University of Memphis
University of Memphis Digital Commons

Electronic Theses and Dissertations

8-4-2021

Semantic Matching Evaluation: Optimizing Models for Agreement Between Humans and AutoTutor

Colin Mackenzie Carmon

Follow this and additional works at: https://digitalcommons.memphis.edu/etd

Recommended Citation
Carmon, Colin Mackenzie, "Semantic Matching Evaluation: Optimizing Models for Agreement Between Humans and AutoTutor" (2021). *Electronic Theses and Dissertations*. 2148.
https://digitalcommons.memphis.edu/etd/2148

This Thesis is brought to you for free and open access by University of Memphis Digital Commons. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of University of Memphis Digital Commons. For more information, please contact khggerty@memphis.edu.

SEMANTIC MATCHING EVALUATION: OPTIMIZING MODELS FOR AGREEMENT BETWEEN HUMANS AND AUTOTUTOR

by

Colin Mackenzie Carmon

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Major: Psychology

The University of Memphis

December 2020

Copyright © 2020 Colin Mackenzie Carmon

All rights reserved

Acknowledgements

This research was supported by the Office of Naval Research (N00014-00-1-0600, N00014-15-P-1184; N00014-12-C-0643; N00014-16-C-3027) and the National Science Foundation Data Infrastructure Building Blocks program (ACI-1443068). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of ONR or NSF.

Abstract

The goal of this thesis is to evaluate the answers that students give to questions asked by an intelligent tutoring system (ITS) on electronics, called *ElectronixTutor*. One learning resource of ElectronixTutor is *AutoTutor*, an instructional module that helps students learn by holding a conversation in natural language. The semantic relatedness between a student's verbal input and an ideal answer is a salient feature for assessing performance of the student in AutoTutor. Inaccurate assessment of the verbal contributions will create problems in AutoTutor's adaptation to the student. Therefore, this thesis evaluated the quality of semantic matches between student input and the expected responses in AutoTutor. AutoTutor evaluates semantic matches with a combination of Latent Semantic Analysis (LSA) and Regular Expressions (RegEx) when assessing student verbal input. Analyzing response-expectation pairings and comparing computer scoring with judge ratings allowed us to look at the agreement between humans and computers overall as well as on an item basis. Aggregate analyses on these data allowed us to observe the overall relative agreement between subject-matter experts and the AutoTutor system. Item analyses allowed us to observe variation between items and interactions between human and computer assessment conditions on various threshold levels (i.e. stringent, intermediate, lenient). As expected, RegEx and LSA showed a positive relationship ρ (5202) = .471. Additionally, F1 measure agreement (the harmonic mean of precision and recall) between the computer and humans was similar to agreement between humans. In some cases, computer-human F1 measure agreement compared to between-humans was as close as F1 = .006.

Table of Contents

Preface

The first two installments of this body of research have been submitted and published as short papers by Educational Data Mining Society and Learning at Scale. The installments were published in the proceedings of *Educational Data Mining* in 2018, and the proceedings of the *Sixth ACM Conference on Learning @ Scale '19*. The citations for the published work are as follows:

Carmon, C.M., Hampton, A.J., Morgan, B., Cai, Z., Wang, L., & Graesser, A.C. (2019). Semantic matching evaluation of user responses to electronics questions in AutoTutor. In *Sixth (2019) ACM Conference on Learning @ Scale* (4 pages). Chicago, IL: ACM. . https://doi.org/10.1145/3330430.3333649

Carmon, C., Morgan, B., Hampton, A.J., Cai, Z., & Graesser, A.C. (2018). Semantic matching evaluation in ElectronixTutor. In K. E. Boyer, & M. Yudelson (Eds.), *Proceedings of the 11th International Conference on Educational Data Mining* (pp. 580–583). Buffalo, NY: EDM Society.

These publications describe data collected that asked students electronics questions. The questions were rated by subject-matter experts and these judge ratings were compared to the computer scoring of the same answers to questions. Optimizing these observations can reveal the degree to which a student response is semantically related to the good answer for any given question. The purpose of this body of work is to highlight the degree to which human judgement and computer scoring may be similar on a task such as automatic short essay grading.

The content published in Educational Data Mining covers the pilot stage of the research where the data set was relatively small (n = 2000). In this paper, we randomly selected 200 student responses out of the 2000 and compared computer scoring to judge ratings. The analysis only

considered judge ratings of 6 as ideal/complete answers and there was a fixed threshold for computer scoring (Regular Expressions and Latent Semantic Analysis).

In the content published in Learning at Scale, two additional thresholds for both judges and computer scoring were considered in the model. Additionally, precision and recall, as well as signal detection theory were included in the analyses to explain the results for researchers in computational linguistics and psychology rather than only reporting results in terms of inter-rater agreement (Cohen's kappa).

Chapter 1

Introduction

Effective learning strategies in traditional face-to-face environments have long been studied, especially in interactions between tutors and individuals or small groups. The literature on traditional learning environments discusses effective learning strategies in classroom contexts as well as in one-on-one interactions (Bloom, 1984). Classroom contexts typically have somewhere around 30 students for every instructor. Students greatly outnumber teachers in traditional classroom environments and students often rely on individual attention from the teacher in order to succeed and meet learning goals.

Research in the field highlights instructional benefits that students receive through traditional tutoring rather than traditional classroom interaction (Cohen, Kulik, & Kulik, 1982). Tutoring occurs in face-to-face interactions where a teacher provides individual attention to a single student or a small group of students. Graesser, Person, & Magliano (1995) documents the pedagogical patterns of tutor-tutee interactions. Tutors and tutees work collaboratively to improve the initial answer to the question, and this is what sets tutoring apart from classroom instruction. Traditional classroom and tutoring approaches each have advantages and disadvantages, but here we focus solely on tutoring approaches in educational software such as an Intelligent Tutoring System. The next section discusses tutoring in computerized systems in greater detail.

Intelligent Tutoring Systems

Though traditional learning has been well-documented and researched, education in practice

constantly shifts to mirror contemporary advances in culture, technology, and ever-evolving best practices. Perhaps one of the most noteworthy trends in modern education is an exponential increase in computer usage. The prevalence of computer usage in education can be observed in applications such as online access to class materials, digital discourse spaces, and new learner technologies. Here, we focus on new learner technologies. Intelligent Tutoring Systems (ITSs), online courses, and easy access to educational tools/software are just a few examples of learner technologies we can expect to see in contemporary educational settings. This thesis focuses on automatic assessment of user input in an ITS that implements conversational agents in a turn-based dialogue to teach electronics.

ITSs provide immediate, individualized instruction and feedback to students without intervention from a human tutor. Some ITSs incorporate natural language communication with the students and have been observed to provide instruction and feedback (hints, prompts, tutoring questions, etc.) to the student without much variation from human tutoring (Graesser, 2016; Olney et al., 2012; VanLehn et al., 2007). Additionally, ITSs can cover a wide range of domains, including physics (*AutoTutor*, Graesser et al., 2004; Nye, Graesser, & Hu, 2014), scientific reasoning (*Operation: ARIES*, Cai et al., 2011; *Operation: ARA*, Halpern et al., 2012), biology (*GuruTutor*, Olney et al., 2012), and electronics (*SHERLOCK*, *BEETLE-II*; Lesgold, Lajoie, Bunzo, & Eggan, 1992; Dzikovska, Steinhauser, Farrow, Moore, & Campbell, 2014; *ElectronixTutor*, Graesser et al., 2018).

ITSs offer convenient benefits in a number of contexts, such as those where students greatly outnumber instructors and therefore may not be able to receive individual attention in a timely

manner as needed. ITSs may also interest students who want to learn but are not as comfortable engaging with material in classroom settings. This is not to say that the purpose of ITSs is to reassign the role of a teacher, but rather such ITSs may be used to supplement the student in addition to classroom learning. The instructional efficacy of modern ITSs is comparable to that of human tutors where large effect sizes on learning gains have been observed (human average d = .79, computer average d = .78; VanLehn, 2011).

In addition to immediate, automatic feedback and individualized engagement, ITSs are free of common grading and consistency errors that humans make. Unlike ITSs, humans are prone to making errors resulting from fatigue, bias, and ordering (Haley, Thomas, Roeck, & Petre, 2007). ITSs are potentially less costly than human tutors in terms of time invested (Dorça, 2015), and, depending on the knowledge domain or task, may combat a shortage of available human tutors. Furthermore, ITSs employ pedagogical strategies and various methods of assessment including but not limited to multiple-choice questions. These systems may contain adaptive mechanisms, such as a recommender system that adapts to the student by suggesting topics to cover based on student performance and engagement.

ITSs that incorporate natural language processing aim to accomplish human-like language understanding in order to properly evaluate user verbal contributions and respond in an appropriate manner. While the concept of multiple-choice may not need additional explanation, conversational ITSs typically operate by means of automatic grading. Automatic grading in natural language can be broken down according to question types. The three question types that are observed in automatic grading are fill in the blank, short answer, and essay.

Fill in the blanks are specific questions that have one or few words in correct the answer. Short answers may be one sentence to one paragraph with the focus being on semantic content rather than precise wording. In automatic essay grading, responses may be two paragraphs to two pages. Automatic essay grading considers both style and semantic content, with a balanced integration of these two fundamental dimensions (Li, Gobert, Graesser, & Dickler, 2018). Conversational ITSs can be viewed as an interactive form of automatic short answer grading (Burrows, Gurevych, & Stein, 2015). Aside from AutoTutor, other early conversational ITSs that incorporate automatic short answer grading are *CIRCSIM-Tutor* (Evens et al., 2001) and *Why2-Atlas* (VanLehn et al., 2002).

Although ITSs can be costly and time-consuming to develop, one recent approach is to broaden the coverage of topics with existing learning resources that have been already developed. A prime example of this is *ElectronixTutor (*Graesser et al., 2018), an ITS that integrates multiple ITSs and other conventional learning resources to teach a curriculum of electrical engineering to students. The integration of multiple ITSs and conventional learning resources empowers ElectronixTutor to have multiple pedagogical strategies to teach students. These learning resources include AutoTutor, *Dragoon* (VanLehn, Wetzel, Grover, & van de Sande, 2016), *Learnform* developed by BBN/Raytheon, and questions adopted from BEETLE-II (Dzikovska et al., 2014). Additionally, ElectronixTutor offers topic summaries and the Navy Electronics and Electricity Training Series that electronics trainees read in the Navy.

This thesis focused on assessment of students' verbal input to electronics questions asked in the AutoTutor learning resource. For AutoTutor to properly respond to students in an intelligent

manner, it must evaluate student input with sufficient accuracy. AutoTutor's assessment of student input is based on semantic matching, which compares student responses to one or more expected answers. This thesis analyzed a sample of responses (n = 5202) that were crowd sourced from Amazon Mechanical Turk (AMT) workers. Crowd sourcing participants offers convenience when collecting large data samples and/or searching for participants in a target population that is small, scattered, or difficult to assemble physically based on location.

In summary, there was an assessment of the computational linguistics algorithms used to automatically compute semantic matches in student responses to questions. Student responses were paired with the ideal answer to the main question as well as to each of the expectations (i.e., correct sentence-like parts of an ideal answer) to the question. For example, a response to a question with three expectations is broken down into four response pairs. One pair for the ideal answer, and there is an additional one for each expectation. In addition, we compared the system's evaluations to those of subject matter experts.

Conversations in the AutoTutor Resource in ElectronixTutor

AutoTutor teaches by holding a conversation with the student in natural language (Graesser, 2016; Graesser, 2020). AutoTutor asks students questions and guides them to an expected answer through conversations with the goal of probing students for concepts and ideas that they may know, but do not initially articulate in their answers to questions. AutoTutor helps the student actively construct an answer to the question by collaboratively improving on the answer in a turn-based conversation similar to human tutors (Graesser et al., 2012). When human tutors ask a question to students, they anticipate and monitor expectations and misconceptions (common incorrect answers)

associated with the question.

AutoTutor's Expectation and Misconception Tailored (EMT) Dialogue models the knowledge of the student. AutoTutor matches the student responses to a pre-defined list of expectations using RegEx and LSA (as defined later). The student response is compared to the ideal answer for the question, or to each expectation in the question. In this data set, we can observe questions with as little as one expectation, or as many as five expectations. The following is an example of a question in ElectronixTutor, the ideal answer, and a breakdown of the ideal answer into expectations. The following example contains four expectations:

> ***Main Question:*** *What are the I-V characteristics related to the threshold and breakdown voltage of a real diode compared to an ideal diode?*
>
> ***Ideal Answer:*** *An ideal diode has a threshold voltage of zero. An ideal diode has no breakdown voltage. A real diode has a threshold voltage greater than zero. A real diode has a breakdown voltage less than zero.*
>
> ***Expectation One:*** *An ideal diode has a threshold voltage of zero.*
>
> ***Expectation Two:*** *An ideal diode has no breakdown voltage.*
>
> ***Expectation Three:*** *A real diode has a threshold voltage greater than zero.*
>
> ***Expectation Four:*** *A real diode has a breakdown voltage less than zero.*

Typically, as the dialogue progresses, the tutor provides more and more hints and other dialogue moves to help the learner until the expectation is covered. Feedback is provided to the learner after

each dialogue turn. Once an expectation has been covered, the system moves to another uncovered expectation, or, if all other expectations have been covered, to a summary of the entire answer. Table 1 provides an example of hints and prompts used in ElectronixTutor. Figure 1 provides the image that matches with the problem observed in Table 1.

Table 1. Hints and prompts for the expectation "An ideal diode has a threshold voltage of zero."

Question Type	Question	Correct Answer
Hint	Consider the I-V voltage parameters. Why does the ideal diode conduct current immediately after the forward voltage is applied to it?	Because it has a threshold voltage of zero.
Hint	Look at the figure on the left. What specific voltage cut-off point does the origin represent for the forward bias voltage?	The threshold voltage of the ideal diode.
Prompt	An ideal diode starts conducting immediately when the applied forward voltage crosses which zero-valued voltage of the diode?	The threshold.
Prompt	Which diode has a threshold voltage of zero?	The ideal.
Prompt	The threshold voltage for an ideal diode is equal to what?	Zero.

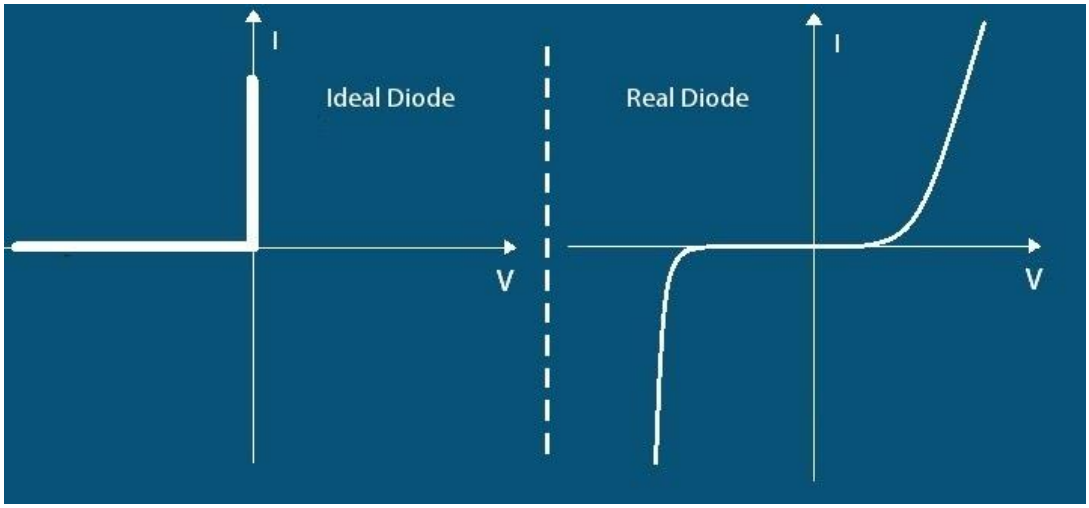

Figure 1. Pictured above are the I-V characteristics of an ideal and real diode.

In this thesis, the sole focus is on the assessment of user responses paired to the ideal answer of the main question and each of the expectations in the question.

Assessment of User Input

AutoTutor's dialogue is driven by semantic matching. In order for AutoTutor to properly respond in conversation, it must be able to meaningfully assess student verbal contributions. AutoTutor assesses student contributions using two matching algorithms, Latent Semantic Analysis (LSA) and Regular Expressions (RegEx). LSA is a natural language processing application used in information retrieval and information extraction. LSA finds relationships between a set of documents and the relevant terms contained in the set of documents. LSA in AutoTutor can be thought of as an assessment of how on topic a student response is. RegExes in AutoTutor are text strings that represent content words we expect to see in student responses. These strings were written at a symbolic level with options to account for synonyms, misspellings, and word ordering as they relate to the expected content words

Regular Expressions

AutoTutor's semantic matching evaluations incorporates regular expressions (Jurafsky & Martin, 2008). Regular expressions are text strings which define expectations represented in AutoTutor. These text strings are used to calculate semantic matches between a student response and an expectation. A RegEx string is a symbolic representation that specifies a set of content words in a text description that may or may not have ordered elements and that may or may not be structured compositionally (i.e., one structure embedded in another structure).

In the ElectronixTutor application the RegEx expressions were a set of content words in an expectation, as specified shortly. One RegEx score for semantic matches was obtained by calculating the proportion of word expressions in the RegEx for an expectation that is matched to the words in the student's verbal contribution. For example, if a RegEx string for an expectation represents 4 word expressions and a student only provides 3 in the body text of their answer, then the RegEx score would be computed as ¾, or .75. The efficacy of RegEx in applications such as these is heavily contingent on the robustness of the RegEx expression used. That is, the quality of a RegEx depends on how explicitly and thoroughly the expressions are written.

RegEx allows for increased flexibility in recognizing student input in three ways. First, they can account for common misspellings (e.g., "sou?r[cs]\w*" would capture "source", "sourse" "sorce", etc). Second, regular expressions can account for anticipated synonyms (e.g., "X will decrease", while the content word is "decrease" can effectively be captured using synonyms "drop", "lower", "smaller", etc.). Third, they also can handle complex student responses. For example, "X will increase, and Y will decrease" can be expressed by the combination of "X.*Y,

increase.*decrease" and "Y.*X, decrease.*increase". This also captures "Y will decrease and X will increase", but does not necessarily capture "X will decrease and Y will increase." Thus, regular expressions capture keywords, synonyms, and complex structures, and common misspellings. In contrast, LSA compares the semantic similarity of the student's answer to the good answer in a very different way, which is especially helpful in recognizing how related user responses are to a given topic based on content word relevance.

Latent Semantic Analysis

LSA (Landauer, McNamara, Dennis, & Kintsch, 2007) is distributional semantics technique for assessing the similarity of pairs of texts expressed in natural language. "Chair" and "table", for example, often appear in the same documents and, as such, have high semantic similarity. The LSA algorithm measures the similarity between a students' input and a good answer (expectation) in the form of a match score from 0 to 1. The good answer for any given question is identified by subject matter experts in the knowledge domain.

LSA spaces are made by combining a classical vector space model with a two-mode factor analysis, Singular Value Decomposition (SVD). SVD is a linear algebraic concept that factors a real (complete) matrix. A bag-of-words (BOW) representation forms a set of texts. BOW models are simplified models of text documents represented as multisets of the words contained in the text. After parsing text and performing the SVD, the BOW representation can be modified with a Term Frequency-Inverse Document Frequency (TF-IDF) matrix and fit into a vector space which represents the semantic field or semantic space. TF-IDF is intended to indicate the relevance of a word to a document within a corpus. Term frequency can be computed as:

$$\boldsymbol{tf_{t,d}} = \frac{\boldsymbol{n_{t,d}}}{\boldsymbol{\#\ terms\ in\ document}}$$

where *n* is the number of times a term (*t*) occurs in the document (*d*). So, each individual term and each document would have its own term frequency. Inverse document frequency can be computed as:

$$\boldsymbol{idf_t} = \mathbf{log}\frac{\boldsymbol{\#\ documents\ in\ corpus}}{\boldsymbol{\#\ documents\ containing\ t}}$$

From here we can compute the TF-IDF for all words contained in the corpus where words with higher scores are more important than words with lower scores. So, TF-IDF can be computed as:

$$\boldsymbol{(tf\_idf)_{t,d} = tf_{t,d} * idf_t}$$

Semantic fields in this context mirror the semantic structure extracted from the original corpus of text documents. Once the semantic field representation is obtained, vectors can be constructed in order to compute similarities between text samples of interest such as an ideal answer and a student response.

Such LSA spaces can be utilized to compute semantic similarity, in our case the semantic similarity between the student's response to an ideal answer or expectation. The metric of similarity is a cosine match score from -1 to 1, with 0 representing no semantic similarity and 1 representing a perfect similarity between the student response and the ideal answer or expectation by virtue of the constructed LSA space. Given two vectors (*X* and Y), we can compute the cosine similarity:

$$\boldsymbol{similarity} = \cos(\boldsymbol{\theta}) = \frac{\boldsymbol{X} * \boldsymbol{Y}}{\|\boldsymbol{X}\|\|\boldsymbol{Y}\|} = \frac{\sum_{i=1}^{n} \boldsymbol{X}_i \boldsymbol{Y}_i}{\sqrt{\sum_{i=1}^{n} \boldsymbol{X}_i^2} \sqrt{\sum_{i=1}^{n} \boldsymbol{Y}_i^2}}$$

Cosines range from -1 (opposite) to 1 (exact or same). However, the cosine similarity or match score between two bodies of text (i.e. an ideal answer and a student response) in this context functionally ranges from 0 to 1 as a result of TF-IDF weighting where TF-IDF matrix term frequencies will never be negative. A student response to a question in AutoTutor that yields an LSA score of .9 has high semantic similarity and is therefore considered to be highly related to the topic, whereas a student response to a question in AutoTutor that yields an LSA score of .1 or .2 has a low degree of semantic similarity and is considered off topic. A 0 score indicates total dissimilarity according to the model and negative match values will not occur.

The most commonly used LSA space, the TASA LSA space (*Touchstone Applied Science Associates, Inc.*; Ivens & Koslin, 1991; Landauer et al., 2007) uses a variety of news articles, novels, and other texts to create a corpus of relevant words and documents. The TASA space is regarded as a general English language LSA space. For the purpose of this research, rather than using the popular TASA LSA space, an Electronics LSA space was developed by creating a corpus of texts from electronics manuals and relevant curriculum materials. An LSA space trained on a specific subject (i.e. LSA spaces trained on electronics for the purpose of grading student quiz answers in an electronics class) is expected to assess student input more accurately and minimize false alarms attributable to the use of general English language terms. In the following section, we discuss the most current version of the customized electronics corpus and LSA space that are used in this research.

Optimization of Computer Combination Model

One goal of using computer models to assess student input is to identify and utilize a model that assesses student verbal contributions (e.g. grading an open-ended student response to a quiz question) similarly to human subject-matter experts. In this context, we can assume that higher agreement between human subject-matter experts and similar agreement between subject-matter experts and the computer indicates a properly functioning model for the assessment of verbal contributions. By computing the agreement between human judges and the computer, we can compare it to agreement between human judges for a relative understanding of the performance of the computer model. We view these agreement observations relative to previous versions of the computer model (Carmon et al., 2019) or similar computer models used in similar contexts. In order to optimize the computer model and improve the agreement between humans and the computer, we should modify the build of the LSA space whose output represents the semantic field of the topic (i.e. electronics). Further, we should consider all possible values of the discrimination threshold.

LSA Model Specification

In order to optimize the computer combination model for semantic matching in the grading of student responses, we need to address the LSA space used in the combination. LSA is the one semantic component of the model, and RegEx being the other. RegEx can be written manually by experts to accommodate misspellings and also semantic variation (e.g. synonyms, functionally comparable words in a given context), but to a lesser extent than LSA. In this context, RegEx expressions can be viewed primarily as a lexical component where words in the student response

are being matched to the content words that define the RegEx for each ideal answer. Any synonym that is recognized by a RegEx string is considered a semantic extension of a lexical item.

A statement can sometimes be semantically true even though it has no match to semantic extensions of a lexical item (as defined by a RegEx). Assessing these types of statements for relevance to an ideal answer is appropriately handled by LSA. For example, imagine a photograph of an exceptionally tall man and being given the instructions to talk about the man's stature. Let us assume that the ideal answer is "The man is tall." Regular expressions may offer expected semantic extensions (i.e. synonyms) of "tall" such as big, large, long, lengthy, etc. So, a student could say "The man is large", which would still satisfy the lexical item "tall" for RegEx. However, if the student responded, "You could tell by his stature that he was a natural-born basketball player", then RegEx would not satisfy the lexical item in the context.

The student is clearly suggesting that the man is tall and therefore, the student's statement is semantically like the ideal answer. However, the language contained unforeseen or unaccounted for semantic extensions of the lexical item "tall". Running the student basketball answer against the ideal answer and computing the cosine similarity between those two bodies of text using an appropriately trained LSA space (e.g. LSA trained on sports texts with documents about typical physical characteristics of basketball players) is much more likely to return a helpful match value. For this reason, it is important to modify the build of the LSA space as to increase the appropriateness of LSA's semantic field representation for a given topic.

In this context, we can modify the LSA space in ways to increase the appropriateness of the LSA space for a given topic (i.e. electronics). Topic appropriateness for an LSA space can be

encouraged through proper selection and structuring of corpus materials as well as modification of topic/dimension frequency and content. An LSA space that is appropriate for a topic such as electronics in this context should yield more precise agreement with subject-matter expert ratings than an off-topic LSA space (e.g. sports LSA space).

When creating a corpus to train an LSA space, it is necessary to select relevant corpus documents as to create a semantic representation that is appropriate for the target application and topic. For example, for an electronics LSA space trained to grade student short answers in a beginners' electronics class, it may be appropriate to create a corpus made from beginner-level electronics texts such as introduction/fundamentals electronics textbooks or electronics manuals. In contrast, training an LSA space entirely on fashion magazine articles is less likely to yield as meaningful a semantic representation as it relates to subject-matter in an electronics class. Apart from selecting the proper documents to feed into the corpus is the issue of chunk size or relevance. For any given document (e.g. introductory electronics textbook), the body text is essentially stripped out and broken down into smaller document sizes, or what will be called *chunks*, that are fed into the corpus to train the text model. In this electronics corpus, we typically group chunks by the paragraph level. This follows the basic assumption that within the body text of these electronics documents, each paragraph break indicates the end of one topic and beginning of a new topic or sub-topic. For this reason, each chunk in the electronics corpus ranges from one to three paragraphs.

Customized Corpora for Training Subject-specific LSA Spaces

Previously, Carmon et al. (2018; 2019) used an LSA space trained on a corpus of

indeterminate electronics documents to examine the agreement between human judges and the computer. In the current study, computer judgement is still represented by a model that combines RegEx and LSA, however, we combined the same RegEx with a new LSA space trained on a carefully selected corpus comprised of electronics and physics documents. Physics texts were added to the model as some believe that physics and electrical engineering may share some overlap in subject-matter and application. In fact, certain subfields of physics interface with subfields of engineering (e.g. relationship between optics/photonics and electrical engineering). The physics texts also appear to slightly improve model performance regarding F1 and Cohen's Kappa metrics.

In addition to 32,000 text chunks (roughly one to three paragraphs per text chunk) worth of physics materials, the Carmon Electronics corpus includes 24 volumes of Navy Electrical Engineering Training Series (NEETS) manuals, three introduction to electronics text books, two electrical engineering handbooks, three sets of course notes from introductory electrical engineering courses, three electronics fundamentals manuals, one AC-DC power supplies manual, and five circuitry textbooks of varying degrees of complexity (i.e. ranging from beginner's level circuitry to more advanced circuitry). After raw texts were extracted from the documents and chunked, the corpus contains roughly 80,000 chunks and over 12 million words. Additionally, the current study assumed 310 topics or dimensions from the corpus of 12 million words. This 12 million word corpus satisfies the criterion laid out by Landauer, McNamara, Dennis, & Kintsch (2007) which suggests as a rule of thumb that LSA spaces used in tasks such as automatic grading should be trained on a corpus of no less than 10 million words. The new corpus and LSA space were created in order to optimize model performance, and indeed, the combination model containing the newer LSA space used in the current study ended up agreeing more with human

subject-matter experts than did the combination model containing the previously used LSA space from Carmon et al. (2018; 2019).

Model Simulation for Optimizing Agreement

After the RegEx threshold was set and the LSA space had been modified and trained on the customized electronics corpus, we made a simulation in Python. To obtain performance metrics for model performance, raw scores from LSA and RegEx were coded into 1's and 0's in comparisons to human judge ratings. When analyzing human and computer decisions, we computed confusion matrices that also yielded precision, recall, F1 measure, Cohen's kappa, and d'. Each observed performance metric was computed based on a single discrimination threshold decided upon for each RegEx and LSA.

In the simulation, we produced one analysis nested within a for-loop that simulates each value of LSA's discrimination threshold from 0 to 1 in thousandths (i.e. .001, .002, .003; all the way to .999) and outputs each set of performance metrics. So, by simulating a thousand values of the LSA discrimination threshold, we get a look at a thousand different sets of performance metrics. By doing this, we can search the highest performance metric value represented for any target statistic according to stringent, intermediate, and lenient thresholds. In this study we optimize thresholds for the highest possible F1 measure values, but the simulation can also be modified to identify highest measures of Cohen's kappa, or d' assuming any given set of answer data with human judgements. Annotated Python code for the simulation can be found in the Appendix section of the paper.

Current Study

This thesis assessed the quality of the semantic matches between student input and expected responses in AutoTutor. Previously, there was a higher degree of agreement between human judges (κ = .699, n = 194) than between AutoTutor's semantic match scores and humans (κ = .493, n = 194) (Carmon, Morgan, Hampton, Cai, & Graesser, 2018) when only stringent thresholds of matching were considered, and when standard computational linguistics metrics (i.e., precision, recall, F-measures) were not calculated. However, models in the current study included these standard computational linguistics metrics. Also, the current study observed AutoTutor semantic matching scores across two additional thresholds rather than observing the stringent threshold only. The three thresholds were considered for assessing user input in the current study: Stringent (*S*), Intermediate (*I*), and Lenient (*L*). These categories of threshold will be explained more in the Method section.

Ideally, the ranges of agreement would conform to similar studies that used natural language processing methods to assess user input or classify response ratings in different knowledge domains (e.g., Gautam, Swiecki, Shaffer, Graesser, & Rus, 2017), where precision reached 96%, and recall 78%. Precision is the proportion of computer responses that that signify a correct match (between the student response and the expectation) that are also deemed as a correct match according to human judgments. In contrast, recall, or sensitivity, is the proportion of human judgments of correct matches that are also computed as correct matches by the computer. Precision and recall are explained and contextualized more in the Analysis section.

One plausible hypothesis is that RegEx and LSA should yield a positive correlational

relationship. Although regular expressions and LSA both share the goal of assessing user responses, they do so differently. That is, RegEx matches keywords (flexibly specified) from the student response to keywords in the expected answer while accounting for common misspellings, synonyms, and word order. In contrast, LSA detects semantic similarity of all of the words between two texts, based on the higher dimensional semantic space. A second plausible prediction is that analyses would show an agreement between the human judges that is equal to or higher than between judges and AutoTutor's semantic match scores.

The judges that were used are both qualified subject-matter experts in electronics and expected to reliably distinguish between various categorizations of response pairings. Although automatic assessment of student input in these systems has substantially improved over the years, human judges are still regarded as the gold standard. The third prediction is that the semantic match scores of AutoTutor should be similar to agreements between human experts. Available studies have analyzed LSA and RegEx match scores in other subject matters such as scientific reasoning. In Cai et al., (2011), models combining LSA and RegEx were found to be as reliable as human tutors. One day the automated systems will hopefully show the same level or higher degrees of agreement between computer and human than can be observed between two or more humans. The fourth prediction is that semantic matching scores would reflect higher overall agreement with humans in lenient threshold categories. The assumption behind this prediction is that agreement increases from vague to precise specifications of meaning.

Chapter 2

Method

Materials

This analysis was conducted on student responses to electronics questions that were developed in *ElectronixTutor.* The responses analyzed in this work were drawn from the verbal reasoning learning resource in the system called AutoTutor. The electronics questions were developed, modified, and eventually approved by two subject matter experts in the knowledge domain of electronics. Each of the subject matter experts were qualified as experts on the basis that they held a master's degree in the given knowledge domain. The items created and used in the verbal reasoning component of the system were typically adapted from an electronics curriculum (e.g. class lectures, exercises, textbooks, etc.) and manuals.

Answers to the electronics questions were collected from users on Amazon Mechanical Turk (AMT). Each of the 219 AMT users that participated in the data collection was pre-screened. In the pre-screen, there was a short electronics questionnaire to establish that the AMT user attempting to participate in the study possessed at least a baseline ability to answer simple questions about electronics basics. The pre-screen consisted of 5 multiple-choice electronics questions that tested the AMT users' ability to perform calculations and demonstrate knowledge on basic electronics concepts such as Ohm's law. The questions in the pre-screen were selected based on topics covered in the AutoTutor resource in ElectronixTutor. The questions in the pre-screen were of similar difficulty to the questions used in data collection.

The purpose of the pre-screen was simply to distinguish between AMT users who could answer basic electronics questions and those that cannot. This pre-screen process attempted to avoid the issue of AMT users who are not qualified to answer electronics questions signing up to receive pay. Additionally, AMT users were asked to describe their background in electronics. Before answering questions, AMT users received the instructions: “Please tell us about your background in electronics. For example, how many courses have you taken, high school or college, etc.” The following example displays one of the problems used in the pre-screen process:

> ***Problem:*** *Calculate the Resistance value in a closed circuit supplied with 110V and power consumed in the circuit is 100 watts.*
>
> ***A:*** *1.1 ohms*
>
> ***B:*** *11 ohms*
>
> ***C:*** *121 ohms* ***(correct)***
>
> ***D:*** *60.5 ohms*

AMT users only had one attempt to complete the pre-screen. Only those who correctly answered 3/5 of the pre-screen problems were permitted to take part in the study. After the pre-screen approval, AMT users filled out an AMT Human Intelligence Task (HIT) and submitted it for financial compensation. A HIT represents a task that a user can work on, submit an answer, and collect a reward for completing.

Each AMT HIT was presented in the form of questionnaire made up of electrical engineering questions of varying topics and difficulty that were borrowed from the verbal reasoning component

of the AutoTutor system. Each HIT contained 6 questions except for the last HIT which was created using the remaining 4 questions in the set of 118. Each question presented in a HIT was matched with an image or figure that provides valuable context for the electronics question that the student reads. For each figure and question, a blank text box was displayed which allowed the AMT user to respond to the question in an open-ended fashion however they saw fit. The questionnaires were written in HTML format and created in such a way that the answers collected from questionnaire text boxes were directly stored in a CSV file. Once answers were collected from all 219 participants, the CSV output of answers to questions collected on AMT were transformed to columns that specifies user ID, question ID, and answer.

Additional experts on electronics received a training session on evaluating semantic similarity between expectations and student responses. Judges were trained on a few of the responses to questions and asked to share ratings. The judges then justified to each other their ratings for the training items with the goal of developing a standard when rating test items independently. The response rating procedure included a judge rating tool to simplify the rating process, save time, and reduce cognitive load taken on by the judges. The rating tool displayed the question or part of the question applicable, the user response, the expectation, and a field for specifying a 1–6 rating.

The 6 ratings are defined according to specific criteria:

- A judge rating of **1** indicates no attempt to answer the question.

- A rating of **2** indicates that the answer is not on topic or contains metacognitive language.

- A rating of **3** indicates that the answer is on topic, but completely incorrect.

- A rating of **4** indicates that the answer is mostly incorrect but contains a small degree of truth value.

- A rating of **5** indicates that the answer is mostly correct.

- A rating of **6** indicates that the answer seems ideal.

All user responses in the rating tool were sorted into tabs according to item. The tool was designed to simplify and speed up the process of rating for judges as opposed to judges interacting with the raw data in CSV or XLS format. The procedure section offers a detailed table about the scoring definitions that the judges used in rating student responses.

LSA and RegEx scores of user responses were computed for comparison to judge ratings. To compare the judge ratings to the computer, we observed three thresholds for humans' ratings. The three thresholds to observe are stringent, intermediate, and lenient. We selected each threshold value based on the simulation program written in Python that considered every possible value of the LSA discrimination threshold to identify the optimal threshold value where highest F1 measure is observed in stringent, intermediate, and lenient threshold categories.

For the stringent human thresholds, human judgment between 1 and 5 was coded as a 0 and

a score of 6 as a 1. The stringent threshold for humans considers student responses that the judges consider to be correct and complete (i.e. a rating of 6). The stringent threshold for computers consists of raw scores at or above the .75 threshold for RegEx, or .892 for LSA and was be coded as a 1, whereas below was coded as a 0. These stringent computer thresholds apply to RegEx only, LSA only, and RegEx and LSA combination (RegEx threshold is met *or* LSA threshold is met).

For the intermediate human thresholds, human judgment between 1 and 4 was coded as a 0, and 5 or 6's as 1. The intermediate threshold for computer scoring considers a RegEx threshold that is placed at .5, and one for LSA at .769. These intermediate computer thresholds apply to RegEx only, LSA only, and RegEx and LSA combination (RegEx threshold is met *or* LSA threshold is met).

In the lenient human threshold, human judgment between 1 and 3 was coded as a 0 and 4–6 as 1. The lenient threshold for humans considers any student responses with varying degrees of truth value as 1. The lenient threshold in computer scoring considered a RegEx threshold that is placed at .33, and one for LSA at .494. These lenient computer thresholds apply to RegEx only, LSA only, and RegEx and LSA combination (RegEx threshold is met *or* LSA threshold is met).

So, analyzing agreement between human judgement and the computer (RegEx only) across 3 thresholds yielded a 3 x 3, or 9 statistical observations. Another 9 observations occurred between human judgement and computers with the only comparison being that the computer threshold observed the LSA only condition instead of RegEx only. A third analysis repeated this 3 x 3 design where the computer observed RegEx and LSA combination thresholds rather than RegEx only and LSA only. A final analysis repeated the 3 x 3 design for Judge 1 and Judge 2.

A RegEx bank of relevant electronics terms created by a team of university researchers provided easier RegEx application to each of the 118 items in the system. The bank contains the semantic field of content words associated with all items in the system. When creating AutoTutor items in ElectronixTutor, researchers may refer to the bank for word expressions that have been created, as they may be represented in ideal answers and expectations for multiple items in the system. The RegEx bank for the electronics items allowed for word expression matches of relevant terms in the body text of the user's response when compared to the ideal answer or expectation.

A RegEx expression is written so that it allows for flexible user response comprehension in a few ways. The RegEx written for these items allows for common misspellings, complex student responses (word ordering), and commonly recognized synonyms. Additionally, this research used an electronics LSA space rather than the TASA LSA space, the most popular LSA space for general English language. LSA and RegEx can be computed using a custom-made tool where the input is a user response and the outputs are computed scores for RegEx and LSA. These scores represent the semantic matching capabilities of AutoTutor, and by using these computed semantic matching scores, we compared AutoTutor's performance to that of a human judge.

Procedure

The data had 219 unique AMT workers who answered 118 questions asked by AutoTutor in ElectronixTutor. An AMT worker answered only a subset of the 118 questions, thus many responses from different items were collected from repeat AMT workers. AMT workers responded to these questions in an open-ended fashion, saying as much or as little as needed in order to

answer the questions asked. Each question received up to 20 user responses for a corpus of roughly 2350 responses to the main questions. Of the 2350 collected responses to the main question, each response was paired with the main question *and* each expectation to the main question, resulting in 5202 (n=5202) total pairings that were used as the sample units for the analyses. Two subject-matter experts independently rated the user responses on a scale ranging from 1–6. See Table 2 for a description of the scoring values used by judges.

Table 2. Scoring values used by human judges rating student responses to electronics questions.

1	No attempt to answer the question.
2	Answer is not on topic/includes metacognitive.
3	Answer is on topic, but completely incorrect.
4	Answer is mostly incorrect.
5	Answer is mostly correct.
6	Answer is completely correct.

In selecting participants for the data collection process, AMT workers were asked to describe their background in electronics and answer questions to the best of their ability without any help (e.g. internet searches, reading materials, asking other individuals). Workers were also required to complete the pre-screen (described in the Materials section) to qualify as eligible. Workers were compensated $1 for each question answered. Compensating $1 per question typically pays a user $6 per HIT with the exception of the one HIT that only contained 4 questions.

Chapter 3

Analysis

The analyses computed LSA and RegEx scores for student responses, whereas two subject-matter experts also judged each response. These analyses aimed to explore the relationship between regular expressions and LSA, interrater agreement between the two judges, and agreement between responses in human judgment and AutoTutor semantic match scores.

This thesis observed agreement by calculating Cohen's kappa, precision and recall scores, and d' ("d prime") scores from signal detection theory. The scores for each metric were calculated between human judges, and between human judges and AutoTutor's semantic matches according to LSA, RegEx, and both. These analyses explain the data using this variety of metrics in order to appeal to multiple audiences. In this thesis, the concept of explaining the data for multiple audiences was informed by Graesser, Wiemer-Hastings, Kreuz, Wiemer-Hastings, and Marquis (2000). Explaining the results for researchers in computational linguistics and psychology rather than only reporting results in terms of inter-rater agreement (Cohen's kappa) offers the benefit of reaching a broader audience.

The first performance metric observed in analyses is Cohen's kappa (κ). Cohen's kappa measures inter-rater reliability for categorical items. Cohen's kappa is regarded as a more robust measure than a simple percent agreement as it considers the possibility of observations that may have occurred by chance. Cohen's kappa is a popular metric in social sciences research. Cohen's

kappa can be calculated as:

$$\kappa \equiv \frac{p_o - p_e}{1 - p_e} = 1 - \frac{1 - p_o}{1 - p_e}$$

Where p_o is accuracy and p_e is the probability of chance. The analyses calculate κ between human judges, and between human judges and AutoTutor's semantic matching mechanism.

Next, the analyses calculate metrics for precision and recall. The metrics for this portion of the analysis include precision, recall, and F1. Precision and recall are common measurements for assessing accuracy of information retrieval, classification, and identification tasks in computers. As stated previously, precision is the proportion of computer responses that that signify a correct match (between the student response and the expectation) that were also deemed as a correct match according to human judgments. In contrast, recall is the proportion of human judgments of correct matches that were also computed as correct matches by the computer.

Precision and recall were calculated in this context using a 2 x 2 matrix where AutoTutor semantic match represents the predicted condition, and where human judgement represents the true or observed condition. Each condition contains 2 levels, positive or negative. The matrix only contains 4 possible outcomes that were factored into precision and recall calculations. These 4 possible outcomes are true positives (*TP*, *alternatively* called *hits*), true negatives (*TN*, i.e., correct rejections), false positives (*FP*, i.e., false alarms), and false negatives (*FN*, i.e., misses). So, precision is calculated as:

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FP}}$$

Recall is calculated as:

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}}$$

F1 scores are calculated as the weighted average (or harmonic mean) between precision and recall where:

$$\boldsymbol{F1} = \frac{\boldsymbol{2} * \boldsymbol{TP}}{(\boldsymbol{2} * \boldsymbol{TP}) + \boldsymbol{FP} + \boldsymbol{FN}}$$

The last metric that the analyses observed comes from signal detection theory. The metric d' ("d prime"), also referred to as the sensitivity index, separates the means of the signal and the noise distributions, in comparison to the standard deviation of the signal or noise distribution. The metric d' can be calculated as:

$$\boldsymbol{d'} = \frac{\boldsymbol{\mu_{SI}} - \boldsymbol{\mu_N}}{\sqrt{\frac{\boldsymbol{1}}{\boldsymbol{2}}\left(\boldsymbol{\sigma_{SI}^2} + \boldsymbol{\sigma_N^2}\right)}}$$

Where (*SI)* represents signal and (*N*) is noise. These analyses calculate d' between human, and between human judges and AutoTutor's semantic matching mechanism.

As previously stated, analyzing agreement between human judgement and the computer (RegEx only) across 3 thresholds yielded a 3 x 3, or 9 statistical observations. An additional 9 observations were made for LSA only versus human. A third 3 x 3 observation was made for RegEx and LSA combination versus human, and a final observation was made for Judge 1 versus Judge 2. The thresholds observed for humans and computers are stringent (*S*), intermediate (*I*), and lenient (*L*). The following describes stringent thresholds in the RegEx and LSA combination model for computer (*c*) and human (*h*), respectively, where *T* is threshold, LSA is *LSA*, and RegEx is *RE*:

$$\boldsymbol{T_{Sc} = 1, T_{SLSA} \geq .892 \vee T_{SRE} \geq .75}$$

$$\boldsymbol{else};\ \boldsymbol{T_{Sc}} = 0$$

$$\boldsymbol{T_{Sh} = 1, rating > 5}$$

$$\boldsymbol{else};\ \boldsymbol{T_{Sh} = 0}$$

The next expression describes intermediate thresholds for computer (*c*) and human (*h*), respectively:

$$\boldsymbol{T_{Ic} = 1, T_{ILSA} \geq .769 \vee T_{IRE} \geq .5}$$

$$\boldsymbol{else};\ \boldsymbol{T_{Ic}} = 0$$

$$\boldsymbol{T_{Ih} = 1, rating \geq 5}$$

$$\boldsymbol{else;\ T_{Ih} = 0}$$

The last expression describes lenient thresholds for computer (*c*) and human (*h*), respectively:

$$\boldsymbol{T_{Lc} = 1,\ T_{LLSA} \geq .494\ \vee\ T_{LRE} \geq .33}$$

$$\boldsymbol{else;\ T_{Lc}} = 0$$

$$\boldsymbol{T_{Lh} = 1, rating \geq 4}$$

$$\boldsymbol{else;\ T_{Lh} = 0}$$

Additionally, a secondary analysis repeated this 3 x 3 design twice more where there was a focus on RegEx and LSA models *independently*, rather than a combination of the two. Further, this thesis included two more 3 x 3 observations. One where agreement between humans and the computer (RegEx and LSA combination) was calculated and one where the same was done between judges. By completing these analyses, we can interpret the statistics to determine whether or not any agreement observations significantly differed between human judges or between humans and the computer. The differences will be observed across the three thresholds mentioned, and human judgement will be compared against RegEx and LSA combination, RegEx only, and LSA

only, respectively.

For interpreting differences between observed agreements (human versus RegEx and LSA combination) we refer to a simple difference in scores for Cohen's kappa. Cohen's kappa is calculated from 0 to 1, where 1 is equal to perfect agreement observed. When interpreting Cohen's kappa, $\boldsymbol{\kappa}$ values are sometimes classified into five somewhat arbitrary categories that Cohen identified: poor (low) value being less than 0.20, fair between 0.20 and 0.30, moderate between 0.30 and 0.40, substantial (good) between 0.40 and 0.70, and outstanding agreement values observed between 0.70 and 1.00.

By adhering to these interpretations of κ values and using them as guidelines, we assume that any two single κ values with an observed difference of .15 or greater can be considered to have notable differences, as they have at least differed by roughly 1 Cohen's kappa agreement category defined above.

Chapter 4

Results

We began by examining the non-parametric relationship between RegEx and LSA using a Spearman's correlation. As expected, we observed a moderate, positive relationship between the two, $\rho\ (5202) = .471$. To compare human judges to the computer, all ratings were coded as either a 1 (positive) or a 0 (negative), and were compared across stringent, intermediate, and lenient thresholds. Before addressing the data below, refer to Table 3. Table 3 lists all F1 measure scores at the aggregate level between judges and between the judges and the computer.

Table 3. A breakdown F1 measure agreement between the computer and human judges, and between human judges according to stringent (*S*), intermediate (*I*), and lenient (*L*) threshold conditions.

F1 Measure Agreement Between Computer and Human Judges

Condition	F1 Measure	
	Judge 1	Judge 2
S: LSA	.368	.300
I: LSA	.466	.470
L: LSA	.555	.561
S: RegEx	.444	.509
I: RegEx	.520	.542
L: RegEx	.6	.579
S: RegEx-LSA Combination	.526	.462
I: RegEx-LSA Combination	.548	.524
L: RegEx-LSA Combination	.589	.620
S: Between Judges	.532	
I: Between Judges	.599	
L: Between Judges	.653	

Note. At the aggregate level, F1 measure values in the combination model match the values between humans more closely than either of the stand-alone models.

Aggregate Data Analyses

Aggregate: Between Judges

We analyzed agreement of the human judge ratings on responses from AMT users using Cohen's kappa, precision and recall measures, and d' from signal detection theory. The interrater reliability between judges in the stringent threshold (*S*) was good, $\kappa = .456$, $n = 5202$, while precision reached .467 and recall .618, with F1 = .532, d' = 1.593. According to intermediate (*I*) thresholds, the interrater reliability between judges was good and slightly higher than judge agreement for stringent thresholds, $\kappa = .466$, $n = 5202$. In the intermediate threshold between

judges, precision reached .636 and recall .565, with F1 = .599, d' = 1.36. The interrater reliability between judges in lenient (*L*) thresholds was similar, κ = .460, n = 5202, while precision reached .710 and recall .604, with F1 = .653, d' = 1.274.

Aggregate: LSA Versus Judges

For the stringent (*S*) human threshold, human judgment between 1 and 5 was coded as a 0 and a score of 6 as a 1. For stringent computer thresholds in LSA, a raw score at or above the .495 threshold was coded as a 1, whereas below was coded as a 0. All optimal thresholds for the LSA model were revealed through the program which simulates a thousand values of the discrimination threshold and compares it to human judge ratings. The reliability between LSA and the first judge was low, κ = .195, n = 5202, whereas the reliability between LSA and the second judge was even lower, κ = .150, n = 5202. When the model was compared with the first judge, precision reached .547, and recall .277, F1 = .368, d' = .649. For the second judge, precision reached .542, and recall .207, F1 = .300, d' = .575.

For the intermediate (*I*) human threshold, human judgment between 1 and 4 was coded as a 0 and 5 or 6's as 1. For intermediate computer thresholds in LSA, the threshold was placed at .423. The reliability between LSA and the first judge was fair, κ = .241, n = 5202, whereas the reliability with the second judge was similar, but slightly lower, κ = .223, n = 5202. When the model was compared with the first judge, precision reached. 631, and recall .369, F1 = .466, d' = .762. For the second judge, precision reached .592, and recall .390, F1 = .470, d' = .636.

For the lenient (*L*) human thresholds, human judgment between 1 and 3 was coded as a 0 and 4–6 as 1. For lenient computer thresholds in LSA, the threshold was placed at = .362. The

reliability between LSA and the first judge was fair, κ = .272, n = 5202, whereas the reliability between LSA and the second judge was again similar, but lower, κ = .236, n = 5202. When the model was compared with the first judge, precision was .673, while recall reached .472, F1 = .555, d' = .755. For the second judge, precision was .621, while recall reached .511, F1 = .561, d' = .618.

Aggregate: RegEx Versus Judges

For the stringent (*S*) human thresholds, human judgment between 1 and 5 was coded as a 0 and a score of 6 as a 1. For stringent computer thresholds in RegEx, a raw score at or above the .75 threshold was coded as a 1, whereas below .75 was coded as a 0. We examined agreement between human judge ratings and RegEx. The reliability between RegEx and the first judge was moderate, κ = .366, n = 5202, whereas the reliability between RegEx and the second judge was good, κ = .428, n = 5202. When the model was compared with the first judge, precision reached .448, and recall .440, F1 = .444, d' = 1.487. For the second judge, precision reached .450, and recall .585, F1 = .509, d' = 1.275.

For the intermediate (*I*) human thresholds, human judgment between 1 and 4 was coded as a 0 and 5 or 6's as 1. For intermediate computer thresholds in RegEx, the threshold was placed at .5. The reliability between RegEx and the first judge was fair, κ = .313, n = 5202, whereas the reliability with the second judge was similar but higher, κ = .362, n = 5202. When the model was compared with the first judge, precision reached .607, and recall .455, F1 = .520, d' = 1.07. For the second judge, precision reached .677, and recall .452, F1 = .542, d' = .882.

For the lenient (*L)* human thresholds, human judgment between 1 and 3 was coded as a 0 and 4–6 as 1. For lenient computer thresholds in RegEx, the threshold was placed at = .33. The

reliability between RegEx and the first judge was fair, $\kappa = .306$, $n = 5202$, whereas the reliability with the second judge was again similar, $\kappa = .307$, $n = 5202$. When the model was compared with the first judge, precision was .679, while recall reached .545, F1 = .600, d' = .870. For the second judge, precision was .713, while recall reached .487, F1 = .579, d' = .815.

Aggregate: RegEx/LSA Combination Versus Judges

For the stringent (*S*) human thresholds, human judgment between 1 and 5 was coded as a 0 and a score of 6 as a 1. For stringent computer thresholds in RegEx and LSA, a RegEx score at or above the .75 threshold or LSA at or above .892 was coded as a 1, whereas below was coded as a 0. We examined agreement between human judges and RegEx/LSA Combination. The reliability between RegEx/LSA Combination and the first judge was good, $\kappa = .446$, $n = 5202$, whereas the reliability between RegEx/LSA Combination and the second judge was moderate and lower, $\kappa = .384$, $n = 5202$. When the model was compared with the first judge, precision reached .479, and recall .584, F1 = .526, d' = 1.508. For the second judge, precision reached .482, and recall .444, F1 = .462, d' = 1.312.

For the intermediate (*I*) human thresholds, human judgment between 1 and 4 was coded as a 0 and 5 or 6's as 1. For intermediate computer thresholds in RegEx, the threshold was placed at .5, and for LSA, .769. The reliability between RegEx/LSA Combination and the first judge was moderate, $\kappa = .365$, $n = 5202$, whereas the reliability with the second judge was similar, but slightly lower, $\kappa = .311$, $n = 5202$. When the model was compared with the first judge, precision reached .710, and recall .446, F1 = .548, d' = 1.096. For the second judge, precision reached .633, and recall .448, F1 = .524, d' = .883.

For the lenient (*L*) human thresholds, human judgment between 1 and 3 was coded as a 0 and 4–6 as 1. For lenient computer thresholds in RegEx, the threshold was placed at .33, and for LSA, .494. The reliability between RegEx/LSA Combination and the first judge was fair, $\kappa = .291$, $n = 5202$, whereas the reliability with the second judge was again similar, $\kappa = .292$, $n = 5202$. When the model was compared with the first judge, precision was .809, while recall reached .463, $F1 = .589$, $d' = .939$. For the second judge, precision was .772, while recall reached .519, $F1 = .620$, $d' = .850$. At the aggregate level, F1 and d' for combination models reached values closest to those between judges in all three threshold conditions. Agreement values between the combination model and judges were higher than between either the LSA only model versus judges or the RegEx only model versus judges. While RegEx alone agrees with humans more than LSA alone, the combination model agrees more with humans than both stand-alone models. This suggests that by pairing RegEx and LSA, there is added benefit in a combination model compared to using each model separately.

Repeated Measures ANOVAs

Repeated Measures: Between Judges

Figure 2 shows the cell means of a 3 x 3 repeated measures design conducted to compare Judge 1 and. Judge 2 on F1 measure agreement in stringent (S), intermediate (I), and lenient (L) threshold conditions. A repeated measures analysis of variance (ANOVA) was conducted on the F1 agreement scores where the unit of analysis was the item ($n = 118$). There was a significant main effect of Judge 1, $F (2, 234) = 27.12$ $p = .001$, $\eta 2 = .188$, and of Judge 2, $F (2, 234) = 30.617$, $p = .001$, $\eta 2 = .207$, with a significant interaction between human judges, $F (2, 234) = 46.96$, $p = .001$,

η2 = .286. Here, F1 agreement on stringent items (items rated 6 by judges) was lowest, and lenient agreement was highest. Agreement on the stringent threshold is practically indistinguishable from agreement between Judge 1 stringent, and Judge 2 intermediate. Judges agreements in intermediate and lenient thresholds appear to show more distinct categorization.

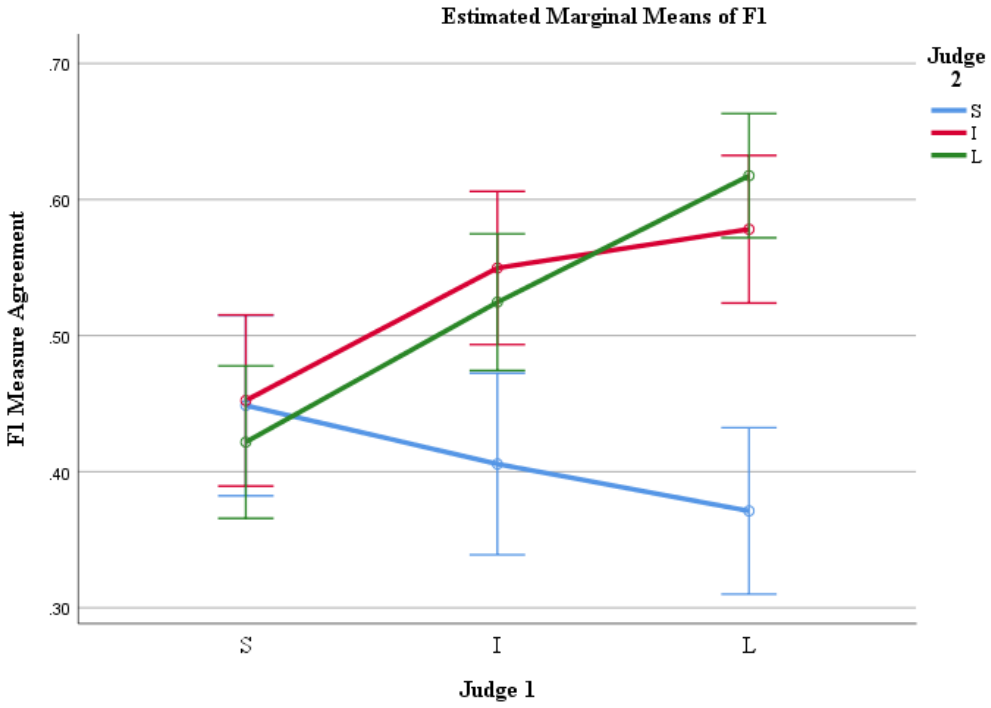

Figure 2. Mean F1 measure as a function of the thresholds of Judge 1 and Judge 2 where *S* is stringent, *I* is intermediate, and *L* is lenient.

Repeated Measures: LSA Versus Judges

Figure 3 shows mean F1 scores as a function of stringent (S), intermediate (I), and lenient (L) thresholds of LSA versus Judge 1 in a 3 x 3 repeated measures design. An ANOVA was conducted on the F1 agreement scores where the unit of analysis was the item (question) in

AutoTutor (n = 118), as opposed to the expectation unit in an ideal answer, noting that some questions had multiple expectations. There was a significant main effect of Judge 1, F (2, 234) = 4.352, p = .026, η^2 = .036, and a a significant main effect of LSA, F (2, 234) = 42.438, p = .001, η^2 = .266, with a significant interaction between LSA and Judge 1, F (2, 234) = 6.11, p = .001, η^2 = .05. F1 measure agreement was highest for LSA lenient in every human judge threshold condition.

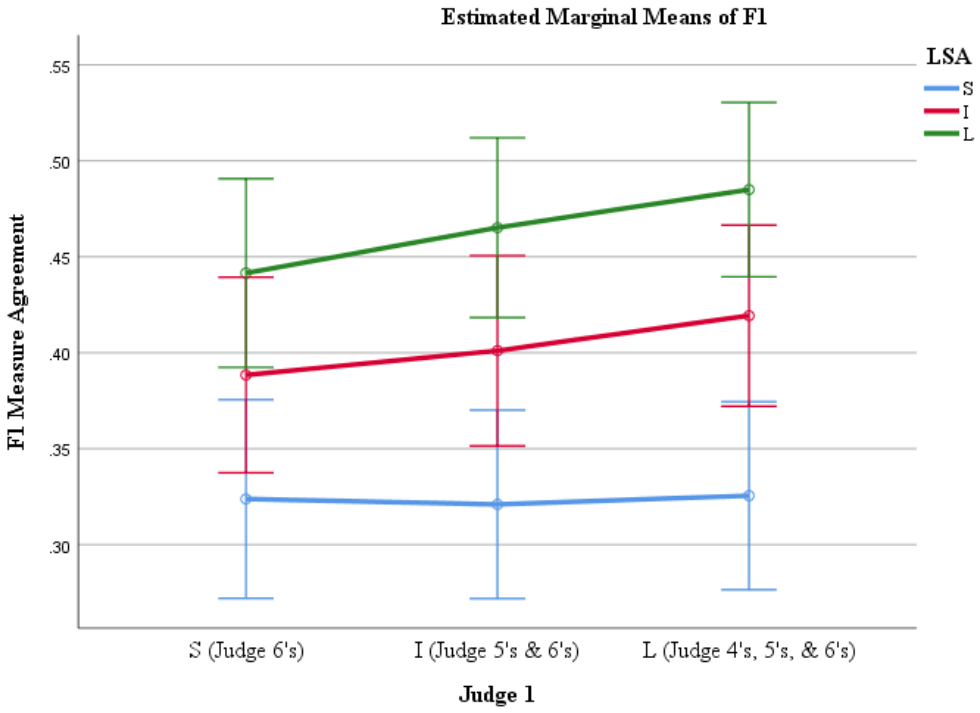

Figure 3. Mean F1 measure as a function of the thresholds of Judge 1 and LSA where *S* is stringent, *I* is intermediate, and *L* is lenient.

Additionally, Figure 4 shows mean F1 scores as a function of stringent (S), intermediate (I), and lenient (L) thresholds of LSA versus Judge 2 in a 3 x 3 repeated measures design. An ANOVA

was conducted on the F1 agreement scores where the unit of analysis was the item (question) in AutoTutor (n = 118). There was a significant main effect of Judge 2, F (2, 234) = 3.800, p = .042, η^2 = .032, and a significant main effect of LSA, F (2, 234) = 62.360, p = .001, η^2 = .348, with significant interaction between LSA and Judge 2, F (2, 234) = 13.950, p = .001, η^2 = .107. Again. F1 measure agreement was highest for LSA lenient in every human judge threshold condition. By itself, LSA performs reasonably well in lenient, but not in stringent or intermediate threshold categories.

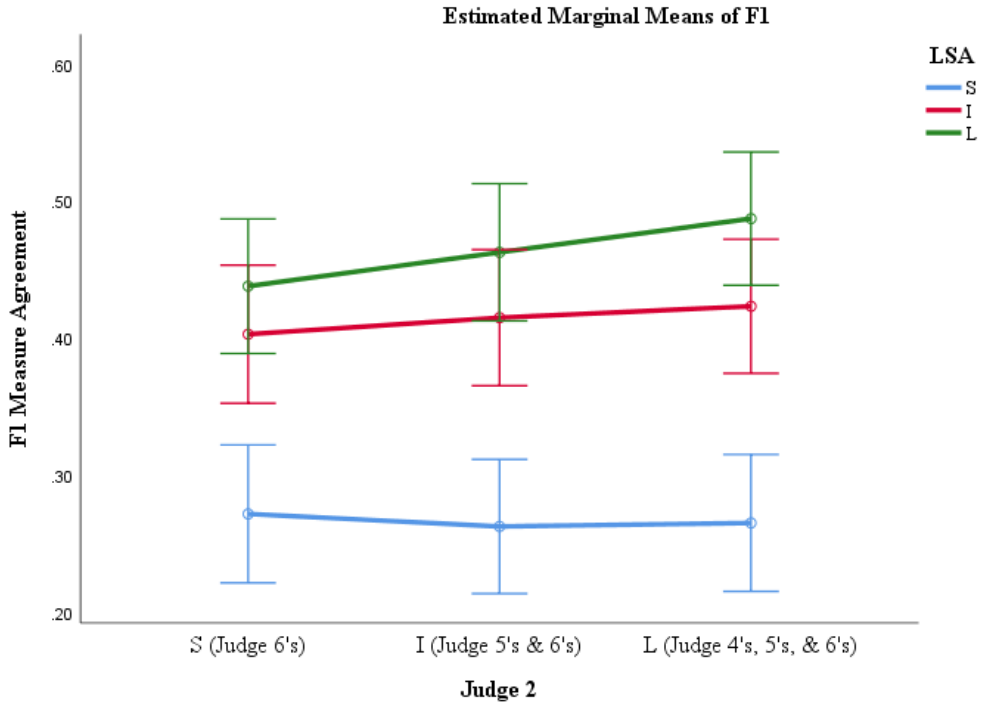

Figure 4. Mean F1 measure as a function of the thresholds of Judge 2 and LSA where *S* is stringent, *I* is intermediate, and *L* is lenient.

Repeated Measures: RegEx Versus Judges

Figure 5 shows mean F1 scores as a function of stringent (S), intermediate (I), and lenient (L)

thresholds of RegEx versus Judge 1 in a 3 x 3 repeated measures design. An ANOVA was conducted on the F1 agreement scores where the unit of analysis was the item (question) in AutoTutor (n = 118). There was a significant main effect of Judge 1, F (2, 234) = 25.530, p = .001, η2 = .181, and a significant main effect of RegEx, F (2, 234) = 26.390, p = .001, η2 = .184, with a significant interaction between RegEx and Judge 1, F (2, 234) = 41.010, p = .001, η2 = .260. Here, we continue to see a trend where lenient F1 value observations are the highest, and stringent and lenient conditions appear to be more distinct than intermediate conditions.

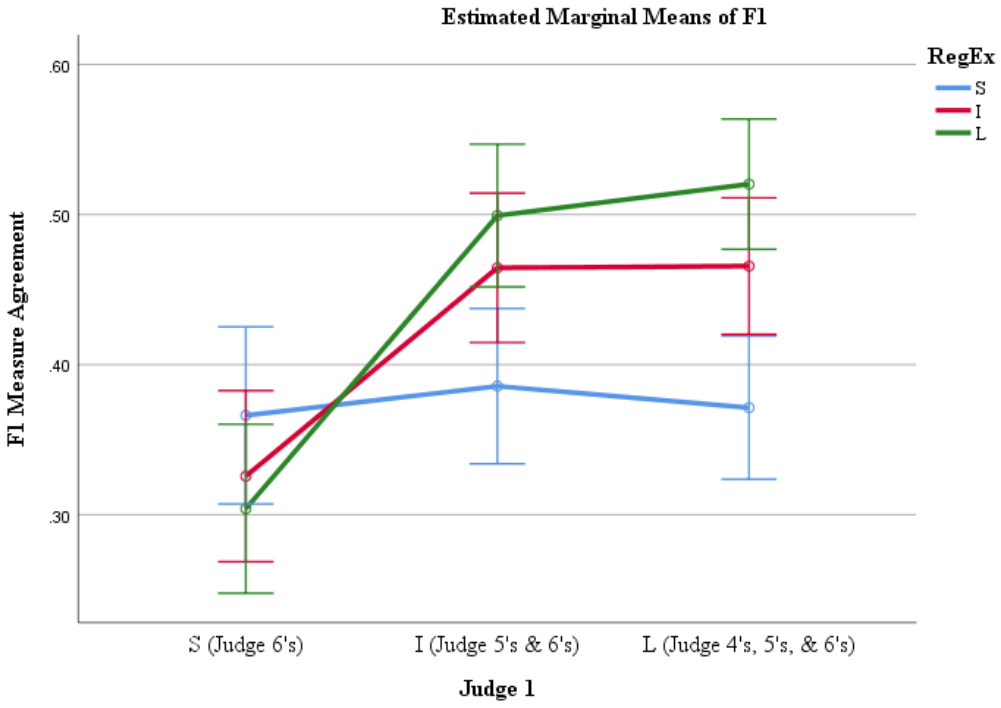

Figure 5. Mean F1 measure as a function of the thresholds of Judge 1 and RegEx where *S* is stringent, *I* is intermediate, and *L* is lenient.

Additionally, Figure 6 shows mean F1 scores as a function of stringent (S), intermediate (I), and lenient (L) thresholds of RegEx versus Judge 2 in a 3 x 3 repeated measures design. An ANOVA was conducted on the F1 agreement scores where the unit of analysis was the item (question) in AutoTutor (n = 118). There was a significant main effect of Judge 2, F (2, 234) = 10.460, p = .001, η2 = .082, and a significant main effect of RegEx, F (2, 234) = 59, p = .001, η2 = .335, with a significant interaction between RegEx and Judge 2, F (2, 234) = 43.55, p = .001, η2 = .271. Agreement between Judge 2 and RegEx is similar to agreement between Judge 1 and RegEx, however, agreement on a stringent threshold is less distinct for Judge 2 and RegEx.

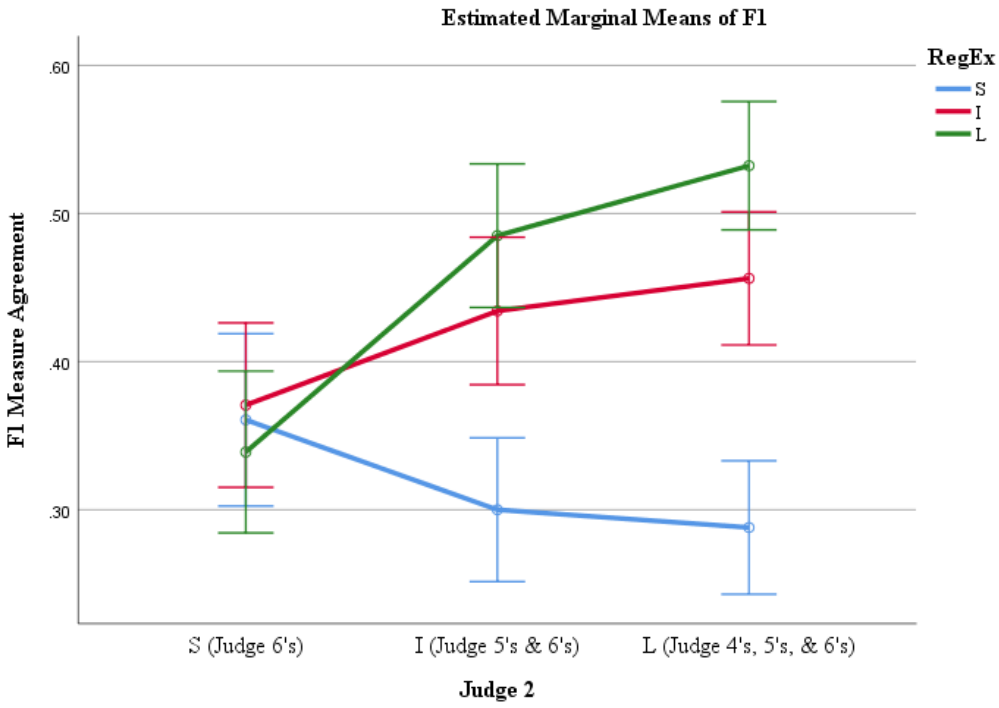

Figure 6. Mean F1 measure as a function of the thresholds of Judge 2 and RegEx where *S* is stringent, *I* is intermediate, and *L* is lenient.

Repeated Measures: RegEx/LSA Combination Versus Judges

Figure 7 shows mean F1 scores as a function of stringent (S), intermediate (I), and lenient (L) thresholds of the RegEx-LSA combination model versus Judge 1 in a 3 x 3 repeated measures design. An ANOVA was conducted on the F1 agreement scores where the unit of analysis was the item (question) in AutoTutor (n = 118). There was a significant main effect of Judge 1, $F(2, 234) = 18.440$ $p = .001$, $\eta 2 = .136$, and a significant main effect of Combo, $F(2, 234) = 38.500$, $p = .001$, $\eta 2 = .248$, with a significant interaction between Combo and Judge 1, $F(2, 234) = 48.380$, $p = .001$, $\eta 2 = .293$. Here, we see F1 measure agreement values are most comparable in the study to values between humans. While intermediate thresholds are less distinct, we see distinction in stringent and lenient threshold agreements. This follows the same trend in all observations where F1 measure agreement is highest in lenient thresholds and overall lowest in stringent thresholds.

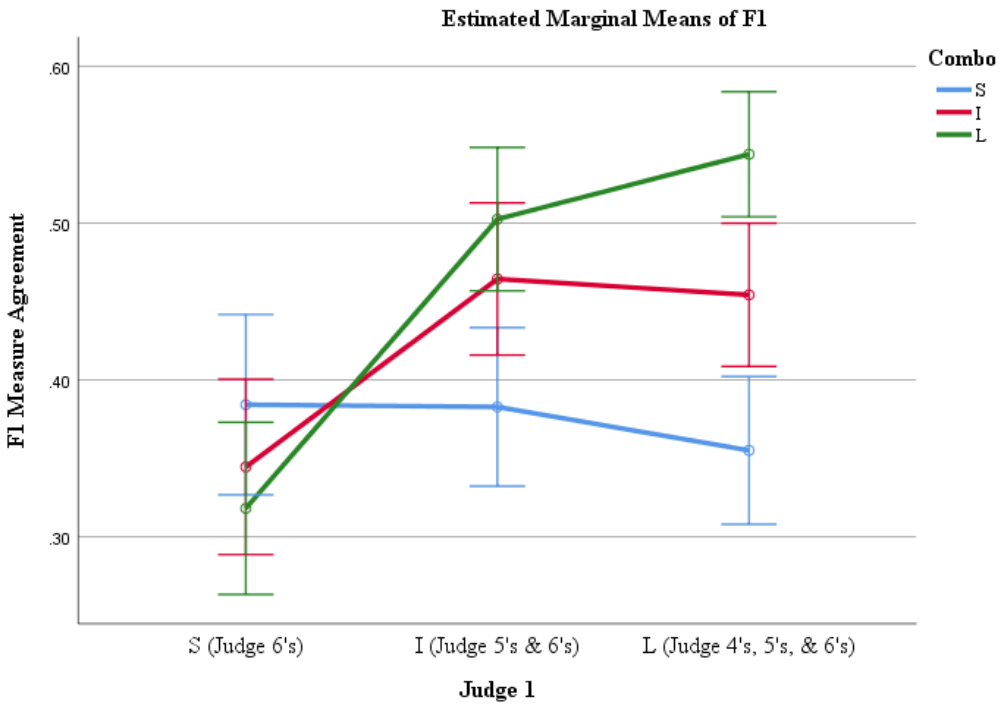

Figure 7. Mean F1 measure as a function of the thresholds of Judge 1 and RegEx-LSA combination where *S* is stringent, *I* is intermediate, and *L* is lenient.

Additionally, Figure 8 shows mean F1 scores as a function of stringent (S), intermediate (I), and lenient (L) thresholds of the RegEx-LSA combination model versus Judge 2 in a 3 x 3 repeated measures design. An ANOVA was conducted on the F1 agreement scores where the unit of analysis was the item (question) in AutoTutor (n = 118). There was a significant main effect of Judge 2, F (2, 234) = 23.500, p = .001, η2 = .167, and a significant main effect of Combo, F (2, 234) = 85.640, p = .001, η2 = .423, with a significant interaction between Combo and Judge 2, F (2, 234) 4= 62.120, p = .001, η2 = .347. Here, we follow the same trend as we have seen previously where F1 measure agreement is highest in lenient and overall lowest in stringent. We see the highest overall F1 measure

agreements in the study (aside from between humans) in the lenient computer by lenient judge threshold observation.

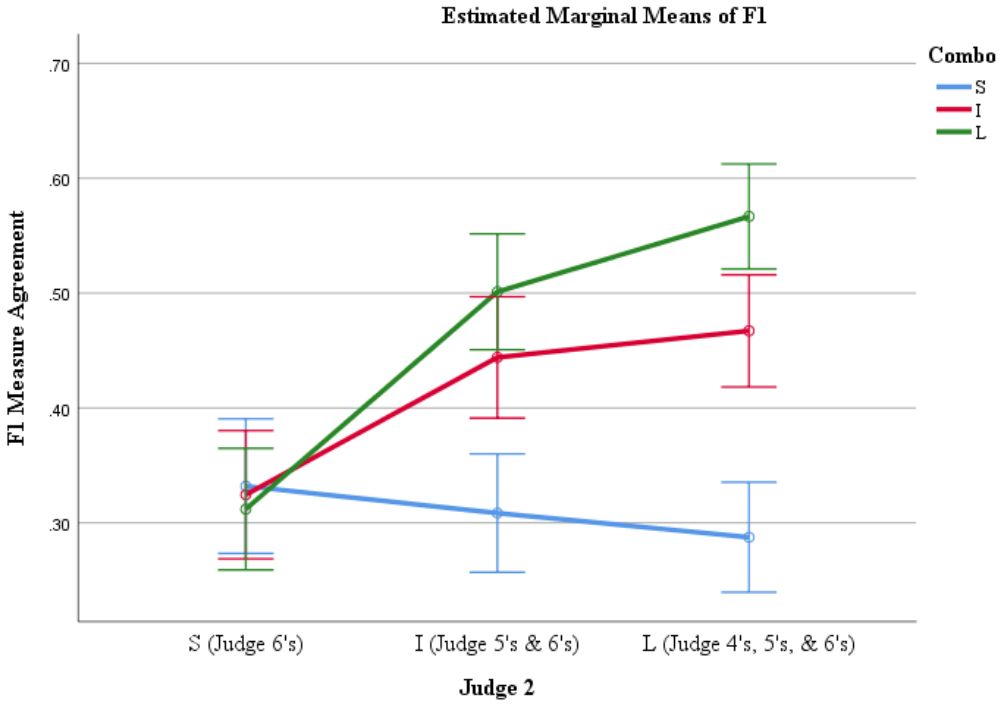

Figure 7. Mean F1 measure as a function of the thresholds of Judge 2 and RegEx-LSA combination where *S* is stringent, *I* is intermediate, and *L* is lenient.

Chapter 5

Discussion

The first analysis examined the non-parametric relationship between two automated methods, LSA and RegEx, which provide complementary evaluations using different text features to assess response relevance. As a result, we expected to see a moderate, positive relationship between LSA and RegEx which was indeed the case. We expected a moderate, positive

relationship because RegEx and LSA share the same purpose of producing match values for user responses based on response relevance, but they each do so by considering different features in the text. Thus, a higher correlation of .9 (or closer to 1) would indicate that the use of both methods together may be redundant, and a much lower correlation would indicate that both are not measuring the same variable.

Although statistics for Cohen's kappa were not impressively high, we expected to see a trend based on results from previous analyses in Carmon et al. (2018; 2019) where agreement between human judges was consistently higher than between human judges and the computer. In this study, we observed that trend across all 3 categories of threshold (stringent, intermediate, and lenient). However, the RegEx/LSA combination model agreed more closely with human judges than either the RegEx only or LSA only model, or to any other model previously fit to the dataset (Carmon et al., 2018, Carmon et al., 2019). This is true across multiple performance metrics (F1, 'd, and Kappa) reported for the models at the aggregate level Additionally, the same simulation developed for finding all values of the discrimination threshold can be easily modified to optimize for kappa, 'd, precision, or recall. In this study, we optimize discrimination thresholds of the combination model to optimize for F1 measure agreement.

These findings suggest that there may be benefits to human-computer agreement in automatic short answer grading (ASAG) contexts by using a combination model in rather than solely relying on RegEx only or LSA only models individually. Also suggested by the findings is the notion that models can be custom-trained and simulated to optimize relative agreement between the computer and categorized judge ratings.

Based on previous analyses, we expected to see higher agreement in lenient thresholds than in stringent thresholds. This is generally true apart from Cohen's kappa. Cohen's kappa adjusts for agreement by chance, so in this context, we may expect more lenient thresholds to yield higher F1 agreement, and similar but slightly lower kappa than one would observe in stringent thresholds. This is rather intuitive, because the more leniency you lend to response appropriateness, the more likely it is that two or more assessors may display overlapping agreement. For example, in stringent thresholds, only ratings of 6 (out of 6) displayed by both judges may be considered agreement, whereas in lenient thresholds, a judge rating of 4 paired with another judge rating of 5 would be considered an agreement. Precision reached .809 in lenient thresholds for response assessment, but at the expense of yielding lower recall (.463).

In the combination model, we observe F1 measure agreement between humans and the computer similar to the F1 measure agreements observed between humans. This is true in all three threshold categories. These F1 measure agreements between humans and the computer are more similar than any previous model used to assess this response data to date (Carmon et al., 2018, Carmon et al., 2019). In stringent thresholds, between-human F1 measure agreement was .532, and human-computer F1 measure agreement reached .526. In intermediate thresholds, between-human F1 measure agreement was .599, and human-computer F1 measure agreement reached .548. In lenient thresholds, between-human F1 measure agreement was .653, and human-computer F1 measure agreement reached .620.

As demonstrated in this study, semantic text models for ASAG can be modified in various ways to optimize agreement. Relevant literature in semantic matching for student response grading

typically trains computational models based on 5 holistic response categories (Crossley et al., 2014), whereas this study details 6 response categories for human raters. For this reason, one alternative may recode the response data on a categorization scale using judgements of 1–5 in the future. Using a 5-category scale, the data can be used in preexisting models. However, simulations for discrimination threshold values show promise in fitting match values in the response data to atypical response categorization scales for holistic human judge ratings such as 6-category scales like the one used in this study.

Moving forward, it may be of interest to weight the ratings of first human judge as more consistent. Although the difference was slight, the first judge (Judge 1) consistently agreed more closely with the computer than did the second human judge. The agreement between the first human judge and the computer was generally higher and closer to agreement between-humans in all three threshold categories observed in analyses.

Ideally, agreement between three or more subject-matter experts (as opposed to two) should be observed to identify potentially poor or inconsistent human judgement. If agreement between two judges is consistently high while a third judge has exceedingly low agreement with each other judge, then it is likely that the third judge is not as reliable which can have negative consequences for the internal validity of the study. Though agreement is relative in the context, these findings identify a trend where judges consistently agree more than AutoTutor versus judges across different levels of thresholds In this case, F1 measure agreement between the computer and humans is substantially similar to agreement between humans. This offers a valuable framework for future tests observing new AutoTutor models and the degree to which they may improve. New

semantic combination models are not necessarily limited to including LSA and RegEx.

Due to the nature of open-ended responses collected, it is appropriate to analyze main question and expectations. However, deciding on a length cap for these open-ended responses may help to avoid false positives in LSA which may be attributed to lengthy responses increasing the likelihood that relevant terms are used which increase the LSA score while they are not satisfying the expected answer.

In future studies we may also want to collect similar data on participants that appear in person and interact with these questions through the AutoTutor system rather than collecting on AutoTutor materials through AMT. We may sacrifice some sample size, but it is critically important to see how users interact with the system in an AutoTutor trialogue with hints, pumps, prompts, conversational agents, and dialogue turns as opposed to assessment of an open-ended user response made in attempt to only the main question. We may also want to rerun similar analyses on the data where we modify the length or content of the corpus used to train the LSA space (e.g. removing physics texts), or the features of RegEx (e.g., removing anticipated synonyms versus complex student answers, etc.).

A future analysis may use the response data to look at ratings of 1 and 2 to identify whether the system correctly detects metacognitive language in responses. Successfully distinguishing partial answers from complete answers and recognizing certain response features such as meta-cognitive language (represented by a response rating of 2) is necessary in helping the system select the most appropriate hint or prompt. It is of critical importance that the system correctly and consistently recognizes these response categories. Additionally, two future analyses, an error

analysis and item difficulty analysis, are of immediate interest and show promise in acquiring necessary and relevant information for this body of research.

An error analysis would explore whether the system is better at detecting correct answers or incorrect answers in accordance with judge response ratings. In human-computer interaction contexts, and especially in automatic answer grading, it is important to consider the types of errors that the system is committing while scoring student answers. The ElectronixTutor system may commit two types of error, misses or false alarms. In these contexts, we would rather have the system minimize false alarms or misses? It is not ideal to pass a student answer that should not have been counted as correct (i.e., a false alarm), but it is also problematic to miss a student answer when it should have been detected as correct. Rejecting correct student answers may frustrate or discourage students and cause dropout (Graesser, 2016). Depending on whether correct or incorrect answers are detected more consistently by the system, it may be justifiable to weight the priority of either precision or recall on the data.

In the item difficulty analysis, we could explore whether item or question difficulty affects the degree of agreement between humans or between the computer and humans. We currently are interested in two approaches in this effort. One can be thought of as a top-down approach, and the other can be thought of as a bottom-up approach. In the top-down approach, we use an existing scaling of electronics topics conducted by experts to analyze item difficulty based on knowledge components in the ElectronixTutor system (Graesser et al., 2018). Using this expert model, we can observe the relationship between item difficulty and agreement between humans as well as agreement between the computer and humans. Similarly, we can compare question items that

require lengthier answers and reasoning (2-5 expectations) versus knowledge-check questions that focus on only 1 or 2 expectations. In the bottom-up approach we would observe the relationship between agreement and item difficulty where item difficulty is informed by average score on each item within the dataset. This normative definition of item difficulty would consider how learners perform and vary in answering each individual question.

References

Aleven, V., Ogan, A., Popescu, O., Torrey, C., Koedinger, K. (2004). Evaluating the effectiveness of a tutorial dialogue system for self-explanation. In J.C. Lester, R.M. Vicari, F. Paraguacu (Eds.), *Proceedings of the 7th international conference on intelligent tutoring systems volume 3220 of lecture notes in computer science* (pp. 443–454). Maceio: Springer.

Bloom, B. S. (1984). The 2 sigma problem: the search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher, 13*, 4-16.

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

Cai, Z., Graesser, A. C., Forsyth, C., Burkett, C., Millis, K., Wallace, P., Halpern, D., & Butler, H. (2011). Trialog in ARIES: User Input Assessment in an Intelligent Tutoring System. In W. Chen, & S. Li (Eds.), *Proceedings of the 3rd IEEE international conference on intelligent computing and intelligent systems* (pp. 429–433). Guangzhou: IEEE Press.

Carmon, C. M., Hampton, A. J., Morgan, B., Cai, Z., Wang, L., & Graesser, A. C. (2019). Semantic matching evaluation of user responses to electronics questions in AutoTutor. In *Sixth (2019) ACM Conference on Learning @ Scale* (4 pages), Chicago, IL: ACM. https://doi.org/10.1145/3330430.3333649

Carmon, C., Morgan, B., Hampton, A. J., Cai, Z., & Graesser, A. C. (2018). Semantic matching evaluation in ElectronixTutor. In K. E. Boyer, & M. Yudelson (Eds.), *Proceedings of the 11th International Conference on Educational Data Mining* (pp. 580–583). Buffalo, NY: EDM

Society.

Cohen, P. A., Kulik, J. A., & Kulik, C. L. C. (1982). Educational outcomes of tutoring: a meta-analysis of findings. *American Educational Research Journal, 19*, 237-248.

Crossley, S., Kyle, K., Allen, L. K., Guo, L., & McNamara, D. (2014). Linguistic microfeatures to predict L2 writing proficiency: a case study in automated writing evaluation. *The Journal of Writing Assessment, 7*, 1-16.

Dorca, F. (2015). Implementation and use of simulated students for test and validation of new adaptive educational systems: *A practical insight. International Journal of Artificial Intelligence in Education 25*, 319–345.

Dzikovska, M., Steinhauser, N., Farrow, E., Moore, J., & Campbell, G. (2014). BEETLE II: deep natural language understanding and automatic feedback generation for intelligent tutoring in basic electricity and electronics. *Int J Artif Intell Educ, 24*, 284–332.

Evens, M. W., Brandle, S., Chang, R. C., Freedman, R., Glass, M., Lee, Y. H., Shim, L. S., Woo, C. W., Zhang, Y., Zhou, Y., Michael, J. A., Rovick, A. A. (2001). CIRCSIM-Tutor: an intelligent tutoring system using natural language dialogue, In *Proceedings of the 12th midwest artificial intelligence and cognitive science conference* (pp. 16–23). Oxford.

Gautam, D., Swiecki, Z., Shaffer, D. W., Graesser, A. C., & Rus, V. (2017). Modeling classifiers for virtual internships without participant data. In X. Hu, T. Barnes, A. Hershkovitz, L. Paquette (Eds), *Proceedings of the 10th International Conference on Educational Data Mining* (pp. 278–283). Wuhan, China: EDM Society.

Graesser, A. C. (2016). Conversations with AutoTutor help students learn. *International Journal of Artificial Intelligence in Education, 26,*124-132.

Graesser, A. C. (2020). Learning science principles and technologies with agents that promote deep learning. In R.S. Feldman (Ed.), *Learning science: Theory, research, and practice* (pp. 2–33). New York: McGraw-Hill.

Graesser, A. C., D'Mello, S. K., Hu. X., Cai, Z., Olney, A., & Morgan, B. (2012). AutoTutor. In P. McCarthy and C. Boonthum-Denecke (Eds.), *Applied natural language processing: Identification, investigation, and resolution* (pp. 169-187). Hershey, PA: IGI Global.

Graesser, A. C., Hu, X., Nye, B., VanLehn K., Kumar, R., Heffernan, C., Heffernan, N., Woolf, B., Olney, A. M., Rus, V., Andrasik, F., Pavlik, P., Cai, Z., Wetzel, J., Morgan, B., Hampton, A. J., Lippert, A. M., Wang, L., Chen, Q., Vinson IV, J. E., Kelly, C. N., McGlown, C., Majmudar, C. A., Morshed, B., and Baer, W. (2017). ElectronixTutor: an intelligent tutoring system with multiple learning resources for electronics. In *International Journal of STEM Education: Innovations and Research*. DOI:10.1186/s40594-017-0072-5.

Graesser, A. C., Lu, S., Jackson, G. T., Mitchell, H. H., Ventura, M., Olney, A. M., Louwerse, M. M. (2004). AutoTutor: A tutor with dialogue in natural language. *Behavior Research Methods, Instruments, & Computers, 36*, 180-193.

Graesser, A. C., Person, N. K., & Magliano, J. P. (1995). Collaborative dialogue patterns in naturalistic one-to-one tutoring. *Applied Cognitive Psychology, 9*, 495–522.

Graesser, A. C., Wiemer-Hastings, K., Kreuz, R. J., Wiemer-Hastings, P., & Marquis, K. (2000). QUAID: A questionnaire evaluation aid for survey methodologists. *Behavior Research Methods, Instruments, & Computers, 32*(2), 254-262.

Haley, D. T., Thomas, P., Roeck, A. D., Petre, M. (2007). Measuring improvement in latent semantic analysis-based marking systems: using a computer to mark questions about HTML. In S. Mann & Simon (Eds.), *Proceedings of the 9th australasian conference on computing*

education, volume 66 of ACE (pp. 35–42). Ballarat: Australian Computer Society.

Halpern, D. F., Millis, K., Graesser, A. C., Butler, H., Forsyth, C., & Cai, Z. (2012). Operation ARA: A computerized learning game that teaches critical thinking and scientific reasoning. *Thinking Skills and Creativity, 7*, 93–100.

Ivens, S., & Koslin, B. (1991). *Demands for Reading Literacy Require New Accountability Methods*. Brewster, NY: Touchstone Applied Science Associates.

Jackson, G. T., Ventura, M. J., Chewle, P., Graesser, A. C., & the Tutoring Research Group. (2004). The Impact of Why/AutoTutor on learning and retention of conceptual physics. In J. C. Lester, R. M. Vicari, & F. Paraguacu (Eds.), *Intelligent Tutoring Systems 2004* (pp. 501–510). Berlin, Germany: Springer.

Jurafsky, D., & Martin, J. (2008). *Speech and language processing*. Englewood: Prentice Hall.

Landauer, T. K., McNamara, D. S., Dennis, S., & Kintsch, W. (Eds.). (2007). *Handbook of latent semantic analysis*. Mahwah, NJ: Erlbaum.

Lesgold, A., Lajoie, S. P., Bunzo, M., & Eggan, G. (1992). SHERLOCK: a coached practice environment for an electronics trouble-shooting job. In J. H. Larkin & R. W. Chabay (Eds.), *Computer assisted instruction and intelligent tutoring systems: Shared goals and complementary approaches* (pp. 201–238). Hillsdale, NJ: Erlbaum.

Li, H., Gobert, J. & Graesser, A. C., & Dickler, R. (2018). Advanced educational technology for science inquiry assessment. *Policy Insights from the Behavioral and Brain Sciences, 5*, 171-178.

Nye, B. D., Graesser, A. C., & Hu, X. (2014). AutoTutor and family: a review of 17 years of natural language tutoring. *Int J Artif Intell Educ, 24*(4), 427–469.

Olney, A., D'Mello, S. K., Person, N., Cade, W., Hays, P., Williams, C., Lehman, B., & Graesser, A. C. (2012). Guru: a computer tutor that models expert human tutors. In S. Cerri, W. Clancey, G. Papadourakis, & K. Panourgia (Eds.), *Proceedings of Intelligent Tutoring Systems (ITS) 2012* (pp. 256–261). Berlin: Springer.

VanLehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems and other tutoring systems. *Educational Psychologist 46*, 197–221. 17.

VanLehn, K., Chung, G., Grover, S., Madni, A., & Wetzel, J. (2016). Learning science by constructing models: can Dragoon increase learning without increasing the time required? *International Journal of Artificial Intelligence in Education, 26*, 1033–1068.

VanLehn, K., Graesser, A. C., Jackson, G. T., Jordan, P., Olney, A., & Rose, C.P. (2007). When are tutorial dialogues more effective than reading? *Cognitive Science, 31*, 3-62.

VanLehn, K., Jordan, P. W., Rosé, C. P., Bhembe, D., Böttner, M., Gaydos, A., Makatchev, M., Pappuswamy, U., Ringenberg, M. A., Roque, A., Siler, S., & Srivastava, R. (2002). The Architecture of Why2-Atlas: A Coach for Qualitative Physics Essay Writing. *Intelligent Tutoring Systems*.

Appendix: A

Table 4

Agreement Measures Between Humans and between Computer and Humans

Condition	*k*	Precision	Recall	F1	d’
S: Judge 1 x Judge 2	.456	.467	.618	.532	1.593
I: Judge 1 x Judge 2	.466	.636	.565	.599	1.360
L: Judge 1 x Judge 2	.460	.710	.604	.653	1.274
S: LSA x Judge 1	.195	.547	.277	.368	.649
I: LSA x Judge 1	.241	.631	.369	.466	.762
L: LSA x Judge 1	.272	.673	.472	.555	.755
S: LSA x Judge 2	.150	.542	.207	.300	.575
I: LSA x Judge 2	.223	.592	.390	.47	.636
L: LSA x Judge 2	.236	.621	.511	.561	.618
S: RegEx x Judge 1	.366	.448	.440	.444	1.487
I: RegEx x Judge 1	.313	.607	.455	.52	1.07
L: RegEx x Judge 1	.306	.679	.545	.600	.870
S: RegEx x Judge 2	.428	.450	.585	.509	1.275
I: RegEx x Judge 2	.362	.677	.452	.542	.882
L: RegEx x Judge 2	.307	.713	.487	.579	.815
S: Combo x Judge 1	.466	.479	.584	.526	1.508
I: Combo x Judge 1	.365	.710	.446	.548	1.096
L: Combo x Judge 1	.291	.809	.463	.589	.939
S: Combo x Judge 2	.384	.482	.444	.462	1.312

I: Combo x Judge 2	.311	.633	.448	.524	.883
L: Combo x Judge 2	.292	.772	.519	.620	.850

Note. At the aggregate level, overall agreement values in the combination model match the values between humans more closely than either of the stand-alone models.

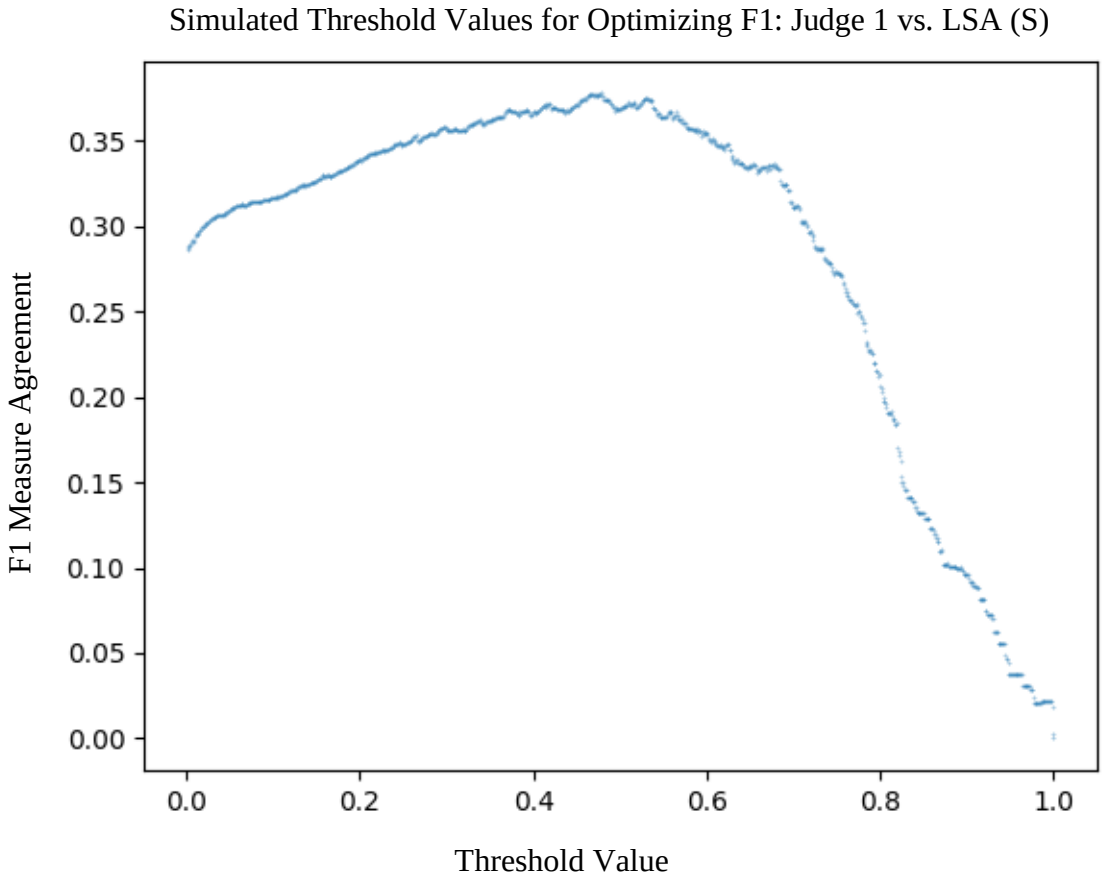

Figure 8. F1 measure agreement simulated for a thousand values of the discrimination threshold between Judge 1 and LSA in stringent thresholds.

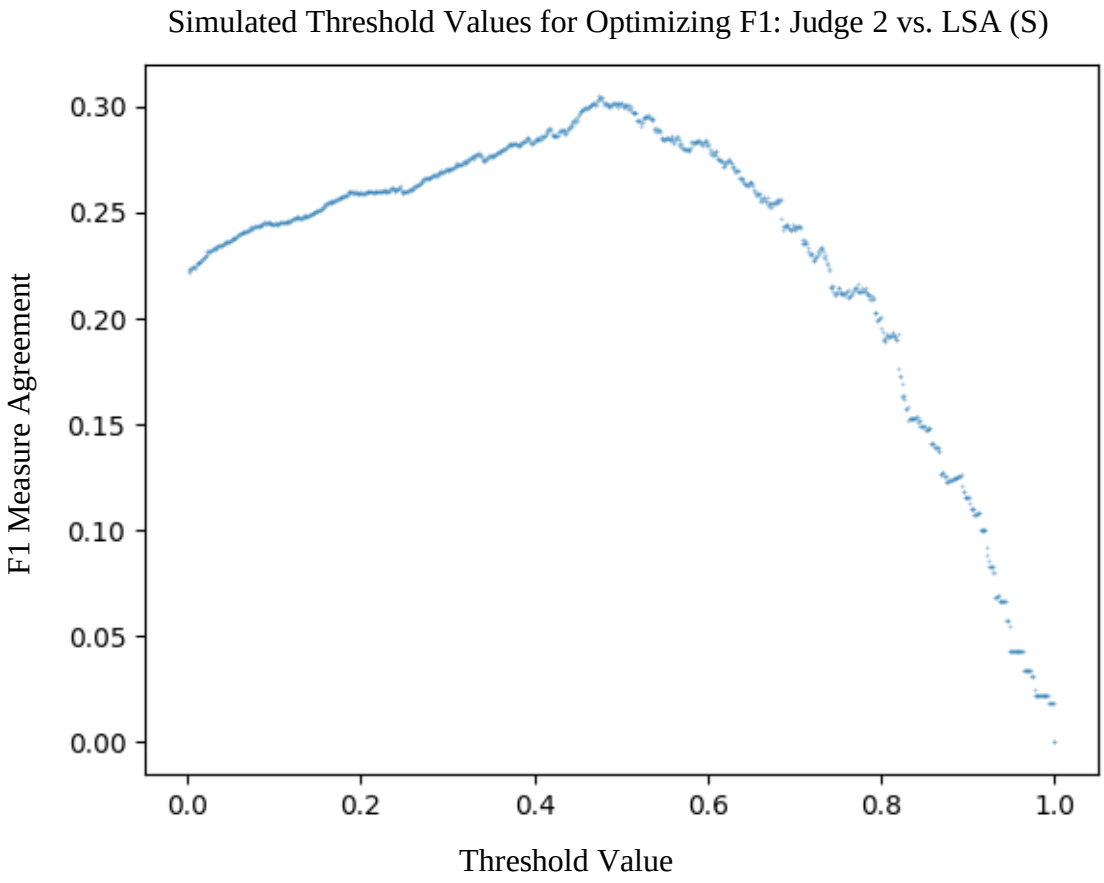

Figure 9. F1 measure agreement simulated for a thousand values of the discrimination threshold between Judge 2 and LSA in stringent thresholds.

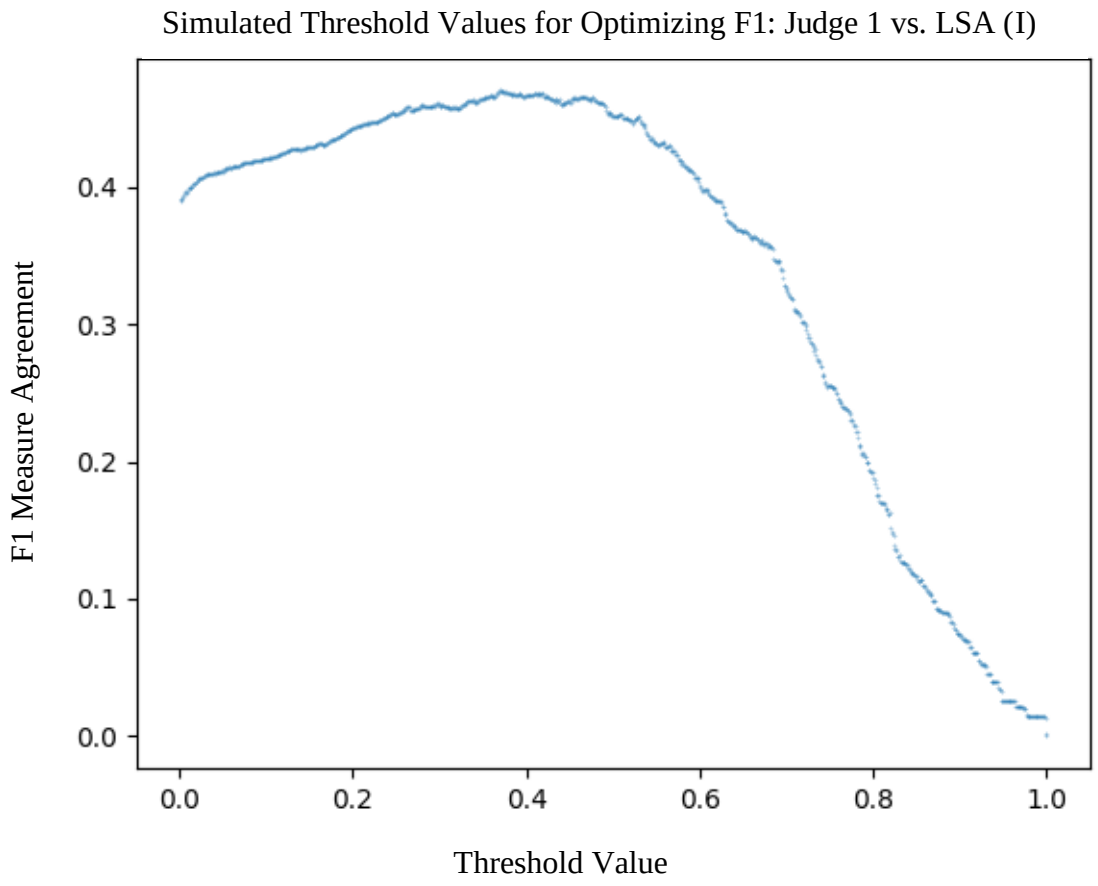

Figure 10. F1 measure agreement simulated for a thousand values of the discrimination threshold between Judge 1 and LSA in intermediate thresholds.

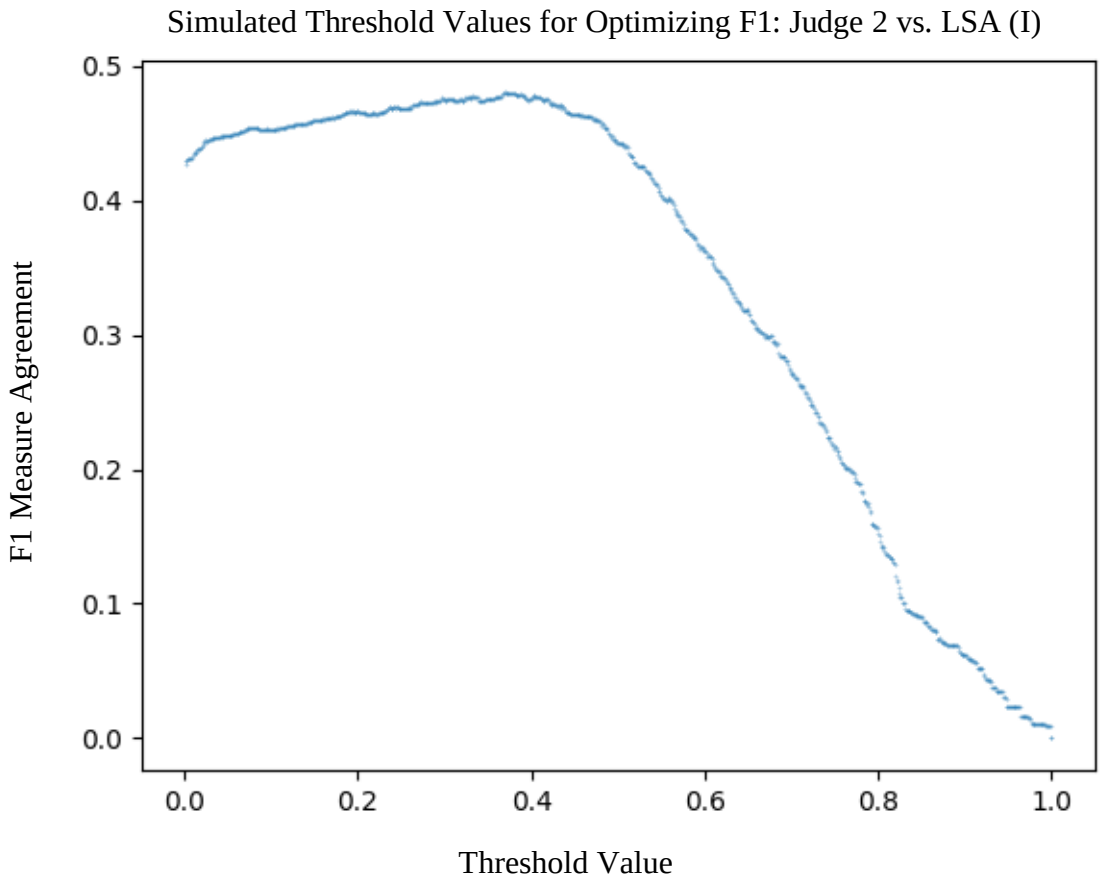

Figure 11. F1 measure agreement simulated for a thousand values of the discrimination threshold between Judge 2 and LSA in intermediate thresholds.

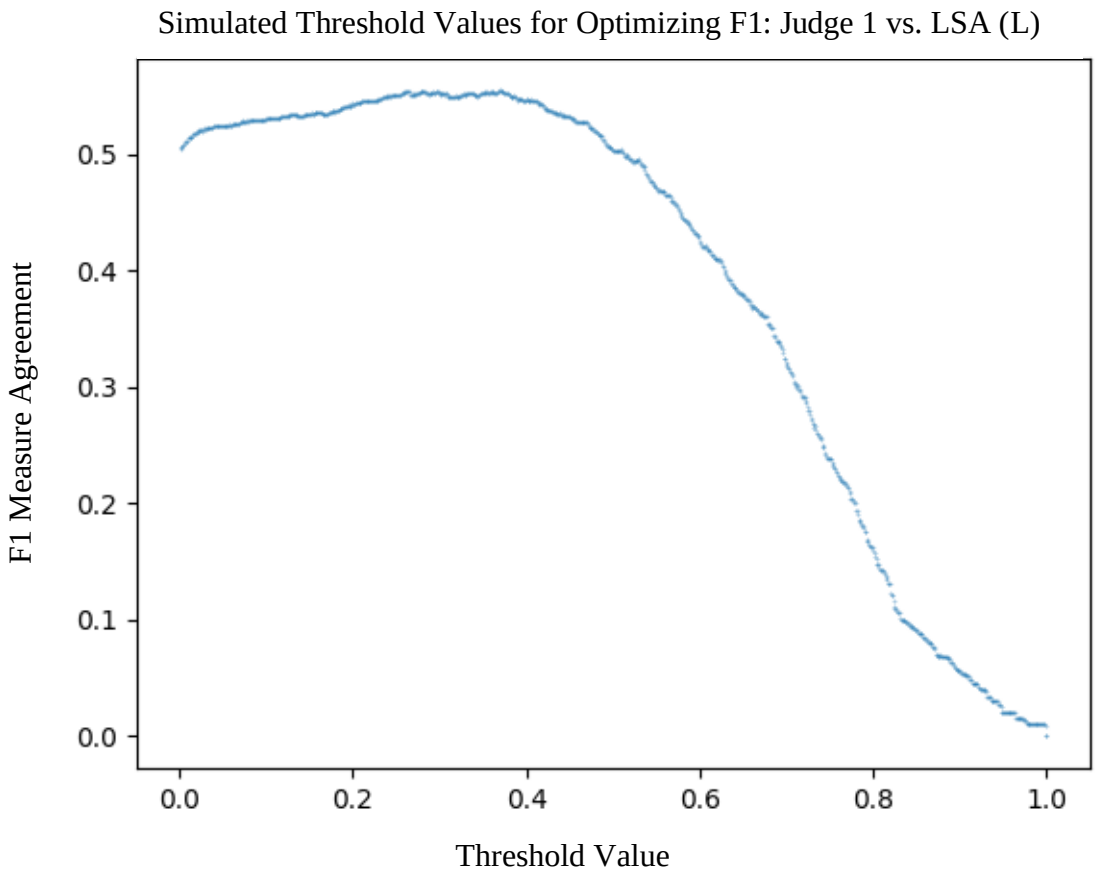

Figure 12. F1 measure agreement simulated for a thousand values of the discrimination threshold between Judge 1 and LSA in lenient thresholds.

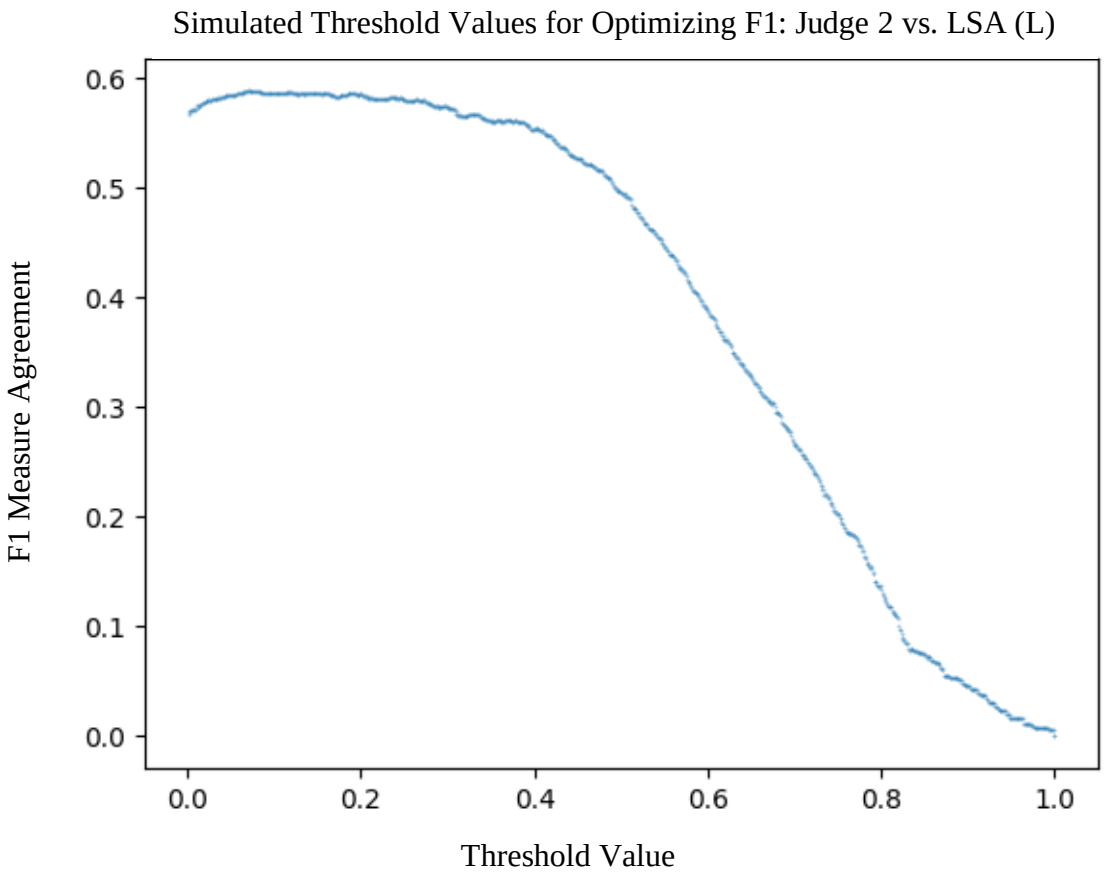

Figure 13. F1 measure agreement simulated for a thousand values of the discrimination threshold between Judge 2 and LSA in lenient thresholds.

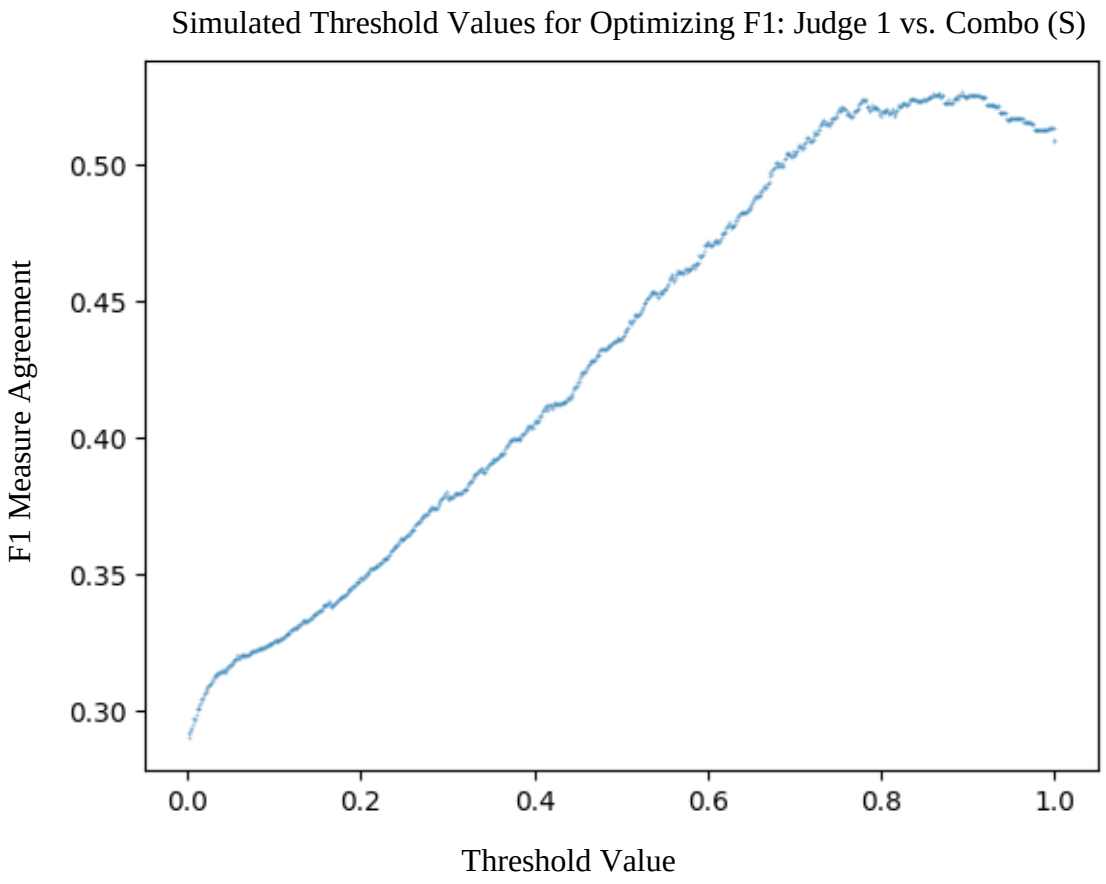

Figure 14. F1 measure agreement simulated for a thousand values of the discrimination threshold between Judge 1 and LSA/RegEx combination model in stringent thresholds.

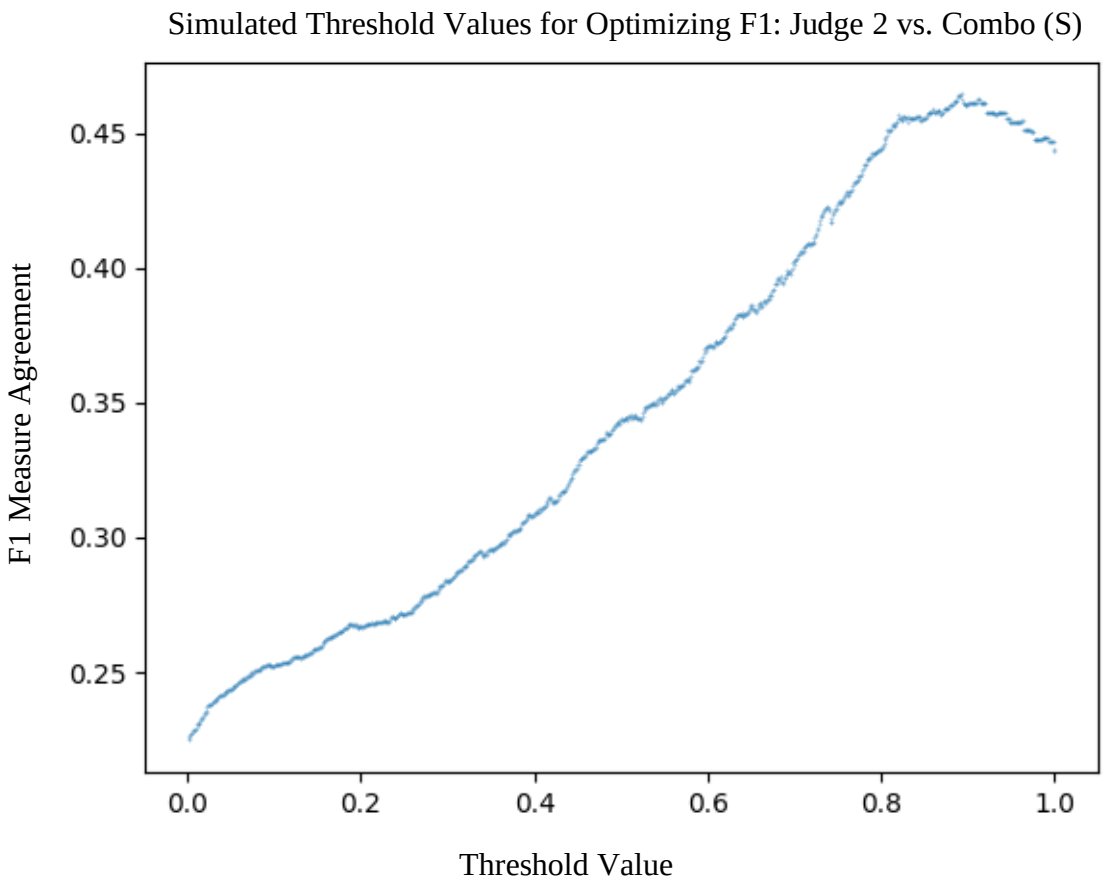

Figure 15. F1 measure agreement simulated for a thousand values of the discrimination threshold between Judge 2 and LSA/RegEx combination model in stringent thresholds.

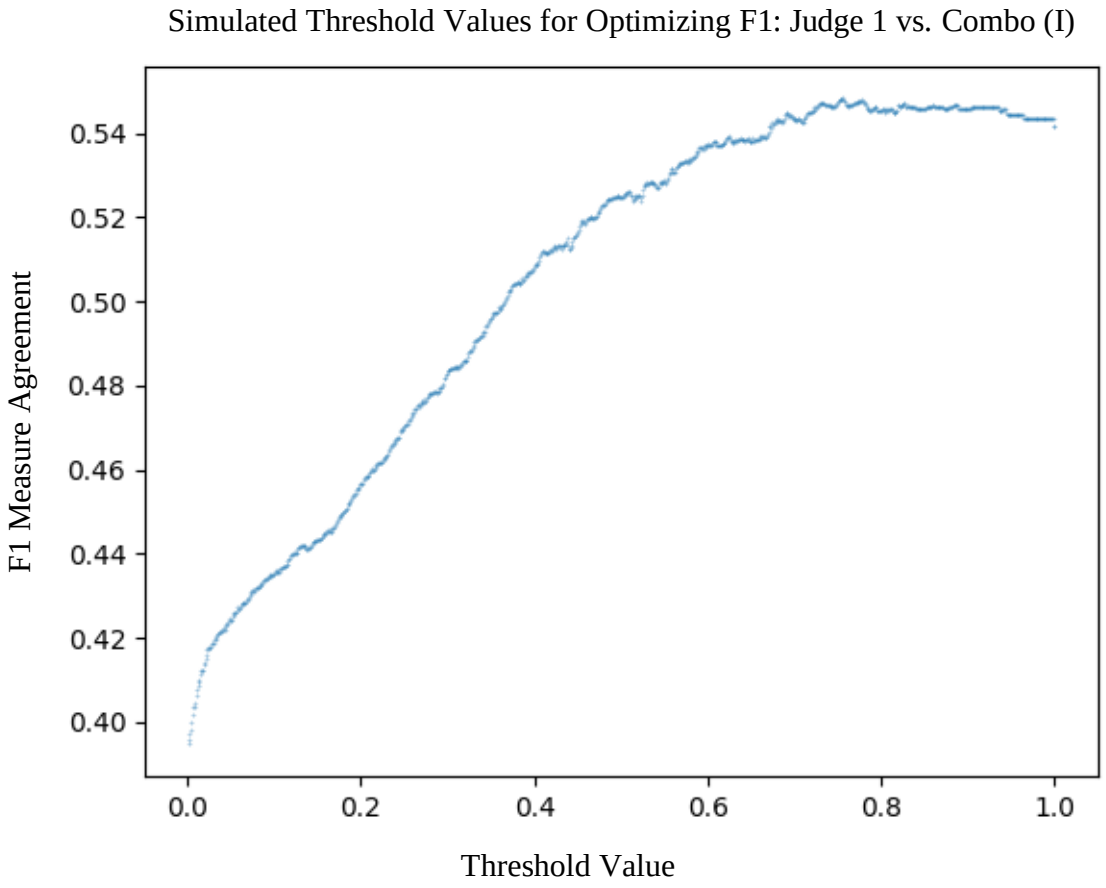

Figure 16. F1 measure agreement simulated for a thousand values of the discrimination threshold between Judge 1 and LSA/RegEx combination model in intermediate thresholds.

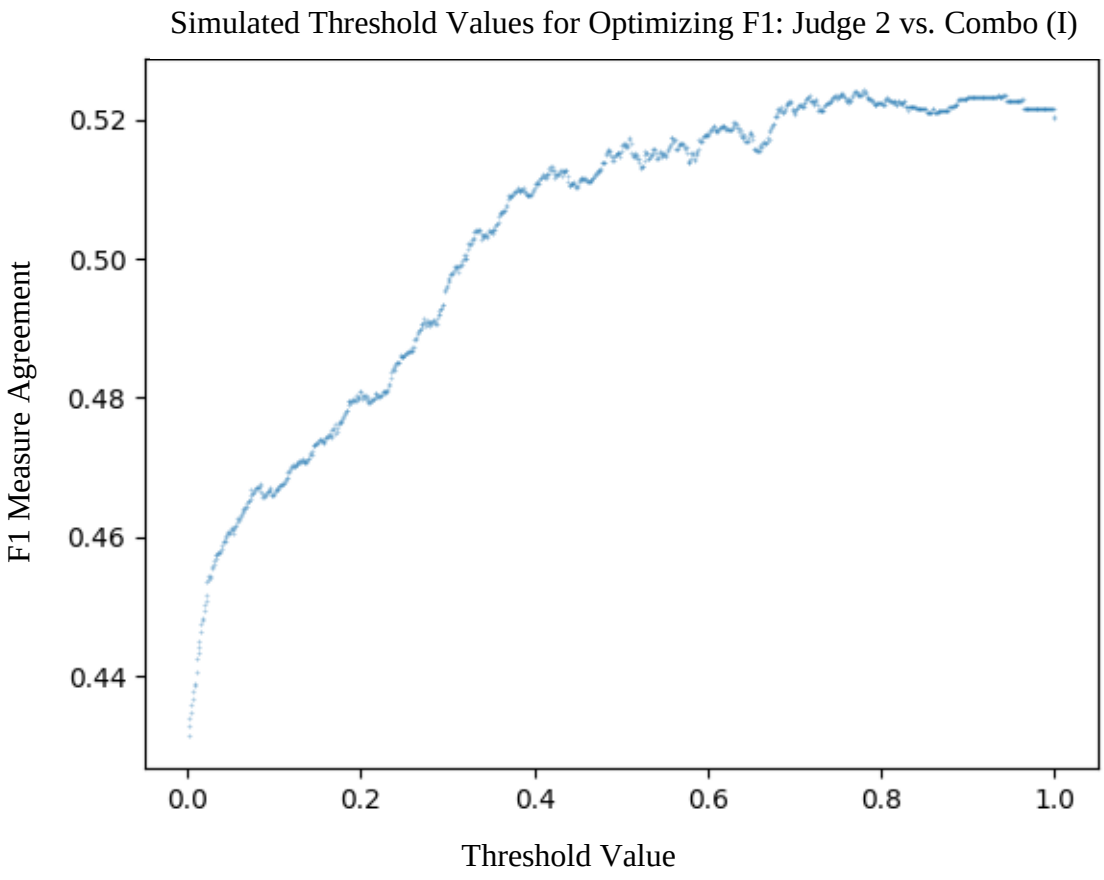

Figure 17. F1 measure agreement simulated for a thousand values of the discrimination threshold between Judge 2 and LSA/RegEx combination model in intermediate thresholds.

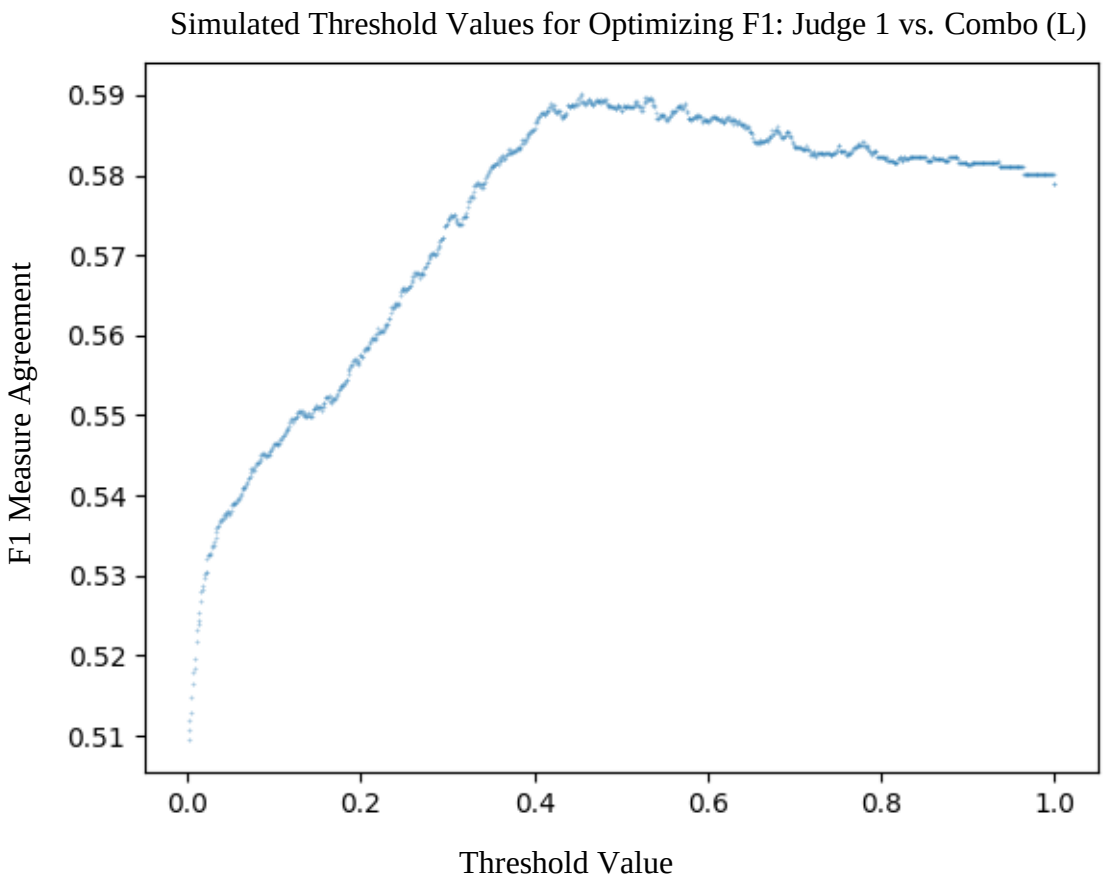

Figure 18. F1 measure agreement simulated for a thousand values of the discrimination threshold between Judge 1 and LSA/RegEx combination model in lenient thresholds.

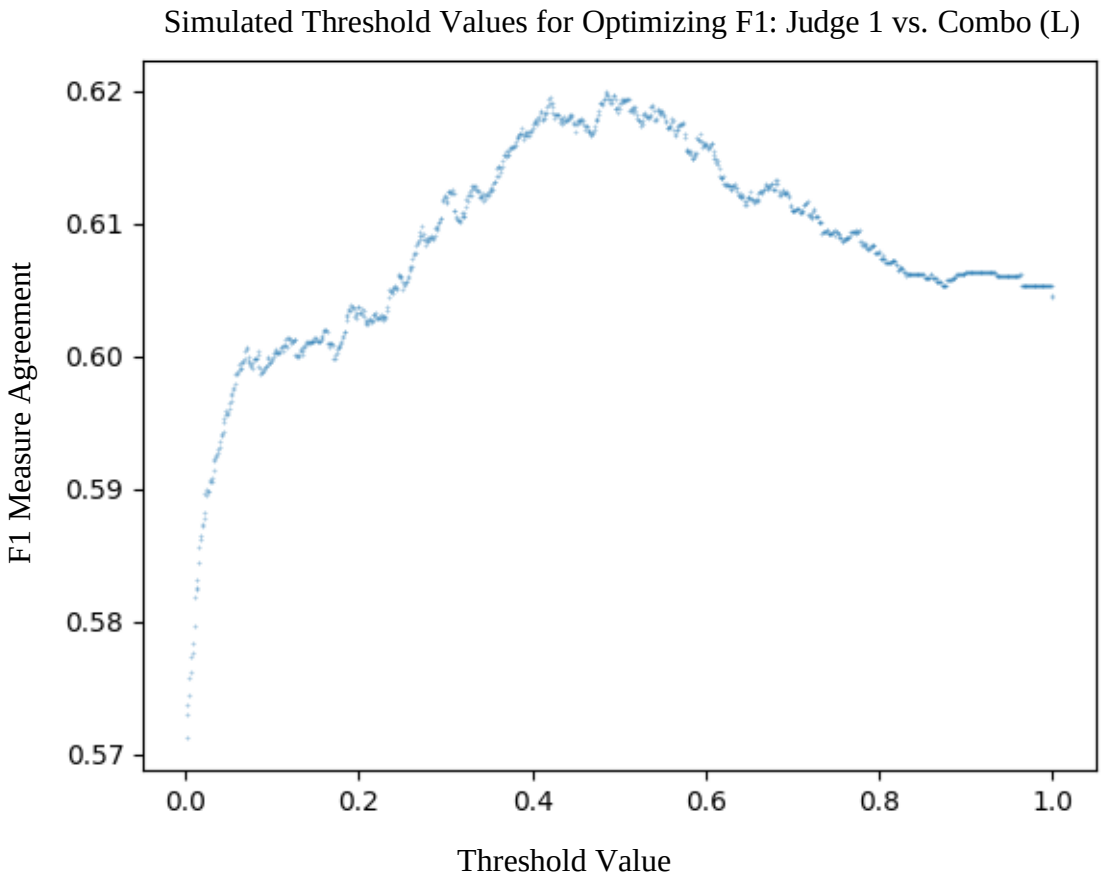

Figure 19. F1 measure agreement simulated for a thousand values of the discrimination threshold between Judge 2 and LSA/RegEx combination model in lenient thresholds.

Appendix: B

Annotated Python Code

Codec-to-txt PDF Version/ Text Chunker/Corpus Writing Tool

```
from urllib import request
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfpage import PDFPage
import re
import urllib
from bs4 import BeautifulSoup
# the line below is for pulling a pdf from a link and downloading it to
your python path folder. the pdf will be later converted to text and then
modified and written to a text corpus.
request.urlretrieve("http://www.rollanet.org/~n0klu/Ham_Radio/(eBook)%20Ele
ctronics%20-%20The%20Electrical%20Engineering%20Handbook.pdf",
"download.pdf")
## the 2 arguments given are ("direct link to pdf", "what you want to name
the download (e.g. "download.pdf", "download13.pdf", "Fundamentals-of-
Electronics.pdf", etc.)")


# this step is not necessary if you download a pdf manually and put it in
the python path folder
def pdf_to_txt(path):
```

```
    rsrcmgr = PDFResourceManager()
    retstr = io.StringIO()
    codec = 'utf-8'
    laparams = LAParams()
    device = TextConverter(rsrcmgr, retstr, laparams=laparams,
imagewriter=None)
    fp = open(path, 'rb')
    interpreter = PDFPageInterpreter(rsrcmgr, device)
    password = ""
    maxpages = 0
    caching = True
    pagenos=set()
    for page in PDFPage.get_pages(fp, pagenos,
maxpages=maxpages,password=password,caching=caching,
check_extractable=True):
        interpreter.process_page(page)

    text = retstr.getvalue()

    fp.close()
    device.close()
    retstr.close()
    return text
text = pdf_to_txt("war-and-peace.pdf") # again, the name of the pdf that
you want to convert to text
print(text[:100])
```

```
#text = convert_pdf_to_txt("war-and-peace.pdf")


listt = []
for i in text.split('\n\n'):
    listt.append(i)
listylistt = []
listylisttlen = []
for i in listt:
    listylisttlen.append(i)
    listylistt.append([i])


listycleaned = []
for i in range(len(listylistt)):                    #getting rid of unwanted
short paragraphs
    if len(listylisttlen[i]) > 50:
        listycleaned.append(listylistt[i])
#where i is the beginning of the body text, and where range is end of body
text minus i:
#                                here we say i=25(beginning of body text),
and range(x)= 10779 (10804-25)
i = 25
for j in range(10779):
    for value in listycleaned[i]:
        i += 1
```

```
        f = io.open("foldy/TestCorpusWriter/snippet01_%s.txt" % i, 'w',
encoding="utf-8")       ## change chunk name ('snippet01_01' becomes
'snippet01_02')
        f.write(value)
        f.close()
```

Codec-to-txt HTML Version/ Text Chunker/Corpus Writing Tool

```
url = "https://olney.ai/neets-web/Mod01%20-%20Matter%20Energy%20and%20DC.pd
f-extracted/Mod01%20-%20Matter%20Energy%20and%20DC.pdf.xhtml-pretty.html"
html = request.urlopen(url).read().decode('utf8')
response = request.urlopen(url)
raw = response.read().decode('utf8')
splits = BeautifulSoup(html, 'html.parser').find_all('p')
len(raw)
listy= []
with open("paragraphsplit.txt","wb") as outfile:
    for i in splits:
        outfile.write(bytes(i.text+'\n', 'UTF-8'))
        listy.append(i.text)


listycleaned = []
for i in range(len(listy)):                    #getting rid of unwanted shor
t paragraphs
    if len(listy[i]) > 140:
        listycleaned.append(listy[i])
```

```
print(listycleaned[90])

print(len(listycleaned))

for j in range(len([listycleaned])):

    i = 0

    for value in listycleaned:

        i += 1

        f = io.open("foldy/TestCorpusWriter/mod01_%s.txt" % i, 'w', encodin
g="utf-8")        ## change volume name ('mod01' becomes 'mod02'), chunk nam
e is 'mod01_01', 'mod01_02', etc.

        f.write(value)

        f.close()
```

LSA Model Builder with Performance Metrics

```
import sklearn.datasets as datasets             ##all imports used in the co
de

import gensim.utils as gensimUtils

import nltk

from gensim import corpora

from gensim import models

import pandas as pd

import scipy.spatial.distance as scipyDistance

from gensim.test.utils import common_texts, get_tmpfile

from gensim.models import Word2Vec

import gensim.parsing.preprocessing as preprocessing

from gensim.models import Phrases
```

```
import numpy as np
from nltk.corpus import stopwords
import re
from gensim.models.doc2vec import TaggedDocument
from gensim.models import Doc2Vec
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import accuracy_score


from nltk.tokenize import word_tokenize,sent_tokenize
from nltk.stem import SnowballStemmer
from gensim.models.doc2vec import LabeledSentence
from nltk.metrics import scores
from sklearn.metrics import confusion_matrix
from nltk.tokenize import word_tokenize,sent_tokenize
from sklearn.metrics import cohen_kappa_score
from easy_table import EasyTable
import sys


sys.argv = [""]
textCorpus = (datasets.load_files('NEETS-Electronics Corpus+Physics32k', sh
uffle=False)) #upload text corpus (46k total texts) best so far has been at
46k texts


ET=pd.read_csv('ETrespclean2.csv', encoding='latin1') #get data from studen
t responses set
df = pd.DataFrame(ET)    #set ET as dataframe
```

```
dfIdeal = pd.DataFrame({'Gsentences': ET.GA}) #define ideal answers for tok
enization (good answers/Gans).
dfIdeal['tokenized_sents'] = dfIdeal.apply(lambda row: nltk.word_tokenize(r
ow['Gsentences']), axis=1)
dfStudent = pd.DataFrame({'Ssentences': ET.SA}) #define student answers for
tokenization.
dfStudent['tokenized_sents'] = dfStudent.apply(lambda row: nltk.word_tokeni
ze(row['Ssentences']), axis=1)
Gans = dfIdeal['tokenized_sents'] ##renaming the 2 tokenized sent sets for
ease.
Sans = dfStudent['tokenized_sents']
GansDict = (dfIdeal['tokenized_sents'].to_dict)  #dictionary for tokenized
sents
SansDict = (dfStudent['tokenized_sents'].to_dict)


tokenizedSentences = [(gensimUtils.simple_preprocess(i, deacc=True, min_len
=2, max_len=18)) for i in textCorpus.data] #tokenize text corpus to build/t
rain models on
tokenizedGans = [(gensimUtils.simple_preprocess(i, deacc=True, min_len=1, m
ax_len=14)) for i in ET.GA] #tokenize Gans and Sans for use in w2v, w2vB, a
nd D2V models matching(LSA allows for unknown terms in tokenized strings. t
hese other models do not.)
tokenizedSans = [(gensimUtils.simple_preprocess(i, deacc=True, min_len=1, m
ax_len=14)) for i in ET.SA]
```

```
tokenizedGansclean = []   #this is done for Gans only to remove answer labe
ls in the text (Gans:). labels like E1:, E2:, E3: etc. are removed in prepr
ocesssing
for i in range(5166):
    hehe = []
    for haha in tokenizedGans[i]:
        if haha != 'gans':
            hehe.append(haha)
    tokenizedGansclean.append(hehe)

englishStop = set(stopwords.words("english"))

frequency = nltk.FreqDist(nltk.flatten(tokenizedSentences)) #frequency dist
ribution

processedCorpus = [[i for i in j if frequency[i] > 1 and i not in englishSt
op] for j in tokenizedSentences]

dictionary = corpora.Dictionary(processedCorpus) # building inverse documen
t frequency matrix to be used by the LSA model
termdocMatrix = [dictionary.doc2bow(i) for i in processedCorpus]
tfidf = models.TfidfModel(termdocMatrix)
tfidfMatrix = tfidf[termdocMatrix]
lsaPhys = models.LsiModel(tfidfMatrix, id2word=dictionary, num_topics=310)
#LSA physics model with 200 topics/dimensions
lsaSpacePhys = lsaPhys[tfidfMatrix]
```

```
lsaPhys.save("NEETS-ELECTRONICS+PHYSICS-LSAmodelv1.2-310")



length = len(ET.GA)
i = 0
LSAlistylist = []
while i < length:
##cosine similarity for ideal answer and student response LSA model
    try:
        GansBow = dictionary.doc2bow(tokenizedGansclean[i])
        SansBow = dictionary.doc2bow(tokenizedSans[i])
        GansVector = pd.DataFrame(lsaPhys[GansBow], columns=['dim','val'])
        SansVector = pd.DataFrame(lsaPhys[SansBow], columns=['dim','val'])
        matchscore = scipyDistance.cosine(SansVector['val'], GansVector['va
l'])
        j = (1 - matchscore)
        LSAlistylist.append(j)
        i += 1
    except (ValueError, ZeroDivisionError):
        b = 0
        LSAlistylist.append(b) # do nothing!
        i += 1
        if RuntimeWarning:                          # these bad boys can surviv
e for miles without water.
            pass
```

```
        for value in LSAlistylist:              # recodes string n/a into f
loat = 0 for missing values in LSA match scores
            if value == 'n/a':
                LSAistylist.append(value)
LSAnum = [i for i in LSAlistylist]      #update entire set of cosine simila
rity scores for LSA model as a new column in dataframe
LSAse = pd.Series(LSAnum)
df['LSAp'] = LSAse.values
```

LSA Model Loader

```
lsaPhys = models.LsiModel.load("NEETS-ELECTRONICS+PHYSICS-LSAmodelv1.2-500"
)



length = len(ET.GA)
i = 0
LSAlistylist = []
while i < length:
##cosine similarity for ideal answer and student response LSA model
    try:
        GansBow = dictionary.doc2bow(tokenizedGansclean[i])
        SansBow = dictionary.doc2bow(tokenizedSans[i])
        GansVector = pd.DataFrame(lsaPhys[GansBow], columns=['dim','val'])
        SansVector = pd.DataFrame(lsaPhys[SansBow], columns=['dim','val'])
```

```
        matchscore = scipyDistance.cosine(SansVector['val'], GansVector['va
l'])
        j = (1 - matchscore)
        LSAlistylist.append(j)
        i += 1
    except (ValueError, ZeroDivisionError):
        b = 0
        LSAlistylist.append(b) # do nothing!
        i += 1
        if RuntimeWarning:                          # these bad boys can surviv
e for miles without water.
            pass
        for value in LSAlistylist:                  # recodes string n/a into f
loat = 0 for missing values in LSA match scores
            if value == 'n/a':
                LSAistylist.append(value)
LSAnum = [i for i in LSAlistylist]      #update entire set of cosine simila
rity scores for LSA model as a new column in dataframe
LSAse = pd.Series(LSAnum)
df['LSAp'] = LSAse.values
```

Automated Model Simulation with Full Performance Metrics

```
listStringentJudge1 = []
listStringentJudge1P = []
listStringentJudge1R = []
```

```
ClistStringentJudge1 = []
ClistStringentJudge1P = []
ClistStringentJudge1R = []
i = 0
for value in range(1000):
    i += .001
    Jthresh = []                          ##the next 8 for loops are about co
ding match values for human judges and computer models(LSA, LSA/RegEx Combo
)
    for value in ET.J1:
        if value == Jthreshv:
            Jthresh.append(1)
        else:
            Jthresh.append(0)
    LSAthresh = []
    for value in df['LSAp']:
        if value >= i:
            LSAthresh.append(1)
        else:
            LSAthresh.append(0)
    RegExLSA = merge(RegExthresh, LSAthresh)           ##Very important. Us
ed for Judges either/or as well as LSA/RegEx combination thresholds.
    Combothresh = []
    for value in RegExLSA:
        if value[0] or value[1] == 1:
            Combothresh.append(1)
```

```
        else:
            Combothresh.append(0)



    LSAcm = confusion_matrix(Jthresh, LSAthresh)  #confusion matrix to plug
into precision and recall tool
    tp = LSAcm[1,1]
    tn = LSAcm[0,0]
    p = (LSAcm[1,0] + LSAcm[1,1])                    ##LSA vs Judge 1
    n = (LSAcm[0,0] + LSAcm[0,1])
    fp = LSAcm[1,0]
    fn = LSAcm[0,1]
# accuracy: (tp + tn) / (p + n)
    accuracy = (tp + tn) / (p + n)
# precision tp / (tp + fp)
    precision = tp / (tp + fp)
    # recall: tp / (tp + fn)
    recall = tp / (tp + fn)
# f1: 2 tp / (2 tp + fp + fn)
    f1 = 2*tp / (2*tp + fp + fn)
    listStringentJudge1.append((f1, i))
    listStringentJudge1P.append((precision, i))
    listStringentJudge1R.append((recall, i))


    Combocm = confusion_matrix(Jthresh, Combothresh)  #confusion matrix to
plug into precision and recall tool
```

```
    Ctp = Combocm[1,1]
    Ctn = Combocm[0,0]
    Cp = (Combocm[1,0] + Combocm[1,1])                    ##Combo vs Judge
1
    Cn = (Combocm[0,0] + Combocm[0,1])
    Cfp = Combocm[1,0]
    Cfn = Combocm[0,1]
# accuracy: (tp + tn) / (p + n)
    Caccuracy = (Ctp + Ctn) / (Cp + Cn)
# precision tp / (tp + fp)
    Cprecision = Ctp / (Ctp + Cfp)
    # recall: tp / (tp + fn)
    Crecall = Ctp / (Ctp + Cfn)
# f1: 2 tp / (2 tp + fp + fn)
    Cf1 = 2*Ctp / (2*Ctp + Cfp + Cfn)
    ClistStringentJudge1.append((Cf1, i))
    ClistStringentJudge1P.append((Cprecision, i))
    ClistStringentJudge1R.append((Crecall, i))


listLenientJudge2P = []
listLenientJudge2R = []
listLenientJudge2 = []
ClistLenientJudge2P = []
ClistLenientJudge2R = []
ClistLenientJudge2 = []
i = 0
```

```
for value in range(1000):
    i += .001
    J2threshL = []
    for value in ET.J2:
        if value >= J2threshLv:
            J2threshL.append(1)
        else:
            J2threshL.append(0)
    LSAthreshL = []
    for value in df['LSAp']:
        if value >= i:
            LSAthreshL.append(1)
        else:
            LSAthreshL.append(0)
    RegExLSAL = merge(RegExthreshL, LSAthreshL)           ##Very important.
Used for Judges either/or as well as LSA/RegEx combination thresholds.
    CombothreshL = []
    for value in RegExLSAL:
        if value[0] or value[1] == 1:
            CombothreshL.append(1)
        else:
            CombothreshL.append(0)



    LSAcmL2 = confusion_matrix(J2threshL, LSAthreshL)  #confusion matrix to
plug into precision and recall tool
```

```
    tpL2 = LSAcmL2[1,1]
    tnL2 = LSAcmL2[0,0]
    pL2 = (LSAcmL2[1,0] + LSAcmL2[1,1])
    nL2 = (LSAcmL2[0,0] + LSAcmL2[0,1])
    fpL2 = LSAcmL2[1,0]
    fnL2 = LSAcmL2[0,1]
# accuracy: (tp + tn) / (p + n)
    accuracyL2 = (tpL2 + tnL2) / (pL2 + nL2)
# precision tp / (tp + fp)
    precisionL2 = tpL2 / (tpL2 + fpL2)
    recallL2 = tpL2 / (tpL2 + fnL2)
# f1: 2 tp / (2 tp + fp + fn)
    f1L2 = 2*tpL2 / (2*tpL2 + fpL2 + fnL2)
    listLenientJudge2.append((f1L2, i))
    listLenientJudge2P.append((precisionL2, i))
    listLenientJudge2R.append((recallL2, i))

    CombocmL2 = confusion_matrix(J2threshL, CombothreshL)  #confusion matri
x to plug into precision and recall tool
    CtpL2 = CombocmL2[1,1]
    CtnL2 = CombocmL2[0,0]
    CpL2 = (CombocmL2[1,0] + CombocmL2[1,1])
    CnL2 = (CombocmL2[0,0] + CombocmL2[0,1])
    CfpL2 = CombocmL2[1,0]
    CfnL2 = CombocmL2[0,1]
# accuracy: (tp + tn) / (p + n)
```

```
    CaccuracyL2 = (CtpL2 + CtnL2) / (CpL2 + CnL2)
# precision tp / (tp + fp)
    CprecisionL2 = CtpL2 / (CtpL2 + CfpL2)
    CrecallL2 = CtpL2 / (CtpL2 + CfnL2)
# f1: 2 tp / (2 tp + fp + fn)
    Cf1L2 = 2*CtpL2 / (2*CtpL2 + CfpL2 + CfnL2)
    ClistLenientJudge2.append((Cf1L2, i))
    ClistLenientJudge2P.append((CprecisionL2, i))
    ClistLenientJudge2R.append((CrecallL2, i))


listLenientJudge1P = []
listLenientJudge1R = []
listLenientJudge1 = []
ClistLenientJudge1P = []
ClistLenientJudge1R = []
ClistLenientJudge1 = []
i = 0
for value in range(1000):
    i += .001
    JthreshL = []                          ##the next 8 for loops are about c
oding match values for human judges and computer models in lenient threshol
ds.
    for value in ET.J1:
        if value >= JthreshLv:
            JthreshL.append(1)
        else:
```

```
            JthreshL.append(0)
    LSAthreshL = []
    for value in df['LSAp']:
        if value >= i:
            LSAthreshL.append(1)
        else:
            LSAthreshL.append(0)
    RegExLSAL = merge(RegExthreshL, LSAthreshL)           ##Very important.
Used for Judges either/or as well as LSA/RegEx combination thresholds.
    CombothreshL = []
    for value in RegExLSAL:
        if value[0] or value[1] == 1:
            CombothreshL.append(1)
        else:
            CombothreshL.append(0)


    LSAcmL = confusion_matrix(JthreshL, LSAthreshL)  #confusion matrix to p
lug into precision and recall tool
    tpL = LSAcmL[1,1]
    tnL = LSAcmL[0,0]
    pL = (LSAcmL[1,0] + LSAcmL[1,1])
    nL = (LSAcmL[0,0] + LSAcmL[0,1])
    fpL = LSAcmL[1,0]
    fnL = LSAcmL[0,1]
# accuracy: (tp + tn) / (p + n)
    accuracyL = (tpL + tnL) / (pL + nL)
```

```
# precision tp / (tp + fp)
    precisionL = tpL / (tpL + fpL)
    recallL = tpL / (tpL + fnL)
# f1: 2 tp / (2 tp + fp + fn)
    f1L = 2*tpL / (2*tpL + fpL + fnL)
    listLenientJudge1.append((f1L, i))
    listLenientJudge1P.append((precisionL, i))
    listLenientJudge1R.append((recallL, i))
    CombocmL = confusion_matrix(JthreshL, CombothreshL)  #confusion matrix
to plug into precision and recall tool
    CtpL = CombocmL[1,1]
    CtnL = CombocmL[0,0]
    CpL = (CombocmL[1,0] + CombocmL[1,1])
    CnL = (CombocmL[0,0] + CombocmL[0,1])
    CfpL = CombocmL[1,0]
    CfnL = CombocmL[0,1]
# accuracy: (tp + tn) / (p + n)
    CaccuracyL = (CtpL + CtnL) / (CpL + CnL)
# precision tp / (tp + fp)
    CprecisionL = CtpL / (CtpL + CfpL)
    CrecallL = CtpL / (CtpL + CfnL)
# f1: 2 tp / (2 tp + fp + fn)
    Cf1L = 2*CtpL / (2*CtpL + CfpL + CfnL)
    ClistLenientJudge1.append((Cf1L, i))
    ClistLenientJudge1P.append((CprecisionL, i))
    ClistLenientJudge1R.append((CrecallL, i))
```

```
listIntermediateJudge2 = []
listIntermediateJudge2P = []
listIntermediateJudge2R = []
ClistIntermediateJudge2 = []
ClistIntermediateJudge2P = []
ClistIntermediateJudge2R = []
i = 0
for value in range(1000):
    i += .001
    J2threshI = []
    for value in ET.J2:
        if value >= J2threshIv:
            J2threshI.append(1)
        else:
            J2threshI.append(0)
    LSAthreshI = []
    for value in df['LSAp']:
        if value >= i:
            LSAthreshI.append(1)
        else:
            LSAthreshI.append(0)
    RegExLSAI = merge(RegExthreshI, LSAthreshI)           ##Very important.
Used for Judges either/or as well as LSA/RegEx combination thresholds.
    CombothreshI = []
```

```
    for value in RegExLSAI:
        if value[0] or value[1] == 1:
            CombothreshI.append(1)
        else:
            CombothreshI.append(0)

    LSAcmI2 = confusion_matrix(J2threshI, LSAthreshI)  #confusion matrix to
plug into precision and recall tool
    tpI2 = LSAcmI2[1,1]
    tnI2 = LSAcmI2[0,0]
    pI2 = (LSAcmI2[1,0] + LSAcmI2[1,1])
    nI2 = (LSAcmI2[0,0] + LSAcmI2[0,1])
    fpI2 = LSAcmI2[1,0]
    fnI2 = LSAcmI2[0,1]
# accuracy: (tp + tn) / (p + n)
    accuracyI2 = (tpI2 + tnI2) / (pI2 + nI2)
# precision tp / (tp + fp)
    precisionI2 = tpI2 / (tpI2 + fpI2)
# recall: tp / (tp + fn)
    recallI2 = tpI2 / (tpI2 + fnI2)
# f1: 2 tp / (2 tp + fp + fn)
    f1I2 = 2*tpI2 / (2*tpI2 + fpI2 + fnI2)
    listIntermediateJudge2.append((f1I2, i))
    listIntermediateJudge2P.append((precisionI2, i))
    listIntermediateJudge2R.append((recallI2, i))
```

```
    CombocmI2 = confusion_matrix(J2threshI, CombothreshI)  #confusion matri
x to plug into precision and recall tool
    CtpI2 = CombocmI2[1,1]
    CtnI2 = CombocmI2[0,0]
    CpI2 = (CombocmI2[1,0] + CombocmI2[1,1])
    CnI2 = (CombocmI2[0,0] + CombocmI2[0,1])
    CfpI2 = CombocmI2[1,0]
    CfnI2 = CombocmI2[0,1]
# accuracy: (tp + tn) / (p + n)
    CaccuracyI2 = (CtpI2 + CtnI2) / (CpI2 + CnI2)
# precision tp / (tp + fp)
    CprecisionI2 = CtpI2 / (CtpI2 + CfpI2)
    CrecallI2 = CtpI2 / (CtpI2 + CfnI2)
# f1: 2 tp / (2 tp + fp + fn)
    Cf1I2 = 2*CtpI2 / (2*CtpI2 + CfpI2 + CfnI2)
    ClistIntermediateJudge2.append((Cf1I2, i))
    ClistIntermediateJudge2P.append((CprecisionI2, i))
    ClistIntermediateJudge2R.append((CrecallI2, i))



listIntermediateJudge1 = []
listIntermediateJudge1P = []
listIntermediateJudge1R = []
ClistIntermediateJudge1 = []
ClistIntermediateJudge1P = []
ClistIntermediateJudge1R = []
```

```
i = 0
for value in range(1000):
    i += .001
    JthreshI = []                          ##the next 8 for loops are about c
oding match values for human judges and computer models in intermediate thr
esholds.
    for value in ET.J1:
        if value >= JthreshIv:
            JthreshI.append(1)
        else:
            JthreshI.append(0)
    LSAthreshI = []
    for value in df['LSAp']:
        if value >= i:
            LSAthreshI.append(1)
        else:
            LSAthreshI.append(0)
    RegExLSAI = merge(RegExthreshI, LSAthreshI)           ##Very important.
Used for Judges either/or as well as LSA/RegEx combination thresholds.
    CombothreshI = []
    for value in RegExLSAI:
        if value[0] or value[1] == 1:
            CombothreshI.append(1)
        else:
            CombothreshI.append(0)
```

```
    LSAcmI = confusion_matrix(JthreshI, LSAthreshI)  #confusion matrix to p
lug into precision and recall tool
    tpI = LSAcmI[1,1]
    tnI = LSAcmI[0,0]
    pI = (LSAcmI[1,0] + LSAcmI[1,1])
    nI = (LSAcmI[0,0] + LSAcmI[0,1])
    fpI = LSAcmI[1,0]
    fnI = LSAcmI[0,1]
# accuracy: (tp + tn) / (p + n)
    accuracyI = (tpI + tnI) / (pI + nI)
# precision tp / (tp + fp)
    precisionI = tpI / (tpI + fpI)
# recall: tp / (tp + fn)
    recallI = tpI / (tpI + fnI)
    # f1: 2 tp / (2 tp + fp + fn)
    f1I = 2*tpI / (2*tpI + fpI + fnI)
    listIntermediateJudge1.append((f1I, i))
    listIntermediateJudge1R.append((recallI, i))
    listIntermediateJudge1P.append((precisionI, i))


    CombocmI = confusion_matrix(JthreshI, CombothreshI)  #confusion matrix
to plug into precision and recall tool
    CtpI = CombocmI[1,1]
    CtnI = CombocmI[0,0]
    CpI = (CombocmI[1,0] + CombocmI[1,1])
    CnI = (CombocmI[0,0] + CombocmI[0,1])
```

```
    CfpI = CombocmI[1,0]
    CfnI = CombocmI[0,1]
# accuracy: (tp + tn) / (p + n)
    CaccuracyI = (CtpI + CtnI) / (CpI + CnI)
# precision tp / (tp + fp)
    CprecisionI = CtpI / (CtpI + CfpI)
    CrecallI = CtpI / (CtpI + CfnI)
# f1: 2 tp / (2 tp + fp + fn)
    Cf1I = 2*CtpI / (2*CtpI + CfpI + CfnI)
    ClistIntermediateJudge1.append((Cf1I, i))
    ClistIntermediateJudge1P.append((CprecisionI, i))
    ClistIntermediateJudge1R.append((CrecallI, i))



listStringentJudge2R = []
listStringentJudge2P = []
listStringentJudge2 = []
ClistStringentJudge2R = []
ClistStringentJudge2P = []
ClistStringentJudge2 = []
i = 0
for value in range(1000):
    i += .001
    J2thresh = []
    for value in ET.J2:
        if value == J2threshv:
```

```
            J2thresh.append(1)
        else:
            J2thresh.append(0)
    LSAthresh = []
    for value in df['LSAp']:
        if value >= i:
            LSAthresh.append(1)
        else:
            LSAthresh.append(0)
    RegExLSA = merge(RegExthresh, LSAthresh)           ##Very important. Us
ed for Judges either/or as well as LSA/RegEx combination thresholds.
    Combothresh = []
    for value in RegExLSA:
        if value[0] or value[1] == 1:
            Combothresh.append(1)
        else:
            Combothresh.append(0)


    LSAcm2 = confusion_matrix(J2thresh, LSAthresh)  #confusion matrix to pl
ug into precision and recall tool
    tp2 = LSAcm2[1,1]
    tn2 = LSAcm2[0,0]
    p2 = (LSAcm2[1,0] + LSAcm2[1,1])                    ##LSA vs Judge 2
    n2 = (LSAcm2[0,0] + LSAcm2[0,1])
    fp2 = LSAcm2[1,0]
    fn2 = LSAcm2[0,1]
```

```
# accuracy: (tp + tn) / (p + n)
    accuracy2 = (tp2 + tn2) / (p2 + n2)
# precision tp / (tp + fp)
    precision2 = tp2 / (tp2 + fp2)
# recall: tp / (tp + fn)
    recall2 = tp2 / (tp2 + fn2)
# f1: 2 tp / (2 tp + fp + fn)
    f12 = 2*tp2 / (2*tp2 + fp2 + fn2)
    listStringentJudge2.append((f12, i))
    listStringentJudge2R.append((recall2, i))
    listStringentJudge2P.append((precision2, i))


    Combocm2 = confusion_matrix(J2thresh, Combothresh)  #confusion matrix t
o plug into precision and recall tool
    Ctp2 = Combocm2[1,1]
    Ctn2 = Combocm2[0,0]
    Cp2 = (Combocm2[1,0] + Combocm2[1,1])                    ##Combo vs Jud
ge 1
    Cn2 = (Combocm2[0,0] + Combocm2[0,1])
    Cfp2 = Combocm2[1,0]
    Cfn2 = Combocm2[0,1]
# accuracy: (tp + tn) / (p + n)
    Caccuracy2 = (Ctp2 + Ctn2) / (Cp2 + Cn2)
# precision tp / (tp + fp)
    Cprecision2 = Ctp2 / (Ctp2 + Cfp2)
    # recall: tp / (tp + fn)
```

```
    Crecall2 = Ctp2 / (Ctp2 + Cfn2)
# f1: 2 tp / (2 tp + fp + fn)
    Cf12 = 2*Ctp2 / (2*Ctp2 + Cfp2 + Cfn2)
    ClistStringentJudge2.append((Cf12, i))
    ClistStringentJudge2P.append((Cprecision2, i))
    ClistStringentJudge2R.append((Crecall2, i))
```