved.

Table II summarizes the performance of all three models. "P" stands for precision, "R" stands for recall, and F1 stands for F1-score.

C. *Error Type Classification Task*

The goal of the error classification task is to classify the specific error type given an error word. For each word, we used all the features described in Section II, and experimented with three different models including softmax regression, support vector machine, and neural network. Balanced class weight is used for both softmax regression and neural network.

The baseline softmax regression model does not perform very well mainly because it can only fit a linear decision boundary to a complex dataset.

Support vector machine is feasible for this task since the training set includes only the error words and is thus small. We scale the features to zero mean and unit variance, and train seven one-against-all models with Gaussian kernel. The hyperparameters for SVM mainly include σ for the Gaussian kernel and regularization parameter C. We tuned these two parameters by grid searching with five-fold cross validation ($\sigma \in \{10^{-3}, 2\times10^{-3}, 5\times10^{-3}, 10^{-2}\}$, $C \in \{10, 100, 300, 500, 1000\}$). Table III shows the dev set performance of softmax regression and support vector machine on all seven error types. The SVM model ($\sigma = 2\times10^{-3}$, $C = 100$) gives quite decent performance.

TABLE II
TRAIN/DEV SET PERFORMANCE FOR ERROR DETECTION TASK

Classification Model	**Training Set**			**Dev Set**		
	P	**R**	**F1**	**P**	**R**	**F1**
Logistic Regression (w=10:1)	0.12	0.10	0.11	0.12	0.09	0.10
Logistic Regression (w=20:1)	0.11	0.13	0.12	0.11	0.13	0.12
Neural Network (L=4, w=10:1, p=0.1)	0.46	0.46	0.46	0.26	0.24	0.25
Neural Network (L=4, w=10:1, p=0.2)	0.40	0.37	0.38	0.29	0.26	0.27
Neural Network (L=4, w=20:1, p=0.1)	0.44	0.45	0.44	0.29	0.25	0.26
Neural Network (L=4, w=20:1, p=0.2)	0.40	0.36	0.38	**0.30**	**0.28**	**0.29**
Random Forest (F=30, E=20)	0.84	1.00	0.91	0.26	0.36	0.30
Random Forest (F=30, E=200)	0.88	1.00	0.93	0.28	0.39	0.33
Random Forest (F=20, E=20)	0.84	1.00	0.91	0.27	0.39	0.32
Random Forest (F=20, E=200)	0.87	1.00	0.93	**0.32**	**0.42**	**0.36**

* w: class weight, L: # of hidden layers, p: dropout, F: # of feature candidates at each split, E: # of estimators

TABLE III
DEV SET PERFORMANCE FOR ERROR CLASSIFICATION TASK

Error Type	**Softmax Dev Set**			**SVM Dev set**		
	P	**R**	**F1**	**P**	**R**	**F1**
Article or Determiner	0.59	0.57	0.57	0.71	0.78	0.75
Noun Number	0.48	0.52	0.50	0.67	0.67	0.67
Verb Tense	0.52	0.36	0.42	0.56	0.57	0.57
Preposition	0.62	0.63	0.62	0.82	0.77	0.79
Verb Form	0.32	0.31	0.32	0.46	0.40	0.42
Word Form	0.32	0.46	0.38	0.43	0.39	0.41
Subj-verb Agreement	0.26	0.33	0.29	0.32	0.28	0.30
Average	**0.49**	**0.48**	**0.48**	**0.62**	**0.63**	**0.62**

The neural network model gives the best performance for this task. We use a weighted categorical cross entropy loss, and tune the network in a similar way as described in the previous subsection. We grid search for the number of hidden layers, L2 regularization parameter, and dropout probability using 5-fold cross validation. Based on experiment results, the best performing network has three hidden layers, $\lambda = 0.05$ and $p_{drop} = 0.2$; there are 300, 200, and 100 neurons in the three hidden layers, respectively.

Figure 2 shows the train/dev set performance versus the dropout probability. The model significantly overfits the training set without dropout. As we increase the dropout fraction, the training score gradually decreases, and the gap between the training error and dev error becomes smaller. The best dev set performance is achieved at $p_{drop} = 0.2$.

Finally, Table IV gives the performance of the optimized neural network model.

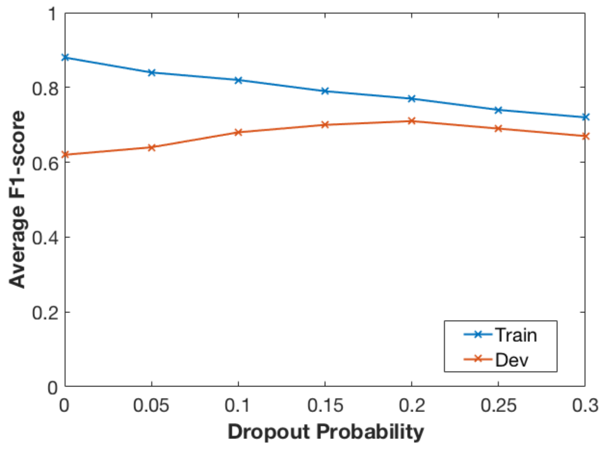

Fig. 2 Neural Network Train/Dev Average F1-Score vs. Dropout Probability (three hidden layers, $\lambda = 0.05$)

TABLE IV
PERFORMANCE OF NEURAL NETWORK ($\lambda = 0.05$, $p_{drop} = 0.2$)

Error Type	Training Set			Dev Set		
	P	**R**	**F1**	**P**	**R**	**F1**
Article or Determiner	0.89	0.89	0.89	0.85	0.87	0.86
Noun Number	0.81	0.81	0.81	0.77	0.73	0.75
Verb Tense	0.82	0.48	0.61	0.79	0.44	0.57
Preposition	0.88	0.95	0.92	0.88	0.94	0.91
Verb Form	0.68	0.64	0.66	0.52	0.49	0.50
Word Form	0.50	0.82	0.62	0.43	0.75	0.55
Subj-verb Agreement	0.53	0.63	0.58	0.39	0.45	0.42
Average	**0.79**	**0.78**	**0.77**	**0.73**	**0.71**	**0.71**

D. *Qualitative Error Analysis*

After some qualitative error analysis, we found that the biggest problem with the model is that it performs poorly when a grammatical error is directly associated with multiple words or the grammatical error results from a missing word, even if the overall sentence structure is very simple. For example, for both of the two following sentences, the model classifies all words in the sentences as error-free.

"Nuclear technology had come a long way since ...." *(Verb tense error: "had come" -> "has come")*
"Dubai is good example for this." *(Preposition error: missing preposition "a")*

The behavior of the model is expected since for each sentence, our approach classifies all the words separately. This problem can be mitigated by using sequence models.

V. Conclusion and Future Work

In this project, we successfully built a system to detect and classify grammatical errors made by non-native English writers. Based on experiment results, random forest achieves the best performance in error detection task (precision 0.32, recall 0.42, F1-score 0.36), and neural network achieves the best performance in error classification task (avg. precision 0.73, avg. recall 0.71, avg. F1-score 0.71).

We believe that the system still has huge potential for future improvement. Most importantly, we should build a model that works at the sentence level to address the problem resulting from associating each grammatical error to a single word. Specifically, we plan to experiment with the recurrent neural network (RNN) architecture which can effectively integrates information of all words in a sentence.

VI. Acknowledgement

We want to say thank you to Prof. Andrew Ng and Prof. Dan Boneh for their guidance throughout the semester, to the CS229 teaching group for providing us with general project support, and to Ziang Xie for providing us with the project idea and giving us many helpful technical suggestions.

VII. Team Contribution

Each team member contributed fairly to this project.
Zihuan Diao: Modeling experiment, hyperparameter tuning
Junjie Dong: Feature engineering, modeling experiment
Jiaxing Geng: Feature engineering, modeling experiment.

References

[1] C. Bryant, M. Felice, and T. Briscoe. Automatic annotation and evaluation of Error Types for Grammatical Error Correction. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Vancouver, Canada. 2017.

[2] A. Rozovskaya, K-W. Chang, M. Sammons, D. Roth, and N. Habash. The Illinois-Columbia System in the CoNLL-2014 Shared Task. 10.3115/v1/W14-1704. 2014.

[3] M. Junczys-Dowmunt and R. Grundkiewicz. The AMU system in the CoNLL-2014 shared task: Grammatical error correction by data- intensive and feature-rich statistical machine translation. In Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task. 2014.

[4] Z. Xie, A. Avati, N. Arivazhagan, D. Jurafsky, and A. Y. Ng. Neural Language Correction with Character-Based Attention. CoRR, vol. abs/1603.09727, 2016.

[5] 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 Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task (CoNLL-2014 Shared Task). Baltimore, Maryland. 2014.

[6] T. Mikolov, K. C., G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space”, CoRR, vol. abs/1301.3781, 2013.

[7] J. Pennington, R.Socher, and C. D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[8] Gupta, Anubhav. 2014. Grammatical Error Detection and Correction Using Tagger Disagreement. CoNLL-2014, 21860(26282), 49.

[9] D. Chen and C. D. Manning. A Fast and Accurate Dependency Parser using Neural Networks. Proceedings of EMNLP 2014.

[10] Apache OpenNLP. Web. https://opennlp.apache.org/

[11] K. B. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. CoRR, vol. abs/1106.1813, 2011.

[12] Pedregosa *et al.* Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.

[13] M. Abadi *et al*. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015.

[14] F. Chollet *et al*. Keras. https://github.com/fchollet/keras