THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Studies of Linear and Deep-Neural Network (DNN) Models for Automated Short Answer Grading

YAJUR TOMAR
SPRING 2023

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Science
with honors in Engineering, B.S.

Reviewed and approved* by the following:

Rebecca Passonneau
Professor of Computer Science
Thesis Supervisor

John Hannan
Professor of Computer Science
Honors Adviser

* Electronic approvals are on file.

ABSTRACT

The Automated Short Answer Grading (ASAG) task is one of the foci at the Penn State NLP Lab. It is focused on finding automated methods to grade and provide feedback for students' short answer questions, based upon rubrics and guidelines set by the instructor or course designer and applied by instructors, teaching assistants or graders. Grading hundreds of short answer questions, especially in larger college classes, can be difficult to complete in a consistent and timely fashion. To help tackle this issue, the Penn State NLP lab looks into machine learning models as a method for assisting with this task. However, one of the main issues with ASAG is the lack of data, which is important especially in the context of implementing deep learning models. Deep learning models have been showing great performance in the NLP domain, however their effectiveness relies on an abundance of data. To help resolve this problem, the NLP lab has collaborated with education researchers in STEM topics to jointly create labeled data that can be used for this task. The models developed on this data consist primarily of neural network architectures, but the baselines they are compared with include both linear models, which require feature engineering, and non-linear neural networks. This paper will explore the different linear and nonlinear machine learning algorithms used as baselines for the lab's work. One important focus was to determine whether one aspect of the feature engineering used in one of the available baselines could be re-implemented, specifically, automatic generation of regular expression features. An attempt was made to replicate other investigators' results using these engineered features with random forest models applied to a KAGGLE dataset. In addition, the thesis describes work completed to create a new ASAG dataset for middle school physics, and results on this data and on a benchmark dataset of a baseline that use LSTM networks to generate features for a logistic regression. The effort involved in engineering features manually versus learning them using LSTM

clearly demonstrates the tradeoff between access to sufficient data to use neural networks versus the effort involved in feature engineering for linear models.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to sincerely thank Dr. Passonneau and Zhaohui Li for allowing me to work with them for the past two years in Penn State's NLP Lab. I have learned so much, and they have provided me with invaluable feedback not only for this thesis but in the NLP domain in general. I appreciate the time and patience the two of them have made for me throughout the years in supporting me with my research.

Chapter 1

Introduction

Utilizing explicit feature engineering and conventional machine learning algorithms from early NLP work provides an interesting baseline for current deep learning models in the ASAG task. This thesis is a detailed investigation of a successful approach of this type, which required detailed understanding of the functionality of the code, and a reimplementation of many components to utilize current libraries and produce both an informative analysis and a codebase that could be utilized today.

Before diving into the complexities of some of the various NLP algorithms utilized for short answer grading, the initial step is to understand what the problem is. In the context of assessments given by grade schoolteachers and undergraduate instructors, most questions can be separated into two large categories: multiple choice and short answer. Grading multiple choice answers has already been automated with Scantron, which does not require any interpretation from a processor other than "which bubble was filled in." However, when grading short answer styled questions interpretation becomes more open ended, making it nearly impossible to make accurate assessments of a students' answers with automated tools that do not rely on artificial intelligence methods, such as machine learning and natural language processing. For example, certain words may be mentioned in a particular order, synonyms and antonyms come into play, or a student can accidentally slightly misspell a word. With all these variations to answer a single question, a model will need more information than a simple text match. As will be seen later in this paper, current linear algorithms require a large assortment of features to help provide a

model with sufficient information to make decisions about short answer responses, from bag of words, average sentence length, tf-idf, and number of manually generated regular expression patterns. An alternative machine learning approach is also applied, relying on deep neural networks (DNNs) to learn a feature representation through a training process where the training process where the training process where the training objective is to classify the correctness of students' answers.

Section 1.1: The ASAG Task

The Automatic Short Answer Grading (ASAG) is a task for assessing short answer questions by computational methods (Burrowes 2015). The goal is to find an algorithm that completes this task time-efficiently, space-efficiently, and accurately. The task specification uses classification of correctness on a narrow scale (e.g., 3 points) rather than assigning a numeric value. Many studies highlight different techniques and algorithms that are used to incrementally improve the efficiency of this task, such as paraphrase recognition (Leacock et al., 2003), text similarity (Burrowes et al., 2013), and BERT-based models (Zhu 2022). The main focus of this paper is on machine learning (ML) algorithms. The bulk of the work examines a high-performing ASAG linear machine learning algorithm from a Kaggle competition, and re-implements the algorithm to provide a baseline for ongoing work in the Penn State NLP Lab. An ancillary task was to help prepare a new, as yet unpublished dataset, and test another baseline that uses neural machine learning.

Section 1.2: The Hewlett Foundation: Short Answer Scoring Competition

Through a competition hosted by Kaggle and The Hewlett Foundation in 2012, a new dataset was released and required participants to "develop a scoring algorithm for student-written short-answer responses." From the results of the competition, Luis Tandalla finished first with an unconventional solution. Tandalla focused on the preprocessing stage, and manually generated regular expressions to accurately score the short answer essays (Tandalla 2012). He used these regular expressions as features within his Random Forest models. Although this idea was successful, the concept of manually generating regular expressions for each prompt was time consuming. Thus, the idea of automating the regular expression feature generation was introduced.

This idea was picked up by Lakshmi Ramachandran, and her algorithm used various stemmer, tagger, and synonym generator algorithms to generate a large amount of regular expression patterns for Tandalla's models to utilize (Ramachandran 2015). As reported by her paper, her automated pattern generator algorithm (AutoP) was able to slightly surpass Tandalla's manually generated patterns, measured by quadratic weighted kappa. Tandalla reported a mean quadratic weighted kappa of 0.77 on all 10 datasets, while Ramachandran reported a mean quadratic weighted kappa of 0.78 (Ramachandran 2015). The comparisons between both algorithms along with detailed explanations of how they work, and their accuracies will be covered. What is important now is that Ramachandran's research changes the perspective of how to tackle the ASAG task. Rather than viewing the task from the perspective of developing innovative ML models, Tandalla and Ramachandran flipped the view to focus on the feature engineering of the data representation, and ensuring their Random Forest models provided a sufficiently informative representation of the data to make decisions.

Chapter 2

Datasets

For the supervised machine learning (ML) algorithms and models discussed in this thesis, utilizing labeled datasets is a crucial step. These datasets are generally preprocessed into some form of feature-representation that can then be fed into classification models, for linear ML, or into cleaned plain text, for DNNs. When reimplementing models for the ASAG task, two major datasets are utilized for training and evaluating. The first dataset, Kaggle's ASAP dataset, was utilized when reimplementing the AutoP algorithm and Tandalla's Random Forest Models. The description of both AutoP's feature generation and Tandalla's models will be explained in further detail in Chapters 4 and 5. However, the reason for utilizing the ASAP dataset for AutoP and the Random Forest Models was due to the fact that Tandalla's preprocessing code was written specifically for the ASAP dataset. Similarly, AutoP is built to automatically generate features for Tandalla's model. If utilizing the SemEval dataset for these algorithms there would be no way to compare performance of the reimplementation, since there are no published results for AutoP on SemEval.

The second dataset, SemEval, was utilized when reimplementing the LSTM and Logistic Regression models. SemEval 2013 Task 7 is the main benchmark dataset for evaluating ASAG, and Penn State's NLP Lab already had preprocessing code for this dataset (Dzikovska et al. 2013). The other dataset is an unpublished dataset that recently was collected by the NLP Lab in collaboration with researchers in STEM education. This dataset consists of an assortment of middle school physics questions with student answers labeled for correctness, and is currently being used for the ASAG task in Penn State's NLP Lab. The following three sections will go

over the organization of the Kaggle's ASAP dataset, SemEval Dataset, and the middle school physics dataset respectively, to help provide a better understanding when later discussing using these datasets for modeling.

Section 2.1: Kaggle's ASAP-SAS Dataset

Kaggle's ASAP-SAS dataset is used for both Tandalla's algorithm of manually generated regular expressions and Ramachandran's AutoP algorithm which automates the process of generating regular expression patterns, which will be discussed further in Chapters 4 and 5. The ASAP-SAS Dataset is constructed slightly differently than the SemEval Dataset, but contains the same information (i.e., rubric information, reference answers, questions, and answers). There are 10 prompts (questions), all of which have a provided document explaining the question and scoring rubric, as well as separate documents containing handwritten reference answers. Each prompt can be organized into one of two domains: english comprehension or lab sciences. All the training and test data are already split in the tsv files; however, these files contain all 10 prompts worth of student answers within a single tsv. As shown in a subset of the training data in Figure 1 the dataset is split into an essay id column, essay set column (one through ten), two score columns, and finally the actual text of the essay. The ID column simply provides a numeric identification to each student answer. The Essay Set column denotes which prompt the student is answering. The Score 1 and Score 2 columns are the scores given by two manual graders, and finally, the essay text column represents the student's answers. Rather than the XML format provided with SemEval, the ASAP-SAS dataset is much easier to read, as all student answers can

be provided in a single file, and separated into their respective prompts just by looking at the EssaySet field.

Id	EssaySet	Score1	Score2	EssayText
1	1	1	1	Some additional information that we would need to replicate the experiment is how much vinegar should be pl
2	1	1	1	After reading the expirement, I realized that the additional information you need to replicate the expireim
3	1	1	1	What you need is more trials, a control set up, and an exact amount of vinegar to pour in each cup/beaker.
4	1	0	0	The student should list what rock is better and what rock is the worse in the procedure.
5	1	2	2	For the students to be able to make a replicate, they would need to tell use how much vinegar is used and w
6	1	1	0	I would need the information of why you would let the different samples dry out of the containers. And what
7	1	1	0	The information I would need in order to sucessfully replicate the experiment is the correct measurement th
8	1	3	3	You would need many more pieces of information to replicate the experiment. You would need the type of samp
9	1	3	3	Some additional information you will need are the material. You also need to know the size of the contanein
10	1	2	2	Inorder to replicate the experiment, we will need to add information such as sticlenx statement of what the
11	1	0	0	An additional information that i would need in order to replicate the experiment
12	1	3	3	The additional infomation you would need to replicate the experiment would include the size of the samples.

Figure 1. ASAP-SAS Sample train dataset

In Figure 1, two separate scores (Score1 and Score2) are shown and are a result of assigning two graders to each question to provide a more consistent label to the data. Additionally, due to the ASAP-SAS dataset belonging to a Kaggle competition, the labels for the test data that scores the model was not released until after the competition was complete. Thus, there are additional labeled tsv files released that were used as optional validation and test datasets.

The difficulties with using this dataset came from translating the rubrics and reference answers from Word (.docx) into text files that the AutoP algorithm can utilize in its logic. Because Tandalla's algorithm and AutoP are heavily dependent on deriving patterns from rubrics and reference answers, it is clear how the setup of the ASAP-SAS Dataset can hinder the results. Due to the rubric being presented in the form of a Word document and reference answers being presented as handwritten PDF files, it is up to the interpretation of those creating the preprocessing algorithms of how they should extract and clean the data. Tandalla was able to get around this issue by physically looking through the rubrics and top-scoring answers of each prompt and manually generating the regular expressions, which allowed him to ignore the step of translating the rubrics and reference answers into files the computer could process. Ramachandran manually modified the dataset to be able to read the prompts and reference

answers provided in documents and handwritten PDF's. Chapter 4 will go into detail on the preprocessing steps required on the Kaggle dataset prior to using it as an input in AutoP's reimplementation.

Section 2.2: SemEval Dataset

The first dataset that will be discussed is the SemEval Data Set. This dataset consists of two smaller component datasets named "SCIENTSBANK" and "BEETLE." The SciEntsBank dataset is the larger of the two, with 10,000 students responses across 15 science related domains; whereas the Beetle dataset contains only 3,000 student responses focusing on the electricity and electronics domain (Dzikovska et al., 2013). The large dataset is broken up into three classification tasks, 2-way, 3-way, and 5-way, all of which are different class label counts. Each task denotes the number of possible labels a short answer question can receive. For example, 2-way means that there are only two possible ways for a student to be scored on their answer (correct and incorrect), 3-way means that there are three possible ways for a student to be scored (correct, incorrect, and contradictory), and finally 5-way means that there are five possible ways for a student to be scored (correct, partically_correct_incomplete, contradictory, irrelevant, and non-domain) (Dzikovska et al., 2013). All these answers are deduced based on comparisons with reference answers. Thus, a contradictory answer is one that contradicts with the reference answer, an irrelevant answer is one that provides domain content but not the actual information to make the answer correct, and non-domain means that the student does not even reference the domain content (Dzikovska et al., 2013).

Figure 2. Example Dataset from SemEval 2013 Task 7

Figure 2, above, depicts an example of the SemEval Dataset. It is provided in an XML format, where the outer section is wrapped with the question that the students have to answer. The subsections inside the question wrapper are for reference answers and student answers respectively. In Figure 2 the reference answers are tagged with different styles of student answers. For the question, “Why does an open switch impact a circuit?”, the reference answers provided are “the open switch creates a gap” which is categorized as the “BEST” type of answer, “a gap” is categorized as “KEYWORD”, “there is a gap in a circuit”, “the path is not closed”, “there is an open path” are categorized at “GOOD”, and finally the reference answer “there is an open path” is categorized as “MINIMAL.” The student answer subsection above has a list of student answers that have been provided with an id, the student answer, and is labeled with their graded accuracy. SemEval is used when creating the baseline Logistic Regression and LSTM (Long Short Term Memory) models that will be discussed in Chapter 3.

Section 2.3: Middle School Physics Dataset

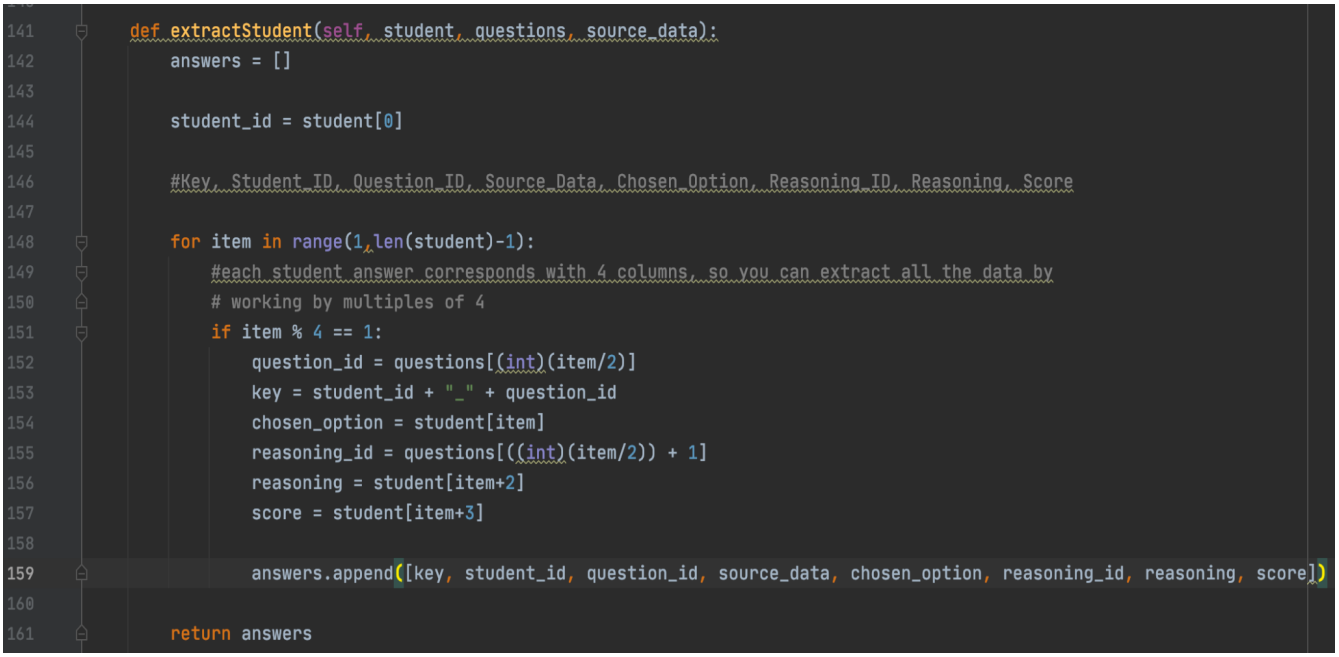
One of the main issues with the ASAG task is the lack of data. When it comes to finding a dataset that contains realistic, structured, accurate, clean data to be able to be used in an NLP setting, it can be difficult. One of the larger tasks prior to reimplementing AutoP was cleaning datasets so that it can be utilized with the current Penn State NLP Models. In particular, this section goes over a dataset provided by collaborators in the learning sciences, and that we in the NLP lab cleaned and prepared for use as training and test data. The lab has applied this dataset in two kinds of algorithmic settings, using a machine learned classifier for assessment, and integrating a machine-learned classifier into a human-in-the-loop system, where a classifier and human each label different parts of the data to achieve maximum accuracy and minimize human effort.

Key	Student_ID	Question_ID	Source_Data	Chosen_Option	Reasoning_ID	Reasoning	Score
Pulley9-10-99_q3a	Pulley9-10-99	q3a	Pulley_Post_Y2	1	q3b	I said that the distance would increase because when you use a movable pulley the fc	2
Pulley9-10-99_q3c	Pulley9-10-99	q3c	Pulley_Post_Y2	2	q3d	The applied force required would go down, because the distance is going up... we alsc	2
Pulley9-10-99_q4	Pulley9-10-99	q4	Pulley_Post_Y2	1	q4a	I say this, because when you have a rusty pulley, there is much more friction, so causc	2
Pulley9-10-99_q7a1	Pulley9-10-99	q7a1	Pulley_Post_Y2	2	q7a2	The applied force needed would decrease because the distance is increasing, so that	2
Pulley9-10-99_q7b1	Pulley9-10-99	q7b1	Pulley_Post_Y2	3	q7b2	The work done would stay the same becasue the distance pulled is less on the single	2
Pulley9-10-99_q9	Pulley9-10-99	q9	Pulley_Post_Y2	3	q9a	Jane andd Mary are doing the same amount of work because the distance for Mary is	2
Pulley9-10-99_q10	Pulley9-10-99	q10	Pulley_Post_Y2	2	q10a	A rusty pulley that needs to be oiled will require more force because there is a lot mor	2
Pulley9-10-99_q11	Pulley9-10-99	q11	Pulley_Post_Y2	2	q11a	I said lifting the box to 2 meters becasue the distance pulled is going up (as you can s	2
Pulley9-10-99_q13	Pulley9-10-99	q13	Pulley_Post_Y2	4	q13a	I say the double compound becasue MA is distance pulled divided by distance object	2
Pulley9-10-99_q14	Pulley9-10-99	q14	Pulley_Post_Y2	2	q14a	I said this becasue the distance pulled willl go up for set up B and since the moved dis	0
Pulley9-10-98_q3a	Pulley9-10-98	q3a	Pulley_Post_Y2	1	q3b	adding a nother pully would make it need to be pulled father to lift but it would be easy	0
Pulley9-10-98_q3c	Pulley9-10-98	q3c	Pulley_Post_Y2	2	q3d	it would increas the ma making the applied force decreas	2
Pulley9-10-98_q4	Pulley9-10-98	q4	Pulley_Post_Y2	1	q4a	it would slide easyer and make it easyer to slide the string and it also matter in how m	1
Pulley9-10-98_q7a1	Pulley9-10-98	q7a1	Pulley_Post_Y2	1	q7a2	the single movable pullys have less ma making the applied force increas	0
Pulley9-10-98_q7b1	Pulley9-10-98	q7b1	Pulley_Post_Y2	1	q7b2	it could also be a very rusty pully making it harder to pull the watermelen up	0
Pulley9-10-98_q9	Pulley9-10-98	q9	Pulley_Post_Y2	4	q9a	the pully could be a double triple or single compound increasing the ma	0
Pulley9-10-98_q10	Pulley9-10-98	q10	Pulley_Post_Y2	2	q10a	it would get stuck and be hard to pull	1
Pulley9-10-98_q11	Pulley9-10-98	q11	Pulley_Post_Y2	2	q11a	it is a higher hight making the work go up because you would do double the work	1
Pulley9-10-98_q13	Pulley9-10-98	q13	Pulley_Post_Y2	4	q13a	the double compound has more ma because if the single compound has 3 ma the dou	1
Pulley9-10-98_q14	Pulley9-10-98	q14	Pulley_Post_Y2	2	q14a	if 1 pully has 1 ma then 2 pullys will have 2 ma	0
Pulley9-10-97_q3a	Pulley9-10-97	q3a	Pulley_Post_Y2	3	q3b	you are pulling it the same distance and the load is staying the same.	0
Pulley9-10-97_q3c	Pulley9-10-97	q3c	Pulley_Post_Y2	2	q3d	with pulling it straight up you would not have to pull down you would pull harder than it	0

Figure 3. Pulley-Post Physics Lab Dataset

One difficulty with utilizing this dataset was that multiple questions could be linked where one question could be multiple choice and another could be short answer. Specifically the short answer question asks for the “reasoning” for the multiple choice question. When the dataset was

initially provided to the lab, it was not organized in a way that can be used instantly into the baseline models the lab had already created. Another key pre-processing task pertains to the difference in perspective that the education researchers have with respect to their data, versus the NLP researchers. The prior focuses on the student, thus each row in the data table represents all the answers to all questions from a given student. The NLP researchers, however, focus on the question-answer pair. Thus, through python scripts, the acquired dataset must be reshaped in a custom preprocessing that allows better organization of information, where the data is now question-answer centered and not student centered.



```
141    def extractStudent(self, student, questions, source_data):
142        answers = []
143
144        student_id = student[0]
145
146        #Key, Student_ID, Question_ID, Source_Data, Chosen_Option, Reasoning_ID, Reasoning, Score
147
148        for item in range(1,len(student)-1):
149            #each student answer corresponds with 4 columns, so you can extract all the data by
150            # working by multiples of 4
151            if item % 4 == 1:
152                question_id = questions[(int)(item/2)]
153                key = student_id + "_" + question_id
154                chosen_option = student[item]
155                reasoning_id = questions[((int)(item/2)) + 1]
156                reasoning = student[item+2]
157                score = student[item+3]
158
159                answers.append([key, student_id, question_id, source_data, chosen_option, reasoning_id, reasoning, score])
160
161        return answers
```

Figure 4. Segment of Python Script for Extracting Student Data

The first step for this preprocessing was extracting the question and prompt itself. Because multiple question-answer pairs for the same student were aligned in the same rows of the dataset, they needed to be extracted separately, so that all the question-answer pairs for a single question were organized together. This process can be seen from Line 151-159 of Figure 4, where for every

4th column (denoted by the if statement: item % 4), a new question-answer pair is appended to the entire answer list (shown in line 159 of Figure 4). Another issue was that the initial datasheet did not have a concrete way of identifying which row of the dataset was a question and which was an answer. Fortunately, based on the number of columns identified with the question versus the number of columns identified with the answer there was a way to programmatically discern the two. Thus, the algorithm implemented that logic to extract and the student answers and questions into a newly created csv. However, this dataset is unpublished because at the time it was collected, the IRB protocols did not include making the data available to the public. However, it has been utilized to further train and evaluate DNN and baseline models in the NLP Lab. For example, the SFRN classifier reported in (Li et al., 2021) was tested on this dataset as well, and compared to a logistic regression baseline (with LSTM encoder). It was also used to train SFRN for a human-in-the-loop task, as reported in (Li et al., 2023), with SFRN achieving an accuracy of 79 on the dataset, and the LSTM encoder with logistic regression (explained in further detail in Chapter 3) achieved an accuracy of 74.

Chapter 3

ASAG Algorithms and Models

This chapter will address the linear and nonlinear models that will be applied to the datasets that have been introduced in the previous chapter. Specifically, the chapter will provide background on some of the different methods and algorithms used for the automatic short answer grading task. Specifically, this chapter will discuss the logistic regression baseline model with a LSTM encoder. It will also introduce Tandalla's Random Forest Models with manually generated regular expression features, and AutoP's algorithm to automatically generate regular expression features, as background for chapters four and five that describe my implementations for these two approaches. All of these models and algorithms have been implemented within the Penn State NLP Labs, utilizing the datasets mentioned in the prior section.

Section 3.1: Tandalla's Model

In contrast to the LSTM, which like other DNN's learns a representation of the input data, Tandalla used conventional linear machine learning with manual feature engineering to represent the input. Tandalla's approach was to manually generate regular expressions based on top scoring answers and rubrics that were provided with the dataset. With these regular expressions they will take data that has been through a spell correction algorithm, and check if a certain student answer has a match with any of the regular expressions that were manually generated. The number of patterns that are matched becomes part of the features that are fed into the Random Forest model, and this information is inputted into a Random Forest Model and a

Gradient Boosted Model. To complete the algorithm, Tandalla takes the average of both models and utilizes this average model for evaluation.

The benefit of this approach is primarily the accuracy. If time is taken to manually generate regular expressions for each rubric/assessment, then the model can be utilized to provide accurate results in short answer grading. This was indicative after Tandalla's approach won first place in the Kaggle Short Answer Grading Competition.

However, there are many drawbacks to Tandalla's approach. First, the use of regular expression matching requires an accurate spell correction algorithm. Matching student answers with regular expressions would only work if the student answers have been spelled correctly. Going through large datasets and conducting a spell correction algorithm can add additional time complexity in addition to training the models. Second, manually generating regular expressions for each dataset reduces the versatility of the model. If different schools or classes wanted to use the same algorithm, they would have to generate their own regular expressions in addition to their rubrics, which Tandalla's entire code base would have to be modified significantly to tailor to any dataset.

Focusing on Tandalla's regular expression code base, the reimplementation of the code only required pulling from his GitHub repository and running the test code with Python 2.7. For Tandalla's model, the data is preprocessed and a variety of features are constructed, through his preprocessing scripts. The features are saved off into csv files, where they are fed into the Random Forest models he constructed in R. This process will be observed in greater depth in chapter 5.

The key aspect to Tandalla's model is the way he generates his features. Utilizing all the manually created patterns that Tandalla created for each essay question, Tandalla does a count to

see how many times each phrase occurs in a student answer (1 or 0). This information is part of the features that get fed into Tandalla's models, and the largest focus for Tandalla's feature engineering.

Section 3.2: Auto P

To improve the generality of Tandalla's regular expression model the largest step to take is to implement Lakshmi Ramachandran's algorithm to generate automated patterns (AutoP). Unlike Tandalla's code, Ramachandran's code uses Java and utilizes models she developed that are not publicly available for extracting parts-of-speech tagging, tokenizing, and dependency parsing information. An effort was made to reach out to Ramachandran for these models, however, there was no success in retrieving them. This makes running her code impossible, leading to reimplementing her code in Python utilizing Spacy's NLP libraries.

Before going into detail on Ramachandran's AutoP algorithm, regular expressions need to be explained. Regular expressions are a context-independent syntax that can represent a wide variety of character sets and character set orderings (OpenGroup). Utilizing special syntax characters can help generalize these expressions to represent a larger set of characters. For example the characters "\w" denote any alphanumeric character, so when incorporated into the following regular expression pattern "1\w2", it can match any sequence of characters that begin with 1, end with 2, and have any alphanumeric character in the middle. This pattern can match strings such as 1e2, 142, 122, 1A2, and so on. As you can see, by adding the character "\w" to a single regular expression pattern the number of phrases it can match increases substantially, in comparison to simply matching for "1e2", "142", "122", and "1A2" separately. Another such character is the Kleene star (*), which can be appended to any character allowing for it to be

repeated any number of times or no times at all. Thus the regular expression pattern “2*” can match 2, 22, 222, and so on. These are just two of the many characters utilized in regular expressions to help generalize text-based matching. Through the power of the regular expression syntax, a few patterns can be created to match a large number of varying student answers, one of the major reasons why Tandalla thought to utilize this method to generate features.

Utilizing this concept of regular expressions in the context of the ASAG task, Tandalla thought to manually create patterns that would appear in good scoring answers based on the provided rubric. Thus, the more matches that are found from these patterns in the student answer, the stronger the assumption is that the student answered the question correctly.

The key idea behind Ramachandran’s algorithm to generate automated regular expressions, is to first take rubric and top scoring answers as an input and construct both “context phrases” and “content words”. These phrases are constructed through generating a word order graph, which strips a sentence of its nonessential words (prepositions), and creates a form of a dependency graph that maintains the ordering of the sentence. In a word order graph edges can only be created between nouns-verbs, nouns-adjectives, verbs-adverbs, where at the current token you can only construct an edge with a previous token. This is the major difference between this technique and a normal dependency graph, because in a dependency graph as long as there is a dependency with another token an edge will be formed, regardless of if the token is before or after the current token.

Once the word ordered graph is constructed, the edges are removed and the remaining vertices (words) are concatenated to become “skeleton phrases.” The next step is to extract content words, by removing all nonessential words, and then creating a count for the rest of the essential words. This count is then used to determine which of the essential words appear the

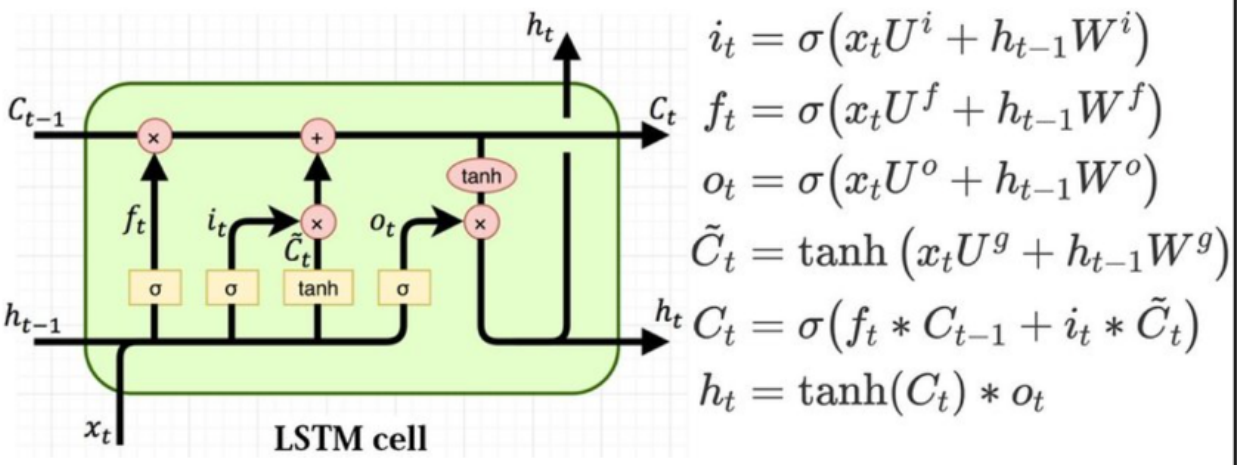
most. This process will be covered in further detail in Chapter 5, however it is these skeleton phrases and content words that provide the backbone of the automatically created regular expressions.

Section 3.3: Logistic Regression Classifier with LSTM Encoding

Two initial tasks performed for the NLP Lab were to generate a logistic regression and Long Short Term Memory Model (LSTM) through Python's machine learning library, PyTorch, for the SemEval Data Set. The reason logistic regression was used was to establish a baseline NLP ASAG model against which to compare a new model developed in the NLP lab (Li et al., 2021). A logistic regression model is one the basic classifiers in machine learning, as it provides a baseline performance when compared to a more complex DNN model on the ASAG task. Utilizing PyTorch made this process simple, as the creating the logistic regression model only took a few lines of code. The preprocessing of the SemEval dataset required the student answers to be converted into a vector representation through Stanford's model GloVe (Pennington et al., 2014), which initializes the representation of text as vectors.

A Long Short Term Memory (LSTM) network was trained as an encoder for the logistic regression model, to learn an embedding representation for effective classification by the logistic regression. An LSTM model is a type of Recurrent Neural Network (RNN) that allows past information to persist within the network. Like many RNN's, an LSTM works as a sequence of cells that feeds the learned representation at a given timestep of one sequence as the input to the

next timestep.



Structure of the LSTM cell and equations that describe the gates of an LSTM cell.

Figure 5. LSTM Cell (Olah 2015)

Figure 5 shows a LSTM cell and its set of functions that determines which information is utilized or forgotten by the model itself. C_{t-1}denotes the cell state of the previous LSTM cell. The input h_{t-1} is one of the outputs of the previous cell and is utilized with input x_t in different activation functions. The first step is to determine the current cell state (C_t). This is done in the following two steps. The f_t function is known as the "forget-state" layer, where a sigmoid activation function on the inputs (h_{t-1} and x_t multiplied by their according weights) determines if the previous cell state (C_{t-1}) will be used. Note that in each of the cell functions, U and W are the weights that the LSTM model is training and updating. After it is determined if the previous cell state will be used, the result gets added to the product of the functions i_t and $\check{C}_t$, where $\check{C}_t$ uses a tanh activation function. The equation to determine the current cell-state is: $C_t = C_{t-1} * f_t + i_t * \check{C}_t$.

To generate the o_t, the same equation as the forget-state layer is used, but with a different set of weights. Finally, to generate the h_t, one of the two outputs of the cell, the newly generated

cell-state, C_t is used in a tanh activation function and multiplied with the generated o_t, which then gets used as an input to the next LSTM cell.

Through these added functions, the LSTM network can utilize past timesteps form the input in addition to make decisions. Utilizing a PyTorch implementation of this LSTM encoder and Logistic Regression model with the SemEval dataset generated a quadratic weighted kappa of 0.70 for the unseen answers dataset, a 0.59 for the unseen questions dataset, and 0.57 for the unseen domain dataset (Li et al., 2021).

Quadratic weighted kappa is the standard evaluation metric to measure performance of the NLP models on educational data (essay grading, short answer grading), and will be used throughout this thesis. The quadratic weighted kappa measures agreement by factoring out the agreement that would occur by chance. Thus it uses the ratio of the observed agreement (O) to the expected agreement (E) to only credit the amount of agreement that is above the level that would occur by chance, given the observed frequency at which any rater chooses a given value. The formula is calculated as: $(O - E)/(1 - E)$, where O and E are observed agreement and expected agreement, respectively (Warrens 2015). From this equation a smaller quadratic weighted kappa score will be generated if the observed agreement is similar to the agreement occurring by chance (from the numerator of the formula).

Chapter 4

Reimplementation of AutoP

To study the effects regular expressions can have as features in the context of short answer grading, Ramachandran devised her AutoP algorithm that generates regular expression patterns automatically based off of rubrics and top scoring answers. These patterns would capture semantic information that should be found in high scoring answers. This chapter will focus on all the steps of reimplementing Ramachandran's AutoP algorithm. Unfortunately, the code base for AutoP, described by Ramachandran, was not readily available, as mentioned in the last section of chapter 3. Thus, an attempt to reimplement the algorithm based on the explanations and pseudocode provided in her paper was made. The high level ideas of the algorithm were replicated in Python; however, at a lower level there seemed to be steps either missing or misinterpreted, as the final result did not perform as well as Ramachandran's paper reported.

At a high level, AutoP is an algorithm that takes rubric text and top scoring answers and automatically generates regular expression patterns, based on recurring context phrases and words found within the text. As Figure 6 depicts, there are five major steps in Ramachandran's pipeline for AutoP that ultimately results in the regular expression patterns that will be used as features in Tandalla's Random Forest model. As will be discussed in further detail in chapter 5, Tandalla's model is the baseline for determining AutoP's effectiveness. By calculating performance using quadratic weighted kappa after running Tandalla's Random Forest models with all of his features and comparing his manually generated regular expression patterns with AutoP's patterns, it will be determined if AutoP's pattern features will be reliable for the short answer grading task.

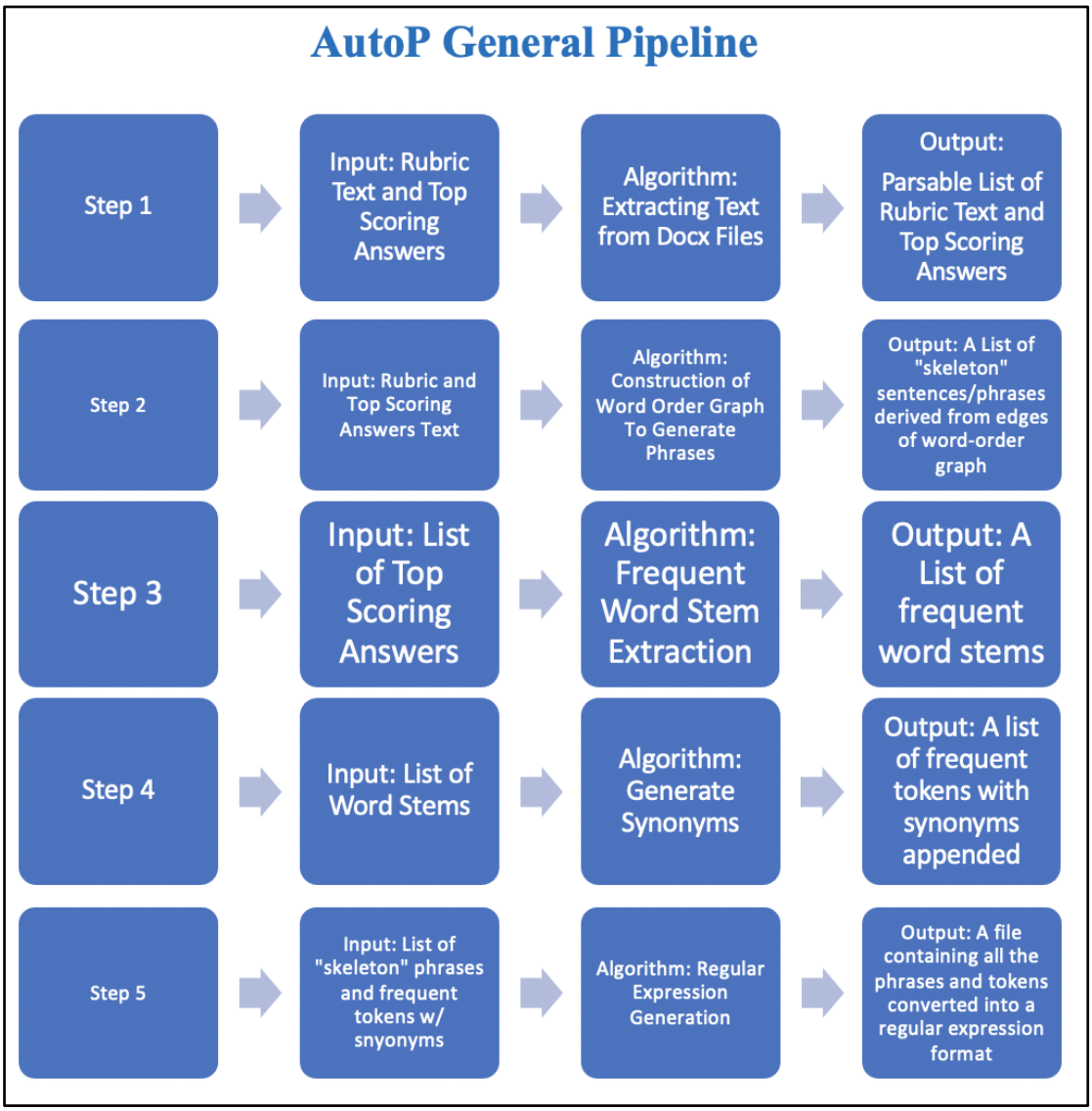


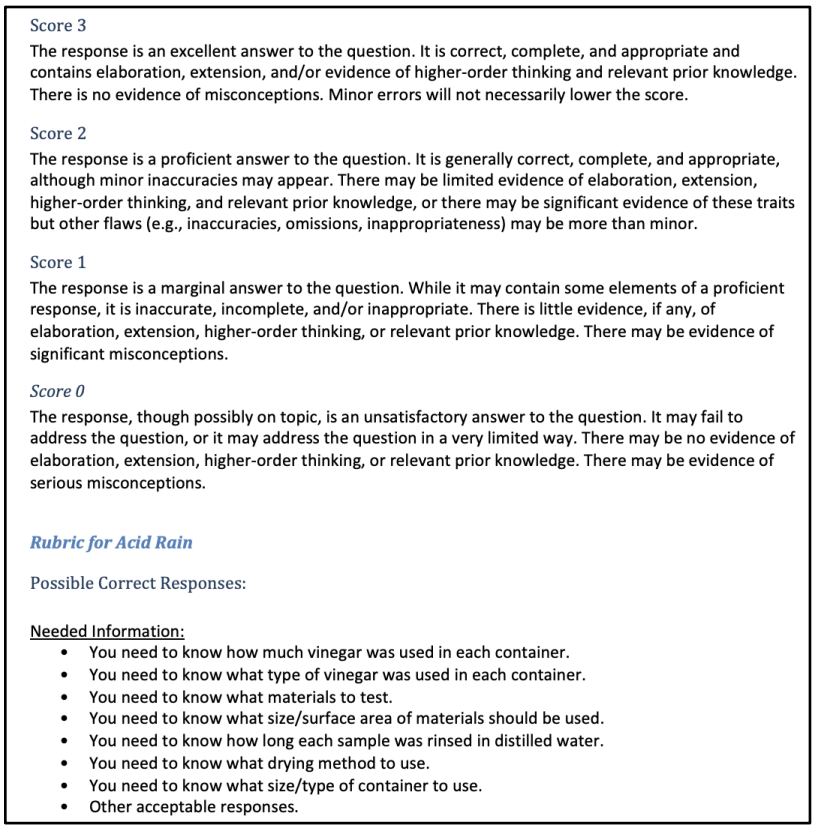
Figure 6. General Pipeline of AutoP

The AutoP algorithm itself is split into five major steps: extracting rubric text and answers of top scorers, converting rubric and top scorers's answers into a list of context phrases, tokenizing and identifying most frequently used tokens in top scoring answers, creating synonyms for tokens in text, and finally merging tokens and phrases into a file of generated regular expressions. This algorithm has to be run separately for each prompt, thus creating 10 separate regular expression files. These files then get fed into Tandalla's preprocessing algorithm, which generates some of the features for the Random Forest model.

This chapter will dive deeper into all of these five steps, and how they have been interpreted and reimplemented in Python.

Section 4.1: Extraction of Rubric and Top Scoring Answers

As mentioned before, the ASAP-SAS dataset utilized for this algorithm is split into ten separate prompts. The dataset provides rubrics for each prompt in the form of Word (*.docx) documents. These documents provided general information about the dataset (i.e. grade level of students, subject, training set size, score range, etc.) at the top, and then proceeded to explain the prompt that the students had to answer. Finally, at the bottom of the document there is a breakdown of scoring criteria that explains what information must be contained in a student answer to achieve that particular score. Figure 7 provides an example of scoring criteria for Prompt #1 in the ASAP-SAS dataset. As can be seen, Figure 7 depicts in detail under the "Needed Information" section what types of phrases would be contained in a good scoring answer, in bullet point format. The rubric criteria provides the best details when it comes to extracting phrases from actual rubric text.



Score 3

The response is an excellent answer to the question. It is correct, complete, and appropriate and contains elaboration, extension, and/or evidence of higher-order thinking and relevant prior knowledge. There is no evidence of misconceptions. Minor errors will not necessarily lower the score.

Score 2

The response is a proficient answer to the question. It is generally correct, complete, and appropriate, although minor inaccuracies may appear. There may be limited evidence of elaboration, extension, higher-order thinking, and relevant prior knowledge, or there may be significant evidence of these traits but other flaws (e.g., inaccuracies, omissions, inappropriateness) may be more than minor.

Score 1

The response is a marginal answer to the question. While it may contain some elements of a proficient response, it is inaccurate, incomplete, and/or inappropriate. There is little evidence, if any, of elaboration, extension, higher-order thinking, or relevant prior knowledge. There may be evidence of significant misconceptions.

Score 0

The response, though possibly on topic, is an unsatisfactory answer to the question. It may fail to address the question, or it may address the question in a very limited way. There may be no evidence of elaboration, extension, higher-order thinking, or relevant prior knowledge. There may be evidence of serious misconceptions.

Rubric for Acid Rain

Possible Correct Responses:

Needed Information:

- You need to know how much vinegar was used in each container.
- You need to know what type of vinegar was used in each container.
- You need to know what materials to test.
- You need to know what size/surface area of materials should be used.
- You need to know how long each sample was rinsed in distilled water.
- You need to know what drying method to use.
- You need to know what size/type of container to use.
- Other acceptable responses.

Figure 7. Rubric Breakdown for Prompt #1: Acid Rain

To convert these documents into a usable form for AutoP, they are converted from docx documents into plain text files. To extract the top scoring answers, the algorithm parses a training file that will not be used or seen by the Random Forest Model and separates all of the student answers that earned the maximum score (usually a score of 2 or 3) for each prompt. Figure 8, below, provides a numerical breakdown of the number of these top scoring answers provided per prompt. As can be seen some prompts have a much smaller number of top scoring answers provided, as their prompts are more difficult for students to answer. In particular prompt 5, with only a 1.89% top scoring answer rate, requires students to describe the four major steps of protein synthesis. A top scoring answer is required to answer all four of these steps correctly.

Distribution of Top Scoring Answers in the ASAP-SAS Training Dataset by Prompt			
Prompt (Max Score)	Number of Top Scoring Answers	Total Number of Student Answers	Percentage of Top Scoring Answers
1 (3)	339	1672	20.28%
2 (3)	317	1278	24.8%
3 (2)	363	1808	20.1%
4 (2)	127	1657	7.66%
5 (3)	34	1795	1.89%
6 (3)	51	1797	2.83%
7 (2)	419	1799	23.3%
8 (2)	777	1799	43.19%
9 (2)	622	1798	34.59%
10 (2)	580	1640	35.37%

Figure 8. Top Scoring Answer Breakdown Per Prompt for the Training Dataset

All the extracted top scoring student answers are placed into individual files separated by prompt. These top scoring answer files are then taken in as inputs in the beginning of the pipeline and stored into list data structures, so that they can be used as inputs for the following steps.

Section 4.2: Context Phrase Generation

The second step of the pipeline is to extract phrases that are "contextually" relevant from the rubric text and top scoring answers. This step is crucial in the construction of the regular expressions, because it helps provide an idea of what kinds of semantically relevant phrases appear in the rubric and top-scoring answers. The premise of this step is the more closely a student answer

follows the phrasing of a rubric or top-scoring answer, the higher chance of the student scoring higher. The output of this phase in the pipeline is to create a list of "skeleton phrases," where only semantically relevant words and their grammatical functions are preserved in each sentence and the rest of the irrelevant details (stop words, conjunctions) are omitted.

In order to create these skeleton phrases, a word-order graph is constructed to keep the order of the words within a sentence and provide grammatical relationships of neighboring tokens. A word-order graph is constructed by organizing the words as vertices, and their grammatical relationship within a sentence as edges. A grammatical relationship is a structurally defined relation between words in phrases and clauses (Payne 2012). A key characteristic of word-order graphs is that edges never point backwards within a sentence. This differentiates from traditional dependency graphs, where dependencies can be constructed between tokens no matter where they appear in a sentence. Although it is true that grammatical dependencies do not have to be restricted by locality, it makes sense to use a word-order graph in this context, because constructing phrases for a regular expression to locate text makes the order of the tokens within that phrase important. Figure 9 provides two examples of word-order graphs from basic sentences. As can be seen from the below figure from the original sentence "Generalists are favored over specialists", the nonessential words "are" and "over" are removed. From the remaining words "generalists favored specialists" connections are created with each adjacent word. "Generalists" is the subject of the verb "favored" and "specialists" is the object of the verb "favored", as Figure 9 displays below. Simple sentences like the previous example may only create a single graph, but complex sentences (meaning sentences with multiple clauses) may need multiple graphs to be represented. For each graph that is constructed, all the words are concatenated to generate the "skeleton phrases."

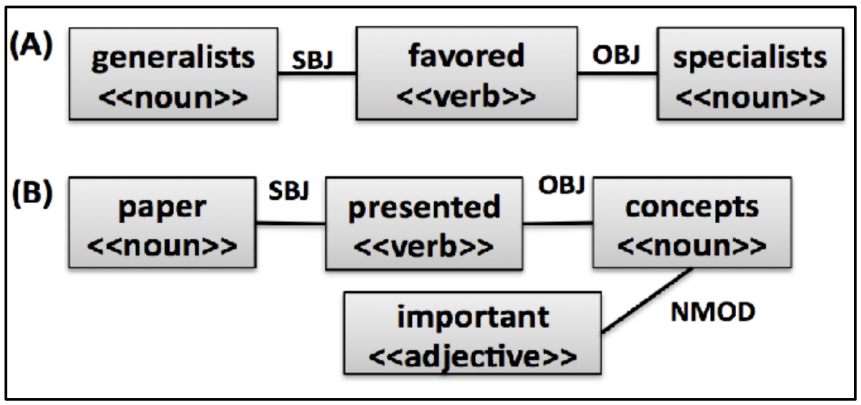


Figure 9 (from Ramachandran 2015). Word-order graph of (A) "Generalists are favored over specialists" and (B) "The paper presented important concepts"

Specifically, the algorithm to generate these word-order graphs simply traverses through each token in the text and determines the part-of-speech (POS) utilizing a POS tagger provided by the Spacy NLP library. Given a sequence of two part-of-speech tags, POS_1 and POS_2, the algorithm consults rules to determine if the two parts-of-speech can be connected by an edge for a grammatical relation like 'subject' or 'object'. For example, the last case of the pseudo code in Figure 10 shows that an adverb before a verb would form a descriptor edge. Similarly an adjective before a noun would form a noun modifier edge. A noun before a verb would form a "subject edge," however a verb before a noun would form an "object edge." The pseudo-code for the word-order graph algorithm is provided below in Figure 10.

Algorithm 1 Vertex and Edge Creation Algorithm

```
for each token w in segment s do
  switch(w)
  case subject component: //checking w's POS tag
  if previous vertex was a subject then
    Concatenate w with previous vertex //updating vertex
  else
    Create a new vertex w //creating a new vertex
  end if
  if an adjective j was the previous vertex then
    if j has been assigned to a subject vertex then
      Delete edge between j and previous subject
      Add edge between j and w //adding adjective-subject edge
    else
      Add edge between j and w
    end if
  end if
  if verb v exists AND edge (v, w) does not exist then
    Add an edge between v and w //adding verb-subject edge
  end if

  case verb or modifier:
  Similar to subject creation. A verb vertex is updated or cre-
  ated, and edges between an adverb and a verb, and a subject
  and a verb may be added in this case.

  case adjective:
  if previous vertex was an adjective then
    Concatenate w with previous vertex
  else
    Create a new vertex w
  end if
  if subject n exists AND edge (n, w) does not exist then
    Add edge between n and w //adding subject-adjective edge
  end if

  case adverb:
  Similar to adjective creation. An adverb vertex is updated or
  created, and an edge between a verb and an adverb may be
  added in this case.
  end switch-case
end for
```

Figure 10 (from Ramachandran 2012). Pseudocode for Word-Order Graph Generation

Figure 10 shows the pseudocode for constructing the word-order graph, with the input being the list of rubric text and top-scoring answers. The code begins by iterating through each word from the top scoring answer and rubric sentence. Utilizing the POS-tag on each token, each current token and previous token are stored at each step, and specific edges are constructed (verb-noun, adverb-verb, adjective-noun, adjective-adjective). Once all these vertices and edges for the graph are stored into a data structure, the vertices are parsed through and appended into a new list of "skeleton" phrases. These skeleton phrases composed by the vertices (words) are the final output of this step of the pipeline.

Section 4.3: Frequent Token Extraction

After the construction of phrase patterns, the third step of AutoP's pipeline is generating context tokens. The importance of this step is to determine from the top scoring answers which key words provide the most context towards the correct answer. Rather than focusing on the grammatical relationships of entire sentences, this step simply focuses on the individual words. This step and the previous step can be interchangeable, as their inputs and outputs are independent of each other, but the pipeline is presented this way to reflect the order of the reimplementation.

Before the rubric and top scoring answers can be extracted, stop words must be removed. Common prepositional words like "of, a, and by" can cause confusion in the algorithm when ranking tokens based on frequency, thus removing them helps keep the focus on the tokens that provide context towards the answers themselves. Once the text has been cleaned, then the list of student answers are tokenized through Spacy's NLP API. This step was not required for generating the word-order graph, because that algorithm focuses only on creating edges from nouns, verbs, adjectives, and adverbs, utilizing a POS-tagger, thus, the stop-words would already be ignored.

In this counting algorithm a simple hash map is utilized to keep track of how many times each token has been mentioned in all the student answers of a prompt. The input is a list of all the top-scoring student responses. With stop words removed, the only tokens remaining are those that provide context. Specifically, words that actually provide semantic meaning to a sentence like the subject of a sentence, a descriptor of a noun, or an action verb. A threshold value was created based on the average count of each token. This average count is calculated using the dictionary that stored all the counts of each distinct token and taking the total number of tokens (the sum of all the frequencies of each distinct token) and dividing it by the number of distinct tokens (the number of

keys within the count map). Only tokens that exceed the average count would be included in the token extraction, as this was the criteria that Ramachandran deemed as “frequent.”

Section 4.4: Generate Synonyms

The next step is to generate synonym classes (Ramachandran loosely used the term equivalence classes) for the most frequent tokens and extracted phrases. This step aids in counteracting one of the weaknesses of regular expressions: regular expressions look for exact matches. Students may write their answers in different ways, but can still achieve the same score. One way this can happen is through synonyms. Because there can be different words that have the same meaning, a regular expression may overlook this synonym, because of its ability to only search for exact matches. This step will target the frequent tokens that were derived from the previous step and use the WordNet library to append synonyms to each token, in order to maximize the number of ways an answer can be represented.

This is done utilizing Python’s WordNet Library, where each token is paired with synonyms whose “similarity score” (a score generated by the WordNet Library) surpassed a certain threshold. These similar tokens then get appended onto the original tokens and returned back to the pipeline.

Section 4.5: Generating the Regular Expressions

Now that the lists of phrases and tokens are created, the file of regular expressions can be constructed. Both lists are appended to create a sing input for the function to parse through and generate patterns. For each phrase, the beginning and end is wrapped with “.*.” In regular

expressions, ".*" represents that any character repeated any number of times can exist, before the rest of the phrase. Thus, by surrounding each phrase or token with ".*" when building the regular expression, it allows for any student response with that given phrase to be matched.

Another key aspect of generating the regular expressions is adding "|" (bitwise-or) symbols for each of the equivalent tokens generated. For example, the phrase "going fast" can generate the regular expression "(?=.*(going fast|quick|swift).*).", where the regular expression could successfully match "...going fast…", "...going quick…", and "...going swift…", all with one expression. The phrase is surrounded with the characters ".*", to represent that this phrase can occur anywhere within a sentence. Once this conversion happens, and a list of these regular expressions are generated, they are written out into a file, which finally terminates the program. Then the algorithm repeats this process for all ten prompts.

Chapter 5

Results of Ablation Experiments

This chapter will discuss how Tandalla's Random Forest model was reimplemented to utilize AutoP's constructed features, along with the results from the ablation study with different combinations of the feature types. Specifically, this chapter will describe how the regular expression files get converted into one of the many types of features Tandalla utilizes in his Random Forest models, and describe the reimplementation of the models in Python and the ablation study conducted to test the power of the regular expression features generated from AutoP and its comparison with other types of features. Once the regular expressions from AutoP were constructed, Tandalla's model needed to be modified to adapt to the new regular expression. Prior to reimplementation, Tandalla's preprocessing algorithm was hard coded to only generate features from his manually generated regular expressions. The following four sections of this chapter will go over how preprocessing is used to filter student answers, how features are generated, the reimplementation of the random forest model, and the results from the ablation experiment conducted on a combination of features with the model.

Section 5.1: Preprocessing

Before features can be generated, all student answers are initially filtered through a spell check algorithm and a stemmer algorithm. Due to the possibility of students misspelling certain words in their answer, it is important to filter each student answer with a spell check algorithm, to ensure that the model does not get confused by two different spellings of a word. The specific spell

check algorithm utilized is Peter Norvig's spell check algorithm, which uses language model probabilities. The algorithm converts a word to its correctly spelled form by leveraging a large language model and using word sequence probabilities to determine which word would take the smallest number of corrections and highest chance of being used (Norvig).

Because words, especially verbs, can have different suffixes but also have the same meaning, a stemming algorithm is used on each student's answer, which normalizes different word forms to the same word stem. In particular the Porter Stemmer algorithm is used, which iterates through each word and strips the suffixes off of each word (Karaa 2013). Both the spell correction and stemming are done to help normalize the student answers that will be used for feature generation by ensuring consistency among the different spellings and word forms used.

Section 5.2: Feature Generation

The filtered student answers are now fed into Tandalla's feature extraction algorithms. The first major feature Tandalla extracts is general information about each student's answer: number of words, number of sentences, and sentence average size.

The next group of features are n-gram features, specifically unigrams, bigrams, and trigrams. These features track the number of times words and sequences of words appear together in a text. Once the n-grams have their appearances counted, they can be defined as Tandalla's "bag of word" (BOW) features. BOW is a representation of text that provides counts for each word that appears in that text. Tandalla assembles these bags-of-words from the training data and calculates the number of times they appear within all the student answers, essentially creating a matrix of counts for each n-gram. Tandalla splits unigrams (single words), bigrams (sequence of two words),

and trigrams (three words in a row) into their own unique features and stores them into files for later use.

Next, Tandalla also calculates the term frequency-inverse document frequency (tf-idf) for each of the unigram, bigram, and trigram features (Tandalla 2012). The tf-idf is a statistic that calculates the weight each of these n-gram has on a current document with respect to the entire text. Larger tf-idf values indicate that the n-gram appears frequently in the current document, but not as frequently in the overall text. Similarly, a smaller tf-idf value either indicates that the n-gram either does not appear as much in the current document or appears frequently both in the current document and overall text. These are another representation of determining the significance of each of the bag of words features, where unlike specific word counts, tf-idf focuses on each bag-of-words' impact on an entire text, which in this case would be all the training samples.

Finally, Tandalla uses his manually generated regular expressions to create features based on the number of occurrences of each regular expression pattern in the student answers. A large matrix is constructed where all the rows represent student answers, and all the columns represent different regular expression patterns. Each cell within that matrix is either a 0 or 1, depending on if the pattern exists within the student answer. Through this variety of features generated, Tandalla wanted to utilize a rich set of features to represent the same student answers in several different ways, to help the machine learner make more informed decisions on the text. For example, if the model was only using bag-of-words features, a lengthier answer with a few more keywords may perform better than an efficiently shorter answer with all the keywords used accurately and precisely, due to BOW only accounting for n-gram count. Implementing tf-idf helps account for that bias, as tf-idf compares the frequency (weight) of each n-gram with respect to all student

answers as a whole. After these feature types have been collected and stored in individual files for each prompt, they are fed into the Random Forest Models.

Section 5.3: Reimplementing the Random Forest Model

In Tandalla's codebase, the Random Forest models are written in R, deviating from his preprocessing and feature generation algorithms that were written in Python. The main issue with Tandalla's codebase was it was over 10 years old and utilizing old Python 2 libraries and merging with legacy R code when constructing the models. To ensure that all the libraries and techniques were up to date all of Tandalla's model was reimplemented in Python 3, including all of the model code in R, to streamline and update the codebase.

The first step Tandalla took before feeding all the features into his model, was to apply Boruta's feature selection algorithm. A feature selection algorithm filters through a large number of features, which in this case would be the bag of words, tf-idf, and regular expression features, and selects which of those features provide the largest significance towards the accuracy of the model.

Boruta's algorithm in particular, is implemented as a wrapper around a random forest classification model (Kursa 2010). In contrast to other feature selection methods, the idea of a wrapper is to train a model with only a subset of the features to draw inferences on which features should be incorporated into the final model. The general idea behind this algorithm is to compare the true feature values with noisy feature values, to verify that the true feature values help the learner, on the assumption that the noisy values do not provide information that is useful. The idea behind this is if a feature is not significant enough to perform against shuffled features, it should not be incorporated into the model. The algorithm begins by duplicating the data and randomizing

the copied values in each column to generate "pseudo features." Then it trains the random forest and measures the importance of each feature through Mean Decrease Accuracy. Mean Decrease Accuracy, as its name suggests, measures how much of the accuracy of the model is lost from excluding particular features. The higher the Mean Decrease Accuracy, the more significant a feature is to the model's accuracy. Thus the Mean Decrease Accuracy values along with the Z-scores (the number of standard deviations away each Mean Decrease Accuracy score is from the mean) of these values are calculated for all the features (the original and the shadow features). After these are calculated, each of the original features' Z-score is compared with the maximum of all of the shadow features' Z-score. If an original feature exceeds this value, it is called a "hit" and the original feature is considered significant enough to add to the final model. This process iterates a set number of times, and if certain original features cannot manage to get a "hit" by the end of these iterations they are dropped from the feature selection process. The algorithm finally outputs a table of features that were "hits", and those features are used in Tandalla's final Random Forest model.

Implementing Boruta's feature selection algorithm in Python, however, did not perform well. Utilizing the Boruta library in Python, the quadratic weighted kappa, when running the feature selection prior to training the model, was an average of 0, with only five features reported as significant to the model. Thus with these insignificant results, the Boruta selection step was dropped, and all the features were fed into a basic Random Forest Model provided by Python's Scikit Learn library. The following section will go over the results of this experiment.

Section 5.4: Ablation Study Results

By skipping the feature selection and just implementing all the features in a basic Random Forest, performance drastically improved. Figure 10 depicts all the results of the ablation study conducted on the data. The ablation was constructed to see which groups of features provided better performance on the model, including the comparison between the AutoP features and Tandalla's manually generated features. Specifically, all the feature types were all used independently to see which feature type provided the greatest significance in model accuracy. From the table it is evident that n-grams and tf-idf features alone scored the highest, with quadratic weighted kappas of .541 and .536 respectively. Next the regular expression features from AutoP and Tandalla's manually created expressions were incorporated into the n-grams and tf-idf features, with the two best combinations being tf-idf, n-gram, Tandalla's manual regular expressions and just the tf-idf and n-grams, with quadratic weighted kappas of .678 and .651.

Ablation Study Table: Quadratic Weighted Kappa Scores for Different Set of Features on All 10 Prompts

Features	1	2	3	4	5	6	7	8	9	10	All
Reg Ex	.352	-.002	.097	.010	.103	0	0	.087	.248	.069	.166
N-grams	.518	.446	.361	.255	.292	.274	.280	.324	.581	.541	.541
Tf-idf	.491	.545	.452	.220	.332	.313	.155	.221	.614	.530	.536
AutoP	0	0	0	0	0	0	0	0	0	0	0
Tf-idf + N-gram	.646	.627	.514	.380	.327	.441	.323	.431	.728	.615	.651
Tf-idf + N-gram + Reg Ex	.669	.646	.532	.439	.556	.447	.347	.501	.714	.636	.678
Tf-idf + N-gram + AutoP	.553	.417	.430	.508	.186	.446	.160	.281	.619	.477	.533
Tf-idf + N-gram + Reg Ex + AutoP	.614	.531	.385	.398	.560	.636	.301	.366	.667	.510	.621
Baseline (All of Tandalla's original features)	.847	.774	.642	.645	.847	.875	.693	.619	.839	.780	.780

Figure 11. Ablation Study Results

From looking at all feature rows that incorporate n-grams and tf-idf in Figure 11, it is evident that the tf-idf and n-gram (bag of words) features had the greatest positive significance on model performance. Removing tf-idf and n-gram features from the experiment greatly reduced performance as can be seen from the .166 and 0 quadratic weighted kappa scores displayed from Tandalla's regular expressions and AutoP rows respectively (Rows 1 and 4 in Figure 11). Only utilizing Tandalla's manual regular expression features or only utilizing AutoP features provided the worst results, specifically AutoP showing 0 model significance with the worst performance out of the entire ablation. From this result, it is clear that the implementation of AutoP was missing

the high performance that Tandalla's regular expressions provided. Another key indication is when combining n-gram and tf-idf features with the manual patterns they produce better results than when AutoP is paired with n-gram and tf-idf features, again indicating that AutoP performed worse than the manual expressions.

It is also crucial to note that with the reimplemented Random Forest models, the quadratic weighted kappa using all of Tandalla's features was .68, where Tandalla reported .78 in his paper. Replacing Tandalla's regular expression features with the AutoP regular expression features significantly dropped the overall quadratic weighted kappa to .53. The following chapter will provide further insight into why these results may have underperformed when compared to the reported scores in Ramachandran's and Tandalla's papers.

Chapter 6

Discussion/Conclusion

This thesis has addressed replicating various linear machine learning algorithms that were used to perform short answer grading, especially focusing on the reimplementation of AutoP and utilizing regular expression features in Tandalla's Random Forest models. This chapter will discuss the outcomes of the ablation study, and why the reimplementation underperformed in comparison to the reported quadratic weighted kappa scores in both AutoP (Ramachandran 2015) and manually generated regular expressions (Tandalla 2012). This chapter will discuss the significance of regular expression features in context with the reimplementation. The following four sections will split the analysis into the general issues with AutoP, the problems with incorporating the AutoP features with Tandalla's model, the issues with reimplementing Tandalla's Random Forest models, and a final conclusion.

Section 6.1: AutoP Analysis

There were a number of issues that arose when it came to reimplementation, especially with AutoP. The pseudo code provided in Ramachandran's paper explains each step of the algorithm at a high level, but there were some key details left up to interpretation.

The first issue was the preprocessing of the data. In the pseudocode that was provided, it was explained that the rubric text and top scoring answers were used as inputs into the context phrase and token generation steps in the AutoP pipeline. However, there was no mention of how that data was extracted and preprocessed prior to that step. As mentioned in Chapter 4, the rubric

text that was provided as a docx document was not immediately usable by a Python parser. It was not specifically mentioned if Ramachandran utilized only the rubric text at the bottom of the document, or if she also incorporated the prompt that is located above the rubric. Here an assumption was made during the reimplementation that the entire prompt and rubric was used as the input. Similarly, with the top scoring answers it was mentioned that they were extracted as the highest scoring answers from the training set, but there was no breakdown of how many top scoring answers were utilized in the algorithm. Thus, an assumption was made that all of the top scoring answers were used.

However, if the prompt or rubric information used in the context phrase and content token generation steps of AutoP are not directly related to the correct student answers, the generated regular expressions can be cluttered with phrases and tokens that do not have as much significance to the correctness of the answer. This can be seen in the ablation study table provided in Figure 10, where the quadratic weighted kappas of all 10 sets were 0 when AutoP was the sole feature of the model. This clearly indicates that the regular expression features generated by AutoP's implementation had no success in grading student answers on their own.

Another issue with AutoP's reimplementation was its lack of capturing all of the relevant phrases and tokens with respect to Tandalla's manually generated expressions. When comparing the regular expressions Tandalla created by hand, it was clear that each regular expression that Tandalla would create captures a different aspect that the rubric was looking for. Figure 11 provides a screenshot of the first thirty regular expressions that Tandalla generated for prompt 1.

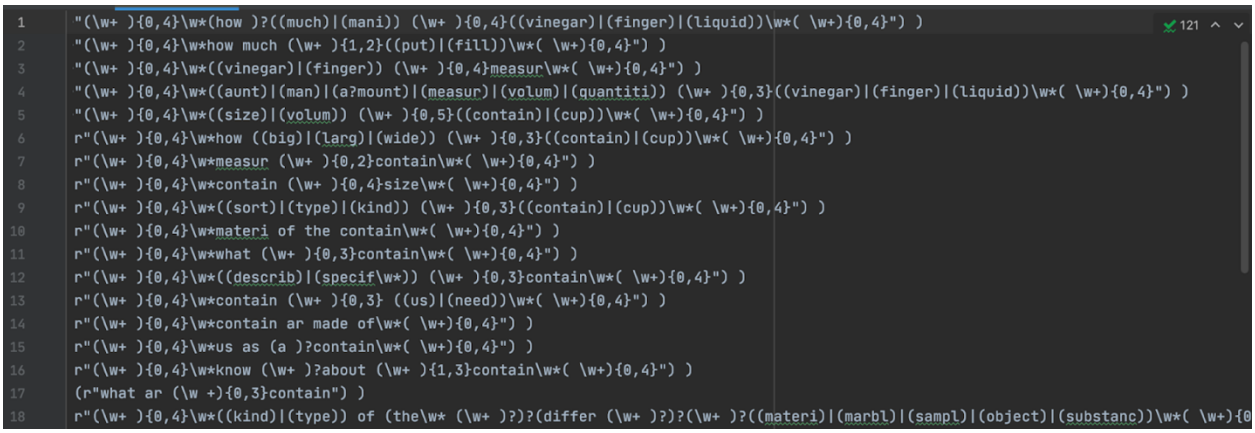


Figure 12. A Sample of Regular Expressions Generated by Tandalla for Prompt 1

From Figure 12, it can be seen how much detail is incorporated into each expression. Tandalla incorporates complex phrases, with sufficient synonyms of the major content tokens. In line 18, the word “sample” is stemmed to just “sampl” and is paired with “substanc” (substance), “materi” (material), and “object” to help provide different variations of the sentence through the use of synonyms. Throughout his expressions, each mention of “is” is paired with “were” and “are” to signify different verb forms students may use. Tandalla rephrases each expression in a different way, to help account for different sentence structures students may have. For only prompt 1, Tandalla manually generated 103 unique regular expressions.

However, when comparing this with the regular expressions that were generated by AutoP’s reimplementation, there is an obvious difference in complexity. Figure 12 depicts the regular expressions of all 30 of the regular expressions that were generated for prompt 1.

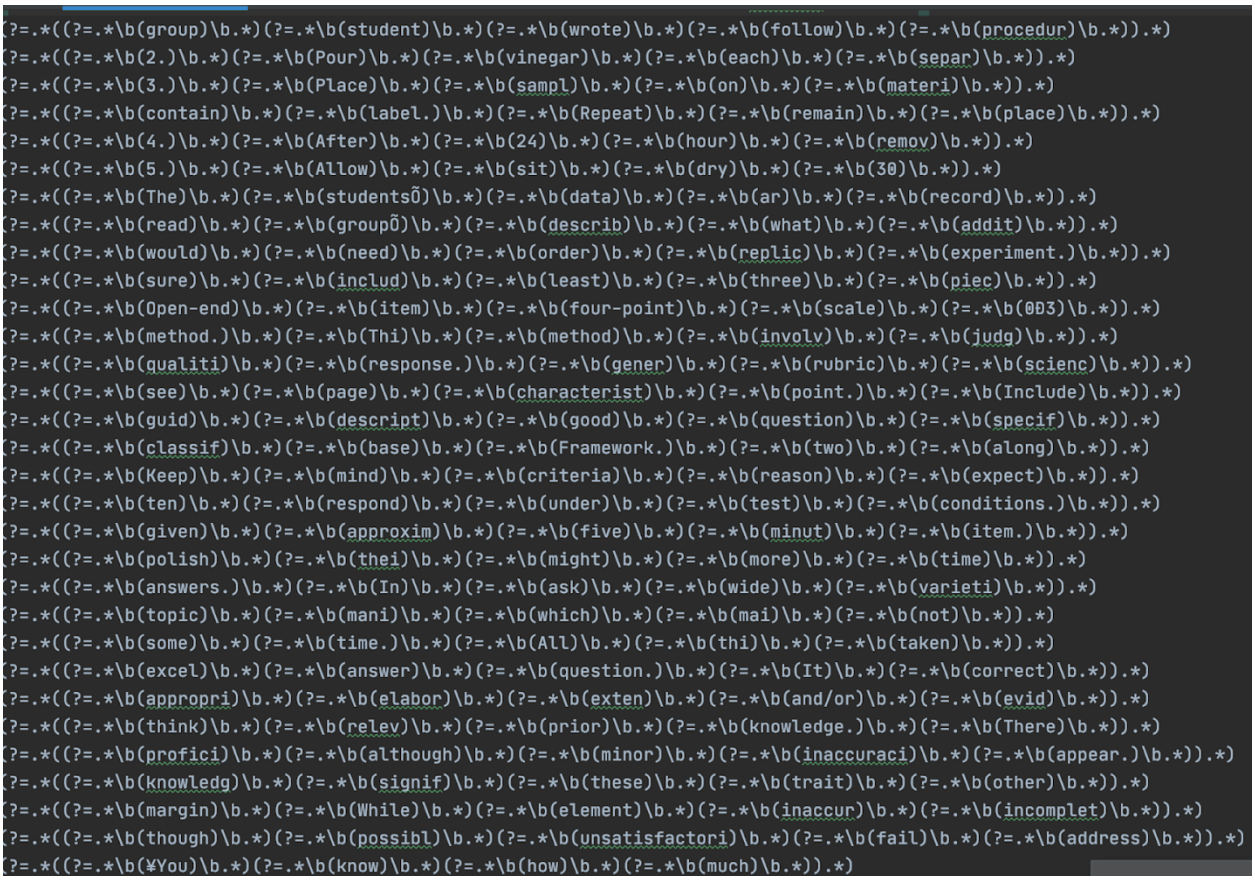


Figure 13. A Sample of Regular Expressions Generated by AutoP's Reimplementation for Prompt 1

In Figure 13, it is immediately apparent that the complexity of the phrases generated are much lower than Tandalla's manually generated expressions. The phrases from Figure 13 only contain a maximum of four to five words, each of which were unable to include synonyms for all the content tokens. The failure to generate synonym tokens was due to the inability to reimplement the synonym generation algorithm with Python's implementation of WordNet. Due to AutoP's original code base utilizing Java's edition of WordNet, it was not immediately apparent how the synonyms were extracted using this library. In Python's implementation of WordNet there is a similarity score provided for each token and its provided synonyms. An arbitrary threshold value is set to only capture synonyms if their similarity score exceeds this threshold value. However,

there was no mention of a threshold value in Ramachandran's paper. Rather, she describes ranking tokens as either "exact, synonym, hypernym/hyponym, meronym/holonym" (i.e. exact matches have the highest "score" and meronyms or holonyms have the lowest score) (Ramachandran 2015). From these rankings, however, it is confusing what she does with these ranking scores in order to actually generate the synonyms for these words.

In Ramachandran's pseudo code there was no mention of how WordNet would actually be used to complete this task, and when viewing Ramachandran's code base of AutoP written in Java, the synonym generation code using WordNet was all commented out, indicating that it was not even used in the final version of her algorithm.

Because there was no confirmation if this aspect of the algorithm was integrated with AutoP, and due to the failure of the reimplementation to capture those synonyms, the regular expressions that were automatically generated were not nearly as strong as those written by Tandalla. This is one of the reasons why AutoP's reimplementation did not contribute to the ablation study results.

Section 6.2: Analysis of Merging with Tandalla's Code

Another reason why AutoP may have underperformed was due to Tandalla's code base being tailored around his manually created regular expressions. When generating features from his regular expressions, there existed functions that would perform the count on the number of times the pattern appeared in each of the student's answers. When incorporating the AutoP regular expressions, it seemed evident that the only major change would be to replace the function that read in Tandalla's regular expressions to AutoP's regular expressions. However, it

was hard to tell if that replacement had any other hidden impacts scattered across his massive code base.

From tracking the process of feeding in the training set to Tandalla's preprocessing algorithm and getting all the feature files as an output, there was no apparent sign that the code was hard coded to only work with Tandalla's manual expressions. However, just because the feature files were successfully generated with the AutoP expressions, it does not mean that those features were generated correctly. This can be seen when looking back at the result of obtaining 0 as the quadratic weighted kappa for only using AutoP features in the ablation study. The fact that the average score 0, would denote that the features themselves may have been corrupted or not outputting correctly. However, when examining these features side by side with Tandalla's manual regular expression features there was no glaring evidence that the features were incorrectly generated. Thus, it was difficult to discern how and where in Tandalla's code the features may have gotten corrupted.

Section 6.3: Analysis of Reimplementing Tandalla's Random Forest

The final step where errors could have occurred in the reimplementation is with Tandalla's Random Forest. As mentioned before, Boruta's feature selection algorithm was not implemented as it stripped away all of the feature columns (originally 14,500 features) to only just use 5 features. This led to an average quadratic weighted kappa of 0, so the feature selection algorithm was abandoned. Instead, all of the features were utilized, and an ablation study was conducted on which groups of those features performed the best.

It was clear that there was something incorrect with the reimplementation of the Random Forest, because Tandalla had reported reaching a quadratic weighted kappa of .70 with only

utilizing regular expressions as features (Tandalla 2012). However, when looking at the ablation study table, and using those same features on the reimplemented model, the model only achieved a quadratic weighted kappa of 0.17.

Aside from not incorporating feature selection, Tandalla averaged two random forest models along with two gradient boosted machines and used that averaged model to generate his predictions. For the ablation study the majority focus was to reimplement a model in Python to see if the AutoP features could be merged with Tandalla's code base. Due to time constraints, reimplementing an additional random forest model and two more gradient boosted machines did not seem productive, as the single random forest model was already performing poorly.

The only minor success from reimplementing the random forest model was from the bag of words features and the tf-idf features, as from the ablation study table they were able to reach quadratic weighted kappa scores of up to 0.678 and 0.621 as the two highest scores in the experiment. This supports the idea that there may have been something wrong with the features that were generated from AutoP and Tandalla's manually generated expressions, as mentioned in Section 6.1 and 6.2.

Section 6.4: Conclusion

From conducting the ablation study and working with regular expression based features in the Random Forest models, it is clear that regular expressions can be useful, however not sufficiently so to perform well on their own. Specifically referencing the ablation study table it was clear that ~~alone~~ Tandalla's manually generated features (QWK of 0.17) performed poorly alone. However when comparing using Bag-of-Words and Tf-idf (QWK of 0.651) with Bag of Words, Tf-idf, and the regular expression features (QWK of 0.672) it is evident that the regular

expression features do help increase the accuracy of the model. The issue with Tandalla's method of creating regular expressions is time. Rarely is it possible to write hundreds of regular expression patterns tailored to each prompt within a dataset to incorporate as features within a model. Although AutoP attempted to mitigate that issue with automating this process, it is clear that from the reimplementation it still requires a lot of work. The strength of AutoP is highly dependent on the rubric data and top scoring answers that are fed in, so if these are not able to be provided in certain datasets, AutoP may not be able to function as well. Additionally, being able to generate synonyms is a crucial step in AutoP, because if a student were to use different wording the generated regular expressions would not be able to find a match, even if the student answered the question correctly. With these two issues resolved, AutoP should be able to construct regular expressions with similar performance as Tandalla's.

This paper examines some of the different approaches when it comes to implementing ML models for the ASAG task. When comparing the performance between linear models like logistic regression and deep neural networks, it has been shown that DNN's tend to have better performance. However, for a deep neural network to be trained requires a large amount of data, which for the ASAG task, is difficult to obtain. This tradeoff between amount of data and model complexity is a prevalent consideration when building these models. Data modification on datasets like the middle school physics dataset become important when finding sufficient data for these complex models. Recent experiments on these unpublished datasets, have shown promising results on some of the lab's deep neural network models when combined with data augmentation techniques to increase the size of the training data (Li, et. al 2021). Linear approaches, on the other hand, which may not require as much data, also may not be complex enough to effectively represent the data. As people continue to work on the ASAG task and acquire more data, the use

of even more complex DNN's will be introduced, and implementing basic linear models will just be utilized as a baseline for comparison.

BIBLIOGRAPHY

Burrows, S., Gurevych, I. & Stein, B. The Eras and Trends of Automatic Short Answer Grading. *Int J Artif Intell Educ* **25**, 60–117 (2015). https://doi.org/10.1007/s40593-014-0026-8

Dzikovska, M., Nielsen, R., Brew, C., Leacock, C., Giampiccolo, D., Bentivogli, L., Clark, P., Dagan, I., and Dang. H. T. 2013. SemEval-2013 Task 7: The Joint Student Response Analysis and 8th Recognizing Textual Entailment Challenge. In Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013), pages 263–274, Atlanta, Georgia, USA. Association for Computational Linguistics.

Karaa, Wahiba Ben Abdessalem, and Nidhal Gribâa. "Information retrieval with porter stemmer: a new version for English." *Advances in Computational Science, Engineering and Information Technology: Proceedings of the Third International Conference on Computational Science, Engineering and Information Technology (CCSEIT-2013), KTO Karatay University, June 7-9, 2013, Konya, Turkey-Volume 1*. Springer International Publishing, 2013.

Kaur, Gaganpreet. "Usage of regular expressions in NLP." *International Journal of Research in Engineering and Technology IJERT* 3.01 (2014): 7

Kursa, Miron B., and Witold R. Rudnicki. "Feature selection with the Boruta package." *Journal*

of statistical software 36 (2010): 1-13.

L. Yao and Y. Guan, "An Improved LSTM Structure for Natural Language Processing," 2018 IEEE International Conference of Safety Produce Informatization (IICSPI), 2018, pp. 565-569, doi: 10.1109/IICSPI.2018.8690387.

Leacock, C., Chodorow, M. C-rater: Automated Scoring of Short-Answer Questions. *Computers and the Humanities* 37, 389–405 (2003). https://doi.org/10.1023/A:1025779619903

Li, Zhaohui, Yajur Tomar, and Rebecca J. Passonneau. 2021. A Semantic Feature-Wise Transformation Relation Network for Automatic Short Answer Grading. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6030–6040, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Li, Zhaohui, Chengnin Zhang, Jin Yumi, and Rebecca J. Passonneau, 2023. Learning When to Defer to Humans for Short Answer Grading. 24th International Conference on Artificial Intelligence in Education.

Norvig, Peter. *How to Write a Spelling Corrector*, Feb. 2007,

https://norvig.com/spell-correct.html.

Olah, Christopher. “Understanding LSTM Networks.” *Colah's Blog*, 27 Aug. 2015, https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Payne, T. (2006). Grammatical relations. In *Exploring Language Structure: A Student's Guide* (pp. 210-236). Cambridge: Cambridge University Press. doi:10.1017/CBO9780511806483.010

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014.

Ramachandran, Lakshmi, and Edward F. Gehringer. "A word-order based graph representation for relevance identification." *Proceedings of the 21st ACM international conference on Information and knowledge management*. 2012.

Ramachandran, Lakshmi, Jian Cheng, and Peter Foltz. "Identifying patterns for short answer scoring using graph-based lexico-semantic text matching." *Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications*. 2015.

Regular Expressions, The Open Group Base Specifications Issue 6, 2004, https://pubs.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap09.html.

Tandalla, Luis. *Scoring Short Answer Essays*. Kaggle, Sept. 2012, https://storage.googleapis.com/kaggle-competitions/kaggle/2959/media/TechnicalMethodsPaper.pdf.

Warrens, Matthijs J. "Five ways to look at Cohen's kappa." *Journal of Psychology & Psychotherapy* 5.4 (2015): 1.

X. Zhu, H. Wu and L. Zhang, "Automatic Short-Answer Grading via BERT-Based Deep Neural Networks," in *IEEE Transactions on Learning Technologies*, vol. 15, no. 3, pp. 364-375, 1 June 2022, doi: 10.1109/TLT.2022.3175537.

ACADEMIC VITA

Yajur Tomar

Schreyer Honors College
Pennsylvania State University, University Park, PA **Aug 2019 – Current**
Bachelor: Computer Science
Minors: Statistics, Engineering Leadership Development

Work Experience

AI Intern Pathways Program - Lockheed Martin, Moorestown, NJ May 2022-Aug 2022

- **Developed an image classification model to discriminate buildings and background from satellite images utilizing Python, Kubeflow, a UNet Model architecture, and the SpaceNet dataset**
- **Constructed masks by using a rasterizer from the SpaceNet dataset as labels and trained the model to achieve a 91% acc and .92 f1 score**
- **Achieved the Nvidia Fundamental of Deep Learning Certificate upon completion of their training course, utilizing JupyterLab to train Image Classification Models utilizing LSTM, Logistic Regression, and convolutional neural network (CNN) models with MNST and custom datasets**
- **Implemented data augmentation to generate pseudo training samples to provide more data for models to increase training/test accuracy by 5%**

Systems Programming Software Intern - Lockheed Martin, Moorestown, NJ May 2020-Aug 2022

- **Worked with product managers in delivering new features and functionalities using embedded systems, real time programming, and multi-processing functionality through the OpenMPI library in C++ in an agile environment**
- **Conducted code reviews and wrote 15+ unit tests utilizing Google Unit Tests' C++ library to successfully validate inputs and outputs**
- **Diagnosed and debugged system issues in C++ during L2 testing and constructed a GUI using Python's Tkinter library to plot and provide visual analysis in L1 testing**

Automated Regular Expression Generator-Natural Language Processing Researcher, PSU Dec 2020-Current

- **Built Logistic Regression, LSTM, and Manhattan LSTM models using PyTorch libraries to create baseline models for the SemEval Dataset, and achieved a score of .60, .65, .64 quadratic weighted kappa respectively (evaluation metric of ML models)**
- **Developed an algorithm using SpaCy's NLP API to generate patterns based on rubrics to compare with student answers to determine correctness, and achieved a score of .80 quadratic weighted kappa**

Projects

Mobile App Developer - Nittany AI Challenge MVP Finalist Dec 2019-Aug 2020

- **Designed a cross platform mobile application utilizing JavaScript and NativeScript to assist practitioners treating college students with mental health issues, and earned $11,000 in funding through the Nittany AI Challenge**
- **Programmed a prototype for a smart journal with activity recommendations using Google's Tone Analyzer and Maps API, created functioning login pages utilizing Firebase's Authentication and NoSQL database, and deployed a scalable application through GCP's Firestore**