Identifying Patterns For Short Answer Scoring Using Graph-based Lexico-Semantic Text Matching

Lakshmi Ramachandran[1], **Jian Cheng**[2] and **Peter Foltz**[1,3]
[1]Pearson, [2]Analytic Measures Inc., [3]University of Colorado
{lakshmi.ramachandran, peter.foltz}@pearson.com
jian.cheng@analyticmeasures.com

Abstract

Short answer scoring systems typically use regular expressions, templates or logic expressions to detect the presence of specific terms or concepts among student responses. Previous work has shown that manually developed regular expressions can provide effective scoring, however manual development can be quite time consuming. In this work we present a new approach that uses word-order graphs to identify important patterns from human-provided rubric texts and top-scoring student answers. The approach also uses semantic metrics to determine groups of related words, which can represent alternative answers. We evaluate our approach on two datasets: (1) the Kaggle Short Answer dataset (ASAP-SAS, 2012), and (2) a short answer dataset provided by Mohler et al. (2011). We show that our automated approach performs better than the best performing Kaggle entry and generalizes as a method to the Mohler dataset.

1 Introduction

In recent years there has been a significant rise in the number of approaches used to automatically score essays. These involve checking grammar, syntax and lexical sophistication of student answers (Landauer et al., 2003; Attali and Burstein, 2006; Foltz et al., 2013). While essays are evaluated for the quality of writing, short answers are brief and evoke very specific responses (often restricted to specific terms or concepts) from students. Hence the use of features that check grammar, structure or organization may not be sufficient to grade short answers.

Regular expressions, text templates or patterns have been used to determine whether a student answer matches a specific word or a phrase present in the rubric text. For example, Moodle (2011) allows for the use of a "Regular Expression Short-Answer question" type which allows instructors or question developers to code correct answers as regular expressions. Consider the question: "What are blue, red and yellow?" This question can evoke a very specific response: "They are colors." However, there are several ways (with the term "color" spelled differently, for instance) to answer this question. E.g. (1) they are colors; (2) they are colours; (3) they're colours; (4) they're colors; (5) colours; or (6) colors. Instead of having to enumerate all the alternatives to this question, the answer can be coded as a regular expression: `(they('|\s(a))re\s)?colo(u)?rs`.

Manually generated regular expressions have been used as features in generating models that score short answers in the Kaggle Short Answer Scoring competition (ASAP-SAS, 2012). Tandalla (2012)'s approach, the best performing one of the competition, achieved a Quadratic Weighted (QW) Kappa of 0.70 using just regular expressions as features. However, regular expression generation can be tedious and time consuming, and the performance of these features is constrained by the ability of humans to generate good regular expressions. Automating this approach would ensure that the process is repeatable, and the results consistent.

We propose an approach to identify patterns to score short answers using the rubric text and top-scoring student responses. The approach involves

Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications, 2015, pages 97–106,
Denver, Colorado, June 4, 2015.

(1) identification of classes of semantically related words or phrases that a human evaluator would expect to see among the best answers, and (2) combining these semantic classes in a meaningful way to generate patterns. These patterns help capture the main concepts or terms that are representative of a good student response. We use a word order graph (Ramachandran and Gehringer, 2012) to represent the rubric text. The graph captures order of tokens in the text. We use a lexico-semantic matching technique to identify the degree of relatedness across tokens or phrases. The matching process helps identify alternate ways of expressing the response.

An answer containing the text *diet of koalas* would be coded as follows: `(?=.*(diet|eat(s)?|grub).*) of (?=.*(koala(s)?|koala|opossum).*)`. The patterns generated contain (1) positional constraints (`?=`, which indicates that the search for the text should start at the beginning, and (2) the choice operator (`|`), which captures alternate ways of expressing the same term, e.g. `diet` or `eat` or `grub`. We look for match (or non-match) between the set of generated patterns and new short answers.

We evaluate our patterns on short answers from the Kaggle Automated Student Assessment Prize (ASAP) competition, the largest publicly available short answer dataset (Higgins et al., 2014). We compare our results with the those from the competition's best model, which uses manually generated regular expressions. Our aim with this experiment is to demonstrate that automatically generated patterns produce results that are comparable to manually generated patterns. We also tested our approach on a different short answer dataset curated by Mohler et al. (2011).

One of the main contributions of this paper is the use of an automated approach to generate patterns that can be used to grade short answers effectively, while spending less time and effort. The rest of this paper is organized as follows: Section 2 discusses related work that use manually constructed patterns or answer templates to grade student responses. Section 3 contains a description of our approach to automatically generate patterns to grade short answers. Sections 4 and 5 discuss the experiments conducted to evaluate the performance of our patterns in scoring short answers. Section 6 concludes the paper.

2 Related Work

Leacock and Chodorow (2003) developed the use of a short-answer scoring system called C-rater, which focuses on semantic information in the text. They used a paraphrase-recognition based approach to score answers.

Bachman et al. (2002) proposed the use of a short answer assessment system called WebLAS. They extracted regular expressions from a model answer to generate the scoring key. Regular expressions are formed with exact as well as near-matches of words or phrases. Student answers are scored based on the degree of match between the answer and scoring key. Unlike Bachman et al., we do not use patterns to directly match and score student answers. In our approach, text patterns are supplied as features to a learning algorithm such as Random Forest (Breiman, 2001) in order to accurately predict scores.

Mitchell et al. (2003) used templates to identify the presence of sample phrases or keywords among student responses. Marking schemes were developed based on keys specified by human item developers. The templates contained lists of alternative (stemmed) tokens for a word or phrase that could be used by the student. Pulman and Sukkarieh (2005) used hand-coded patterns to capture different ways of expressing the correct answer. They automated the approach of template creation, but the automated ones did not outperform the manually generated templates. Makatchev and VanLehn (2007) used manually encoded first-order predicate representations of answers to score responses.

Brill et al. (2002) reformulated queries as declarative sentence segments to aid query-answer matching. Their approach worked under the condition that the (exact) content words appearing in a query would also appear in the answer. Consider the sample query "When was the paper clip invented?", and the sample answer: "The paper clip is a very useful device. It was patented by Johan Vaaler in 1899." The word *patented* is related in meaning to the term *invented*, but since the exact word is not used in the query, it will not match the answer. We propose a technique that uses related words as part of the patterns in order to avoid overlooking semantically close matches.

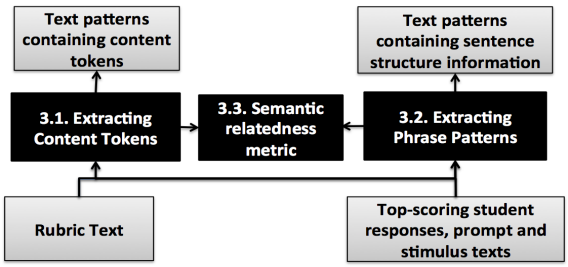

Figure 1: An overview of the approach.

3 Approach

In this section we describe our approach to automatically identify text patterns that are representative of the best answers. We automatically generate two types of patterns containing: (1) content words and (2) sentence structure information. We use the rubric text provided to human graders and a set of top-scored student answers as the input data to generate patterns. Top-scoring responses are those that receive the highest human grades. In our implementation we use the top-scored answers from the training set only. Figure 1 depicts an overview of our approach to automate pattern generation.[1,2]

3.1 Extracting Content Tokens

We rewrite the rubric text in order to generate a string of content words that represent the main points expected to appear in the answer. The aim of our approach is to generate patterns with no manual intervention. The re-writing of the rubric is also done automatically. It involves the removal of stopwords while retaining only content tokens.

We eliminate stopwords and function words in the text and retain only the important prompt-specific content words. Short answer scoring relies on the presence or absence of specific tokens in the student's response. Content tokens are extracted from sample answers, and the tokens are grouped together without taking the order of tokens into consideration.

Students may use words different from those used in the rubric (e.g. synonyms or other semantically related words or phrases). Therefore we have to identify groups of words or phrases that are semantically related. In order to extract semantically similar words specific to the prompt's vocabulary, we look for related tokens in top-scoring answers as well as in the prompt and stimulus texts.

3.1.1 Semantic Relatedness Metric

We use WordNet (Fellbaum, 1998) to determine the degree of semantic match between tokens because it is faster to query than a knowledge resource such as Wikipedia. WordNet has been used successfully to measure relatedness by Agirre et al. (2009).

Match between two tokens could be one of: (1) exact, (2) synonym[3], (3) hypernym or hyponym (more generic or specific), (4) meronym or holonym (sub-part or whole) (5) presence of common parents (excluding generic parents such as *object*, *entity*), (6) overlaps across definitions or examples of tokens i.e., using context to match tokens, or (7) distinct or non-match. Each of these matches expresses different degrees of semantic relatedness across compared tokens. The seven types of matches are weighted on a scale of 0 to 6. An exact match gets the highest weight of 6, a synonym match gets a weight of 5 and so on, and a distinct or non-match gets the least weight of 0.

In the pattern `(?=.*(larg(e)?|size|volum(e)?).*)(?=.*(dry).*)(?=.*(surface).*)`, the set `(?=.*(larg(e)?|size|volum(e)?).*)` contains semantically related alternatives. The pattern looks for the presence of three tokens: any one of the tokens within the first `(?=.*···*)` and tokens `dry` and `surface`. These tokens do not have to appear in any particular order within the student answer. A combination of these tokens should be present in a student answer for it to get a high score. Steps involved in generating content tokens based patterns for the text "size or type of container to use" are described in Algorithm 1.

3.2 Extracting Phrase Patterns

In order to capture word order in the rubric text we extract subject–verb, verb–object, adjective–noun, adverb–verb type structures from the sample answers. The extraction process involves generation of

[1]Prompt: Writing prompt provided to help guide students.

[2]Stimulus: Text presented to students, in addition to the writing prompt, to provide further writing guidance.

[3]We use the part-of-speech of a token to extract the synset from WordNet. This, to an extent, helps disambiguate the sense of a token.

Input: Rubric text, top-scoring answers, and prompt and stimulus texts (if available)
Output: Patterns containing unordered content words.
for *each sentence in the rubric text* **do**
`/* Rubric text: "size or type of container to use" */`
1. Remove stopwords or relatively common words.
`/* Output: size type container use*/`
2. Rank tokens in top-scoring answers, and prompt and stimulus texts based on their frequency, and select the top most frequent tokens.
`/* size container type*/`
3. Identify classes of alternate tokens, for each rubric token, from among most frequent tokens (from Step 2).
`/* {size, large, mass, thing, volume} {container, cup, measure} {type, kind}*/`
4. Stem words and use the suffix as an alternative
`/* container→ (stem: contain, suffix: er) →contain(er)?/`
5. Generate the pattern by AND-ing each of the classes of words.
`/* (?=.*(large|mass|size|thing|volume).*)(?=.*(contain(er)?|cup|measure).*)(?=.*(kind|type).*)`
end

Algorithm 1: Generating patterns containing unordered content tokens.

word-order graph representations for the sample answers, and extracting edges representing structural relations listed above.

Generating word-order graphs: We use word-order graphs to represent text because they contain the ordering of words or phrases, which helps capture context information. Context is not available when using just unigrams.

Word graphs have been found to be useful for the task of determining a review's relevance to the submission. Word-order graphs' f-measure on this task is 0.687, while that of dependency graphs is 0.622 (Ramachandran and Gehringer, 2012). No approach is highly accurate, but word graphs work well for this task.

Structure information is crucial in a pattern-

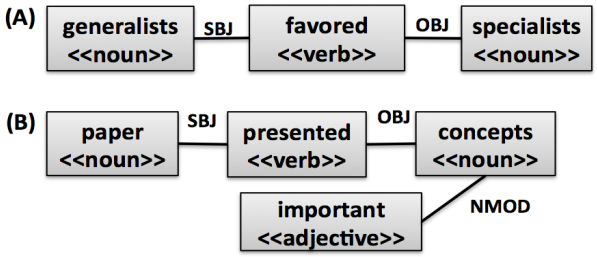

Figure 2: Word-order graphs for texts (A) "Generalists are favored over specialists" and (B) "The paper presented important concepts." Edges in a word-order graph maintain ordering information, e.g. generalists–are favored, paper–presented, important–concepts.

Input: Rubric text, top-scoring answers, and prompt and stimulus texts (if available)
Output: Patterns containing ordered word phrases.
for *each sentence in the rubric text* **do**
`/* Rubric text: "···particles like sodium, potassium ions into membranes···"`
1. Generate word-order graphs from the text, and extract edges from the word-order graph.
`/* The extracted segment: particles like--sodium potassium--ions into membranes. Graph edges are connected with a "--"`
2. Replace stopwords or function words with `\w{0,4}`.
`/* The segment becomes: particles(\s\w{0,4}\s){0,1} sodium potassium ions(\s\w{0,4}\s){0,1}membranes`
3. Rank tokens in top-scoring answers and prompt and stimulus texts based on their frequency, and select the top most frequent tokens.
4. Identify class of alternate tokens, for each rubric token, from among most frequent tokens.
5. Add all synonyms of the rubric token from WordNet to the class of alternatives.
`/* E.g. class of alternate tokens for` *sodium*`: {potassium, bismuth, zinc, cobalt}, for` *potassium*`: {tungsten, zinc, calcium, iron, aluminum, tin}, for` *membrane*`: {film, sheet}`
6. Stem words and generate pattern by AND-ing all classes of words.
end

Algorithm 2: Generating patterns containing sentence structure or phrase pattern information.

generation approach since some short answers may capture relational information. Consider the answer: "Generalists are favored over specialists", to a question on the differences between generalists and specialists. A pattern that does not capture order of terms in the text will not capture the relation that exists between "generalists" and "specialists". Figure 2(A) contains the graph representation for this text.

During graph generation, each sample text is tagged with parts-of-speech (POS) using the Stanford POS tagger (Toutanova et al., 2003), to help identify nouns, verbs, adjectives, adverbs etc. For each sample text consecutive noun components, which include nouns, prepositions, conjunctions and Wh-pronouns are combined to form a noun vertex. Consecutive verbs (or modals) are combined to form a verb vertex; similarly with adjectives and adverbs. When a noun vertex is created the generator looks for the last created verb vertex to form an edge between the two. When a verb vertex is found, the algorithm looks for the latest noun vertex to create a noun–verb edge. Ordering is maintained when an edge is created i.e., if a verb vertex was formed before a noun vertex a verb–noun edge is created, else a noun–verb edge is created. A detailed description of the process of generating word-order graphs is available in Ramachandran and Gehringer (2012).

For this experiment we do not use dense representations of words (e.g. Latent Semantic Analysis (LSA) (Landauer, 2006)) because they are extracted from a large, general corpus and tend to extend the meaning of words to other domains (Foltz et al., 2013). In place of a dense representation we use word-order graphs, since they capture order of phrases in a text.

Substituting stopwords with regular expressions: Stopwords or function words in the extracted word phrases are replaced with the regular expression `(\s\w{0,x}\s){0,n}` where `x` indicates the length of the stopwords or function words, and `n` indicates the number of stopwords that appear contiguously. We use `x=4`, and `n` can be determined while parsing the text. We allow for 0 occurrences of stopwords (`in {0,n}`) between content tokens. Some students may not write grammatically correct or complete answers, but the answer might still contain the right order of the remaining content words, which helps them earn a high score.

Identifying semantic alternatives for content words: Just as in the case of tokens-based patterns (Section 3.1), semantically related words are identified to accommodate alternative responses (relatedness metric described in Section 3.1.1). Tokens in top-scoring answers and prompt texts are ranked based on their frequency, and the most frequent tokens are selected for comparison with words in the rubric text. Apart from that we also add other synonyms of the token to the class of related terms. For instance some synonyms of the token `droplets` are `raindrops, drops`, which are added to its class of semantically related words.

Stemming accommodates typos, the use of wrong tenses as well as the use of morphological variants of the same term (containing singular-plural or nominalized word forms). For instance if "s" is missed in "drops", it is handled by the expression "drop(s)?". These are correctly spelled variants of the same token. We use Porter (1980) stemmer to stem words. The final class of words from the example above looks as follows: {`droplet(s)?, driblet, raindrop(s)?, drop(s)?`}. Humans tend to overlook typos as well as difference in tenses. Therefore the trailing "s" is considered optional.

Algorithm 2 describes steps involved in extracting phrase patterns from a sample answer "$\cdots$particles like sodium, potassium ions into membranes$\cdots$". Output of Algorithm 2 is: `particles(\s\w{0,4}\s){0,1}(?=.*(sodium|potassium|bismuth|zinc|cobalt).*)(?=.*(potassium|tungsten|zinc|calcium|iron|aluminum|tin).*)ions(\s\w{0,4}\s){0,1}(?=.*(membrane|film|sheet).*).` These patterns are also flexible like the token-based ones (with the presence of positional constraints), but it expects content words such as `particles, sodium, potassium, ions` and `membrane` to appear in the text, in that order.

4 Kaggle Short Answer Dataset

The aim of the Kaggle ASAP Short Answer Scoring competition was to identify tools that would help score answers comparable to humans (ASAP-SAS, 2012). Short answers along with prompt texts (and in some cases sample answers) were made avail-

able to competitors. The dataset contains 10 different prompts scored on either a scale of 0–2 or 0–3. There were a total of 17207 training and 5224 test answers. Around 153 teams participated in the competition. The metric used for evaluation is QW Kappa. The human benchmark for the dataset was 0.90. The best team achieved a score of 0.77.

4.1 Tandalla's Approach

Tandalla (2012)'s was the best performing model at the ASAP-Short Answer Scoring competition. One of the important aspects of Tandalla's approach was the use of manually coded regular expressions to determine whether a short answer matches (or does not match) a sample pattern. Specific regular expressions were developed for each prompt set, depending on the type of answers each set evoked (e.g. presence of words such as "alligator", "generalist", "specialist" etc. in the text). These patterns were entirely hand-coded, which involved a lot of manual effort. Tandalla built a Random Forest model with the regular expressions as features. This model alone achieved a QW Kappa of 0.70. Tandalla also manually labeled answers to indicate match with the rubric text. A detailed description of the best performing approach is available in Tandalla (2012).

4.2 Experiment

Our aim with this experiment is to compare system-generated patterns with Tandalla's manually generated regular expressions. The goal is to determine the scoring performance of automated patterns, while keeping everything (but the regular expressions) in the best performing approach's code constant.

We substituted the manual regular expressions used by Tandalla in his code with the automated patterns. We then ran Tandalla's code to generate the models and obtain predictions for the test set. We evaluate our approach on each of the 10 prompt sets from the Kaggle short answer dataset.

The final predictions produced by Tandalla's code is the average of four learning models' (two Random Forests and two Gradient Boosting Machines) predictions. The learners were used to build regression (and not discrete) models. We used content tokens and phrase patterns to generate two sets of predictions, one for each run of Tandalla's code. We stacked the output by taking the average of the two sets of predictions.

We compare our model with the following:

1. *Tandalla's model with manually generated regular expressions:* This is the gold standard, since manual regular expressions were a part of the best performing model.

2. *Tandalla's model with no regular expressions:* This model constitutes a lower baseline since the absence of any regular expressions should cause the model to perform worse. Since the code expects Boolean regular expression features as inputs, we generated a single dummy regular expression feature with all values as 0 (no match).

4.3 Results

From Table 1 we see that Tandalla's base code along with our patterns' stacked output performs better than the manual regular expressions. On 8 out of the 10 sets our patterns perform better than the manual regular expressions. Their performance on the remaining 2 sets is better than that of the lower baseline i.e., Tandalla's code with no regular expressions.

The mean QW Kappa achieved by our patterns is 0.78 and that achieved by Tandalla's manual regular expressions is 0.77. Although the QW Kappas are very close (i.e. the difference is not statistically significant), their unrounded difference of 0.00530 is noteworthy as per Kaggle competition's standards. For instance the difference between the first and second place teams (Luis Tandalla and Jure Zbontar) in the competition is 0.00058.[4]

4.4 Analysis of Behavior of Regular Expressions

While the overall performance of the automated regular expressions is better than Tandalla's manual regular expressions, there are some aspects that it may be lacking in when compared with the manual regular expressions.

In the case of Sets 5 and 7, the stacked model performs worse than the model that uses manual regular expressions. This indicates that the manual regular expressions play a very important role for these

[4]Kaggle Public Leaderboard `https://www.kaggle.com/c/asap-sas/leaderboard/public`

Table 1: Comparing performance of models on the test set from the Kaggle ASAP competition. The table contains QW Kappas for each of the ten prompts in the dataset. AutoP: Stacked patterns model. Tandalla's: Tandalla's model with manually generated regular expressions; Baseline: Tandalla's model with no regular expressions.

Approach	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 8	Set 9	Set 10	Mean
AutoP	**0.86**	**0.78**	**0.66**	**0.70**	0.84	**0.88**	0.66	**0.63**	**0.84**	**0.79**	**0.78**
Tandalla's	0.85	0.77	0.64	0.65	**0.85**	0.88	**0.69**	0.62	0.84	0.78	0.77
Baseline	0.82	0.76	0.64	0.66	0.80	0.86	0.63	0.59	0.82	0.76	0.75

prompts. In the case of set 5, the prompt evokes information on the movement of mRNA across the nucleus and ribosomes. We found that:

1. The answers discuss the movement of mRNA in a certain direction, e.g. out of (exit) the nucleus and into the (entry) ribosome. Although students may mention the content terms such as nucleus and ribosome correctly, they tend to miss the directionality (of the mRNA). Since terms such as *into, out of* etc. are prepositions or function words, they get replaced, in our automated approach by `\w{0,x}`. Hence, if the student answer mentions "the mRNA moved *into* the nucleus" as opposed to saying "*out of* the nucleus", our pattern would incorrectly match it.

2. Another reason why automated regular expressions do not perform well is that WordNet treats terms such as *nucleus* and *ribosome* as synonyms. As a result when students interchange the two terms, the regular expression finds incorrect matches. For example an automated pattern for the text "travels from the cytoplasm into the ribosome" is represented as `travels(\s\w{0,4}\s){0,2}(?=.*(cytoplasm|endoplasm(ic)?).*)(\s\w{0,4}\s){0,2}(?=.*(ribosom(e)?|nucleu(s)?).*)`. An incorrect student answer containing "$\cdots$ mRNA travels from the cytoplasm into the *nucleus* $\cdots$" will match this pattern.

As described above we found that retaining stopwords (e.g. prepositions such as "into" or "out of") in the regular expressions may be useful in the case of some prompts. Our approach to regular expression generation may be tweaked to allow the use of stopwords for some prompts. However, our aim is to show that with a generalized approach (in this case one that excludes stopwords) our system performs better than Tandalla's.

In the case of prompt 7, the answers are expected to contain a description of the traits of a character named Rose, as well as an explanation on why students thought that the character was caring. An automated pattern such as: `(?=.*(hard|difficult).*)(?=.*(work(ing)?).*)` captures some of Rose's traits. The answer "Rose was a very hard working girl. She felt really lonely because her dad had just left and her mother worked most of the day." matches the above pattern. However the explanation provided by the student in the second sentence is *not* correct. This answer was awarded a score of 1 by the human grader, but was given a 2 by the system. Although the pattern succeeds in capturing partial information, it does not capture the explanation correctly for this prompt.

5 Mohler et al. (2011)'s Short Answer Dataset

In this section we evaluate our approach on an alternate short answer scoring dataset generated by Mohler et al. (2011). The aim is to show that our method is not specific to a single type of short answer, and could be used successfully on other datasets to build scoring models.

Mohler et al. use a combination of graph-based alignment and lexical similarity measures to grade short answers. They evaluate their model on a dataset containing 10 assignments and 2 examinations. The dataset contains 81 questions with a total of 2273 answers. The dataset was graded by two human judges on a scale of 0–5. Human judges have an agreement of 57.7%.

Mohler et al. apply a 12-fold cross validation over the entire dataset to evaluate their models. On average, the train fold contains 1894 data points

Table 2: Sample questions from a single assignment. Questions in this assignment are about sorting techniques. Since they discuss the same subject a single model can be built for the assignment.

1. In one sentence, what is the main idea implemented by insertion sort?
2. In one sentence, what is the main idea implemented by selection sort?
3. What is the number of operations for insertion sort under a best-case scenario, and what is the best-case scenario?
4. What is the base case for a recursive implementation of merge sort?

while the test fold contains 379 data points. Models are constructed with data from assignments containing questions on a variety of programming concepts such as the role of a header file, offset notation in arrays and the advantage of linked lists over arrays. Although all the questions are from the same domain (e.g. computer programming) the answers they evoke are very different.

Mohler et al. achieved a correlation of 0.52 with the average human grades, with a hybrid model that used Support Vector Machines as a ranking algorithm. The hybrid model contained a combination of graph-nodes alignment, bag-of-words and lexical similarity features. The best Root Mean Square Error (RMSE) of 0.98 was achieved by the hybrid model, which used Support Vector Regression as the learner. The best median RMSE computed across each individual question was 0.86.

5.1 Experiment and Results

We use the same dataset to extract text patterns. Since patterns are prompt or question specific we cannot create models using the entire dataset like Mohler et al. do. Patterns extracted from across different questions may not be representative of the content of individual questions or assignments. Questions within each assignment are on the same topic. Table 2 contains a list of all questions from Assignment 5, which is about insertion, selection and merge sort algorithms. We therefore extract patterns containing content tokens and phrases for each assignment.

The data for each assignment is divided into train and test sets (80% train and 20% test). The train set contains a total of 1820 data points and the test set contains a total of 453 data points. The train data is used to extract content tokens and phrase patterns from sample answers.

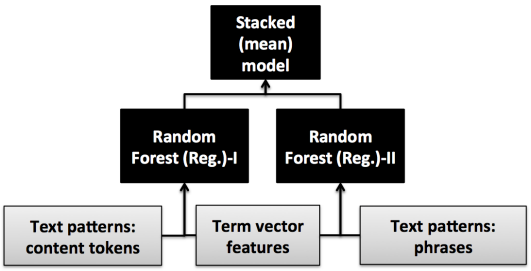

Figure 3: Features used and models built for the experiment on Mohler et al. (2011)'s short answer dataset.

Most short answer grading systems use term vectors as features (Higgins et al., 2014), since they work as a good baseline. Term vectors contain frequency of terms in an answer. We use a combination of term vectors and automatically extracted patterns as features.

We use a Random Forest regressor as the learner to build models. The learner is trained on the average of the human grades. We stack results from models created with each type of pattern to compute final results. Results are listed in Table 3. Our approach's correlation over all the test data is 0.61. The RMSE is 0.86, and the median RMSE computed over questions is 0.77. The improvement in correlation of our stacked model over Mohler et al.'s performance of 0.52 is significant (one tailed test, p-value = 0.02 $<$ 0.05, thus the null hypothesis that this difference is a chance occurrence may be rejected). Correlation achieved by using just term vectors is 0.56 (difference from Mohler et al.'s result is not significant). These results indicate that the use of patterns results in an improvement in performance.

The above process was repeated at the granularity level of questions. Data points from each question were divided into train and test sets, and models were built for each training set. There were a total of 1142 training and 1131 test data points. Results from the stacked model are computed over all the test predictions. This model achieved a correlation of 0.61, and an RMSE of 0.88. The median RMSE computed over each of the questions is 0.82.

Table 3: Comparing performance of models on Mohler et al. (2011)'s dataset. Md(RMSE): median RMSE over questions; AutoP (As): Stacked model over assignments; AutoP (Qs): Stacked model over questions; Baseline: Mohler et al.'s best results; Human: Human Average (Mohler et al., 2011). "*" under column Sig. indicates that the difference between our model and the baseline is statistically significant ($p < 0.05$)

Models	R	Sig.	RMSE	Md(RMSE)
AutoP (As)	**0.61**	*	**0.86**	**0.77**
AutoP (Qs)	**0.61**	*	0.88	0.82
Term vectors	0.56		0.92	0.87
Baseline	0.52		0.98	0.86
Human	0.59		0.66	0.61

As can be seen from Table 3 our stacked model performs better in terms of correlation, RMSE and median RMSE over questions than Mohler et al.'s best models. One of the reasons for improved performance could be that models were built over individual assignments or questions rather than over the entire data. Patterns are particularly effective when built over assignments containing the same type of responses. Short answer scoring can be very sensitive to the content of answers. Hence using data from across a variety of assignments could result in a poorly generalized model.

6 Conclusion

Automatically scoring short answers is difficult. For example, none of Kaggle ASAP short answer scoring competitors managed to consistently reach the level of human-human reliability in scoring. The results of the Kaggle competition, however do show that manually generated regular expressions are a promising approach to increase performance. Regular expressions like patterns are easily interpretable features that can be used by learners to boost short answer scoring performance. They capture semantic and contextual information contained within a text. Thus, determining the best ways to incorporate these patterns as well as making it efficient to develop them is critical to improving short answer scoring.

In this paper we introduce an automated approach to generate text patterns with limited human effort, and whose performance is comparable to manually generated patterns. Further we ensure that the method is generalizable across data sets.

We generate patterns from rubrics and sample top-scoring answers. These patterns help capture the desired structure and semantics of answers and act as good features in grading short answers. Our approach achieves a QW Kappa of 0.78 on the Kaggle short answer scoring dataset, which is greater than the QW Kappa achieved by the best performing model that uses manually generated regular expressions. We also show that on Mohler et al. (2011)'s dataset our model achieves a correlation of 0.61 and an RMSE of 0.77. This result is an improvement over Mohler et al. (2011)'s best published correlation of 0.52 and RMSE of 0.86.

References

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. 2009. A study on similarity and relatedness using distributional and WordNet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27.

ASAP-SAS. 2012. Scoring short answer essays. ASAP short answer scoring competition system description. `http://www.kaggle.com/c/asap-sas/`.

Yigal Attali and Jill Burstein. 2006. Automated essay scoring with e-rater® v. 2. *The Journal of Technology, Learning and Assessment*, 4(3).

Lyle F Bachman, Nathan Carr, Greg Kamei, Mikyung Kim, Michael J Pan, Chris Salvador, and Yasuyo Sawaki. 2002. A reliable approach to automatic assessment of short answer free responses. In *Proceedings of the 19th international conference on Computational linguistics-Volume 2*, pages 1–4. Association for Computational Linguistics.

Leo Breiman. 2001. Random forests. *Machine learning*, 45(1):5–32.

Eric Brill, Susan Dumais, and Michele Banko. 2002. An analysis of the AskMSR question-answering system. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 257–264, Stroudsburg, PA, USA.

Susan Dumais, Michele Banko, Eric Brill, Jimmy Lin, and Andrew Ng. 2002. Web question answering: Is more always better? In *Proceedings of the 25th annual international ACM SIGIR conference on*

Research and development in information retrieval, pages 291–298. ACM.

Christiane Fellbaum. 1998. Wordnet: An electronic lexical database. *MIT Press*.

Peter Foltz, Karen Lochbaum, and Mark Rosenstein. 2011. Analysis of student writing for large scale implementation of formative assessment. In *Paper presented at the National Council for Measurement in Education*, New Orleans, LA.

Peter Foltz, Lynn A. Streeter, Karen Lochbaum, and Thomas K. Landauer. 2013. Implementation and applications of the Intelligent Essay Assessor. In *M. Shermis & J. Burstein, (Eds.). Handbook of Automated Essay Evaluation*, pages 68–88, Routledge, NY.

Evgeniy Gabrilovich and Shaul Markovitch. 2007. Computing semantic relatedness using Wikipedia-based explicit semantic analysis. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 1606–1611.

Derrick Higgins, Chris Brew, Michael Heilman, Ramon Ziai, Lei Chen, Aoife Cahill, Michael Flor, Nitin Madnani, Joel Tetreault, Dan Blanchard, Diane Napolitano, Chong Min Lee, and John Blackmore. 2014. Is getting the right answer just about choosing the right words? The role of syntactically-informed features in short answer scoring. *arXiv*.

Thomas K. Landauer, Darrell Laham, and Peter W. Foltz. 2003. Automated scoring and annotation of essays with the Intelligent Essay Assessor. In *Automated Essay Scoring: A cross-disciplinary perspective*, pages 87–112, Mahwah, NJ: Lawrence Erlbaum Publishers.

Thomas K Landauer. 2006. Latent semantic analysis. *Encyclopedia of Cognitive Science*.

Claudia Leacock and Martin Chodorow. 2003. C-rater: Automated scoring of short-answer questions. In *Computers and the Humanities*, volume 37, pages 389–405. Springer.

Maxim Makatchev and Kurt VanLehn. 2007. Combining bayesian networks and formal reasoning for semantic classification of student utterances. In *Proceedings of the 2007 Conference on Artificial Intelligence in Education: Building Technology Rich Learning Contexts That Work*, pages 307–314, Amsterdam, The Netherlands.

Tom Mitchell, Nicola Aldridge, and Peter Broomhead. 2003. Computerised marking of short-answer free-text responses. In *Manchester IAEA conference*.

Michael Mohler, Razvan Bunescu, and Rada Mihalcea. 2011. Learning to grade short answer questions using semantic similarity measures and dependency graph alignments. *Proceedings of the 49th Annual Meeting of the Association of Computational Linguistics - Human Language Technologies (ACL HLT 2011)*, pages 752–762.

Moodle. 2011. Regular expression short-answer question type. `http://docs.moodle.org/20/en/Regular_Expression_Short-Answer_question_type`.

Martin F Porter. 1980. An algorithm for suffix stripping. In *Program: electronic library and information systems*, volume 14, pages 130–137. MCB UP Ltd.

Stephen G Pulman and Jana Z Sukkarieh. 2005. Automatic short answer marking. In *Proceedings of the second workshop on Building Educational Applications Using NLP*, pages 9–16.

Lakshmi Ramachandran and Edward F. Gehringer. 2012. A word-order based graph representation for relevance identification (poster). *Proceedings of the 21st ACM Conference on Information and Knowledge Management*, pages 2327–2330, October.

Lakshmi Ramachandran and Edward Gehringer. 2013. Graph-structures matching for review relevance identification. In *Proceedings of TextGraphs-8 Graph-based Methods for Natural Language Processing*, pages 53–60, Seattle, Washington, USA, October.

Luis Tandalla. 2012. Scoring short answer essays. ASAP short answer scoring competition–Luis Tandalla's approach. `https://kaggle2.blob.core.windows.net/competitions/kaggle/2959/media/TechnicalMethodsPaper.pdf`.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *In Proceedings of HLT-NAACL*, pages 252–259.

Yu Wang, Yang Xiang, Wanlei Zhou, and Shunzheng Yu. 2012. Generating regular expression signatures for network traffic classification in trusted network management. In *Journal of Network and Computer Applications*, volume 35, pages 992–1000, London, UK, UK, May. Academic Press Ltd.