Confusion Detection on Annotator Affect

by
Tongyu Zhou

Professor Iris Howley, Advisor

A thesis submitted in partial fulfillment
of the requirements for the
Degree of Bachelor of Arts with Honors
in Computer Science

Williams College
Williamstown, Massachusetts

May 19, 2020

Contents

List of Figures

List of Tables

Abstract

Reading is an irreplaceable part of our everyday lives. We read to relax, to satiate our imaginations, and to enrich our understanding of the world around us. In the educational setting, we especially read to gain new knowledge. This information acquisition process does not always proceed smoothly however. When encountering new material, we may be hindered by gaps in understanding and consequently fall into a state of *confusion*. In order to continue learning effectively, it is thus critical for readers to be able to recognize this confusion and resolve it in a timely manner.

However, confusion is often difficult to recognize and qualify, especially for readers themselves who may fall into boredom and passivity instead of confronting their confusion. Such prolonged stays in this state of apathy may lead to reader frustration and burnout, eventually contributing to a final disengagement from the reading material. To avoid this result, there thus arises a need for automatic confusion detection.

In this thesis, I built a confusion detection application to help students recognize confusion during the reading process in real time. The application achieved this by collecting and analyzing reading annotations, critical derivations of the students' thought processes that are reflective of their interpretations and understandings of the text. For this application, I trained a support vector machine (SVM) classifier based on language and discourse features on data from Lacuna Stories, a social annotations platform, that predicted the confusion levels of each annotation on a 1 to 5 Likert scale. The reader was then able to review these classifications and obtain a better grasp of their understanding of different parts of the reading.

I then conducted a pilot user study using the application to identify certain trends in how the accuracy of confusion prediction influenced a reader's actual learning, reported perceived learning, and reported perception of self-predicted and machine-predicted confusion. When assuming the user's self-predicted confusion levels to be correct (perceived accuracy), the overall performance of the trained classifier was not better than random. However, after re-coding the annotations for confusion using the same standards as those on the Lacuna training set (actual accuracy), I achieved much higher accuracies for the SVM model, suggesting that future iterations need to unify user definitions of confusion. This study also suggested that "more accurate" predictions for confusion in terms of the user improved actual learning, did not affect reported perceived learning, and widened the gap between reported perceived self-predicted confusion and reported perceived machine-predicted confusion.

Acknowledgments

I am incredibly grateful to my parents for their endless love and support for everything I have done. Thanks to Hyeongjin for constantly encouraging me and helping me proofread my prose. Thanks to Markus for helping me debug and test my prototype. Thank you to all my wonderful friends and to everyone who volunteered for my user study—this work would not be here without you all.

Thanks to Dan Barowy, the second reader of this thesis, whose comments and feedback were invaluable during this entire process. Finally, I would like to express my deepest gratitude and appreciation for my advisor, Iris Howley. From our first summer working together at Williams, she has inspired a love for research in me that directs everything I do now and will do in the future. Her mentorship has not only significantly impacted this thesis but also changed the manner in which I approach computer science.

Chapter 1

Introduction

1.1 The Problem

Learning is invariably tied to the reading process. The consumption and production of reading material such as books, magazines, journal articles, and websites are seamlessly incorporated into all fields of study at all education levels. By internalizing the information from these texts, students can supplement holes in their understanding and gain new perspectives to preexisting knowledge.

The extent of this internalization, however, is hard to both qualify and quantify. Reading is at its core a solitary activity: despite the plethora of literature courses, discussion groups, and book clubs, group reading is rarely employed as the sole means of digesting text. There is usually no instructor or peer hovering over the student posing check-in questions to assess comprehension. Under these circumstances, the student can only rely on themselves to gauge how well they really understand a piece of text. That is, they must actively monitor their comprehension to identify parts of the reading that they do understand, partially understand, and do not understand. They must also be able to recognize whether their understanding is sufficient enough to apply the knowledge acquired from reading. Otherwise, prolonged frustration due to inabilities to pinpoint the lack of understanding may lead to boredom and eventually learner dropout.

Unfortunately, people are often bad at self-diagnosis [6]. Specifically, students often find it difficult to qualify the progress of their learning processes and their own lack of understanding because they tend to focus on overarching main ideas instead of details in the reading. Especially when a student is moving particularly quickly through a piece of material, they often do not pause to contemplate whether they fully absorbed the necessary information or recognize when they are in a state of confusion. An alternative approach independent of student whim to measure how well a learner understands the reading material thus involves a more systemic method that evaluates their reading annotations.

An annotation, formally defined as an additional note of explanation or comment anchored to a selected piece of text, can reveal critical cognitive processes and areas of potential confusion unbeknownst to the creator of the annotation themselves. While there are current models that can identify the presence of confusion, however, these systems are "incomplete" in the sense that they

are often post-hoc and do not help students recognize the subtleties in their degree of confusion in real time. These systems also fail to consider the direct role the accuracy of confusion prediction plays in student affect as well as how this affect in turn contributes to intention-to-use and the ultimate acceptance of the system.

In other words, the clearest indication of whether students are learning from reading material is their degree of confusion while they are reading. A confusion detection system that makes predictions during the reading process thus becomes necessary.

1.2 Goals

In this thesis, I intend to create a confusion detection application that helps students recognize their confusion as they are actively going through reading material in real time. To create effective confusion classification models, I will make use of feature engineering that specifically targets confusion arising from the reading and reinterpretation of text. I will then run a pilot user study to assess the newly built system and the effects of the accuracy of the machine learning model on student learning and perception.

Chapter 2

Background & Prior Work

A common approach to measuring student comprehension and learning of reading material is to record their cognitive engagement. However, this metric is often inadequate as it fails to capture the exact difficulties experienced by the reader. Alternatively, learning scientists have eventually shifted to directly measure confusion using behavioral, physiological, and language-based methods to gauge the progress of learning. These algorithm-based methods have seen mixed responses, however, with more positive ones associated with more efficient and accurate methods. The effects of confusion detection on learner affect thus directly influences the intention to use and eventual adoption of the system.

2.1 Limitations of Cognitive Engagement

Cognitive engagement (CE) is a term that describes how deeply a student is thinking about course material when reading through a piece of text [41] and is often used when describing the benefits of active learning over passive learning [13]. Specifically, Chi and Wiley [13] hypothesized and proved through a series of laboratory and classroom studies that student learning increased with greater engagement levels. This concept of CE can thus be very useful both in traditional classroom settings and online learning platforms to inform the instructor of the overall effectiveness of their courses. Examining how CE varies throughout different learning material can provide invaluable insight into how supplementary course modules that further reinforce learning should be structured for the future.

Table 2.1: Extended Cognitive Engagement Taxonomy

Label	Name	Description	Example
O	Off-topic	Do not refer to the reading directly	"The picture in page 6 makes me think of a motorboat and its wake."
A2	Active general	Indicates attention to material without referring to content	"The videos are sometimes more useful than the reading because they have more examples."
A1	Active targeted	Indicates engagement by paraphrasing without insight	"What is an isoelectric point of an amino acid?"
C2	Constructive extending	Introduce a statement or question that includes a new idea or refers to external sources, but does not reason about them	"Even though these three parts of protein folding are classified separately, it does not necessarily mean that they occur independently."
C1	Constructive reasoning	Display reasoning by explaining a phenomenon, stating a cause and effect, or drawing a conclusion	"The amino acid sequence, peptide bonds and length of the protein are derived from the primary structure, which is maintained in denaturation."
I	Interactive	Displays C1 and responds to a previous annotation	"Yes, I agree with you, if the interactions between the sheets change, it may cause the protein's overall structure to change thus the function."

Table 2.1 displays a taxonomy for different levels of CE aligning to different activities. These categories were originally created by Wang et al. [38] and expanded upon with descriptions and examples provided by Yogev [41]. Yogev's study, which examined a corpus of in-place comments drawn from MIT's annotation platform, Nota Bene [4], revealed the direct associations between CE and learning through reading materials. Annotations labeled as O, A2, and A1 that are correlated with low CE mostly comprised of questions, indicating that students did not attain firm grasps of the material. Conversely, students who created annotations labeled as C2, C1, and I, categories that are correlated with higher CE, tended to perform better on exams. Kendall rank correlation scores between CE and total course grade further affirmed that higher CE is associated with higher success in the course. Identifying the corresponding CE level for a learning task can thus reveal how deeply a student is thinking about the knowledge being taught and consequent ability to excel in the course.

What Yogev's study failed to capture, however, was a systemic means to identify the lack of understanding. In an interview with the study participants in a Biology course, there was consensus that CE is not a good indicator for difficulty or confusion: the topic students found most difficult is the subject of integrating pH and pKa, but comments within this region exhibited both high and

low CE. In other words, although CE can effectively convey whether a student is trying to learn a particular topic, it is inadequate to identify the obstacles and gaps in understanding they may encounter along the way. This limitation introduces possibilities for confounding variables in the assessment of course success solely based on the CE model as confusion may play critical roles in student attenuation.

2.2 The Confusion Paradigm

Confusion is defined as a cognitive-affect state where students are confronted with an anomaly, contradiction, or an impasse, and are uncertain about how to proceed [40]. This state is often accompanied by a shift in emotions during the learning task, which D'Mello and Graesser [16] identified as the *model of affect dynamics*:

$$Engagement \leftrightarrow Confusion \leftrightarrow Frustration \leftrightarrow Boredom$$

In this model, the student can shift from each state to another as directed by the arrows. If they face contradictions, incongruities, or other obstacles while in the engagement state, they will enter a state of confusion. A prompt response to confusion via teacher feedback or self-assessment allows students to gain a deeper understanding of complex topics, driving them back to engagement, while the lack thereof creates frustration and eventually boredom. As confusion persists, the student becomes more vulnerable to dropout.

Graesser [19] further formalized this relationship between engagement and confusion by framing it as a balance between equilibrium and disequilibrium. When faced with confusion, students usually respond in one of the following two ways.

1. Students view confusion as a challenge and are willing to expend cognitive efforts through reflection, problem-solving, and other active processes to regain understanding and return to *cognitive equilibrium*. This type of student is best left to their own devices.

2. Students are discouraged as they recognize they are not good at the subject matter, do not wish to receive negative feedback, and become stuck in *cognitive disequilibrium*. This type of student needs encouragement and hints from external sources.

Determining the category which students fall into is often a difficult task that varies with their background knowledge and interest in the subject. D'Mello and Graesser continued to theorize that all students fall into a *zone of optimal confusion*, as illustrated in Figure 2.1. This new model is influenced by Vygotsky's *zone of proximal development* [37] which describes the distance between what a learner can do without assistance and what they can do with the support of someone with area expertise. The zone of optimal confusion adopts a similar spectrum-like model that describes the distance between a learner with constructive confusion and one with adverse confusion. The exact "optimal" confusion is dependent on individual variability.

The lower bound of this zone T_a is associated with a minimum level of constructive confusion, which learners can easily escape from and return to engagement. The upper bound of this zone T_b is

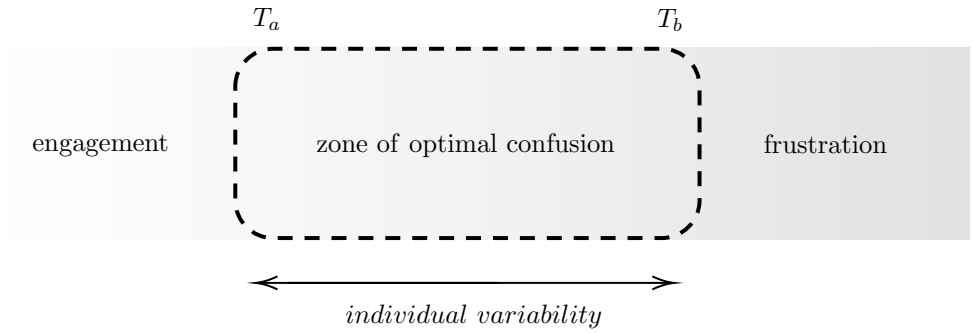

Figure 2.1: Zone of Optimal Confusion

associated with a maximum level of adverse confusion, which without the proper assistance, learners may quickly bypass and enter into the realm of frustration.

To effectively respond and help learners whose confusion is closer to T_b than T_a, a reliable methodology that identifies the presence of confusion thus becomes necessary.

2.3 Detection of Confusion

Prior work to identify confusion involved defining it according to the individual who experienced it firsthand and is achieved by directly asking learners to report their perceived levels of confusion encountered during learning tasks. However, even these self-reports differ in the scales of the measurements. While some studies used binary encoding (*0=not confused, 1=confused*), others elected for finer options like the Likert-scale [16]. To assist with the self-reporting process, other researchers have also considered the option of letting participants continuously voice their feelings in an *emote-aloud* protocol [3] in attempts to capture confusion.

Despite its simplicity and ease of use, self-reporting is often impractical as it lacks sensitivity. That is, due to social and cognitive tendencies that are biased against the lack of understanding, individuals are often unwilling to honestly report their confusion. The self-awareness of confusion also requires some emotional intelligence [3] and openness to vulnerability. Combined with the particular multifaceted nature of emotions, confusion becomes even harder for the participants to consciously recognize. Due to these various factors, many studies have tried to tackle the problem of accurate confusion detection. This section outlines a few of their approaches.

2.3.1 Behavioral & Physiological Sensing

The earliest work on confusion detection dealt with the interpretation of facial expressions, body language, and physiological responses.

Using facial expressions is an easy way to identify certain emotions in others. We smile when we are happy, frown when we are sad, and furrow our eyebrows whenever we experience frustration. In a classroom setting, teachers are able to recognize confusion by reading the expressions on their students' faces. By using systems that can video capture facial expressions, we can thus

better understand instances of confusion when a student is learning about a particular task without being overly intrusive. Some of these systems have already been developed, including the Computer Expression Recognition Toolbox (CERT) [27], a frame-by-frame facial expression tracker based on learner eyebrows, eyelids, mouth cues. Other techniques that can also be used to identify facial expressions include facial electromyography, which involves measuring the electrical activity of contracting muscles by placing electrodes on the surfaces of the skin. Although these technologies can accurately capture facial expressions, the only caveat is that internal confusion may not be always conveyed through these expressions.

Observing the body and conversational cues of a learner may also provide some indication of their affective state. According to Calvo and D'Mello [10], identifying emotions using body cues can actually be more accurate since people are less biased by "social editing," adjusting the self to accommodate social norms, in comparison to using facial expressions. This approach has been commonly used in studies that use trained judges. The judge is able to identify instances of confusion by taking note of behaviors such as head-scratching, changes of upper body and head position, and non-linguistic vocal expressions. One issue with this method, however, is that it does require real-time trained observers and is less extensible to digital learning environments.

Another alternative approach to confusion identification is by considering its effects on peripheral physiology. Heart rate and heart rate variability (HRV) are both potential physiological indicators of changes in emotional arousal. As they are the results of mostly unconscious and uncontrolled reactions of the automatic nervous system, Bradley and Lang [9] suggested that they can be considered objective ways of measuring confusion. However, the accuracy and sensitivity of this physiological measurement is likely to be lowered by the difficulty in collecting such data from someone working on a keyboard while interacting with a computer, since it adds additional noise to the data, and thus may not be entirely reliable.

Unfortunately, all these methods are only applicable in cases where such options are available and are extremely hard to scale past one-on-one learning scenarios. Recent approaches have instead leveraged machine learning in attempts to accurately and efficiently qualify confusion on a larger scale.

2.3.2 Language and Discourse Based Classification

The most recent work has begun to frame confusion detection as a classical classification problem. Such models used statistical procedures to map a set of input features, derived from text that includes annotations, to a set of output categories, the prediction for the presence of confusion within that text. This approach solves the scaling problem since once a sufficient classifier is constructed, this model can then be used to predict on all future annotations.

Table 2.2 displays a summary of prior attempts to create such an ideal set of features that can accurately model confusion using purely language and discourse. Here, note that while these studies did consider other community and activity related metrics such as the number of upvotes, the number of views, and clickstream data in a massive open online course (MOOC) setting, these features were omitted in the interest of prioritizing individual effect over community influence on

Table 2.2: Summary of Feature Space Sets for Confusion Classification

Feature Category	Yang *et al.* [40]	Zeng *et al.* [42]	Atapattu *et. al* [5]
(1) N-gram	✓	✓	✓
(2) Question	✓	✓	✓
(3) Language	✓		✓
(4) Content		✓	✓

confusion.

Each feature category includes the following specific features.

1. ***N*-gram**: In computational linguistics, an n-gram refers a sequence of n words from a given corpus. For example, the *uni*gram refers to collections of singular words, otherwise known as the *bag-of-words* model. Visually, this looks like

```
"I am who I am and I have the need to be."
bow = {"I":3,"am":2,"who":1,"and":1,"have":1,
       "the":1,"need":1,"to":1,"be":1};
    = [3,2,1,1,1,1,1,1,1];
```

   Note that the vector-space embedding depicted in the last line above assumes a shared meaning for each vector dimension. This n-gram model can then be used to identify whether certain domain-specific content words, phrases, or sentences, depending on the value of n, are present in the text. If a word is frequently associated with confused annotations, then it is more likely for a new annotation that contains this word to also indicate confusion.

2. **Question**: Frequently, confusion is conveyed through questions. The frequency of question marks, modal verbs, and sentences that began with a confusion related fixed expression can all be used as markers to identify the presence of such questions.

3. **Linguistic features**: Studying the proliferation of particular linguistic features can allude to the presence of a confused annotation. These features include the following.

   - Pronoun distribution: Atapattu et. al [5] specifically identified the dominance of the first person singular and the third person plural in annotations flagged for confusion.
   - Negative sentiment: texts expressing frustration, anger, or sadness are correlated with adverse confusion (T_b).
   - Type-Token Ratio (TTR): Atapattu et. al also drew particular attention to the TTR, a measure of complexity that conveys the lexical richness of a particular piece of text. It can be computed as

$$TTR = \frac{\text{number of types}}{\text{number of tokens}} * 100 \tag{2.1}$$

Using MANOVA analysis, they showed that TTR is the highest predictive feature (F=1129.50, $p < 0.001$, $\eta^2 = 0.204$) for identifying confusion over all other linguistic features. Particularly, confused learners are more likely to write posts that are diverse in vocabulary.

4. **Content**: The content of the annotation can also reveal information about the presence of confusion. Content can be examined from several perspectives.

   - Automated readability index (ARI): the readability index qualifies how understandable a piece of text is. High ARI, denoting texts that are harder to read is normally associated with a higher probability of confusion.
   - Post length: the length of the text by word count. Longer post lengths appear to be associated with higher chances of confusion.

Table 2.3: Coh-Metrix Measures

Referential cohesion	Lexical diversity	Text easability
noun overlap, argument overlap, stem overlap, content word overlap	type-token ratio, Measure of Textual Lexical Diversity, *vocd*	narrativity, syntactic simplicity, word concreteness, referential cohesion, deep cohesion, verb cohesion, connectivity, temporality

While simple features such as n-gram or length are easy to pull from a piece of text, others such as readability or sentiment require additional tools to extract. The *Coh-Metrix*[29], a text analysis tool that measures cohesion and the interaction between linguistic representations and knowledge representations of the text, provides such a means to explore texts at multiple discourse levels. Table 2.3 displays the measures that are used in the Coh-Metrix to assess selected theoretical constructs. Alternatively, the *Linguistic Inquiry and Word Count* [31], a text analysis program that studies the emotional, cognitive, and structural components present in an individual's writing, is also a good candidate for feature extraction.

Different subsets of these features have already seen usage in various studies [40, 42, 5] to effectively classify confusion. However, these studies largely focused on the *presence* of confusion rather than the *degree* of confusion and were all conducted post-hoc. Greater exploration into real-time degree-wise confusion detection can thus reveal more nuance into the ways confusion influences learning.

2.4 Learner Affect

Although incorporation of real-time confusion detection into learning systems will theoretically aid the learner by informing them of their degree of confusion, such machine predictions may not be fully accepted by the learner and are heavily dependent on the learner's own perceptions and consequent trust in the classifier. This perception and trust of algorithmic decisions are also invariably tied to

the nature of task. For example, an online experiment involving managerial decisions conducted by Lee [26] revealed that in the case of mechanical tasks, human and machine decisions were perceived equally, although human success was attributed to the individual's authority while machine success was attributed to its perceived efficiency. On the other hand, with human tasks, machine decisions were perceived as less fair and evoked more negative responses.

While it is still debatable whether confusion detection is a mechanical or human task, its positive reception and eventual adoption appears inevitably reliant upon the affect and perceived accuracy it relays to the user.

2.4.1 Technology Acceptance Model

The technology acceptance model (TAM), an information systems theory that explains and predicts user acceptance of information technology, provides a method to effectively measure the effects of confusion detection model accuracy on learner affect. It associates a user's intent to use a system with *perceived usefulness*, defined as the extent to which usage of the system will improve user performance, and *perceived ease of use*, defined as the extent to which users can navigate the system with minimal effort. Empirical studies of TAM demonstrated that perceived usefulness had consistently the most leverage on a user's intended usage; by improving the *perception* of a system's usefulness (not necessarily it's *actual* usefulness), users are more likely to adopt the new system.

Venkatesh and Davis [36] created such an extension to TAM that incorporated both social influence and cognitive instrumental processes as major determinants of TAM's perceived usefulness. This new model, referred to as TAM2, is depicted in Figure 2.2.

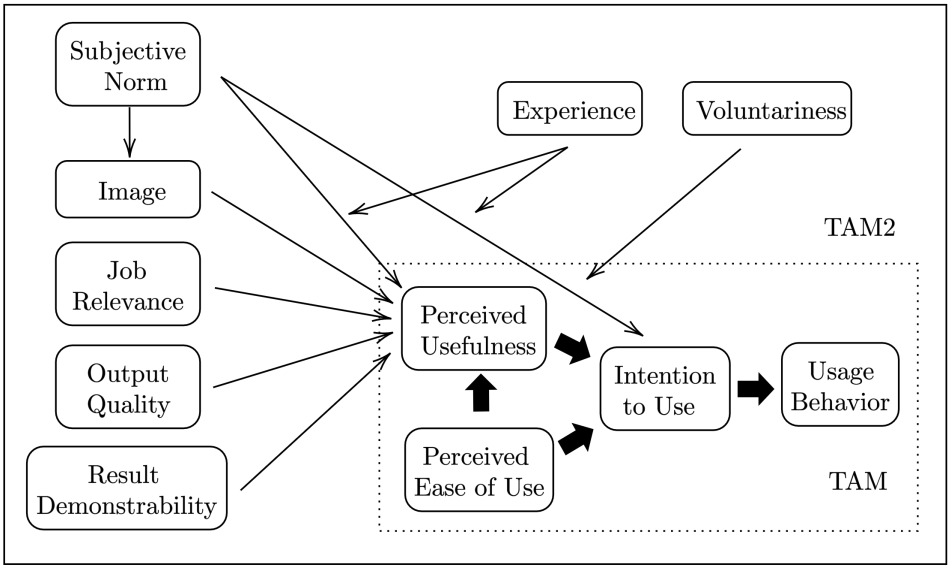

Figure 2.2: Technology Acceptance Model 2

Venkatesh and Davis's new model suggests that the constructs especially relevant to the accep-

tance of the confusion prediction system are job relevance, defined as the user's perception about the applicability of the system, output quality, defined as the ability of the system to perform the designated tasks, and result demonstrability, defined as the tangibility of results using the new system. In the case of the confusion prediction system, job relevance is user perception about whether confusion prediction is useful, output quality is the accuracy of the predictions, and result demonstrability is how much better the student can learn after using the system. By quantifying all three of these features through future user studies, I can verify whether students will actually use the confusion detection application.

Chapter 3

Machine Learning Frameworks

Since the inception of digital documents and large databases in the past few decades, automatic text classification has become an invaluable means to separate and qualify text. Until the late 1980s, the most popular approach to text classification was knowledge engineering, a method that involves emulating the judgment and expertise of a human expert in a particular field. However, favor has quickly shifted to the novel machine learning (ML) approaches by the early 1990s [33] as creating such expert systems was time consuming and expensive. This new framework involves a general inductive process that builds a classifier from a set of pre-classified documents that discriminates according to some parameter of interest, and is generally able to achieve an accuracy comparable to that of domain experts without their personal intervention.

The text classification process can be broken down into the following steps [21]:

1. **Read document**: The document is first read from the database.

2. **Tokenize text**: The document is broken down into minimally meaningful short blocks known as typographic tokens.

3. **Stemming/Lemmatization**: Stemming is the process through which each word is reduced to its word stem through the direct removal of derivational affixes (prefixes and suffixes). Lemmatization works similarly in the sense that it also reduces each word, but it additionally takes into account the morphological analysis of the word. This renders lemmatization more powerful despite being more computationally expensive in comparison to stemming. Consider the words *studies* and *studying*. Stemming will produce *studi* and *study* while lemmatization will produce *study* and *study*.

4. **Delete stopwords**: Stopwords (for example, "a," "an," "the") are commonly used words that are often filtered out to attribute more importance to the keywords of the text.

5. **Vector representation of text**: Since machines are often better at understanding numbers than text, the document is converted into a series of numerical representations. These vectors can then be combined to form their own vector space where the rules of that vector space

will apply. One possible such representation is the one-hot representation, a process in which the document is entirely converted into a binary vector of length $|V|$ where V represents the vocabulary (unique words) in that document [21]. Each value in the vector is assigned a 1 if the document contains that particular vocabulary word and a 0 if it does not. The document is thus positioned in $R^{|V|}$ space.

6. **Feature extraction and selection**: Because raw data is often dense and redundant, feature extraction is often employed to reduce the dimensionality of such data to a derived set of values to facilitate learning. If the initial set of features is still too large to be processed, it can be further reduced through feature selection. A more comprehensive list of features commonly used in text classification models can be found in Section 2.3.2.

7. **Learning algorithm**: This last step is where the ML approach comes in. More formally, it can be defined as follows: if there exists a collection of documents $D = \{d_1, ..., d_n\}$ and a set of categories $C = \{c_1, ..., c_m\}$, then classification is the assignment $(d_i, c_j) \rightarrow \{T, F\}$ for each pair, where a value of true indicates that document d_i belongs to category c_j and a value of false indicates that it does not. If the "true" assignment of these documents is denoted as some target function Φ, then the particular assignment produced by the classifier is the approximation $\hat{\Phi}$ and is governed by the learning algorithm.

Once the classifier is created, its effectiveness should be assessed. This is achieved by initially splitting the data into a training, validation, and testing set. The classifier is then constructed using documents from the former two sets, then tested on the latter set. That is, the goodness of that particular classifier can then be obtained by observing how often $\Phi(d_i, c_j) = \hat{\Phi}(d_i, c_j)$ in the testing set. By comparing this goodness across the classifiers of different learning algorithms, one can then identify the best algorithm for a particular set of data. Metrics for goodness can be found in Section 6.1.1.

The following sections will cover the ML algorithms used for text classification in this thesis.

3.1 Naive Bayes

The Naive Bayes classifier adopts a probabilistic approach and aims to find that, given a vector representation of a document d_i, the probability that it belongs to category c_j, or $P(c_j|d_i)$ [33]. Since this conditional is not easy to directly compute, Bayes' theorem can be applied here to obtain

$$P(c_j|d_i) = \frac{P(c_j)P(d_i|c_j)}{P(d_i)}. \tag{3.1}$$

In (3.1), $P(c_j)$ refers to the probability that a randomly selected document belongs to category c_j and can be easily computed by counting all the documents in the collection that belong to c_j. Similarly, $P(d_i)$ refers to the probability that a randomly selected document is represented by vector d_i and can be computed by counting all the documents that can be represented with d_i. Computing $P(d_i|c_j)$ requires the additional independence assumption that, given the vector $d_i = \{x_{1i}, ...x_{|d_i|i}\}$,

any two entries x_{ai} and x_{bi} are independent when viewed as random variables. Thus, the joint probability formula can be applied to get

$$P(d_i|c_j) = \prod_{k=1}^{|d_i|} P(x_{ki}|c_j). \tag{3.2}$$

However, it is important to note here that this independence assumption is *not* always satisfied in practice, hence why this classifier is deemed the "naive" approach. As a result of this assumption, even if words are dependent in the text, they contribute weight individually, resulting in the magnitudes of weights in certain categories with strong word dependencies to be larger than those with weaker word dependencies.

Another problem with the Naive Bayes classifier lies in the properties of the training set. If one category has significantly more training samples than the other, Naive Bayes will then choose poor weight assignments for the decision boundaries due to a skewed data bias effect that reduces weights for categories with less training samples.

However, the classifier still retains some popularity as it is both fast and easy to implement. Some of the faults discussed above can also be mitigated through simple corrections to the algorithm. Rennie et al. [32] attempts to solve the issue with the independence assumption by normalizing classification weights to keep categories with more dependencies from dominating. They also attempt to handle the imbalance of training samples per category by introducing a "complement class" where instead of estimating weights for a category c_i, weights are estimated for data belonging to all categories except for c_i. Altogether, these corrections resulted in a fast yet still accurate algorithm that is competitive with other state-of-the-art strategies.

3.2 Decision Trees

While probabilistic models may be easier to implement, they are often not easily interpretable by humans due to their numeric nature. An alternative approach is the decision tree classifier, a hierarchical tree model where internal nodes consist of decisions and terminal leaf nodes consist of categories. This classifier classifies a document d_i by starting at the root and testing each decision as determined by the internal nodes until a leaf is reached; the category at that leaf is then assigned to d_i. Figure 3.1 depicts an example of a decision tree.

One possible construction of a decision tree involves a recursive divide-and-conquer process [33] as follows. Within the training set, check if all documents belong to the same category. If not, select some term t_k that can partition the documents into separate classes where documents inside each class share the same value for t_k. Each class is then placed in their own subtree, in which the process outlined earlier is repeated until each leaf of the tree contains documents belonging only to one category. The leaf is then labelled with that category.

The crux of this algorithm lies in the determination of the term t_k that splits the classes. The ideal term would split the data such that each child node mostly contains documents belonging to a singular category. This choice is made and optimized according to an *entropy criterion* or

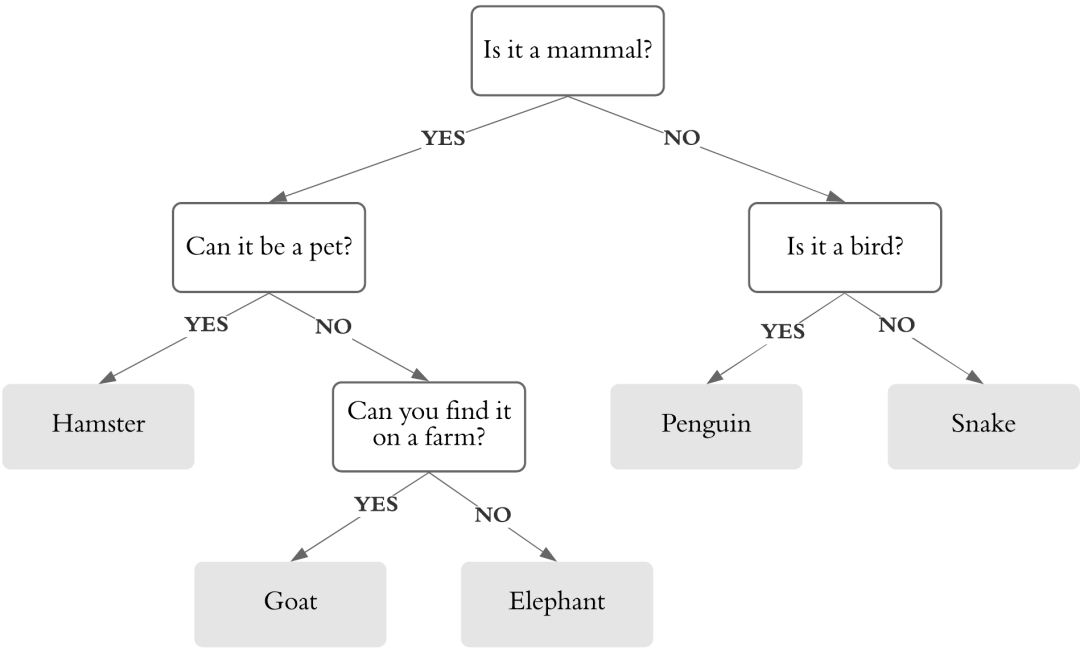

Figure 3.1: Decision Tree Example

information gain, metrics that provide insight on the quality of a split.

In the context of decision trees, entropy can be interpreted as "orderliness," or the amount of variance in the data set. Consider an example where a subset of data, S_i, is split into two classes, where p_1 is the proportion of documents in the first class and $p_2 = 1 - p_1$ is the proportion of documents in the second class. Entropy of the subset is thus defined as

$$E(S_i) = -p_1 \log_2(p_1) - p_2 \log_2(p_2). \tag{3.3}$$

The relationship between entropy and different values of p_1 can be seen in Figure 3.2. Note that when the subset is equally distributed across the two classes and $p_1 = p_2$, entropy is the highest at 1. Conversely, when all documents in the subset belong to the same class, entropy is the lowest at 0. To ensure that the decision tree construction process reaches a leaf where all documents are more likely to be in the same category, the entropy should thus be minimized.

With > 2 classes, the above equation can additionally be generalized (3.3) to get

$$E(S_i) = -\sum_{j=1}^{n} p_j \log_2(p_j), \tag{3.4}$$

where n is the total number of classes (splits) and p_j is the proportion of documents that belong to the j-th class [23]. Note that currently, $E(S_i)$ only represents the quality of a single subset. To

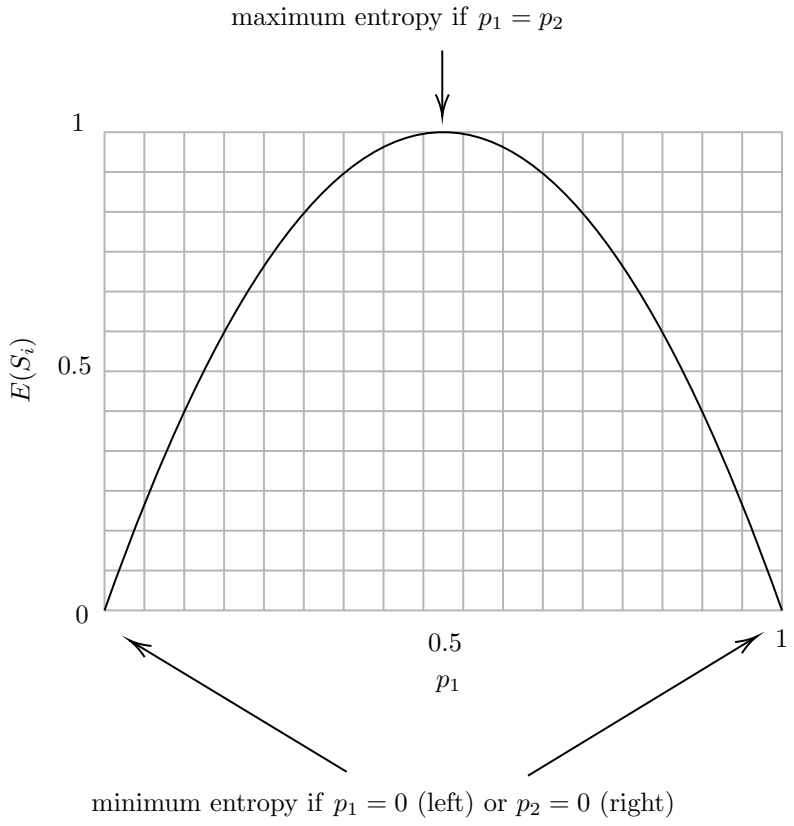

Figure 3.2: Decision Tree: Entropy of Class (p_1) Proportions

obtain the quality for the entire split, one can then take a weighted average,

$$I(S,T) = \sum_i \frac{|S_i|}{|S|} E(S_i), \tag{3.5}$$

to get the average entropy over all subsets $S_i \in S$ using the term T, otherwise known as the *information*.

After obtaining the information for a term t_k, it can be compared to the original entropy of S before the splitting process to see how much improvement was gained. Information gain is thus the amount of entropy removed and can be computed as

$$Gain(S,T) = E(S) - I(S,T). \tag{3.6}$$

Note that maximizing information gain is equivalent to minimizing entropy.

However, such a highly-branched tree may also be problematic as it is prone to *overfitting*, where terms that are not actually useful for prediction are selected. This issue may be solved by *pruning*, a process that either stops a growing branch of the tree when information becomes unreliable (*pre-pruning*) or simplifies a tree post-facto by replacing certain internal nodes with leaves (*post-pruning*) [23]. Pre-pruning is based on a statistical significance test, most commonly the χ^2, and stops the growth of the tree whenever there is no statistically significant association between any term and the class at a node. Post-pruning first grows a full tree, then goes back and either (1) replaces a subtree with a leaf or (2) deletes an intermediary subtree while maintaining performance. Other impurity measures such as the popular Gini index can also be used instead [23].

3.3 Support Vector Machines

The support vector machine (SVM) is a relatively new method that was introduced to text classification by Joachims [22] in 1998. It is based on the idea of the *Structural Risk Minimization Principle* from computational learning theory. Structural risk minimization strives to find a hypothesis h that ensures the lowest true error, defined as the probability that h will err on a randomly selected test sample.

For the sake of simplicity, consider this method in the binary classification setting. Given a training set of documents already converted into their vector-forms and their respective positive and negative labels, the SVM is essentially attempting to find some decision surface (or hyperplane) σ_i that best separates the positive training samples from the negative training samples by the widest margin. This process in depicted in Figure 3.3, where the best decision surface is indicated by the bold line and other potential decision surfaces considered are indicated by the regular lines. *Support vectors*, the training samples closest to the best hyperplane, are indicated by small boxes and ultimately determine the best decision surface. In other words, removal of any support vector will result in a change in position of the best hyperplane.

Given input pairs (x_i, y_i) where each vector x_i is labelled $y_i = \pm 1$ ($+1$ indicates a positive

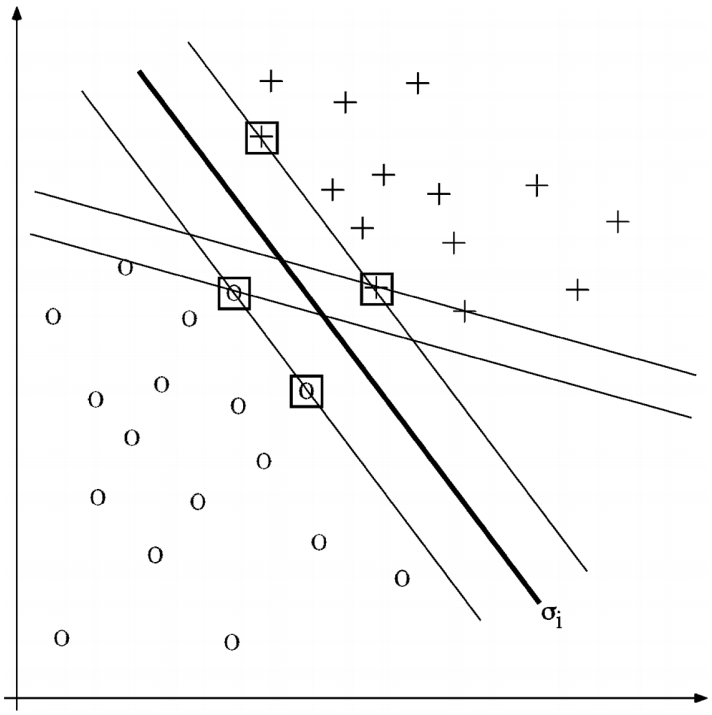

Figure 3.3: Support Vector Machine for Binary Classification [33]

classification and -1 indicates a negative classification), the best decision surface follows the equation

$$\mathbf{w}^T\mathbf{x}_i + b = 0, \tag{3.7}$$

where $\mathbf{w}$ is a vector of weights and b is the bias. Similarly, the bordering hyperplanes that the support vectors lie on can be rewritten as

$$\mathbf{w}^T\mathbf{x}_i + b = 1 \tag{3.8}$$

$$\mathbf{w}^T\mathbf{x}_i + b = -1, \tag{3.9}$$

since this is the furthest they can be placed while still maintaining the correct classifications. The distance between the optimal hyperplane and one of these bordering support vector hyperplanes is

$$\frac{\mathbf{w}^T\mathbf{x}_i + b}{||w||} = \frac{1}{||w||},$$

so the total distance between the two bordering hyperplanes is $\frac{2}{||w||}$; this is essentially the margin to maximize.

Maximizing $\frac{2}{||w||}$ is equivalent to minimizing $||w||$. The following additional constraints are also

necessary to make sure that there are no data points between the two bordering hyperplanes:

$$\mathbf{w}^T\mathbf{x}_i + b \geq 1 \text{ if } y_i = 1 \tag{3.10}$$

$$\mathbf{w}^T\mathbf{x}_i + b \leq -1 \text{ if } y_i = -1 \tag{3.11}$$

Altogether, this results in a quadratic programming problem that can be solved by applying the Langrangian multiplier method.

One major advantage of SVMs is that the classifier's ability to learn can be independent of the dimensionality of the feature space [22]. That is, since the complexity of the SVM hypothesis is based on the margin of separation instead of the number of features, the SVM is always generalizable as long as the data is separable. This property renders the SVM particularly well-suited for text categorization since it can handle high dimensional input spaces and most text categorization problems are indeed linearly separable.

In the case where data is not necessarily linearly separable, however, it may still potentially be separated by a quadratic or any other function. This means that if the data can be mapped to a higher dimensional space, linear separation can be achieved. A kernel function $K(\mathbf{u}, \mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v})$, where Φ is the function that maps space $\mathcal{X}$ to some higher space $\mathcal{F}$ helps achieve this [35]. Two commonly used kernels are the polynomial kernel, $K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^p$, and the radial basis function kernel, $K(u, v) = e^{-\gamma(\mathbf{u}-\mathbf{v})\cdot(\mathbf{u}-\mathbf{v})}$. The former induces polynomial boundaries of degree p in the original space and the latter induces boundaries by placing weighted Gaussians at key training steps.

Chapter 4

Training Dataset

One of the goals of this project is to develop an effective confusion detection model. Since this classifier will ultimately be integrated into a reading annotations system, its performance on the test set will be improved if the initial training set also consisted of reading annotations. One of the difficulties that come with this task lies in the fact that real-world datasets containing such annotations are scraped from online courses and are often sparse and unlabelled. Preliminary data analysis, in addition to text cleaning, thus also required labelling the annotations for confusion.

The challenge that comes with this manual labelling process is that because self-reporting is unreliable, actual "ground truth" classifications will be very difficult to obtain. Thus, the best classifier I can build is one that agrees with the labellings of the original training set.

4.1 Lacuna Stories

The data set used to train the classifiers was collected from Lacuna Stories, a social annotations platform created by the Stanford Poetic Media Lab in 2013. Designed to enhance reading experiences, encourage community learning, and promote discussion, it allows each registered user to freely take notes on all text, images, video, and audio uploaded to the platform within a social environment. Students can share their annotations with one another and have conversations on the readings outside the classroom. The instructors on Lacuna courses are also able to see how their students engage with the text and what particular annotation skills they are developing through the Annotations Dashboard, which records the annotations in real-time.

Figure 4.1 depicts the annotations interface that a student would see when using Lacuna to take notes on a reading. To create the annotation, the user highlights a selection of text with the cursor, which then consequently prompts a pop-up window. This window consists of a text box allowing the user to type their specific annotation and select tags ("comment," "question," "analyze," and "compare") that mostly likely correlate with the type of annotation they are making. The user can also easily share these notes with the outside community by toggling the privacy settings ("private," "instructor,' "peer-groups," and "everyone").

While Lacuna Stories has a very well-established setup, I noticed that although there is the social

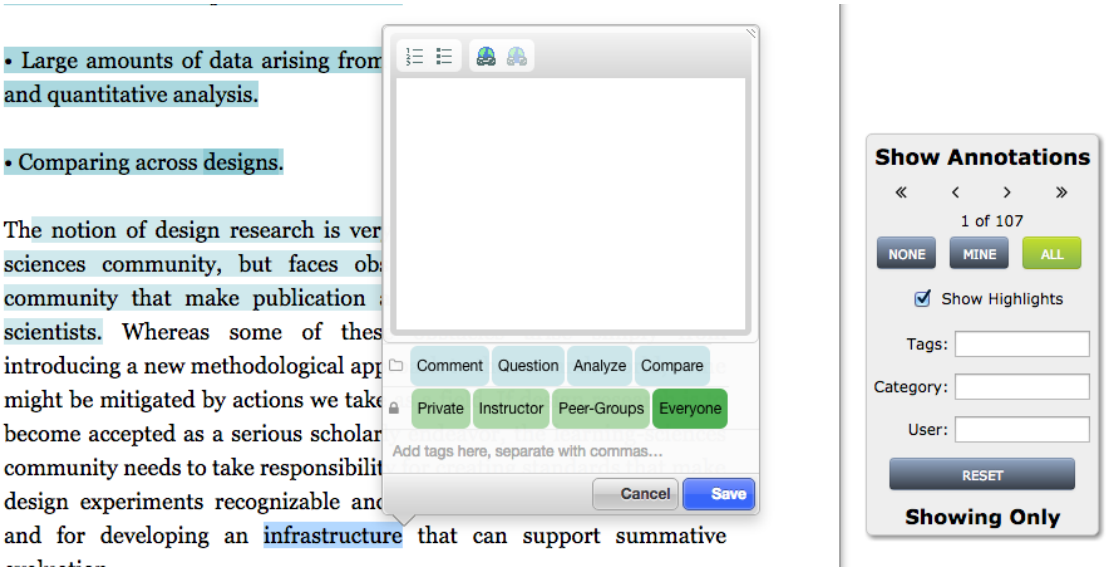

Figure 4.1: Lacuna Stories Annotations Interface

aspect of sharing completed annotations with the outside community, the process of creating the annotation itself is still a solitary one. In addition, if no one is immediately online to specifically look at a student's annotations, the student will virtually receive no feedback. My desire to give some form of immediate commentary on a student's current understanding of the reading material is where this project comes in.

I extracted a collection of 2000 annotations from Lacuna Stories. These annotations were created by a variety of different students reading different articles in different classes. Each entry in this dataset contained one unique annotation as well as other relevant meta-information, including the annotation id, user id, the passage the user highlighted, the title of the article, and the reading strategy employed in that annotation. Unfortunately, these annotations were not pre-labelled with information regarding the student's confusion level.

4.2 The Confusion Coding Manual

Because the Lacuna dataset was not labelled for confusion, I formulated my own confusion coding manual. To do this, I asked two coders to code for confusion on the same dataset so that I can establish a "gold standard" of coding rules. I obtained these rules pseudo-greedily–that is, I add rules to the manual such that their inclusion would ultimately increase the *inter-reliability* score.

Inter-reliability, also called *inter-rater concordance* or *inter-observer reliability*, is a metric that quantifies the degree of agreement among raters. High inter-reliability indicates that there is consensus among the raters and the metric they are rating by is more likely to be similar. In contrast, low inter-reliability indicates that there could potentially be bias in the rating process. Common metrics for inter-reliability are outlined below.

- **Percent Agreement**: the probability of agreement is the simplest and least informative measure. It is computed as the proportion of agreements over the total number of decisions. One major weakness of this method is that it does not account for agreements that occur solely by chance. This side effect is extremely problematic if the number of choices the raters can select from is small.

- **Cohen's Kappa (κ)**: devised by Jacob Cohen [14] in 1960, this statistic describes the proportion of agreement for categorical data after chance is from the equation. It is generally believed to be more robust than percent agreement and computed as

$$\kappa = \frac{p_o - p_e}{1 - p_e}, \tag{4.1}$$

  where p_o is the proportion of actual observed agreement and p_e is the proportion of agreement that is expected by chance. Cohen's kappa behaves similarly to correlation coefficients and range from -1 to 1, although Cohen notes that values under 0 are unlikely in practice. The Kappa results can be interpreted as follows: ≤ 0 indicating no agreement at all, $0.01 - 0.20$ indicating slight, $0.21 - 0.40$ indicating fair, $0.41 - 0.60$ indicating moderate, $0.61 - 0.80$ indicating substantial, and $0.81 - 1.00$ indicating perfect agreement [14]. However, there is some controversy surrounding this metric as the indices are hard to interpret. That is, an agreement of 61% by Cohen's Kappa may still be problematic, but this issue is not captured by the statistic.

- **Scott's Pi (π)**: named after William A. Scott, Scott's Pi is very similar to Cohen's Kappa as it also accounts for chance agreement and shares the same formula. However, it differs from the former in the definition of chance agreement p_e. The p_e in Cohen's Kappa is the squared geometric means of the marginal proportions, while the p_c in Scott's Pi is their squared arithmetic means [15]. Scott's Pi also assumes that coders have the same distribution of responses and can be thus less informative. A Scott's Pi within $0.21 - 0.40$ is considered "fair," $0.41 - 0.60$ considered "moderate," and above 0.60 considered "good [25]."

- **Krippendorff's Alpha** (α): this last metric was named after Klaus Krippendorff [24] and generalizes the metrics mentioned above. Alpha's general form is

$$\alpha = 1 - \frac{D_o}{D_e}, \tag{4.2}$$

  where D_o is the observed disagreement and D_e is the disagreement expected by chance. Ranging from 0 to 1, an alpha value of ≤ 0.667 is considered unacceptable, a value ≥ 0.667 and ≤ 0.800 decent, and a value ≥ 0.800 reliable. This statistic is also more powerful than the previous ones as it is applicable to any number of coders, any type (nominal, ordinal, binary, etc.) of metric, and even to incomplete data.

To establish an effective confusion encoding manual, I would thus want coders using this manual to have high inter-reliability, regardless of the metric used. This is achieved by first instructing the

two coders to code for confusion on a separate "training" dataset that already comes pre-labelled for confusion. For my purposes, I used the Stanford MOOC Forum Posts [1] dataset, a collection of 29,604 anonymized learner forum posts from eleven Stanford online classes. Although this dataset did not necessarily comprise of annotations, I decided that this was still acceptable as students creating forum posts will likely be in a similar cognitive-affective state as students creating annotations. The pre-labelling for confusion on this MOOC Forum Posts dataset by Stanford was obtained by computing the Krippendorff alphas for all possible combinations of coders. After selecting the particular combination of coders that produced the best agreement score, the unweighted average of their scores served as the final confusion label [1]. These additional confusion labellings served as my pseudo "third" coder.

Table 4.1: Stanford MOOC Forum Posts Inter-reliability Scores (3 coders)

Trial	n	Pairwise Agreement (avg)	Pairwise Cohen's Kappa (avg)	Alpha[1] (nom)	Alpha[1] (ord)
1	50	32.653%	0.194	0.177	**0.229**
2	50	15.333%	0.017	-0.064	0.052
3	50	40%	**0.258**	**0.239**	0.139

[1] Krippendorff's Alpha.

The prelabelled confusion levels on the original Stanford MOOC Forum Posts dataset employed a 1-7 Likert scale (1: *extremely knowledgeable*, 2: *very knowledgeable*, 3: *somewhat knowledgeable*, 4: *neutral*, 5: *somewhat confused*, 6: *very confused*, 7: *extremely confused*). To remain consistent with this original categorization, the two coders were instructed to also maintain the same scale. 50 posts were randomly sampled from the dataset and given to the two coders to rate for confusion separately. This process was iterated three times and the resultant reliability coefficients, computed with ReCal OIR ("Reliability Calculator for Ordinal, Interval, and Ratio data") [17] can be found in Table 4.1. After discussions about what would increase inter-reliability, the two coders reached a consensus that it was difficult to differentiate between the subtlety in meaning within levels of confusion (i.e. "extremely knowledgeable" vs. "very knowledgeable") and levels of knowledgeable ("very confused" vs. "somewhat confused"). Moving forward, I thus combined these intermediary levels. While this choice would lower the inter-reliability of the coders vs. the original labelling, I believed that it would raise overall agreement between the two coders. Since condensing the scale resulted in a significant improvement in inter-reliability (40% agreement, kappa of 0.258, and alpha of 0.239 in Trial 3), I decided to reduce the range of the Likert scale from 1-7 to 1-5 moving forward with the actual dataset.

After another around of discussion, the following rule was also established:

- **Definition of "knowledgeable"**: instead of being defined as the opposite of "confused," a text is considered more "knowledgeable" if it contains some additional form of explanation or justification.

Table 4.2: Lacuna Dataset Inter-reliability Scores (2 coders)

Trial	n	Pairwise Agreement	Scott's Pi	Cohen's Kappa	Alpha[1] (nom)	Alpha[1] (ord)
1	50	52%	0.31	0.345	0.317	0.637
2	100	82%	0.726	0.728	0.728	**0.886**

[1] Krippendorff's Alpha.

Now that clearer rules about confusion categorizations were established, the two coders moved on to rating for confusion on the actual Lacuna dataset since it was much closer to the type of data the classifier will be analyzing. Again, 50 and 100 annotations were randomly sampled from the dataset and given to the two coders to rate for confusion separately. The resultant reliability coefficients, computed with ReCal OIR, from these trials can be found in Table 4.2.

Establishing a common definition of "knowledgeble" caused the inter-reliability scores to rise significantly, especially the Krippendorff's Alpha value for ordinal data. However, since this new value of alpha is $0.637 < 0.667$, it clearly needs further improvement. Another round of discussion culminated in the following additional rules:

- **Handling vocabulary words**: there were certain annotations that contained definitions for unknown vocabulary words or provided synonyms for words. These types of annotations should all be designated confusion level 3 (neutral).
- **Rhetorical questions/critiques**: there was some disagreement on whether critique of text, posed as rhetorical questions, actually indicated confusion. Unless the annotator explicitly framed their comment as a question that implied a lack of understanding, the rhetorical question should be labelled as confusion level 2 (somewhat knowledgeable) or even confusion level 1 (extremely knowledgeable), depending on context.

These new rules resulted in a drastic increase in inter-reliability scores in Trial 2 with a pairwise agreement of 82%, a Cohen's Kappa of 0.728, and an ordinal Krippendorff's Alpha of 0.886. Since $0.61 \leq 0.728 \leq 0.80$ and $0.880 < 0.886$, these scores were now sufficient and the confusion coding manual was reliable. Following this manual, one of the two coders was then selected to code the rest of the Lacuna dataset ($n = 2000$) following the outlined rules. A condensed version of this confusion manual can be found in Appendix A.

Overall, the process of constructing this manual demonstrated that creating a universal definition of confusion is an extremely difficult task that requires numerous iterations. This current final version may not even agree with the user's definitions of confusion, a problem which I discuss in greater detail in Section 6.2.

Chapter 5

Methodology

Constructing the complete confusion detection system for reading annotations required both a machine-learning based backend and a friendly user interface as the frontend. Initial evaluation of the system is then conducted through a pilot user study.

5.1 Backend: Confusion Prediction with LightSide

I used LightSide [28], an open-source machine learning platform, as the basis to construct my models. It divides the machine learning process into the following 6 steps, each of which can be manipulated manually:

`Extract` → `Restructure` → `Build` → `Explore` → `Compare` → `Predict`

`Extract` handles the conversion of the training set into feature tables and its interface can be seen in Figure 5.1. The original plugins that were pre-built into the system for feature extraction included basic features, n-grams, column features, regular expressions, and stretchy patterns. This list, although extensive, was still insufficient for my purposes since I wanted to include additional features specific to text analysis of annotations. Thus, I expanded this pool of tools for feature extraction by creating my own plugins. A comprehensive account outlining the steps I took to create new plugins can be found in Appendix B. Using this method, I created the following features: type-token ratio (TTR), automated readability index (ARI), sentiment analysis, and question mark identification/counting, which I stored in a new plugin called "Complexity Metrics," as seen in the third row of feature extractor plugins in Figure 5.1.

`Restructure` allows the user to manually adjust the feature tables obtained from `Extract`, including functionality to combine features, combine feature tables, and filter through unwanted features.

`Build` handles the construction of the machine learning models according to the following algorithms: Naive Bayes, Logistic Regression, Linear Regression, Support Vector Machines, and Decision Trees. An example of a trained SVM model is depicted in Figure 5.2.

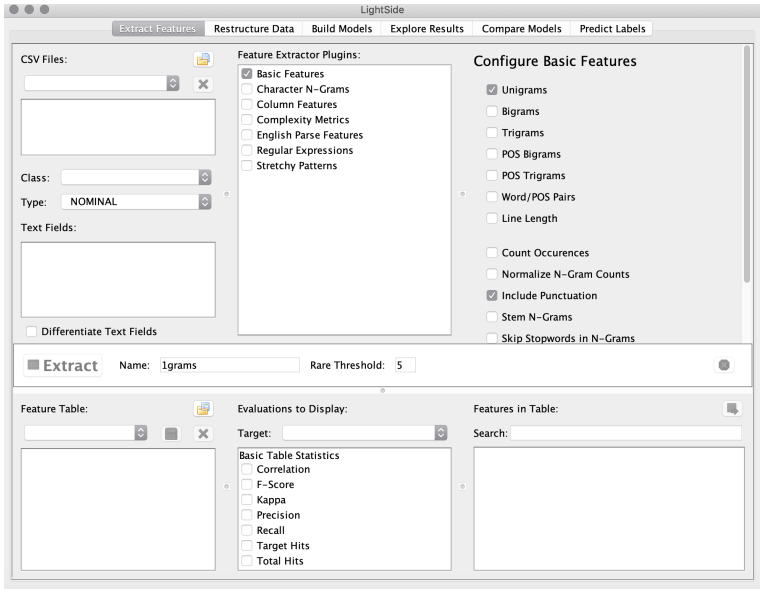

Figure 5.1: LightSide User Interface (Extract)

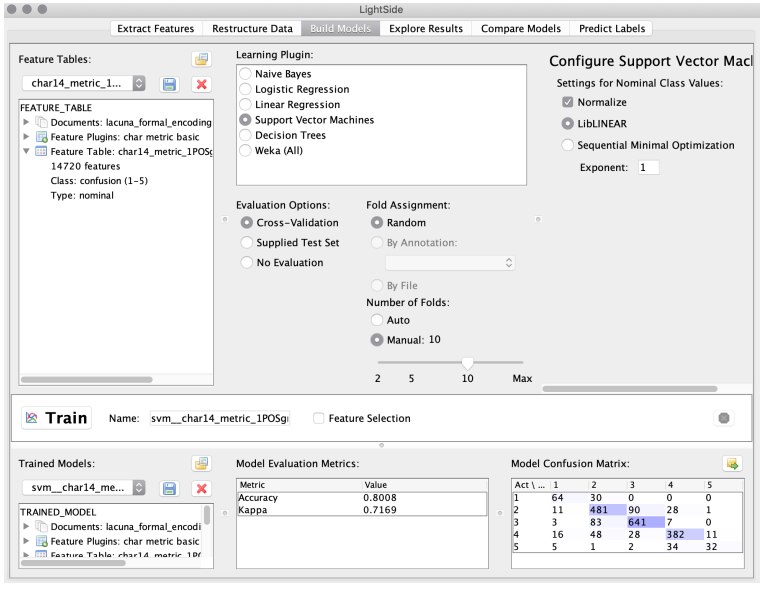

Figure 5.2: LightSide User Interface (Build)

After constructing each model, `Explore` provides insight into the different weights and influences assigned to each feature and offers other error analysis tools.

`Compare` handles the comparison of these metrics across different models.

`Predict` allows the model to predict on new testing data.

For the purposes of confusion classification (1-5), I implemented different combinations of features including n-grams, questions, part of speech, pronoun distribution, sentiment, TTR, ARI, and text length based on prior works as discussed in Chapter 2.3.2. Using these feature tables, I then constructed various Naive Bayes, decision tree, and SVM models and compared their performances using random 10-fold cross-validation. Since LightSide only provided accuracy and kappa as the performance measures, I additionally computed other metrics such as precision, accuracy, F1, and Krippendorff's Alpha using the confusion matrix. The results of these evaluations can be found in Section 6.

5.2 Frontend: Confusion Detection Interface

The interface of the confusion detection annotation software was written entirely in Javascript, HTML, and CSS. I used Annotator.js [2], an open-source JavaScript library that provides the capability to add annotations to any web page, as the basis of annotations creation. Since this baseline package only included the functionality to create, edit, and delete annotations, however, I additionally used PoeticMediaLab's Annotator Categories Plugin [39] that allowed users to choose categories for their annotations and color the specific annotation according to the category it belonged to. I re-purposed this plugin so that the categories for each annotation were restricted to only represent the five confusion levels as established by the final version of the confusion coding manual (1: *extremely knowledgeable*, 2: *somewhat knowledgeable*, 3: *neutral*, 4: *somewhat confused*, 5: *extremely confused*). I designated specific colors to each confusion level, ranging from light-yellow denoting confusion level 1 to dark red denoting confusion level 5.

Progression through this interface proceeded as follows. First, the user is introduced to a starting screen, as seen in Figure 5.3. This screen displays a simple textual explanation of the purposes of the study as well as briefly describes the steps the user will follow. After clicking on the `START`, the user transitions to Figure 5.4. This screen is where the five different confusion levels are introduced to the user, who is then prompted to familiarize themselves with them by clicking on the legend. I decided to include this interactive portion instead of explaining the confusion levels in text form in order to encourage more engagement with the software. This legend also remains on the left-hand side of the screen for the entire duration of the study so that users can always refer back to it to remind themselves what color corresponded to which confusion level.

Figure 5.5 introduces the passage that the user will be annotating for the study. Since I will be recruiting from a pool of computer science students for the future preliminary user study, I selected "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," [34] a paper from Google DeepMind detailing the algorithm behind the new AlphaZero, as the reading passage because it was both sufficiently difficult that it was likely to incite feelings of confusion and particularly interesting to this audience. The user was instructed to thoroughly

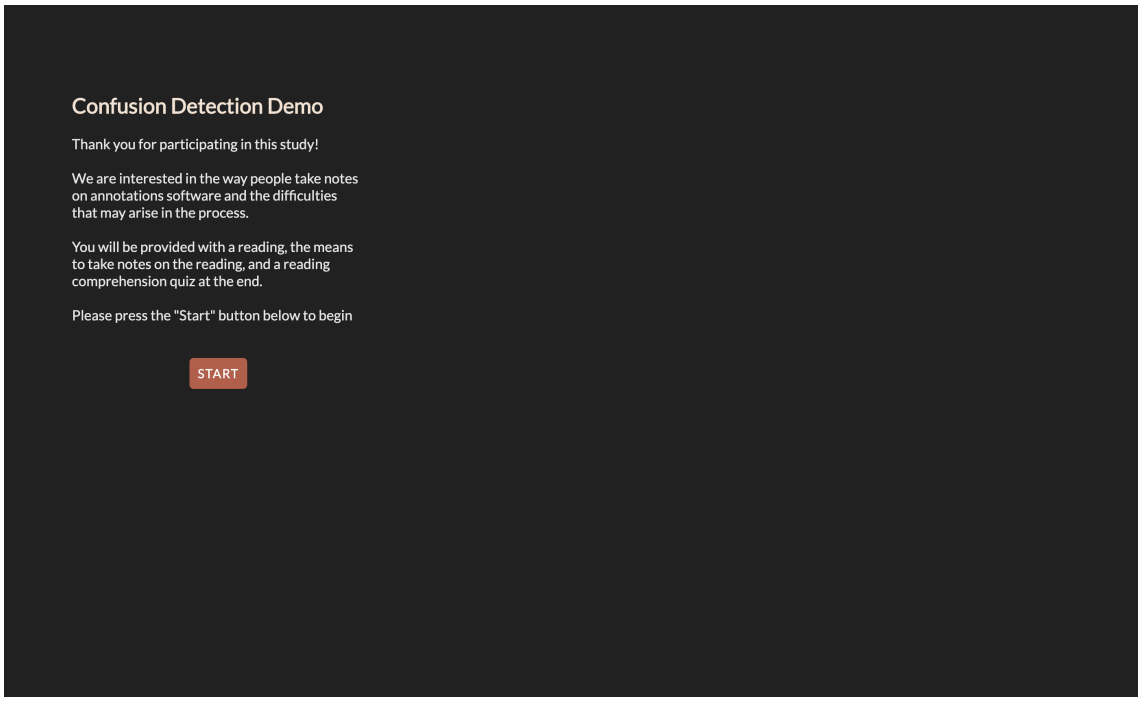

Figure 5.3: Confusion Detection Interface Screen 1

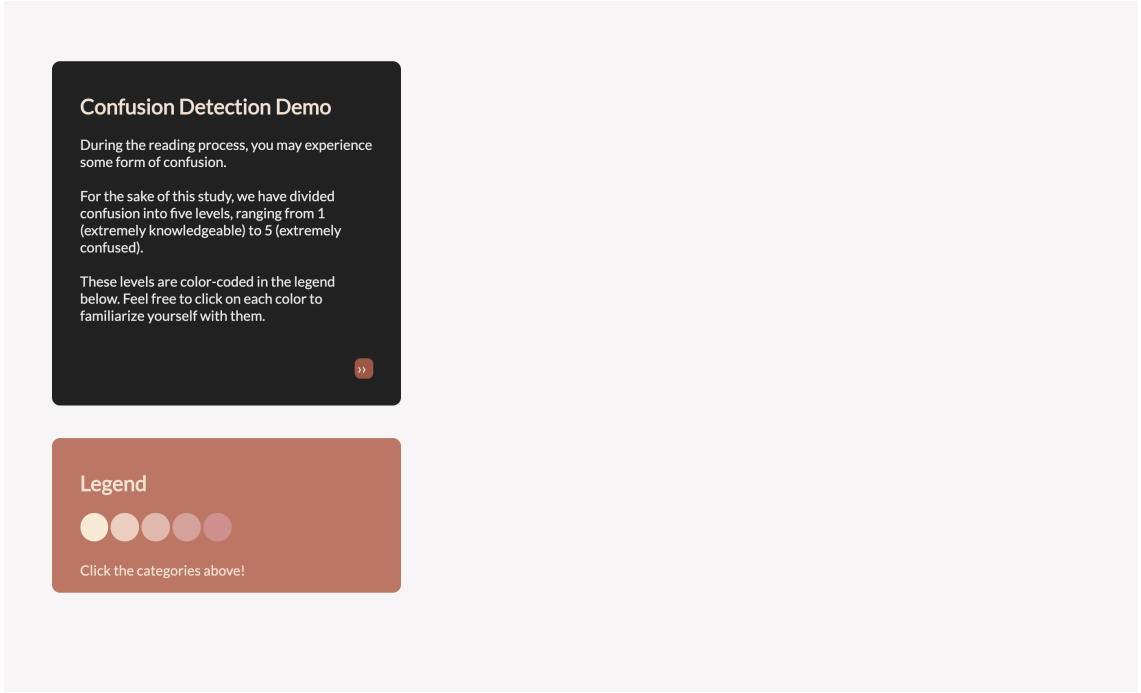

Figure 5.4: Confusion Detection Interface Screen 2

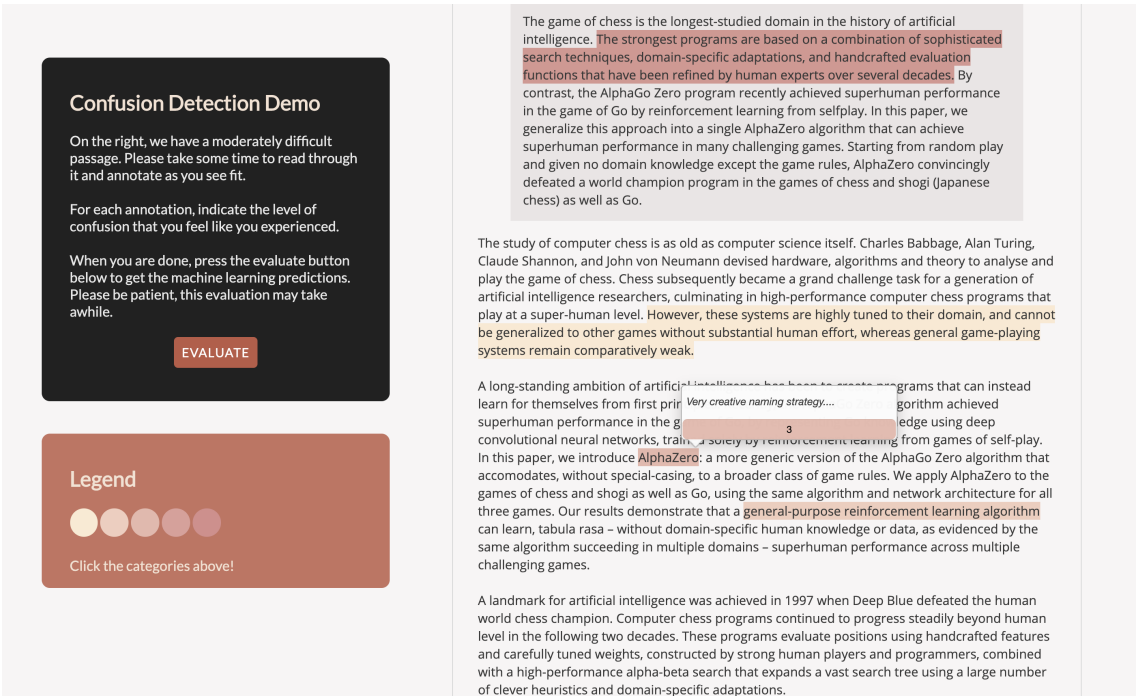

Figure 5.5: Confusion Detection Interface Screen 3

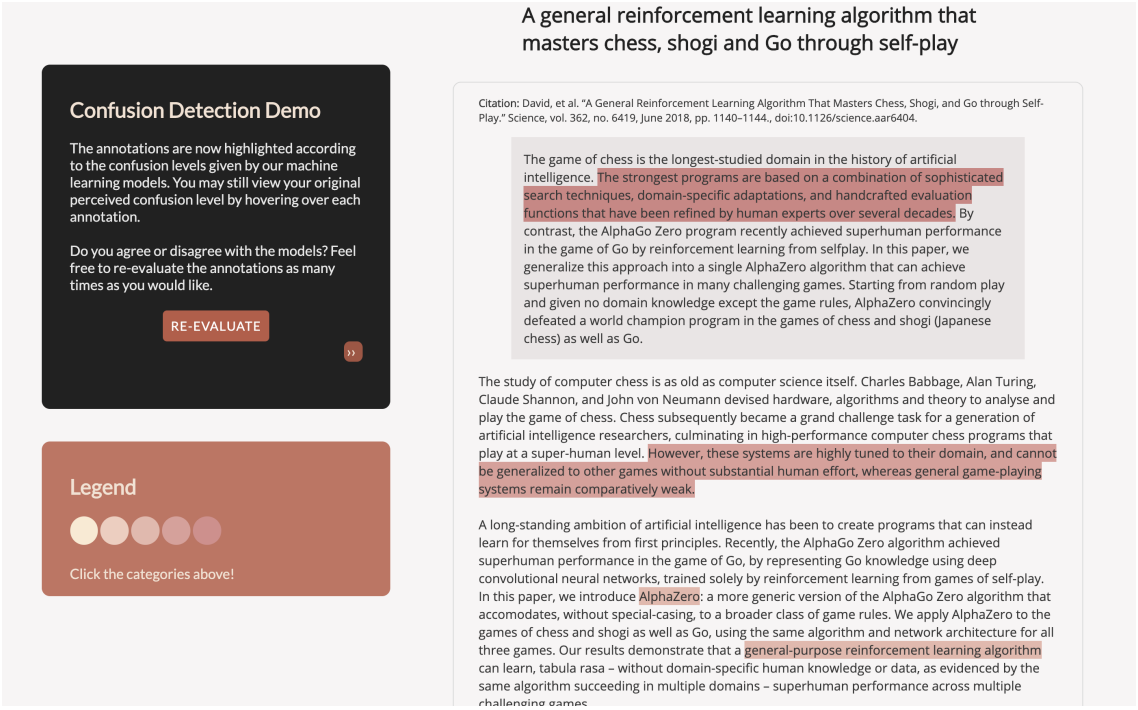

Figure 5.6: Confusion Detection Interface Screen 4

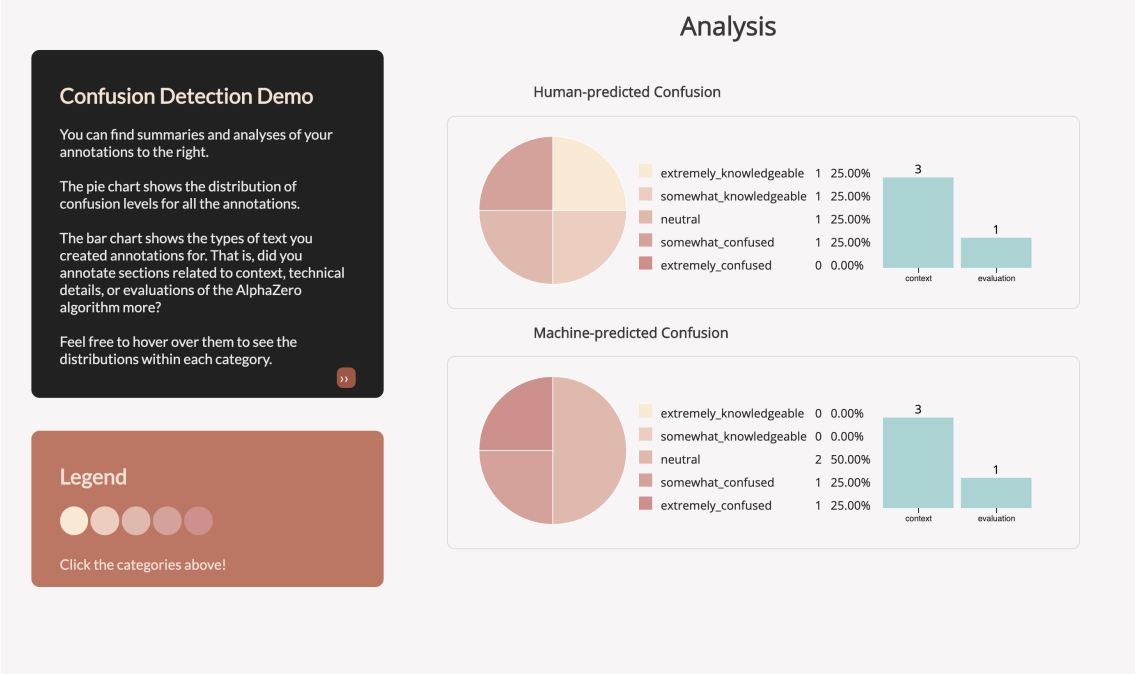

Figure 5.7: Confusion Detection Interface Screen 5

read through the passage and annotate it as they would normally. For each annotation made, they indicated the level of confusion they experienced at that instance.

An example of a completed annotation can be found in Figure 5.5. The text is highlighted based on the color that reflects the confusion level as indicated by the user. Hovering over this highlight then reveals the annotation itself as well as the corresponding confusion level. After the user is done creating all of their annotations, they can click on the `EVALUATE` button to obtain the machine learning predictions.

Figure 5.6 shows what a user would see after clicking on the `EVALUATE` button. The highlights of the text are re-colored to display the confusion levels as determined by the machine learning models. The original categorization made by the user can still be accessed by hovering the mouse over each annotation, although now the user may only view one original categorization at a time. At this point, the user may still add, edit, or remove annotations as they see fit. They may also choose to click the `RE-EVALUATE` button to obtain machine learning predictions for any new or modified annotations until they are ready to proceed.

Figure 5.7 shows an overall summary and analysis of the differences between the human-predicted and machine-predicted confusion levels. The pie chart represents the distribution of confusion levels throughout the entire reading while the bar chart displays the number of annotations created for each topic (i.e. content, technical details, evaluation). Note that these topics were created manually for this specific reading, but can be modified to display topics extracted from latent Dirichlet allocation instead (see Section 7.2.3). These plots are also interactive: for example, the user may hover over the dark red portion of the pie chart to see the number of annotations for each topic that are "extremely

confused." Similarly, they may also hover over the technical details bar to see how confusion levels were distributed within that topic. Students may utilize this information to identify areas that they were more confused about to focus on.

After completing this step, the user's annotations, which are currently stored locally using the Offline Annotator Plugin [11], are automatically outputted to a JSON file. In particular, I was interested in the following meta-data:

- `userid`: unique identifier of the user writing the annotations. Later used to differentiate between them during analysis.
- `ranges`: the location of the annotation, includes a start offset and an end offset.
- `quote`: the specific content of the text the user highlighted and created the annotation for.
- `category`: the category of confusion (1-5) self-selected by the user when writing the annotation.
- `predicted category`: the category of confusion (1-5) predicted by the ML algorithm.
- `text`: the content of the annotation.

5.3 Frontend + Backend Integration

Figure 5.8 depicts the integration process. The goal was to construct a fluid user experience where machine learning serves only an auxiliary role in user understanding of their own confusion. To achieve such an effect, I first allow the user to categorize their own perceived confusion levels as they read through and annotate the text. After this process, I run all of the annotations through the ML models and return the machine-predicted results to the user. The user is then able to compare differences between the two confusion levels.

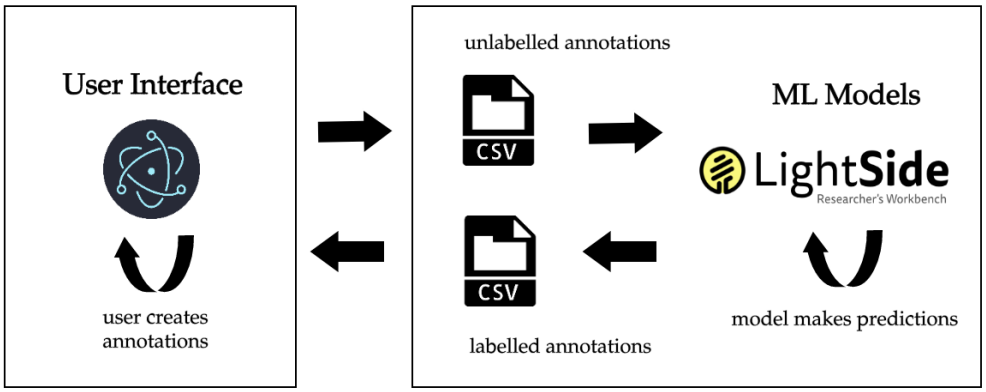

Figure 5.8: Frontend and Backend Intergration Process

The main problem encountered in this process was that the users needed to gain access to the LightSide model in order for it to make predictions. There are two possible solutions:

1. **Host the LightSide models on a server which each client can then send their annotations to as communication requests**. This online model prevents the user from directly accessing the trained ML models but allows me to collect data more easily since I can take advantage of Mechanical Turk. However, it is more difficult to implement. It also means that I will not be doing user studies in person and thus cannot directly observe their interactions with the application.

2. **Host everything locally**. This offline model removes the complications of maintaining an active server. However, it does require that the user either pulls the entire Git repository containing the application or use someone else's device. It also limits the amount of user studies I can do, but since these are in-person studies, the data collected will be more substantive as I will also record behaviors and interactions.

For ease of testing, I opted for the second solution and will be conducting in-person user studies. However, the hosting code is separate from the code of the actual software, the latter of which can be converted to an online model in the future.

One problem specific to making predictions using LightSide models still remains. When the user wants to receive their predictions and presses the `EVALUATE` button in Figure 5.5, all annotations currently stored are scraped from local storage and exported into a csv file. The ML model is then supposed to read in this file, make its predictions, then output the results into another csv file, which in return is read back and displayed on the interface. However, prediction in LightSide only occurs in two ways, either (1) within the provided LightSide interface or (2) via the following command line command which calls a shell script:

```
scripts/predict.sh <model> 'utf-8' <unlabeled data> <output>,
```

where `<model>` denotes the name of the trained model desired, `<unlabeled data>` denotes the name of the file containing user annotations, and `<output>` denotes the name of the outputted predictions. Since launching LightSide and making the prediction from there would be disruptive to the entire user experience, the natural choice is to choose option (2). However, issuing such commands directly from Javascript for the command prompt/terminal would pose a security hazard since I do not want any Javascript code to access internal files. Instead, I used Electron [18], an open-source framework with the Chromium rendering engine and Node.js runtime (thus with access to the `child_process` module) to create a local server. Finally, the pipeline is complete.

5.4 Pilot User Studies

I conducted a series of preliminary user studies to see how the application affected the learning process. To control for prior knowledge, I recruited students currently enrolled in CS 256: Algorithm Design and Analysis, a core course in the Computer Science major at Williams that focuses on the mathematical modeling of computational problems and developing algorithmic techniques to solve them. These students were randomly split into control and test groups, as depicted in Figure 5.9. After a quick introduction to the purposes of the study, the former group was given a randomized

model that predicted confusion levels based on a random number generator while the latter group was presented with the actual trained model.

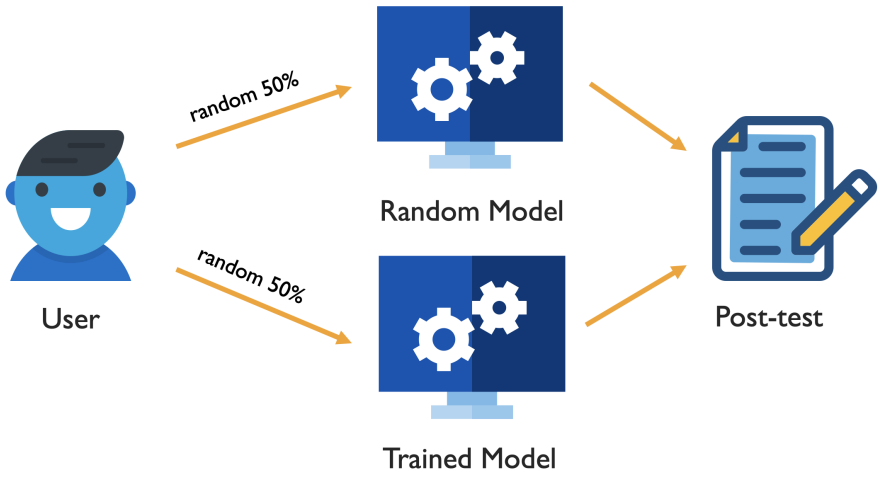

Figure 5.9: Pilot User Study Flowchart

During this entire process, the user is encouraged to think aloud, verbalizing any thoughts or questions they may have going through the application. A moderator sits next to them and records everything that the user does or says. While this moderator is not allowed to directly interfere with the user's interactions with the software, they can answer any clarifying questions if the user asks them.

After navigating through their respective confusion detection demos, each user was directed to a post-test, created using Google Forms, intended to assess both their overall comprehension of the passage and impressions of the experience after using the software. This post-test, in addition to demographic information about previous computer science classes taken, included the following comprehension (C) and perception (P) questions.

- (P) How familiar are you with machine learning algorithms? Options: Likert scale from 1-5
- (C) What is the typical search method used in traditional chess and shogi programs before AlphaZero? Options: Iterative deepening, alpha-beta, DFS, BFS
- (C) What are the cons to this method (referring to the traditional programs)? Options: It requires a domain expert, Training takes too long, It requires too much memory
- (C) Which of the following about the new AlphaZero algorithm is true? Options: It takes in a vector of the next possible moves as input, It outputs a vector of move probabilities, The output is computed through heuristics, The parameters of the deep neural network are trained through supervised learning
- (C) In games against AlphaGo Zero (go), Stockfish (chess), and Elmo (shogi), AlphaZero won by a large margin? Options: True, False

- (C) Do you think the AlphaZero algorithm algorithm can be eventually extended to a general games playing system? Why or why not? Options: Long response answer
- (P) How accurate do you think you are in categorizing your own confusion? Options: Likert scale from 1-5
- (P) How accurate do you think the machine learning model is in predicting confusion? Options: Likert scale from 1-5
- (P) I completely understand the passage given. Options: Likert scale from 1-5

All comprehension (C) questions except for the long answer are automatically graded, tallying up to a maximum total score of 4 points. The long answer question is graded manually, mainly looking for a reasonable justification for the response. These scores serve as measures of understanding of the passage. That is, the more points they receive, the better grasp they are considered to have of the material.

The perception questions try to assess learner affect. That is, I are interested in the following questions:

- Does prior knowledge on the material affect a student understanding of their own confusion?
- How does actual accuracy of the model affect student perception of its accuracy?
- Are there any correlations between the accuracy of perceived self-confusion and the accuracy of perceived machine-predicted confusion?
- Given all these parameters, how is learning itself affected?

The entire study took approximately 30 minutes for each user. Once they completed both the application and the survey, the participants were compensated $10 for their time.

Chapter 6

Results

In this thesis, I attempt to construct a confusion detection application that functions in real time as the user is actively reading. This was achieved by selecting the best combination of language and discourse based features and classification models that specifically targeted confusion encountered in annotations. I then assessed the effectiveness of this system by running a pilot user study to see whether the accuracy of the confusion predictions influenced learning and perception, both of which contribute to intention-to-use.

6.1 Machine Learning Model Evaluations

All models were validated using k-fold stratified cross-validation ($k = 10$). That is, the original dataset was partitioned into k equally sized subsets. One set is retained as the test set and other $k-1$ sets are used as training. The benefit of this method is that each observation is used for testing exactly once. In particular, stratified cross-validation is commonly used in classification problems as it ensures that the folds are selected in a way that maintains the same proportion of categories in each fold.

By averaging performances across the $k = 10$ folds, I can then compare different learning algorithms and determine the best model to use for confusion classification.

6.1.1 Comparison of Learning Algorithms

To evaluate which learning algorithm produced the best results, I mainly used 5 metrics: accuracy, kappa (as discussed in Section 4.2), average precision, and average recall, and average F1. Precision attempts to quantify the percentage of positive results that were actually correct. If tp denotes true positives and fp denotes false positives, then precision is computed as

$$prec = \frac{tp}{tp + fp}. \tag{6.1}$$

On the other hand, recall attempts to quantify the proportion of actual positives that were identified correctly. If fn additionally denotes false negatives, then recall is computed as

$$rec = \frac{tp}{tp + fn}. \tag{6.2}$$

Unfortunately, precision and recall are sometimes in opposition; if the classification boundary was shifted to increase precision, recall would decrease, and vice-versa. To thus account for both metrics, F1 can be used instead, which provides the harmonic mean of precision and recall as

$$F1 = 2 \cdot \frac{prec \cdot rec}{prec + rec}. \tag{6.3}$$

This F1 metric essentially encodes a "values judgment" that balancing precision and recall is a good thing. In many cases, precision is a much more important factor than recall since there is a greater need to minimize false positives [7].

Table 6.1: Model Training Performance

Model	Features[1]	Accuracy	Kappa	Precision[2]	Recall[2]	F1[2]
Naive Bayes	ng	0.620	0.456	0.560	0.558	0.546
Naive Bayes	ng, pun, pos	0.678	0.538	0.628	0.602	0.598
Naive Bayes	pun, pos	0.719	0.595	0.673	0.624	0.632
Naive Bayes	pun, pos, ttr, ari, q	0.723	0.602	0.678	0.632	0.638
DT	pun, pos, ttr, ari, q	0.707	0.584	0.551	0.534	0.537
SVM	pun, pos, ttr, ari, q	0.774	0.677	0.696	0.635	0.651
SVM	pun, pos, ttr, ari, q, ug, ll	0.784	0.692	0.710	0.655	0.664
SVM	pun, pos, ttr, ari, q, ng, ll	0.798	0.712	0.764	0.683	0.703
SVM	pun, pos, ttr, ari, q, ng, ll, qnum	0.801	0.717	0.762	0.712	0.729

[1] The features used are abbreviated as follows: unigrams (ug), ngrams (ng), punctuation (pun), part-of-speech (pos), type-token ratio (ttr), automated readability index (ari), does a question exist (q), line length (ll), and number of questions (qnum).
[2] These metrics were averaged across each confusion level.

Thus, the closer the F1 score is to 1, the better the model is. A comprehensive table providing these assessment metrics and be found in Table 6.1. As expected, the Naive Bayes models performed the worst. However, using this learning algorithm, I was still able to improve the F1 score from 0.546 to 0.638 by removing n-grams and adding other features such as punctuation, part-of-speech, type-token ratio, automated readability index, and existence of questions. While a F1 of 0.638 and accuracy of 0.723 appear to be quite low, these models are still significantly better than random, which would have generated a 0.200 accuracy with a five-class classification problem.

I additionally considered using decision trees (DT) to model the data. However, even the best

DT model, as shown in the fifth row of Table 6.1, produced a F1 score of 0.537 that is still lower than that of the worst Naive Bayes model. I thus decided not to proceed with decision trees due to its poor performance.

Finally, I modeled the data using support vector machines (SVMs). With the same exact features, the switch to this new learning algorithm already produced higher kappa and F1 scores of 0.677 and 0.651, respectively. While this kappa value was already quite good ($0.61 \leq 0.677 \leq 0.80$), I tried to further improve the model. The final best model used the features from the best Naive Bayes model in addition to line length, number of questions, and n-grams and produced a kappa of 0.717 and a F1 score of 0.729.

Table 6.2: Best SVM Model Level-wise Precision, Recall, and F1

Confusion Level	Precision	Recall	F1
1 (extremely knowledgeable)	0.646	0.681	0.663
2 (somewhat knowledgeable)	0.748	0.787	0.767
3 (neutral)	0.842	0.873	0.858
4 (somewhat confused)	0.847	0.788	0.816
5 (extremely confused)	0.727	0.432	0.542

As specified in the notes under Table 6.1, the precision, recall, and F1 were all averaged over each confusion level. To get a better sense of how the classifier is performing on the micro level, I took a look at the level-wise scores, as shown in Table 6.2.

Most notably, the F1 scores for this SVM model appear to be the best at confusion levels 2-4, averaging around a high value of 0.814. However, at the ends of the spectrum with confusion levels 1 and 5, the F1 is significantly lower at 0.663 and 0.543, respectively. A closer look reveals that the model has an especially low recall score of 0.432 at confusion level 5, indicating that out of all the total number of annotations that were confusion level 5, only around 43.2% were identified correctly. This result suggests that the model may be more partial towards the medium levels and can be further verified by looking at a confusion matrix.

The confusion matrix in Figure 6.1 reveals that the model is actually heavily biased towards confusion levels 2-4. Although the F1 score for this model was high, the classifier will actually perform poorly on real data as it will rarely make predictions for confusion level 1 or 5.

6.1.2 Handling Imbalanced Classes

The bias of the SVM classifier can be traced back to class imbalances in the original training set. The bar chart in Figure 6.2 depicts the distribution of confusion levels.

As shown, there are significantly more data points for confusion levels 2-4. In fact, the number of observations for confusion level 3 (734) is almost ten times the number of observations for confusion level 5 (74). Rennie et al. [32] suggest handling the imbalance of training samples by using

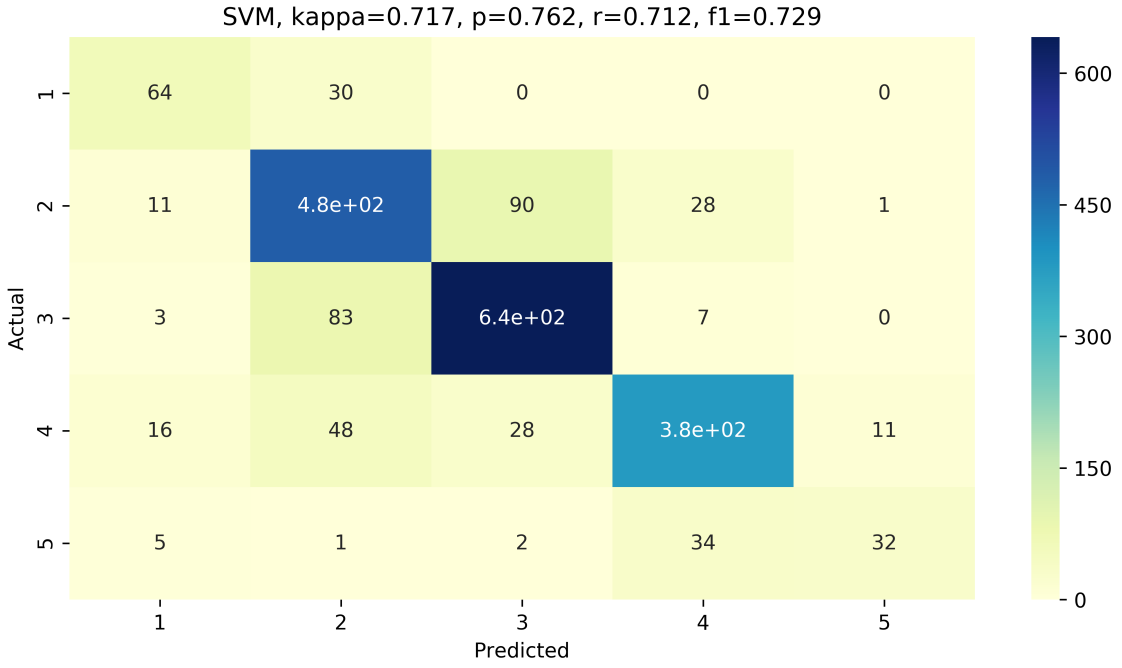

Figure 6.1: Confusion Matrix for best SVM: Kappa=0.717, Prec=0.762, Rec=0.712, F1=0.729

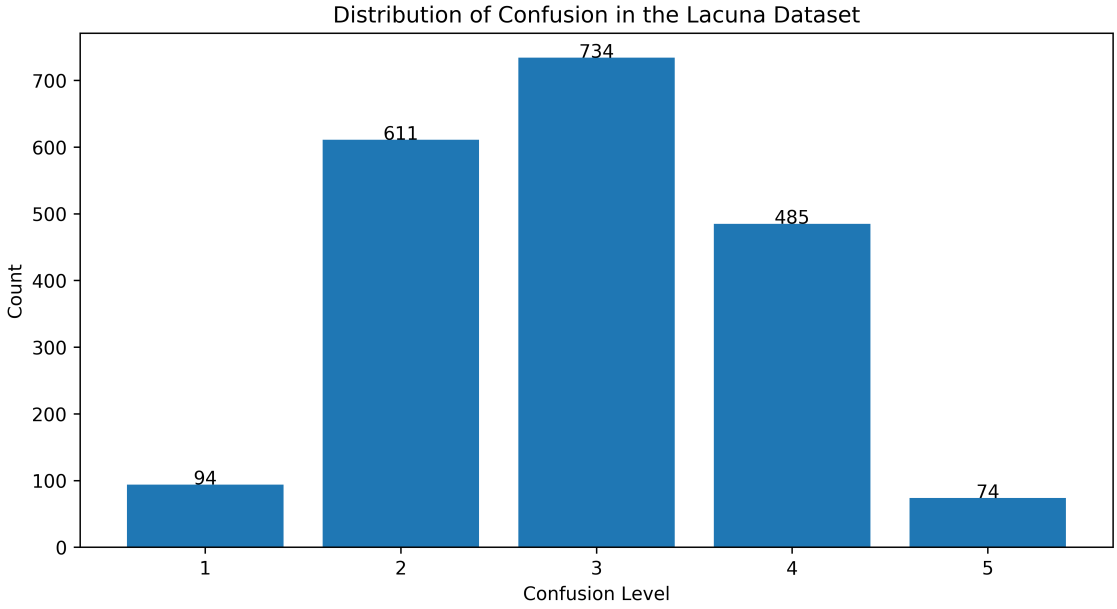

Figure 6.2: Distribution of Confusion in the Lacuna Dataset

the "complement class" method where weights are estimated for data belonging to all categories except for the category of interest. However, the resultant number of such weights will still remain imbalanced. To remedy this, the dataset still needs to be re-balanced. There are two ways to do this: either under-sample the high-count classes or over-sample the low-count classes. Since there are pros and cons to either methods, I will attempt both.

The goal of oversampling is to make up for the class imbalance by supplementing portions of the training data with multiple copies of minority classes. However, because I am also validating the trained model using a subset of that data, a model trained on heavily oversampled data is more likely to overfit. Conversely, undersampling removes samples from the majority class and attempts to solve imbalance without overfitting. However, it introduces new problems: the resultant total sample size may be too small, leading to increased classifier variance and the omission of important samples.

I employed random oversampling and undersampling. Instead of oversampling observations to match the total number of data points in confusion level 3, however, I only oversampled to $n = 400$ observations per class such that the total number of observations ($n = 2000$) is maintained. Classes which exceeded this number of observations were undersampled to $n = 400$. This served as my first new dataset. My second new dataset consisted of undersamplings from all classes to match the number of observations in the minority class, $n = 74$. I then fit SVMs on these datasets.

Table 6.3: Resampled SVM Model Training Performance

Model	Samples per level	Accuracy	Kappa	Precision[1]	Recall[1]	F1[1]
SVM	400	0.922	0.903	0.925	0.923	0.923
SVM	74	0.716	0.649	0.723	0.717	0.713

[1] These metrics were averaged across each confusion level.

Table 6.3 depicts the performance of the best SVM models found. Interestingly, the set of features that generated the best models remained the same. Here, I note that the $n = 400$ model has an extremely high F1 score of 0.923, which may suggest that the classifier is indeed overfitting. The $n = 74$ model has a F1 score of 0.713 that is closer to the F1 score of the previous best model, 0.729.

Looking at the level-wise scores as depicted in Table 6.4 reveals that the $n = 400$ model still maintains very high F1 scores across all levels. In fact, F1 reached 0.993 with recall at a perfect 1.000, further affirming the possibility of overfitting. The $n = 74$ model is comparatively worse with the lowest F1 score at confusion level 4. However, note that this score is still higher than the previous best SVM model's lowest F1 score of 0.542 at confusion level 5. This $n = 74$ model also appears to perform better when working with "more knowledgeable" annotations (confusion levels 1-3).

The confusion matrices for $n = 400$ and $n = 74$ SVM models, as depicted in Figures 6.3 and 6.4, appear to be much more balanced. There is no more classifier bias as previously demonstrated in Figure 6.1, although note that the predictions in Figure 6.3 do appear to be a little too "perfect."

Table 6.4: Resampled Model Level-wise Precision, Recall, and F1

Samples per category	**400**			**74**		
Confusion	Prec	Recall	F1	Prec	Recall	F1
1	0.985	0.998	0.991	0.844	0.878	0.861
2	0.864	0.828	0.845	0.671	0.653	0.662
3	0.822	0.912	0.865	0.656	0.863	0.746
4	0.967	0.875	0.919	0.653	0.635	0.644
5	0.985	1.000	0.993	0.788	0.554	0.651

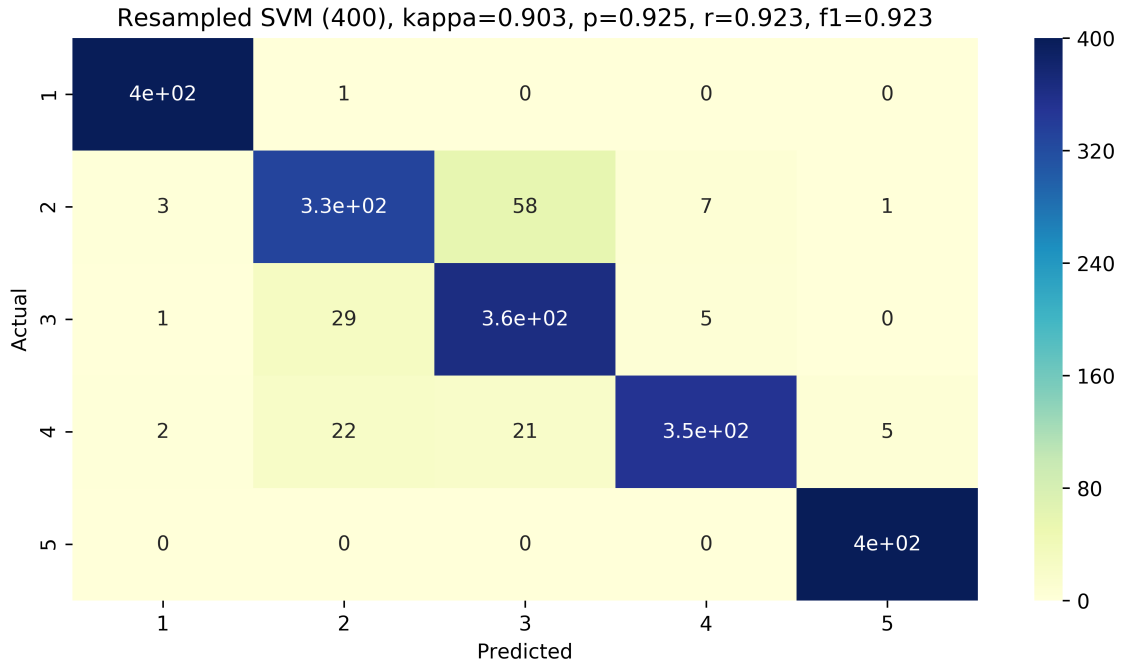

Figure 6.3: Confusion Matrix for Resampled SVM (n=400): Kappa=0.903, Prec=0.925, Rec=0.923

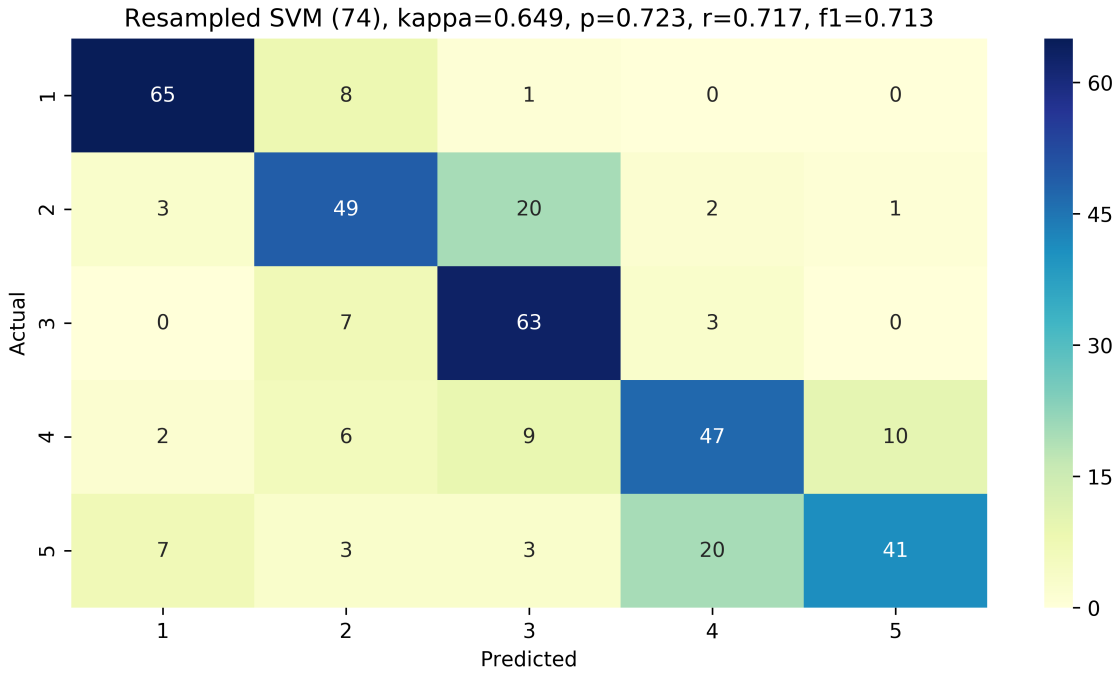

Figure 6.4: Confusion Matrix for Resampled SVM (n=74): Kappa=0.649, Prec=0.723, Rec=0.717

6.1.3 Limitations of the Current Approach

The performance scores of the oversampled model in Table 6.4 reveal that there exists some significant overfitting in the classifier. In particular, since the sample sizes of confusion categories 1 and 5 were very small, at 94 and 74 respectively, and oversampling attempts to push this number to 400 with replacement, this process resulted in numerous replicated samples both in the training and validation sets. This problem can potentially be alleviated in the future since the system itself is built in a way that allows the collection of more data at each iteration of the user study. With more new data points in confusion levels 1 and 5, the number of duplicates created through resampling will also decrease accordingly.

Feature extraction via LightSide also produced too many features. To be exact, extracting the 8 main types of features as specified in Table 6.1 resulted in a total number of 14,720 features. While the models were trained using some built-in feature selection functionalities, these features could be more heavily cut down to reduce overfitting by setting stricter thresholds for the coefficients of each feature when evaluating feature importance.

Another reason that contributes to overfitting could lie in the choice of model. That is, the SVM model with the radial basis function kernel may be too powerful especially with the large amount of features and overlapping data points due to resampling. While this kernel was selected in the earlier tuning process without resampling, returning to a simpler kernel (polynomial or potentially even linear) for the resampled models could help reduce overfitting.

6.2 Pilot User Study Evaluations

The goal of this initial evaluation of the system via a pilot study is to: (1) identify potential issues in the system design of the application for use by readers, (2) determine if machine learning models that predict confusion can be employed to improve learning, and (3) identify trends in the perception of the machine learning models. Due to time limitations, I only tested the application on five users. Three of the users were assigned to the "test" category and provided with the trained model while two of the users were assigned to the "control" category and provided with a random model. The "test" users created a total of 50 annotations while the "control" users created a total of 33 annotations. Of these latter 33 annotations, however, 4 were empty text boxes, despite the application instructing the user to never leave a highlight without annotation, and were thus removed. Each user did fully complete the survey at the end of the application.

6.2.1 Confusion Prediction

To verify the effectiveness of the trained model on actual user annotations, I computed a series of performance metrics. Agreement metrics were obtained using ReCal OIR [17] and F1 scores were obtained using `scikit` [30]. Note that because the users did not have access to the confusion manual, they may have very different notions of confusion that do not correspond with those that the machine learning models were trained on. However, as summarized in Section 2.4.1, TAM suggests that perceived usefulness contributes most to intention to use. Thus, I will be evaluating both the perceived and actual accuracy of the machine learning predictions. To obtain the latter, I re-coded all the annotations according to the confusion encoding manual afterwards.

Table 6.5: Classifier Performance on User Encodings

Classifier	n	Accuracy	Kappa	Alpha (nom) [1]	F1 (micro) [2]	F1 (macro)[3]
Random	29	0.276	0.09	0.085	0.276	0.244
SVM	50	0.22	-0.012	-0.133	0.22	0.132
SVM (400)	50	0.24	-0.007	-0.151	0.24	0.116
SVM (74)	50	0.26	0.012	-0.15	0.26	0.124

[1] Only nominal Krippendorff's Alpha was used since the ML classifications were not ordinal.
[2] Micro: metrics were calculated globally by counting the total TPs, FNs and FPs.
[3] Macro: metrics were calculated for each label then averaged.

Table 6.5 depicts the performance metrics obtained when using the user encoding of confusion as truth (perceived accuracy). Interestingly, the random, untrained classifier performed the "best" with a Cohen's kappa of 0.09, a Krippendorff's Alpha of 0.085, and F1's of 0.276 and 0.244. These results indicate that the ML-trained classifiers were not effective at all in the eyes of the user.

Looking at specific annotations created by the user, however, reveals discrepancies between their annotations and the confusing ratings they gave themselves. Consider the following examples:

- *Highlighted text*: "cannot be generalized to other games"
  Annotation: "Why is that?"
  Confusion Level: 1

- *Highlighted text*: "neural network $(\mathbf{p}, v) = f_\theta(s)$"
  Annotation: "How the neural network is modeled mathematically"
  Confusion Level: 5

The two cases, according to the confusion encoding manual, would be rated as confusion levels 4 and 3, respectively. However, the user self-rated them as confusion levels 1 and 5 instead. When asked about this categorization, the user commented that they recorded their confusion levels based on how they felt about the entire article overall at the time, while they recorded annotations that were reflective of their immediate reactions to the highlighted text.

There was also another user who confused the definitions of the levels of confusion. Although they had a legend outlining the different levels and their respective meanings as they were going through the text, they admitted to forgetting these distinctions and interpreting confusion levels as "a spectrum of confusion," where 1 indicated "slightly confused," 3 indicated "more confused," and 5 indicated "super confused," towards the end of the study. This user misinterpretation of numerical confusion labels suggests that the 1-5 numbers should be replaced with more descriptive text to reduce this confusion.

Table 6.6: Classifier Performance on Confusion Encoding Manual

Classifier	n	Accuracy	Kappa	Alpha (nom) [1]	F1 (micro) [2]	F1 (macro)[3]
Random	29	0.345	0.118	0.08	0.345	0.202
SVM	50	0.74	0.478	0.477	0.74	0.619
SVM (400)	50	0.66	0.301	0.278	0.66	0.434
SVM (74)	50	0.64	0.237	0.199	0.64	0.408

[1] Only nominal Krippendorff's Alpha was used since the ML classifications were not ordinal.
[2] Micro: metrics were calculated globally by counting the total TPs, FNs and FPs.
[3] Macro: metrics were calculated for each label then averaged.

Table 6.6 depicts the new performance metrics obtained when using the confusion encoding manual as truth (actual accuracy). As expected, the metrics for the random classifier did not change by much as the kappa, alpha, and macro F1 values still remained very low. In contrast, the performance of the trained classifiers improved drastically and all seem to outperform the random classifier. Interestingly, the best out of the trained classifiers was the SVM model without resampling with a Cohen's kappa of 0.478, a Krippendorff's Alpha of 0.477, and F1's of 0.74 and 0.619, confirming the earlier suspicions of overfitting.

To get a better sense of how the classifiers performed within each confusion level, I can look at the distribution of predictions across categories of confusion. Figure 6.5 provides a bar chart depicting the number of times each confusion level was encoded and predicted for the random classifier. I

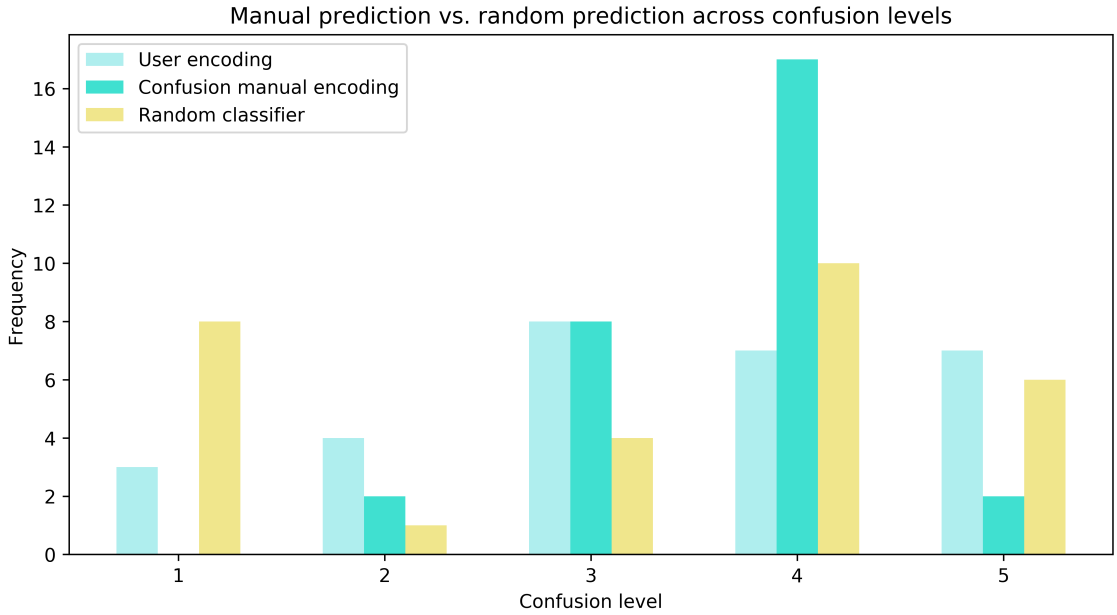

Figure 6.5: Manual vs. Random Prediction Across Confusion Levels

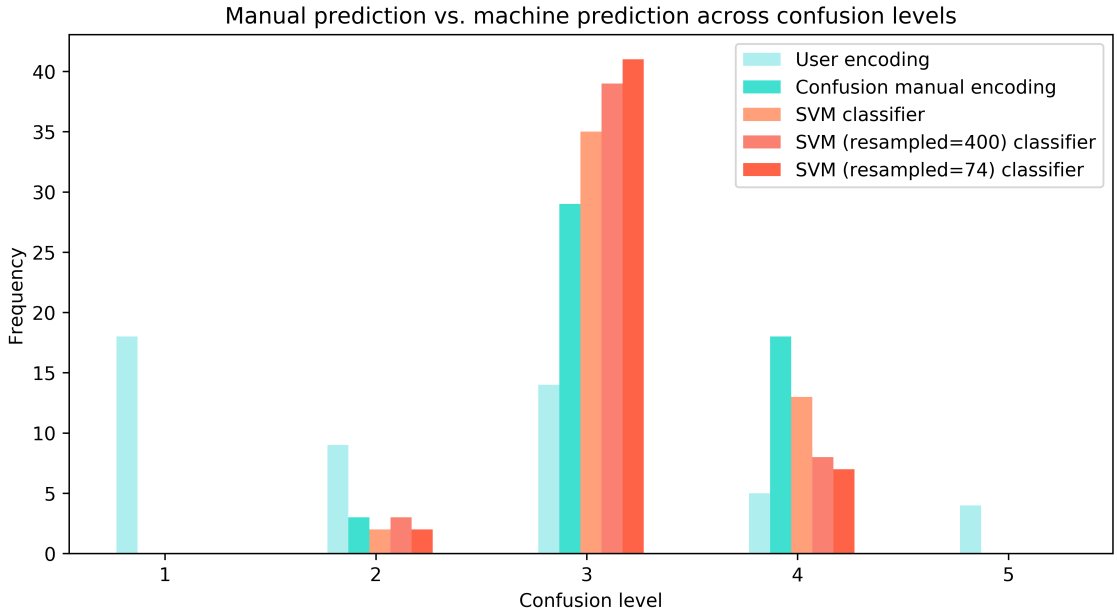

Figure 6.6: Manual vs. Machine Prediction Across Confusion Levels

note here that while the users coded for a generally even spread of confusion, coding according to the confusion manual suggested that most of the annotations actually veered towards confusion level 4, "somewhat confused," while none of the annotations were confusion level 1, "extremely knowledgeable." The latter result is directly linked to the fact that I associated "knowledgeable" with the existence of any sort of justification or explanation, neither of which were found in these user annotations.

The random classifier, due to the small sample size of $n = 29$, did not actually generate a random distribution of confusion predictions. Instead, it appeared to predict confusion level 4 the most, followed by confusion level 1, and predicted confusion level 2 the least. Coincidentally, the coding according to the confusion manual also produced a majority of confusion level 4 ratings; this resulted in the higher than random accuracy of 0.345.

Figure 6.6 depicts the number of times each confusion level was encoded and predicted for all trained classifiers. Note that the predictions given by the original SVM classifier without resampling most closely resembles the confusion manual encodings and is thus the best. The SVM classifier that was both under and over sampled to obtain 400 observations per confusion category is less accurate in the sense that it gives more bias towards confusion level 3 (neutral) predictions. This bias is even more emphasized by the SVM classifier under sampled to obtain 74 observations per category. Contrary to initial expectations, this means that re-sampling did not improve the classifier even though it originally did increase F1 scores on the training set. On this test set however, it appears that even though the original SVM classifier had class imbalances, it still outperformed other re-balanced classifiers.

6.2.2 Learning Survey & User Feedback

All users (anonymized as A, B, C, D, and E) promptly completed a survey that assessed their learning of the article and overall impressions of the application. They took a range from 5 to 10 minutes to finish all the questions. Because there were only 5 users and some had disparate definitions of confusion that were not aptly captured by the trained models (perceived accuracy of predictions being vastly different from actual accuracy), however, more studies must be conducted to affirm these conclusive statements about learning as pertaining to the accuracy of the random versus trained classifiers. However, their individual responses can still indicate important notions of how perception of confusion and learning are affected.

Table 6.7 shows the aggregate responses to the survey questions from all the users in the pilot study. Note that Users A and B who both received the random classifier claimed to be not so familiar with machine learning algorithms (2 on the Likert scale), obtained the same number of correct comprehension questions (3), and had similar understandings of the passage (2). However, depending on the reported perceptions of their own confusion, their reported perceptions of machine predicted confusion vastly differed. User A believed that they are accurate in categorizing their own confusion (4) and the ML model was not so accurate (2). Conversely, User B believed that they were not accurate in categorizing their confusion (2) and the ML model was more accurate (4). Note that since these two users were given the same model, these reported results are independent of

Table 6.7: User Study Survey Results

User ID (R=random, T=trained)	A(R)	B(R)	C(T)	D(T)	E(T)
Total CS classes taken	3	6	8	6	4
How familiar are you with machine learning algorithms? (1-5)	2	2	2	4	3
Total score (max=4)	3	3	2	3	2
Do you think the AlphaZero algorithm can be eventually extended to a general games playing system? Why or why not?	Yes[0]	Yes[1]	No[0]	Yes[1]	Yes[1]
How accurate do you think you are in categorizing your own confusion? (1-5)	4	2	2	3	3
How accurate do you think the machine learning model is in predicting confusion? (1-5)	2	4	4	2	4
I completely understand the passage given. (1-5)	2	2	1	3	2

[0] Provided no justification.
[1] Provided some reasonable justification.

both perceived accuracy (taking the user labelled confusion as correct) and actual accuracy (taking the confusion manual labelled confusion as correct) and more reliant upon their perceptions of themselves.

Looking at the trained classifier group revealed that the users in this study, on average (2.7), scored slightly worse (2, 3, 2) on the comprehension questions in comparison with the control group despite reporting more familiarity with machine learning algorithms (2, 4, 3). However, the same trends, although slightly subdued, in perception of confusion can be seen here. That is, User C also reported low confidence in categorizing their confusion (2) and high confidence in the machine learning model (4). Users D and E reported that they were okay at categorizing their own confusion (3) and perceived ML model accuracies of 2 and 4, respectively. Overall, these survey questions indicate that the random classifier group saw greater differences between reported perceived self-predicted confusion and reported perceived machine-predicted confusion and slightly higher scores on the comprehension test.

Because the users were not presented with the confusion coding manual, they shared varying understandings of what each level of confusion meant. In particular, the following was observed by the moderator during the user studies:

- User B believed that the 1-5 Likert scale indicated the scale of confusion (where 1 meant "slightly confused" instead of "extremely knowledgeable").

- User A commented towards the end of the study that "I would just annotate important parts, not necessarily for confusion"; a large portion of this user's annotation thus solely comprised of asterisks (which the user labelled as 1 or 2 but the ML classifier would label as 3).

- User C felt that they tended to highlight things with lighter colors because red (more confused) was more stressful to see, even though they were actually more confused. In this case, the indicated confusion level was not reflective of their actual confusion at all.

A comprehensive list of moderator observations can be found in Appendix C. Altogether, these mixed beliefs about confusion levels resulted in the random classifier having higher perceived accuracy with respect to the original user labels. Combined with the previous conclusions about perceived self-predicted confusion versus perceived machine-predicted confusion and comprehension scores, it appears that "more accurate" ML models that assume user-predictions of confusion as correct widen the disparity between perceived self-predicted confusion and perceived machine-predicted confusion and improve comprehension. However, since both the difference in accuracy between the random and SVM classifiers ($F1 = 0.112$) and the sample size of the users ($n = 5$) were very small, future studies with a modified user interface that made identifying confusion clearer for the reader need to be conducted to confirm this conclusion.

Conversely, if the confusion coding manual predicted confusion levels was as correct and the focus was on actual accuracy, then "more accurate" ML models actually decrease the disparity between perceived self-predicted confusion and perceived machine-predicted confusion and are more detrimental to comprehension. This result suggests that the confusion predictions could potentially be "distractions" for the reader and draw their attention away from the reading itself. The reader may not need to know their position on the knowledgeable to confused spectrum at all unless they are extremely confused. An alternative design approach to the user interface would thus involve only displaying the confusion level to the reader whenever they are predicted as extremely confused. Another possible "distraction" from learning was having the reader label their own confusion. If the model was sufficiently accurate, the reader can bypass this step completely and consequently only focus on the reading.

Overall, the users in the study did report that the annotations application helped them better engage with the text and improved their overall reading comprehension. By posing as anchor points that the users can return to, the highlights served as "checkpoints" in each learner's understanding. The color gradient used (light yellow $\rightarrow$ dark red) also helped them prioritize which annotations to focus on, although User C did report some disincentivization against the dark red highlights. Users also reported that they focused more on the darker highlighted annotations after clicking the "evaluate" button to see what annotations the ML model reported as more confused.

On the other hand, users also expressed some frustration with using the application itself. In particular, User B was confused by the legend and only figured out how to access the confusion levels after a prompt from the moderator. User C actually did not click on the legend at all, did not ask for instructions, and assumed knowledge on confusion levels. User E had additional complaints about the layout of the annotations, the color choices of confusion, and expressed sentiments that the "edit annotation buttons could be bigger." On a second iteration of user studies, these comments will be incorporated to further enhance the user experience.

6.2.3 Summary

In conclusion, the confusion detection application did not reflect what the students labelled for confusion, but rather attempted to reflect "expert" human annotators. The disparity between the two resulted in vastly different perceived prediction accuracy and actual prediction accuracy for the trained model. However, the pilot user study still demonstrated that the predictions for confusion was still a useful tool for students who want to monitor their learning progress and state of understanding. Running the studies also revealed importance insights on how the accuracy of the model (both perceived accuracy and actual accuracy) affects the divide between reported perceived self-predicted confusion and perceived machine-predicted confusion.

Chapter 7

Conclusion

7.1 Contributions

In this thesis, I have achieved the following:

- Created a manual set of rules for splitting confusion into five categories that is verified through inter-reliability scores.
- Implemented LightSide plugins for the following features: question identification/counting, type-token ratio, automated readability index, and sentiment analysis. Trained and compared different classifiers for their ability to effectively separate student annotations into the five levels of confusion as specified in the manual.
- Built an annotations application that incorporates the trained classifier to give users personalized predictions of their own confusion and facilitates the data collection process of student annotations without third-party websites.
- Performed a preliminary evaluation of the application for perceived accuracy, actual accuracy, student learning, and student perception.

Confusion detection is a difficult task, especially when the exact definitions of confusion are not entirely well-established and can be highly subjective. While I tried to hone in on what it meant for a piece of text to be more "knowledgeable" versus more "confused," the manual constructed by measuring agreement from two coders can always be improved with the addition of more coders. Despite these limitations, however, using this manual to label confusion and construct the corresponding models still resulted in a best SVM model with a 10-fold cross-validated accuracy of 0.801 and F1 score of 0.729. Although these performance scores may not be satisfactory in binary classification, recall that five-class classification is a much more difficult task.

The preliminary evaluation offers several insights for further improving the user interface of the developed system. In particular, the ambiguity associated with the definition of confusion negatively affected the user studies. Because each user had their own slightly different pre-conceived notions

of the subject, even the best trained classifier performed very poorly (accuracy=0.22 , F1=0.132) when considering the user defined confusion levels as truth, resulting in poor accuracy perceived by the user. In fact, the random classifier (accuracy=0.276, F1=0.244) actually performed better than the trained classifier, although not significantly. This does not mean that the trained classifiers were ineffective, however. After each annotation was re-coded according to the guidelines as specified by the previously established confusion manual, the best trained classifier (accuracy=0.74, F1=0.619) significantly outperformed the random classifier (accuracy=0.345, F1=0.202), revealing that the actual accuracy is quite good.

Since perceived accuracy deviated so much from actual accuracy, directly conclusive statements cannot be made about how machine-predicted confusion affected student learning or student perceptions of their learning. However, the pilot user study suggests that a user more confident in categorization of their own confusion is less confident in the machine learning model, and vice versa. The survey results also seem to suggest a trend in how **the accuracy of the model affects the divide between reported perceived self-predicted confusion and reported perceived machine-predicted confusion**. That is, the control group that was presented with the random classifier exhibited greater differences in this divide, while the test group that was presented with the trained classifier exhibited smaller differences in this divide. The control group also received a slightly higher comprehension score (C: avg=3, T: avg=2.33), despite having less familiarity with machine learning algorithms. This suggests that "more accurate" models, as perceived by thee user, encourage learning while increasing the difference between reported perceived self-predicted confusion and reported perceived machine-predicted confusion. However, while the random classifiers were slightly better than the the trained classifiers when using user defined confusion as truth, they were significantly outclassed when using confusion manual defined confusion as truth. Thus, more user studies where the users code for their own confusion by following definitions for confusion closer to the manual need to be conducted to reconcile perceived accuracy and actual accuracy and to substantiate these conclusions.

7.2 Future Work

7.2.1 Improvement of ML Models by Advanced Oversampling

In this thesis, I predicted confusion levels using reading annotations in real time and obtained an actual accuracy of 0.74 and a F1 of 0.619 during the pilot user study. While no current research to my knowledge has achieved this thus far, I note that the ML model can still be drastically improved. Overfitting of the resampled models can be ameliorated by a harsher feature selection process and collecting more data through running more user studies with the system, as suggested in Section 6.1.3. In addition, since the current best model is the SVM model without resampling, this means that this model still suffers from class imbalances. Random sampling, regardless of oversampling or undersampling, was not the solution since it ultimately resulted in models that performed worse in comparison to the original. Thus, alternative sampling methods should be considered.

SMOTE: Synthetic Minority Over-sampling Technique [12] is a strategy that over-samples the

minority class by creating synthetic examples instead of repeatedly picking with replacement. It functions over the "feature space" rather than the "data space" as follows. A feature vector is taken from the minority class its k-nearest neighbors found. The difference between this feature vector and one of its neighbors is multiplied by a random scalar $x \in [0, 1]$ and added back to the original feature vector under consideration. This process is repeated for any many new data points as necessary and is superior to the original oversampling with replacement method as it guarantees that the decision region of the minority class is generalized [12].

ADASYN, or the adaptive synthetic sampling approach [20], further expands upon SMOTE by also ensuring that more synthetic data is generated for minority classes that are harder to learn in comparison to minority classes that are easier to learn. By using a weighted distribution for different minority classes based on learning difficulty, ADASYN ensures that the bias introduced by class imbalance is reduced and the classification decision boundary is dynamically shifted toward the classes that are more difficult to classify. Since the application deals with a five-class classification problem with two minority classes confusion level 1 and confusion 5, using ADASYN could potentially significantly improve classifier performances.

7.2.2 Clarification of the User Interface

To solve the issue of disparate interpretations of confusion among the users, I need to clarify the definition of confusion and incorporate aspects of the confusion manual into the interface of the application. Such an ideal location can be the screen depicted in Figure 5.4, where the user is first introduced to the five levels of confusion via the legend. Instead of just providing them with the abstract descriptions of "extremely knowledgeable," "somewhat knowledgeable," "neutral," "somewhat confused," and "extremely confused," more details from the confusion coding manual and examples of annotations that fall into each category can be provided to more clearly delineate each category.

User E (see Appendix C) also suggested other improvements to the interface that can be further incorporated into its next iteration. Some examples include greater differences in the colors to represent the different confusion levels, clearer instructions on how to create the first annotation, larger edit/remove annotation buttons that are easier to see, and a hoverable legend. The best way to decide which changes should actually be implemented would be through a user experience design approach and several more iterations of user studies to improve the interface. These future experiments would also provide insight into whether having the user label their own confusion was actually beneficial to the learning activity.

7.2.3 Latent Dirichlet Allocation for Confusion Clustering

In addition to confusion identification, students and instructors may also be interested in what *types* of sentences or paragraphs give rise to varying degrees of confusion when the student is reading them. Topic modeling thus allows clustering and extraction of abstract "themes" that are common in text that give rise to confused annotations.

Latent dirichlet allocation (LDA) [8] is a generative statistical model commonly used in the topic modeling of text. It infers topics in a corpus by first imagining the process whereby the documents are created, then reverse engineering it. Under this model, the overall vocabulary is comprised of V words and that each document of interest is comprised of N words, $\mathbf{w} = \{w_1, \ldots, w_N\}$. This document can then be reproduced by sampling a mixture of some k number of latent topics, then sampling words w_i from that mixture of topics. Mathematically, this means the following.

1. Choose $\theta \sim \text{Dirichlet}(\alpha)$
2. For each of the N words w_n:
   (a) Choose a topic $z_n \sim \text{Multinomial}(\theta)$.
   (b) Choose a word $w_n \sim p(w_n|z_n, \beta)$, a multinomial conditioned on z_n.

$\alpha = \{\alpha_1, \ldots, \alpha_k\}$ is denoted to be the k-dimensional Dirichlet parameters and $\beta =$ the $k \times V$ matrix that denotes the parameters controlling the k multinomial distribution over words. Specifically, these relationships can be visualized in Figure 7.1.

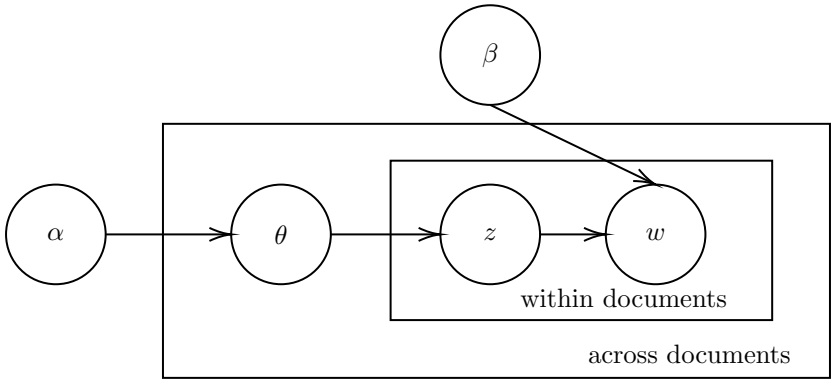

Figure 7.1: Graphical Model of the Latent Dirichlet Allocation [8]

Given these parameters α and β, the marginal probability distribution of a document $\mathbf{w}$ is thus

$$p(\mathbf{w}|\alpha, \beta) = \int_\theta p(\theta|\alpha)\Big(\prod_{n=1}^{N}\sum_{z_n} p(z_n|\theta)p(w_n|z_n, \beta)\Big)d\theta.$$

In this way, LDA has the power to not only identify particular topics that individual students find confusing, they can also suggest overarching themes of what groups of students struggle with. Although this cannot be extensively used to enhance the existing machine prediction models, as "general" topics of confusion for one group cannot be assumed to apply to another individual user, its future inclusion in the application can serve as a helpful summarization tool.

7.2.4 More User Studies & Online Hosting

In order to obtain more data to verify the tentative conclusions from Section 7.1, a lot more user studies need to be conducted. However, the difficulty with this task is that the process of recruiting

users and conducting each 30 minute study is time consuming and unrealistic considering the current social distancing protocols during the COVID-19 epidemic. Thus, moving the application from the local hosting model to an online hosting model like Mechanical Turk allows the user data collection process to proceed even synchronously.

7.3 Summary

The goal of this thesis was to build a confusion detection system that predicts student confusion levels based on their reading annotations in real time. To train the system, I devised a "confusing conding manual," a manual set of rules for splitting confusion into five categories by comparing inter-reliability scores between two independent coders. I then built a SVM classifier that achieved an accuracy of 0.801 and F1 of 0.729 on the training Lacuna dataset. This classifier was integrated into a user interface on which students can create annotations as they read. I then conducted preliminary user studies to evaluate the effectiveness of this system. Although the results of the pilot study revealed the challenges of confusion detection without a fixed universal definition for confusion (accuracy of 0.22 and F1 of 0.132 when assuming user-labelled confusion to be correct and accuracy of 0.74 and F1 of 0.619 when assuming confusion coding manual to be correct), they also suggested patterns in the way accuracy of predictions affected student perception of their confusion and identified future directions for research regarding confusion detection in the reading process.

Appendix A

Confusion Detection Manual

I established a set of rules that would most likely increase inter-reliability between two coders to create a consistent encoding of confusion for the unlabelled Lacuna data set. The metrics used to measure inter-reliability included pairwise agreement, Scott's Pi, Cohen's Kappa, and Krippendorff's Alpha, a complete review of which can be found in Section 4.2.

In this manual, different levels of confusion are mapped to scores as follows

- Extremely knowledgeable → 1
- Somewhat knowledgeable → 2
- Neutral → 3
- Somewhat confused → 4
- Extremely confused → 5

where *confusion* refers to the extent to which the creator of the annotation is expressing an inability to understand the concept discussed in the passage and *knowledge* refers to the extent to which the creator can provide some additional form of explanation or justification.

The following rules were created to handle certain categories of annotations:

- *Vocabulary words*: both synonyms and definitions of highlighted words are confusion level 3
- *Rhetorical questions/critiques*: unless the annotation is explicitly framed as a question, rhetorical questions/critiques are confusion level 1 or 2

The coder must only use the five values of confusion as described and intermediary scores (such as 2.5) are not allowed.

Appendix B

LightSide Plugins Guide

Before creating your own plugin, make sure to familiarize yourself with the LightSide Researcher's Workbench User Manual.

B.1 Creating the Plugin

Navigate to `plugins/example/features` and open `WordMatchingExamplePlugin.java`. This is an example template that you can use for whatever plugin you would like to add. Edit this template as follows.

B.1.1 Setting global variables

Replace the global variable `String matchString = "awesome";` with variable(s) that are critical to computing your feature of interest.

Because the original example attempts to find the number of occurrences of "awesome," this word is thus stored. In the case where storing such information is not necessary (for example, you just want to extract something like sentiment), you may just set the global variable to some toggle such as `Boolean findSentiment = true;` that indicates whether or not you want this feature to be extracted.

B.1.2 Extracting and storing the feature of interest

Edit the `extractFeatureHitsForSubclass` method in the for-loop over the documents as follows.

(a) Set `featureName = documents.getTextFeatureName(XXX, column);` where `XXX` is the variable you declared in part 1.1.

(b) Set `feature = Feature.fetchFeature(prefix, featureName, featureType, this)`. The prefix refers to a String name given to the feature being extracted. The featureType refers to one of the following types of features: (`Type.NUMERIC`, `Type.BOOLEAN`, `Type.STRING`, or `Type.NOMINAL`).

(c) Now, extract your actual feature of interest (if you are still following the sentiment extraction example, this extraction could either be *pos*, *neg*, or *neutral*). Let's call this `featureValue`.

(d) Determine whether your feature of interest is of type `LocalFeatureHit` or of type `FeatureHit`, and initialize it accordingly.

- `FeatureHit` implies that there is only one occurrence of the feature in the document (e.g. sentiment or post length). Its constructor takes the form

```
FeatureHit(feature, featureValue, docIndex).
```

- `LocalFeatureHit` extends from `FeatureHit` and implies that there are two or more occurrences of this feature within the same document (e.g. the number of certain pronouns). Its constructor takes the form

```
LocalFeatureHit(feature, featureValue, docIndex,
                textColumn, startIndex, endIndex).
```

(e) Associate the feature hit with the feature using `documentHits.put(feature, hit);`.

B.1.3 Configuring the UI

- Edit the `generateConfigurationSettings` and `configureFromSettings` methods to make sure the configurations update correctly for each feature of interest stored
- Edit the `getOutputName` and `toString` methods to set the short name and the full name of this plugin
- Edit `makeConfigUI` to change the UI such that interacting with it triggers inclusion or exclusion of the feature

B.2 Compiling the Plugin

Assuming you have a JDK, navigate to `lightside` and type the following in terminal.

```
> javac -classpath $CLASSPATH ./plugins/example/features/*.java
> cd plugins/
> jar -cf example.jar example/
```

B.3 Setting config.xml

Navigate to `lightside/plugins/example`, open the `config.xml`, and add the following code snippet.

```
<plugin>
   <name>Name of Plugin</name>
```

```
   <author>Author Name</author>
   <version>0.1</version>
   <description>Description of Plugin</description>
   <jarfile>example.jar</jarfile>
   <classname>example.features.ExamplePlugin</classname>
</plugin>
```

Appendix C

User Study Observations

As each user navigates through the Confusion Detection application, a moderator sits next to them and records anything that the user does or says. A collection of all these observations, categorized by user, is listed below. Quotation marks are used to denote direct comments made by the user.

C.1 User A

- "What do I need to annotate it for?" (encountered when user first sees the article)
- "I would just annotate important parts, not necessarily for confusion" (user proceeded to enter asterisks as annotations)
- Messed up labelling → verbalized slight confusion but put down 2 as label (changed afterwards)
- Engaging with the text → recalls AlphaGo vs. Lee Sedol tournament
- Recognized that the past programs were rule-based
- Struggled w/ probability part of the network (verbalized) → but did not record as annotation
- "Very interesting" and "huh" utterances when going back to previous annotations after finishing the passage and clicking the evaluate to get machine predictions → after understanding, removed certain annotations
- Surprised that the ML thinks he's "extremely knowledgeable" on certain topics

C.2 User B

- Was confused by legends → figured it out after reading the sidebar
- Confused about confusion labelling → also thought 1-5 indicated the scale of confusion (not necessarily 3 = neutral)

- Highlighted without writing annotation
- Returned to read passages several times before moving on → mentioned how they returned to "more confused" annotations often
- Mentioned how pre-requisite of Go is required for article
- Mentioned how he thought he was more confused → but turned out to be more knowledgeable on technical details

C.3 User C

- Did not know you could make annotations directly by clicking and dragging cursor to highlight before instructed
- User did not click on legend (did not ask, assumed knowledge on confusion levels)
- Chose lighter color not necessarily as the opposite of confusion → but to see if they will mention it later on
- Mentioned that vocabulary/terminology is confusing
- Highlighted terms that were not clear/they did not understand
- Tended to highlight things with lighter colors b/c red (more confused) is stressful to see → but did review red annotations a lot before moving on
- Skim paragraphs believed to not provide pertinent information
- Every time something new is learned → highlight in white to flag it for later perusal
- Highlight clutter: tended not to highlight too much to clutter article
- Highlighted w/o annotations: put asterisks as a bookmark to keep certain terms in mind
- General reading strategy is to skim once briefly and refer back as necessary, so the first time is not necessarily a close reading
- Note that even though user was pretty confused -¿ kept skimming
- Seeing recurring term → mentioned how normally, user would google the term, then put its definition in the annotations
- Mentioned desire to have control F to find unknown terms that were previously defined before

C.4 User D

- Played around with highlight and annotation tool before starting passage

- Had some previous knowledge about DeepBlue/previous chess/AlphaGo algorithms, but did not know about self-play
- Asked a question about "first principles" in annotation but indicated 3 for confusion level
- Figured out how to edit annotations half way through the user study
- Confusion → 3 for vocab, but decided it wasn't critical to the overall passage
- Mentioned they would like to Wiki shogi → 3 but this is not critical to understanding the passage
- Mentioned they won't remember any of the equations after the reading
- Scale was confusing → thought 1 was not confused at all/neutral
- Looked back an reviewed all the previous annotations multiple times after clicking "evaluate" before moving on

C.5 User E

- Wondered what the legend box is → clicked and figured it out
- Mentioned how creating annotations was not obvious
- Felt like the annotation should not go above the text → want ability to exit out from annotation using a double click
- Want more differences between colors → use different colors so they don't have to remember
- "Edit annotation buttons could be bigger."
- Legend → want hover instead of click
- Want scale 1 (most confused) → 5 (least confused)

Bibliography

[1] Agrawal, A., and Paepcke, A. The stanford moocposts dataset. `https://datastage.stanford.edu/StanfordMoocPosts/`, 2014.

[2] Annotation, O. Annotator.js. `http://annotatorjs.org/`, 2015.

[3] Arguel, A., Lockyer, L., Lipp, O. V., Lodge, J. M., and Kennedy, G. Inside out: Detecting learners' confusion to improve interactive digital learning environments. *Journal of Educational Computing Research 55*, 4 (2017), 526–551.

[4] at MIT, H. G. Nota bene, 2009.

[5] Atapattu, T., Falkner, K., Thilakaratne, M., Sivaneasharajah, L., and Jayashanka, R. An identification of learners' confusion through language and discourse analysis. *CoRR abs/1903.03286* (2019).

[6] Baker, L. Comprehension monitoring: Identifying and coping with text confusions. *Journal of reading behavior 11*, 4 (1979), 365–374.

[7] Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S., and Engler, D. A few billion lines of code later: using static analysis to find bugs in the real world. *Communications of the ACM 53*, 2 (2010), 66–75.

[8] Blei, D. M., Ng, A. Y., and Jordan, M. I. Latent dirichlet allocation. *J. Mach. Learn. Res. 3* (Mar. 2003), 993–1022.

[9] Bradley, M. M., and Lang, P. J. Measuring emotion: Behavior, feeling, and physiology. *Cognitive neuroscience of emotion 25* (2000), 49–59.

[10] Calvo, R. A., and D'Mello, S. Affect detection: An interdisciplinary review of models, methods, and their applications. *IEEE Transactions on affective computing 1*, 1 (2010), 18–37.

[11] Carroll, A. Offline annotator plugin. `https://github.com/aron/annotator.offline.js`, 2012.

[12] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research 16* (2002), 321–357.

[13] Chi, M. T., and Wylie, R. The icap framework: Linking cognitive engagement to active learning outcomes. *Educational psychologist 49*, 4 (2014), 219–243.

[14] Cohen, J. A coefficient of agreement for nominal scales. *Educational and psychological measurement 20*, 1 (1960), 37–46.

[15] Craig, R. T. Generalization of scott's index of intercoder agreement. *Public Opinion Quarterly 45*, 2 (1981), 260–264.

[16] D'Mello, S., and Graesser, A. Confusion and its dynamics during device comprehension with breakdown scenarios. *Acta psychologica 151C* (06 2014), 106–116.

[17] Freelon, D. Recal oir: Ordinal, interval, and ratio intercoder reliability as a web service. *Int. J. Internet Sci. 8* (06 2013), 10–16.

[18] Github. Electron. `https://github.com/electron/electron`, 2013.

[19] Graesser, A. Learning, thinking, and emoting with discourse technologies. *The American psychologist 66* (11 2011), 746–57.

[20] He, H., Bai, Y., Garcia, E. A., and Li, S. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)* (2008), IEEE, pp. 1322–1328.

[21] Ikonomakis, M., Kotsiantis, S., and Tampakas, V. Text classification using machine learning techniques. *WSEAS transactions on computers 4*, 8 (2005), 966–974.

[22] Joachims, T. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning* (1998), Springer, pp. 137–142.

[23] Johannes Fürnkranz. Decision-tree learning.

[24] Krippendorff, K. Computing krippendorff's alpha-reliability.

[25] Landis, J. R., and Koch, G. G. The measurement of observer agreement for categorical data. *biometrics* (1977), 159–174.

[26] Lee, M. K. Understanding perception of algorithmic decisions: Fairness, trust, and emotion in response to algorithmic management. *Big Data & Society 5*, 1 (2018), 2053951718756684.

[27] Littlewort, G., Whitehill, J., Wu, T., Fasel, I., Frank, M., Movellan, J., and Bartlett, M. The computer expression recognition toolbox (cert). In *Face and gesture 2011* (2011), IEEE, pp. 298–305.

[28] Mayfield, E., Adamson, D., and Rosé, C. Lightside: Researcher's workbench. `http://ankara.lti.cs.cmu.edu/side/download.html`, 2014.

[29] McNamara, D. S., Graesser, A. C., McCarthy, P. M., and Cai, Z. *Automated Evaluation of Text and Discourse with Coh-Metrix*. Cambridge University Press, New York, NY, USA, 2014.

[30] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[31] Pennebaker, J., Francis, M., and Booth, R. Linguistic inquiry and word count (liwc). *LIWC 2001* (01 1999).

[32] Rennie, J. D., Shih, L., Teevan, J., and Karger, D. R. Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of the 20th international conference on machine learning (ICML-03)* (2003), pp. 616–623.

[33] Sebastiani, F. Machine learning in automated text categorization. *ACM computing surveys (CSUR) 34*, 1 (2002), 1–47.

[34] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science 362*, 6419 (2018), 1140–1144.

[35] Tong, S., and Koller, D. Support vector machine active learning with applications to text classification. *Journal of machine learning research 2*, Nov (2001), 45–66.

[36] Venkatesh, V., and Davis, F. D. A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management Science 46*, 2 (2000), 186–204.

[37] Vygotsky, L. S. *Mind in society: The development of higher psychological processes*. Harvard university press, 1980.

[38] Wang, X., Wen, M., and Rosé, C. P. Towards triggering higher-order thinking behaviors in moocs. In *Proceedings of the Sixth International Conference on Learning Analytics & Knowledge* (New York, NY, USA, 2016), LAK '16, ACM, pp. 398–407.

[39] Widner, M. Annotator categories. `https://github.com/PoeticMediaLab/Annotator-Categories`, 2015.

[40] Yang, D., Wen, M., Howley, I., Kraut, R., and Rose, C. Exploring the effect of confusion in discussion forums of massive open online courses. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale* (New York, NY, USA, 2015), L@S '15, ACM, pp. 121–130.

[41] Yogev, E., Gal, K., Karger, D., Facciotti, M. T., and Igo, M. Classifying and visualizing students' cognitive engagement in course readings. In *Proceedings of the Fifth Annual*

ACM Conference on Learning at Scale (New York, NY, USA, 2018), L@S '18, ACM, pp. 52:1–52:10.

[42] Zeng, Z., Chaturvedi, S., and Bhat, S. Learner affect through the looking glass: Characterization and detection of confusion in online courses. In *10th International Conference on Educational Data Mining* (2017).