“Is the Sky Pure Today?”

AwkChecker: An Assistive Tool for Detecting and Correcting Collocation Errors

Taehyun Park, Edward Lank, Pascal Poupart, Michael Terry
David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, ON, Canada, N2L 3G1
{t2park, lank, ppoupart, mterry}@cs.uwaterloo.ca

ABSTRACT
Collocation preferences represent the commonly used expressions, idioms, and word pairings of a language. Because collocation preferences arise from consensus usage, rather than a set of well-defined rules, they must be learned on a case-by-case basis, making them particularly challenging for non-native speakers of a language. To assist non-native speakers with these parts of a language, we developed AwkChecker, the first end-user tool geared toward helping non-native speakers detect and correct collocation errors in their writing. As a user writes, AwkChecker automatically flags collocation errors and suggests replacement expressions that correspond more closely to consensus usage. These suggestions include example usage to help users choose the best candidate. We describe AwkChecker’s interface, its novel methods for detecting collocation errors and suggesting alternatives, and an early study of its use by non-native English speakers at our institution. Collectively, these contributions advance the state of the art in writing aids for non-native speakers.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Human Factors.

Keywords: Collocation errors, non-native speakers, linguistic tools, writing aids.

INTRODUCTION
Non-native speakers (NNSs) of a language can learn a foreign language's rules for spelling and grammar, but a language's commonly used expressions, idioms, and word pairings either cannot be described by rules or require memorization of many special-case rules. For example, a non-native speaker may ask a person to “take their shoes down” while the more common expression is to “take their shoes off.” While the latter phrase is clearly the “correct” phrase for a native speaker (NS) of English, this “correctness” is determined by consensus usage rather than any set of rules related to the English language. This consensus usage is referred to as a *collocation preference*, with violations of these preferences referred to as *collocation errors* [8,10,15,16].

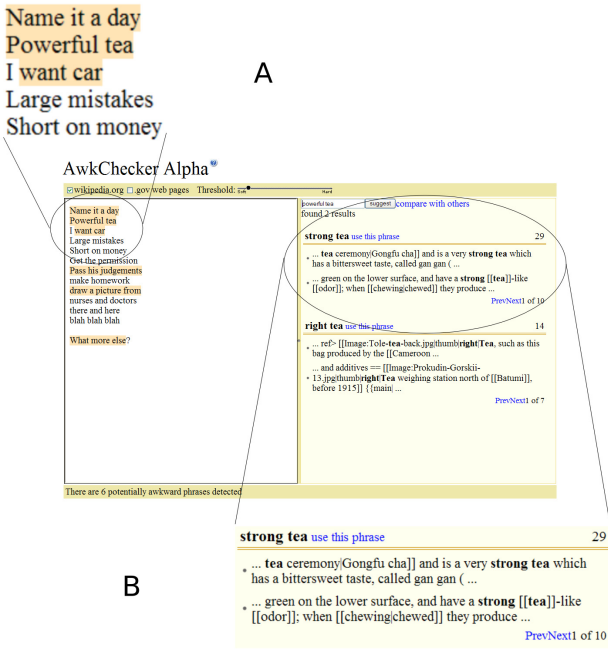

Figure 1: AwkChecker's user interface is depicted with the following callouts: A) flagged phrases in the composition window; and B) One of the suggested alternative phrases for Powerful tea.

The lack of well-defined rules to determine collocation preferences makes it difficult for non-native speakers to detect and correct these errors. Spell checkers and grammar checkers can help ensure one's language is syntactically and grammatically correct, but these *rigid-language tools* offer no assistance in detecting or correcting collocation errors. Instead, non-native speakers must employ ad-hoc methods to detect these types of errors. For example, many NNSs check whether a phrase is commonly used by observing the number of results returned by search engines [9,19]. While a clever reappropriation of search engines, there are obvious limitations to this approach compared to dedicated linguistic tools. For example, search engines are

UIST’08, October 19-22, 2008, Monterey, California, USA.

of limited help when a NNS must find an alternative phrase that is more commonly used than their original phrase.

The reappropriation of search engines indicates a clear need for *soft-language tools,* tools geared toward aiding NNSs with the common expressions, idioms and word pairings that comprise collocation preferences. In this paper, we present a new tool, AwkChecker, which allows NNSs to identify and correct collocation errors in their own writing. AwkChecker uses word-level statistical n-grams and incorporates algorithms to suggest corrections for four common types of collocation errors: insertion, deletion, substitution, and transposition errors.

AwkChecker is implemented within the context of a web-based text editor that flags collocation errors and suggests alternatives from which a user can select (Figure 1). As shown, when users edit text within the window, phrases that are potentially "awkward" are highlighted by the interface (Figure 1a). Users can click on awkward phrases to generate a list of suggested alternatives (Figure 1b). This list of alternatives includes information about the relative ubiquity of each alternative within the training corpus. It also includes example uses for each phrase, showing the phrase in context with surrounding text. Users can click on "Next" links to see additional examples of the phrase, thus allowing them to assess the quality of each alternative. Because collocation preferences are highly idiosyncratic, showing phrases from the corpus in context can be crucial in helping users determine which phrase is most appropriate given the context of their writing.

The use of n-gram statistics confers a number of benefits, both from the perspective of detecting collocation errors and from the perspective of an end-user who must understand the tool's capabilities and limitations when applying the tool. First, because the statistical n-grams are extracted from an underlying corpus, the system flags collocation relative to the style of writing within the corpus. As a result, the tool can be used by NNSs to write in the style appropriate for different writing tasks, provided a relevant corpus exists. An example would be using a medical corpus to flag phrases inappropriate within medical research writing. Second, the use of n-gram statistics results in resilience to "messy" data. As one of our corpora, we use Wikipedia entries, which vary widely in the overall quality of writing. However, provided the corpus is "correct" more frequently than it is "incorrect", collocation errors will be detected and appropriate corrections suggested. Finally, because AwkChecker's likelihood estimate is a function of an underlying corpus, end-users can more easily understand the system's capabilities and limitations. This phenomenon was obvious during thinkaloud evaluations where users commented on their improved understanding of false positives (phrases that were flagged as collocation errors but were not).

There are three primary contributions of this work. First, this work defines a new mechanism for identifying collocation errors, defining collocation errors as a function of the relative frequency of phrase usage within a corpus. Next, this work presents algorithms for suggesting alternatives, based on the specific types of errors made by NNSs. Finally, the web-based interface represents, to the best of our knowledge, the first tool that allows end-users to identify and correct their own collocation errors. Together, these contributions result in a significant advance in tools to aid NNSs in written composition.

BACKGROUND

Second language (L2) refers to a language that is acquired after first language (L1). For many, second language learning is a critical component of their everyday lives. For example, it is estimated that about 70% of the English speaking population consists of non-native speakers [4]. However, few linguistic tools exist that are specifically designed to scaffold language production by second language learners.

In this section, we describe needs unique to non-native speakers when attempting to produce "correct" language in a second language, and contrast these needs with native speakers' needs. We also consider the linguistic tools currently available (e.g., spell checkers, grammar checkers) and argue that the majority of linguistic tools currently available primarily cater to the needs of native speakers and the types of errors they produce, with the unique needs of NNSs largely unmet. We review research efforts focused on linguistic aids for NNSs and identify an opportunity for tools that assist in the detection and correction of one common type of NNS error, collocation errors.

Language Needs for Non-Native Speakers

As with L1 acquisition, L2 acquisition is a complex process, the nuances of which are still being discovered. However, there are some distinct characteristics of L2 acquisition that are noteworthy, especially in the context of computational tools intended to support language production by nonnative speakers.

Grammar and essential vocabulary are the first parts of a language acquired by NNSs. While this acquisition requires a fair amount of rote memorization (e.g., learning vocabulary), there are also well-known rules associated with a language's grammar (e.g. noun-verb agreement) and spelling ("i before e, except after c"). Once learned, these rules help non-native speakers produce new language that is syntactically and grammatically correct.

Despite knowledge of rules of grammar and spelling, non-native speakers continue to make a number of language errors. These errors include those involving determiners (e.g. a, an, the), prepositions (e.g., to, for, in), and noun-verb agreement. Some of these errors involve a language's grammar rules (e.g., noun-verb agreement) and can be detected by tools such as grammar checkers. Others require grammar and language tools geared specifically to the needs of NNSs. Finally, a third class of errors exist that result from collocation preferences.

Collocation preferences refer to habitual word combinations in a language. For example, in English, one can say there is a "clear sky" to express the notion that there are no clouds in the sky. However, one would not say

"clean sky" or "pure sky," even though these phrases, at a basic, fundamental level, convey a similar sentiment to "clear sky." When trying to convey this concept, second language learners cannot know which of the phrases is the "correct," more common phrase, except through experience. That is, there are no rules that dictate that "clear" is the more appropriate adjective in this circumstance. Collocations are thus idiosyncratic in nature since they lack predictable syntactic and semantic features that one can learn and apply.

Collocation conventions lead to a distinct class of errors, *collocation errors*, which are produced by NNSs, but not NSs. For example, in the Chinese-English Learner Error Corpus (CLEC), approximately 30% of English as a Second Language (ESL) writing errors involve different types of collocation errors [19]. Some experimental evidence shows that even advanced NNSs have difficulties with collocation [16]. As such, collocation errors constitute a significant class of errors produced by non-native speakers. However, while it can be difficult for non-native speakers to produce language with correct collocations, they have little difficulty in understanding the meaning of phrases when encountered. As we will show, the fact that collocations can be understood when encountered has implications for computational aids.

In contrast to the errors described above, *native* speakers tend to make relatively simple mechanical errors (e.g. "then" vs. "than," "its" vs. "it's," etc.). These are errors that non-native speakers may also make, but are clearly of a different nature than the types of errors unique to NNSs. For convenience, we refer to errors produced by NNSs, but not NSs, as *L2 errors*. We consider *L1 errors* as those errors common to NSs, even though NNSs may also produce these types of errors. Table 1 summarizes these two classes of errors.

	L1 Errors	L2 Errors
Spelling	Homonyms (e.g., then, than)	Any, including L1
Grammar	Run-on-sentences	Verb-noun agreement Determiner usage
Style	Rare	Use of informal vocabulary
Collocation	Rare	Various

Table 1: A (non-exhaustive) list of common language errors made by native speakers (L1 errors) and non-native speakers (L2 errors). Non-native speakers may make either type of error, but native speakers rarely make L2 errors.

Given these two primary classes of language errors, we turn now to a review of linguistic aids.

Linguistic Aids

A wide range of computational tools have been developed to aid in language production, with spell checkers, grammar checkers, and translation services the most common. A number of tutoring (educational) systems have also been developed to scaffold language learning, though these systems are, by nature, not general-purpose tools one can use while writing.

In considering the two general classes of language errors defined above, it is clear that existing linguistic tools are of most use for addressing L1 errors. Spell checkers and grammar checkers help one produce grammatically correct language, but do not help detect and correct the collocation errors non-native speakers are prone to make. Thus, these tools are useful to both NS and NNS, but do not address L2 errors. As these tools allow detection of violations of linguistic *rules*, we call these tools *rigid-language tools*.

It has been frequently noted that the rigid-language tools designed to detect L1 errors do not effectively address many L2 errors [7]. To help detect L2 errors, many nonnative speakers rely on web search engines, especially when checking for collocation errors [9,19]. To check for a collocation error, a NNS will submit a phrase to a search engine. If the phrase returns few "hits," the NNS can assume something about the phrase is incorrect. At this point, the NNS can generate additional phrases to test until an acceptable candidate is found. While this use of search engines clearly deviates from their intended use, it provides an important stopgap for NNSs in need of L2 error detection. However, there are limitations: Search engines are not tightly integrated with writing tools, users must manually produce alternative phrases to test, and most search engines provide only rudimentary suggestions when a phrase contains an L2 error.

Noting the limited utility of existing linguistic aids for non-native speakers, the research community has begun to actively investigate tools to address this problem space. Research has focused on the development of a set of guidelines for the design of L2 error detection tools, the development of techniques to detect and automatically correct L2 errors in written documents, and learning aids for NNSs.

L2 Tool Design Guidelines

In their work developing a Swedish grammar checker, Knutsson *et al* studied adoption practices of their grammar checker by non-native speakers [12,13]. Based on their study, they developed a set of guidelines for the design of linguistic tools for NNSs. In particular, they suggest:

- Real-time feedback is always desirable, especially since it helps one improve one's understanding of the language as it is produced
- Tools should not only indicate what is wrong, but also provide sufficient information (e.g., examples, grammar rules, etc.) so that users can reason about the error and its solution
- The tool should be transparent with respect to its capabilities and limitations; users should understand what it can and cannot do
- The tool should not be too technical with its terminology and should avoid linguistic terms

- Users should be able to focus on producing content, not on low-level details such as spelling, grammar, etc. That is, the tool should not distract from their primary goal of communication

These proved useful for a subsequent Swedish grammar checker designed specifically for NNSs. In a study involving a class of students learning Swedish with their NNS grammar tool, the researchers found that deviations from these guidelines noticeably reduced the tool's effectiveness. The strongest conclusion from their study is that a user's understanding of the limitations of the tool resulted in fewer questions about why the system failed to detect an obvious error or why it flagged a statement that was clearly correct. As well, providing additional information about the error to the end-user proved helpful in judging false positives.

L2 Error Detection Tools

As noted earlier, L2 errors include both grammar errors (e.g. determiners and prepositions) and collocation errors. Recognizing that L2 errors are distinct from L1 errors, a number of research systems have attempted to perform automatic L2 grammar error checking (e.g. for determiners, prepositions, etc.) [2,5,7,19].

One common approach taken by these systems is to use a large reference corpus to develop L2 grammar rules, essentially a model-based approach to error detection and correction. Eeg-Olofssons *et al* created a rule-based approach to grammar checking geared specifically to L2 errors made by NNS of Swedish [5]. They describe two types of rules: word and phrase form errors, and preposition errors. They performed a minor evaluation of their system on a 2800 word text, manually flagging 40 errors. The system detected 11 of these 40 errors (recall = 28%), and resulted in no false positives (precision = 100%).

More recently, Gamon *et al* have used machine learning algorithms [7] to create rules for detecting eight different L2 errors they specify. They report detection and correction accuracy for their system of 46% for preposition errors and 55% for determiner errors. Performance on the other error types has not yet been reported. Brockett *et al* have applied a statistical machine translation (SMT) technique to detect and correct countability errors associated with mass nouns (i.e. errors involving uncountable nouns such as information, pollution and homework) and report correction accuracy of 61.81% [2]. A strength of the model-based approach is the ability to handle arbitrary input. The systems label parts-of-speech (e.g. noun, verb, verb-phrase, etc.) and then use their model to validate the structure of the fragments. However, these models can over-generalize, producing frequent false negatives that are not understandable by users.

In contrast to model-based approaches, Yi *et al* have used web search hits from search engines to detect and correct determiner and preposition errors [19]. Their system detects verb-noun collocations and determiner errors. The system parses a corpus searching for sentences that are of two forms: verb-phrase, noun-phrase; and verb-phrase, prepositional-phrase; noun-phrase. It also identifies all noun phrases. The system then uses a web search engine to flag errors and suggest replacements. Their system identifies determiner errors 41% of the time, and provides a correct replacement in 63% of these cases. For collocations, however, the system's performance is much poorer (recall = 31%, precision = 37%).

The above L2 error detection tools have been valuable in improving the state-of-the-art in L2 error detection. However, they have typically focused on a sub-set of L2 errors, specifically those errors that can be modeled by grammar rules. These approaches will not detect errors that result from collocation errors.

Furthermore, because the goal of these L2 error detection tools has been to improve the recognition of a certain subset of L2 errors, these tools have not been embedded in end-user systems. For end-user linguistic tools dealing with fuzzily-defined linguistic conventions (such as collocation preferences), the properties of model-based approaches may be problematic. In particular, using the desiderata of Knutsson *et al* [13] and Vernon [20], models may serve to decrease the transparency of the system, making it difficult for users to form a mental model that can account for failures of the system, such as false positives and poor suggested corrections. As these techniques have not been realized in end-user tools, these remain important open questions to investigate.

L2 Tutoring Systems

One area where linguistic tools have been developed for non-native speakers has been in computer-assisted language learning (CALL). While most CALL systems contain a series of pre-specified lessons to support the learning of vocabulary and grammar, rather than collocation preferences, one exception to this general trend is a system proposed by Shei *et al.* to teach collocation preferences [18]. The system will first extract collocations from a reference corpus, a large, correct, collection of English text. The system will also extract errors from a learner corpus, a corpus containing multiple instances of collocation errors. The system will then select collocation errors from the learner corpus, and NNS can work to correct the collocation errors. As well, given short phrases typed by the NNS, the system will determine whether these phrases are collocation errors or not. While Shei et al. have performed some proof-of-concept testing of their system's techniques, their primary interest is in pedagogical issues in training professional translators [17]. The proof-of-concept testing on their proposed system served to inform practices for translator training, but the system was not deployed for end-users.

One shortcoming of tutoring systems is that their overall goal, language instruction, typically results in an interaction style unsuited to aiding arbitrary writing. Tutoring systems consider short passages of text and flag errors. The systems require users to perform corrections immediately, thus scaffolding the learning process. This short, phrase-based interaction is useful for improving one's writing. However, when composing a document, the need to interact phrase-

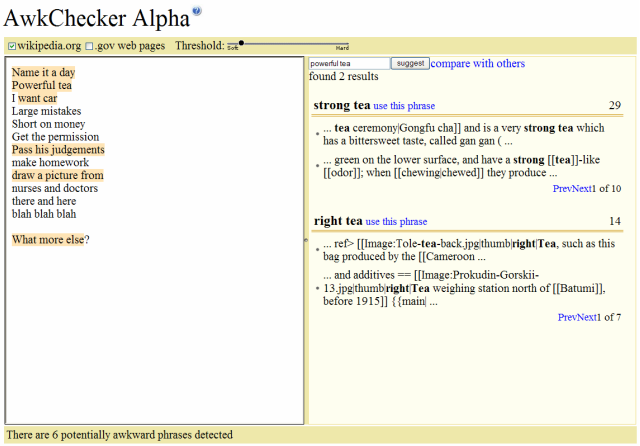

Figure 2: AwkChecker's user interface.

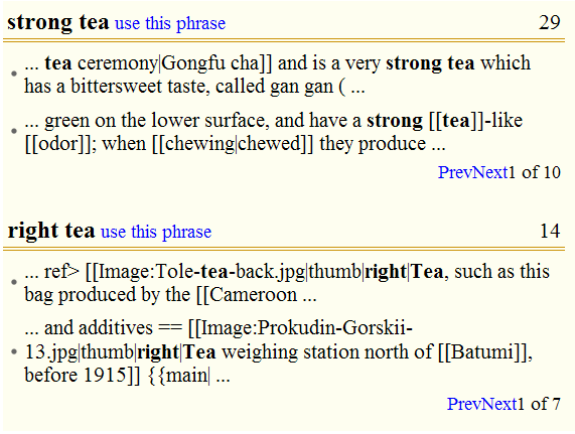

Figure 3: An enlarged view of the suggestion panel.

by-phrase with the system can impede the user's writing task.

Summarizing L2 Learner Needs

To summarize, NNSs make a set of errors, L2 errors, which NSs rarely make. These errors include errors involving determiners, prepositions, noun-verb agreement, and collocation errors. In this paper, we are most concerned with collocation errors, an L2 error that has received little attention in the past. To aid NNSs in detecting and correcting errors, linguistic tools for NNS should provide real-time feedback, information to support reasoning about errors, a high degree of transparency with respect to the tool's functionality and capabilities, and little reliance on specialized terminology [13,20]. We turn now to a system we built to detect collocation errors while following these general guidelines.

AWKCHECKER

System Design: Overview

AwkChecker is a text editor that performs real-time detection of collocation errors (Figure 2). As the user enters and edits text, AwkChecker automatically highlights potential collocation errors. Users can click on highlighted phrases to receive a list of alternative suggestions.

AwkChecker's suggestion list includes a relative ranking of the frequency of each phrase in the corpus. We

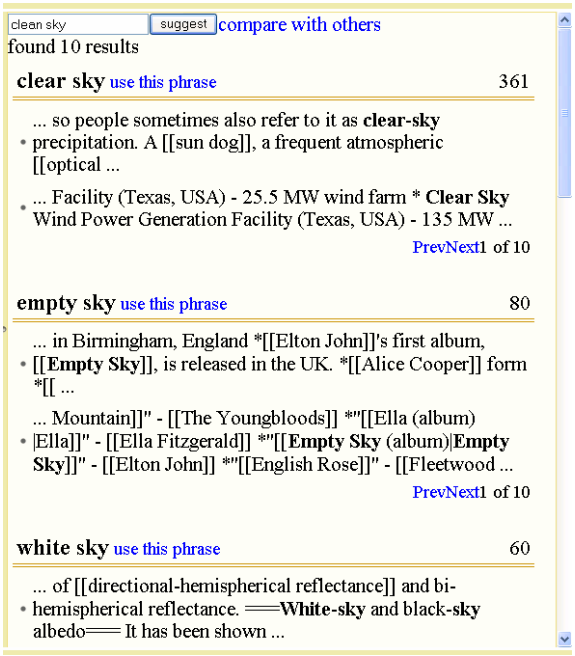

Figure 4: Alternative suggestions for the phrase "clean sky".

experimented with a variety of feedback mechanisms, including a normalized frequency in the range [0, 1], a bar graph representation where the length of bars indicated the frequency of the phrase, and a combination of some numerical score and graphical representation. While it was our goal to allow users to quickly perceive relative scores, these numerical scores became a source of confusion, as the numerical scores did not map intuitively onto an obvious measure of frequency. Currently, a phrase's score is depicted as an integer value that corresponds to the number of occurrences of the phrase in the corpus (Figure 3).

Each suggested alternative is shown in the context of a passage of text from the corpus. The goal of this short passage is to help users assess the best replacement when multiple alternatives exist. As well, for users who wish to improve their language skills, seeing a series of alternative phrases in context can provide them with examples of proper use of the phrases. The short passages are drawn at random from the corpus, and occasionally the alternative phrases are inappropriate. They may, themselves, be examples of collocations, or they may not have the same meaning as the original phrase. To address this, users can press a "Next" link to see additional examples of the specific phrase in context (Figure 3).

When correcting awkward phrases, the need for additional information (rankings, example usage) will naturally vary from user to user: Some users will be experienced enough in a language that they merely need to see a phrase to know it is the desired phrase. For example, they may have seen the phrase before and were unable to recall the phrase, or their command of the language is great enough that they can judge a phrase's "correctness" by simply reading it. Other users will need to examine the context surrounding

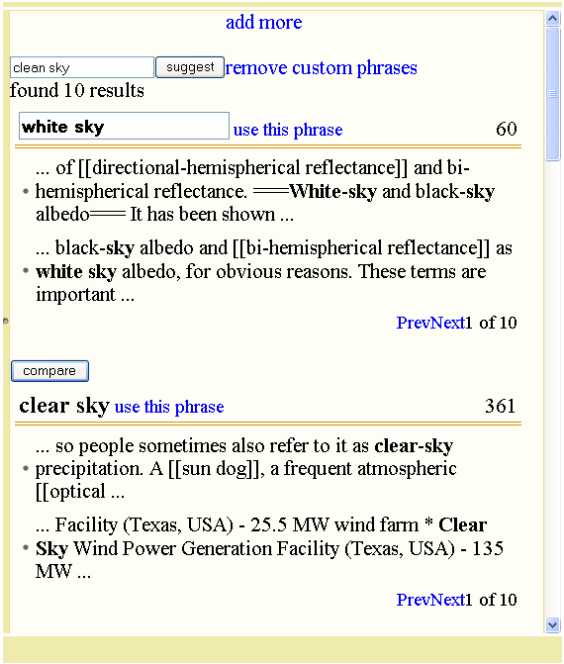

Figure 5: Comparing "clean sky" to "white sky".

phrases to make an informed decision. We refer to these two needs as *recognition vs. education*. Existing linguistic tools typically only support recognition and rarely support the means for end-users to educate themselves about alternatives. Again, since collocation preferences do not conform to rules, but reflect consensus usage, this type of information can be critical for end-users to make effective use of the tool.

At the top of the suggestion list, a text field is supplied to enter a short phrase when one wishes to receive a list of suggested alternatives for a specific phrase (Figure 4). If a user suspects that another phrase may be a better alternative, they can verify this by following the "compare with others" link. A new text field is presented, and the user can enter the second phrase for comparison. This new phrase is located in the corpus, a frequency score is presented, and the user can use contextual information to determine if this phrase is more correct. This comparison feature is shown in Figure 5.

Finally, above the composition area, two selectable options can be used to control the behavior of AwkChecker. These two additional features are shown in Figure 6. First, users can select from available underlying corpora. Currently, our prototype interface includes the Wikipedia corpus and ".gov" web pages. Users can also customize a threshold for collocation error detection. A higher threshold improves recall (more collocation errors are identified), but negatively impacts precision (more false positives occur). Depending on their tolerance for false negatives, users can customize this threshold for their desired system behavior.

Error Detection and Correction: Implementation Details

As noted earlier, to support the detection of collocation errors and the generation of alternative suggestions, AwkChecker uses statistical word-level n-grams. N-grams

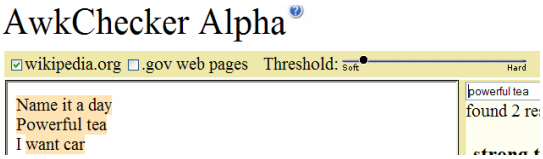

Figure 6: Selecting from available corpora and configuring the threshold for flagging collocation errors.

can be viewed as an n-dimensional table of probabilities. The value stored in any entry in the table represents the likelihood that one would randomly observe the string of n words represented by the entry.

To help NNS detect and correct collocation errors in their writing, there are three tasks that are realized by AwkChecker's back-end algorithms. First, AwkChecker includes a training interface which analyzes a corpus and builds a set of statistical n-grams. Second, AwkChecker analyzes text input by the user against the corpus, now organized as an n-gram dictionary, to determine whether any phrase is a collocation error. Finally, if a phrase is identified as a collocation error, AwkChecker generates a list of alternatives for a phrase.

Analyzing a Corpus

Detecting and correcting collocation preferences shares some similarities with spell checking. In spell checkers, words are compared to a dictionary. If the word is found, it passes and remains unflagged. However, if the word is not present in the dictionary, spell checkers suggest a list of alternatives, typically by measuring the edit distance between the typed word and dictionary words using a function called the Levenshtein distance. While we make use of a similar approach in our L2 error detector, unlike spell checking, no dictionary exists for collocation preferences. Thus, a dictionary must first be constructed before we can detect or correct awkwardness.

AwkChecker builds a dictionary of n-grams (sequences of words) from a given corpus and records frequencies of each sequence within the underlying corpus. At present, AwkChecker builds a dictionary of 2-5 word phrases. All 2-5 word phrases contained in the corpus have associated frequencies in the dictionary. While n-grams are not a novel technique, L2 language error detection and correction typically use more complex linguistic models such as decision trees, statistical machine translators, and others. To date, we are unaware of any systems that use n-grams as their linguistic model for detecting collocation errors.

Algorithmic Basis for Detecting Collocation Errors and Suggesting Alternatives

In this section, we describe the basis of our technique for calculating an acceptability metric, which we use to determine whether a phrase is a collocation error. This acceptability metric is also used to create a list of alternative phrases if the phrase being analyzed is classified as a collocation error.

To detect collocation errors and to find corrections, the system compares the frequency of any end-user input

expression (typed into AwkChecker's text area) of length 2 to 5 to "similar" expressions within the dictionary. A collocation error is indicated if there exist similar phrases with much higher frequency than our input phrase. Similarly, alternatives are suggested by examining phrases that are similar to the phrase typed by the user for more likely candidates. Here we focus on the mechanism for calculating a collocation-error score. We will then discuss detecting an error and suggesting alternatives.

Given an input expression $e = w_1\ w_2\ ...\ w_n$, we want to find similar expressions that have the highest probability of being more acceptable than the current input expression. Mathematically, we want to find an expression E^*, a close derivative of e, such that:

$$E^* = \arg\max_c P(c|e) = \arg\max_c P(e|c)P(c)$$

In this expression, *c* represents a candidate phrase within the dictionary, $P(c|e)$ is the probability of candidate phrase c being the correct phrase given that e was input by the user. We iterate over all candidates in the dictionary that are close derivatives of *e* until we find the most likely candidate expression. Using a Bayesian relationship, we model this probability using $P(e|c)\ P(c)$, where $P(e|c)$ is the probability of transforming phrase *e* into phrase *c*, and $P(c)$ is the probability of candidate phrase *c*. $P(e|c)$ and $P(c)$ are referred to as the *error model* (EM) and *language model* (LM), respectively. Essentially, the above equation states that if the user typed expression *e*, then we should flag it as a collocation error if there is another expression, *c*, that it seems the user should have typed. E^* is the most likely of all candidate expressions considered.

Our identification of candidate phrase E^*, is inspired by the Bayesian model described above. We use two functions, $f(e,c)$ and $g(c)$, to identify E^*.

$$E^* = \arg\max_c f(e,c)g(c)$$

Presently, $g(c)$ is the frequency of phrase *c* in the corpus. The error function, $f(e,c)$ is an analytically derived function based on the edit distance between *e* and *c*. We use the Levenshtein distance as a measure of edit distance, and assume that candidates within edit distance 1 are more probable than candidates at edit distance 2. Given the relatively short n-grams used, we do not consider edit distances greater than 2. We also assume that first and last words are unlikely to be a result of an insertion error, and that article/preposition deletion and substitution errors are more likely to occur than other types of collocation errors.

Detecting Collocation Errors

Collocation errors are somewhat unique from L1 errors (and many L2 errors) in that there is no definitively "correct" phrase. There are only degrees of acceptability for any given phrase. We, therefore, define a function that represents the acceptability of a phrase *e* as follows:

$$A(e) = g(e) - \max_c f(e,c)g(c)$$

Given a phrase *e*, the acceptability of *e* heavily depends on the actual usage frequencies. However, if there are better alternatives, the likelihood of *e* being awkward increases. The function $A(e)$ captures these factors by comparing the frequency of phrase *e*, $g(e)$, to the product of the cost of transforming *e* into *c*, $f(e,c)$ and the frequency of *c*, $g(c)$, for the best alternative phrase in the corpus. If $A(e)$ is less than a user-customizable threshold, the phrase *e* is flagged as a collocation error. To efficiently compute $A(e)$, we employ a search engine (inverted index), the Wumpus Information Retrieval System [3]. A $A(e)$ calculation normally takes less than a millisecond.

Suggesting Alternatives for Flagged Phrases

Correction of an awkward phrase requires a candidate list of alternative phrases to be created. As noted above, we use the Levenshtein distance metric to generate a list of candidates. We apply this metric by first recognizing that a NNS can introduce four different types of errors into a phrase: insertion, deletion, transposition, and substitution errors. We then apply inverse error transforms to the phrase to create a set of alternative phrases which are then ranked according to our acceptability metric.

We refer to the possible L2 errors that result in collocation errors as *error transformations*. *Insertion errors* insert a word in the phrase. For example, the phrase "I went to home" is a collocation error because the preposition "to" is inserted. This type of error transformation is often associated with prepositions and articles. *Deletion errors* occur when a word is deleted from a phrase. This error is commonly associated with articles. For example, in the phrase "I am student," the article "a" is missing. *Transposition errors* occur when two words are swapped. For example, "he's talking with his full mouth" is a transposition error since the phrase should be "he's talking with his mouth full." Finally, *substitution* or *alternation errors* occur when a non-preferred word is used in place of a more commonly used word. Substitution errors frequently result in collocation errors. For example, "make homework" should be "do homework," and "clean sky" should be "clear sky." Given these four types of error transforms, we apply a set of inverse error transforms to generate our candidate list of alternative expressions and rank these alternatives according to the error and language models, $f(e,c)$ and $g(c)$, for each phrase.

Modeling insertion and transposition errors is reasonably simple. For any phrase, we can select individual candidates to delete or transpose to perform inverse transformations of insertion and transposition errors. However, precisely modeling substitution and deletion errors is more challenging. The inverse transformations for these errors require the insertion of words, and without some error model, heuristics must be employed to limit the set of words considered for the inverse transformations. For these two error types, we insert or substitute prepositions and articles (preposition and determiner errors constitute about 12% of ESL errors [7]), synonyms (from WordNet), allowed verb forms, and singular and plural forms of nouns. Any error type can occur multiple times within a

phrase. As a result, we apply inverse transformations up to our limit of two errors per phrase.

EVALUATION

The design of AwkChecker is the result of an iterative design process involving formative testing by non-native speakers. The system was tested by five non-native speakers. To speed evaluation, three of the participants were given an essay written by a non-native speaker to edit, while two edited their own content. Testing concluded with semi-structured interviews on the design of the interface, the features, and the usefulness of AwkChecker.

As noted in the Introduction, AwkChecker is a type of tool we refer to as a *soft-language tool*. Our study revealed a number of insights into how end-users perceive, use, and desire to use this soft-language tool. We describe our findings and implications for the design of tools, like AwkChecker, that support soft-language constraints.

Perceptions of a Soft-Language Tool

As we have argued, the detection and correction of collocation errors is qualitatively different than that of spell or grammar checking. In our user studies, we found that the inherent fuzziness of this component of language required care in how the tool was positioned as well as how certain aspects of its functionality were exposed to users (in particular, the presentation of suggested alternatives).

As users have never encountered a soft-linguistic tool, we tried several ways of describing what it can do. At first, we described it simply as a tool that detects "awkwardness" in one's speech. However, users found this description vague and needed more information and context. We also described its functionality as being similar to using a search engine to check whether one's speech conforms to standard conventions. Users understood this concept, but did not immediately understand how AwkChecker could improve upon this ad-hoc method until they encountered an awkward phrase ("powerful tea") that they could not fix with a search engine alone.

Eventually, we found that explaining the system as a "dictionary of expressions," constructed from an underlying corpus, was the best way to position the system. Furthermore, by indicating that this dictionary of expressions is built on an underlying corpus, users attributed false positives to the underlying corpus rather than any shortcoming of the tool itself. That is, they were able to build a mental model of the system, its functionality, and limitations through this description compared to others.

While users were able to grasp the basic functionality of the system, AwkChecker's presentation of suggested alternatives posed some problems. Originally, AwkChecker displayed the calculated goodness metrics for each phrase, but users were not able to understand what this number was, nor its meaning relative to other scores. We then switched to showing the number of occurrences in the underlying corpus. This was more easily understood, but still required explanation. This particular issue – how to represent varying levels of confidence in a suggestion – remains a problem that deserves further investigation so that end-users can understand, on their own, the notion that the tool is not authoritative, but a *guide*.

Patterns of Use and Desired Uses

One of our users, U1, used the system continuously for a week. The user would have used it as their primary text editor, but the web-based editor lacks undo and other features typical of a basic word processor. Despite these limitations, the user developed regular patterns of usages that are noteworthy.

U1's first language is Korean, which does not make use of articles. Accordingly, U1 employed AwkChecker to check articles and prepositions. U1 developed one interesting workaround which suggests a potential design modification in the future. In particular, when checking prepositions, U1 would often want to check incomplete phrases. For example, in one instance, U1 wanted to check the phrase, "pass judgment" where they were unsure whether the preposition should be "to" or "on" before an object noun. Rather than define the noun, U1 wanted to specify a pattern "pass judgment to <noun>," rather than specifying a noun. As a workaround, U1 replaced the nouns with "dummy" articles to compel the system to provide a suggestion.

When correcting their text, users either used *recognition* or *education* to choose the replacement phrase. In the case of recognition, users relied on past knowledge to choose the replacement phrase. For example, they may have previously encountered the correct phrase, but could not recall it, or they could judge the correctness of a phrase based on how it "sounded." In the case of education, the user needed to educate themselves on the various phrases by reading the provided examples. In some cases, a dictionary was required as part of this process because the alternatives contained new vocabulary to the user. This finding indicates that supplying a dictionary, accessible from any part of the system, would be useful.

Notably, when used to educate oneself on new phrases, AwkChecker is extending one's capabilities with the language, something not possible with linguistic aids such as spell checkers or grammar checkers. It has been found that NNSs frequently use an "avoidance" strategy when producing L2, using only words and structures with which they are confident, while avoiding unfamiliar forms [6]. AwkChecker provides a means by which users can safely leave their "comfort zone" to push beyond their current skill set.

The system was found to be useful for correcting direct translations from one's native language. Since collocation preferences generally do not transfer between languages, this is a natural and perfect use of the system.

Unused Functionality

The original incarnation of AwkChecker contained two additional features that were not used by participants. A comparison tool was provided to directly compare the frequency of phrases in the corpus. This functionality was designed to allow people to investigate two or more phrases in parallel when the alternatives were known *a priori*.

However, users seemed satisfied with the basic functionality of the system and cited the need to manually enter phrases in text boxes as a deterrent to using this comparison tool. Furthermore, most of this tool's functionality is supported at a basic level by the default detection algorithm; it is most useful in cases where two or more phrases are highly distinct to the extent that the inverse transformations on one would not yield the other phrases.

A third tool, a phrase analyzer, was also constructed to help users pinpoint the likely cause of awkwardness in their phrase. This tool subdivides the phrase into sub-phrases and rates each sub-phrase. However, users did not seem to need this additional information; they either wanted to recognize the correct phrase or educate themselves about which is the best to choose.

DISCUSSION

AwkChecker determines collocation errors through the use of n-gram statistics and an underlying corpus. As mentioned above, in our current implementation, users can choose to use either the Wikipedia or Gov2 corpora as their backing corpus.

As we have argued, the choice of collocation error technique can have important implications for actual end-user tool use. We consider three implications related to the use of n-gram statistics and corpora to detect collocation errors: Its robustness to "messy" data within a learning corpus, its ability to allow users to generate reasonably accurate mental models of how the system operates, and its ability to not only detect collocation errors, but help one to conform to particular styles of writing by choosing an appropriate corpus.

Considering Messy Data

Any corpus will contain errors, and the corpora used by AwkChecker are no exception. For example, Wikipedia entries vary widely in the overall quality of writing. However, AwkChecker's use of n-gram statistics, relative rankings of results, and example uses for each suggestion provide a graceful way to handle errors in the underlying corpus. Examining the nature of errors in a corpus makes these points clear: If errors are not *consistently* made for collocation preferences, then the relative frequency of any particular collocation error *in the underlying corpus* will be significantly less than the consensus usage for a collocation preference. What this means is that collocation errors in the underlying text are unlikely to lead to false negatives when checking a user's text. By the same token, these source errors should not, in most cases, appear in the suggestion list. However, even if they do, the relative rankings supplied with each suggestion, along with its example context of use, will help users to discard these examples. Thus, messy data in a corpus is not a significant concern for this type of linguistic aid.

Detecting System Failures

When attempting to detect and correct nuanced language errors such as collocation errors, one must be sensitive to the fact that the absence of well-defined linguistic rules means any system that attempts to detect and correct these types of errors will be prone to false negatives, false positives, and poor correction suggestions; there will be no "perfect" system. What becomes important, then, is ensuring end-users themselves can detect these failings of the underlying system and reasonably cope with them. Relating this to Knutsson's and Vernon's guidelines, there is a need for *transparency* in the system's design.

There are two important implications related to this specific form of transparency (i.e., system failures). First, this high-level need means the choice of underlying algorithms is not without consequence for end-user interaction. In particular, given the choice between two methods of detecting and correcting collocation errors, it may be more desirable to choose the method whose basic functionality end-users can more easily understand. In this case, when the system fails, they will be in a better position to understand how and why it failed, and thus recover from that failure. As we noted in the Background section, there are a number of techniques developed to detect L2 errors, but none have been assessed with respect to this end-user need. That is, none have considered the direct link between the underlying detection algorithms and the user's ability to effectively make use of those algorithms.

The second implication for supporting transparency is that the interface should provide sufficient information so that users can detect and recover from failure. Again, because any such tool will be imperfect, users should be able to determine why a system appears to be making an error in either the detection or correction process.

AwkChecker's use of n-gram statistics helps address both transparency goals. First, basing collocation error detection on consensus usage rather than derived models helps users develop reasonably accurate mental models of AwkChecker's capabilities and limitations. As we found in our user evaluation, once users understand that error detection is based on the notion of the frequency of a phrase in a corpus, they can more easily account for mistakes the system may make when flagging awkward phrases or making suggestions. In contrast, if a model-based approach were used, it would be more difficult to understand false positives generated by an overgeneralized model. Similarly, model-based approaches could lead to suggested phrases being synthesized that are themselves examples of collocation errors. Both types of system failures are potentially more difficult to understand and explain with model-based approaches than AwkChecker's use of n-gram statistics.

Second, as mentioned above, displaying relative rankings and example context for suggestions can also assist users when recovering from suspected failures of the system. The rankings and context guide improve the reliability of end-user corrections and help to identify false positives.

Writing Styles

While the primary goal of AwkChecker is to detect collocation errors that NNSs make, AwkChecker can also be used as a *style checker*. In considering collocation preferences, one can create a spectrum of collocation errors

ranging from awkward phrasing that no NNS would produce, to phrases that are acceptable, but not the preferred phrases, within a particular domain. For example, in describing medical stitches, a surgeon would tend to say "sutures" while a lay person would instead say "stitches." Both are acceptable, but one is the more preferred word in certain contexts. In this case, the collocation error is not one of awkwardness as much as it is one of style.

AwkChecker's approach to detecting collocation errors provides a degree of flexibility and customizability in catering one's language to a desired style of writing. While we currently provide a choice of only two corpora, one could easily generate *sets* of corpora representing a range of writing styles, from scientific writing to literature reviews to urban street slang. One need only obtain a reasonably sized corpus, have AwkChecker index it, and then use that as the primary corpus to check consensus usage.

CONCLUSION AND FUTURE WORK

AwkChecker is the first end-user tool specifically designed to support the detection and correction of collocation errors, a type of error common to NNSs. Collocation is unpredictable using low-level linguistic features such as syntax and grammar, and it is thus difficult to automatically correct collocation errors.

In user testing, we observed very positive reactions to our system. Our participants, all NNSs, had never seen tools such as ours before. As well, our experiments with various techniques for describing the system to users demonstrated that when users understand the characteristics of the underlying corpus and the mechanism used to flag collocation errors, they are better able to predict the limitations of our system, and have a better understanding of false positives and false negatives.

One feature that was desired by one of the participants was some mechanism for part-of-speech tagging, for example that ability to specify a word class as one part of a phrase. As a result of these observations, we may extend AwkChecker with part-of-speech tags for correction. We are also considering developing plug-ins for commercial word-processors.

While the development of tools for non-native speakers is an active area of research, much of the work has been geared toward determiner, preposition, and other grammatical components of speech which can be specified with rules. As well, most systems are designed to test detection and automatic correction algorithms, rather than as end-user tools to aid non-native speakers with their own writing.

While addressing these grammatical components of speech for non-native speakers is an important research goal, equally (or more [16]) problematic for people acquiring a second language are the soft constraints, the collocations, idioms, and common usages that are based on social consensus, not on syntactic and grammatical rules. In this work, we describe a new technique, based on statistical word-level n-grams, to detect collocation errors and suggest alternative phrases. As well, we incorporate our algorithms into a tool, AwkChecker, that helps end-users follow these linguistic conventions in their own writing.

ACKNOWLEDGMENTS

The authors would like to thank the participants in our research study. Funding for this research was provided by the Natural Science and Engineering Research Council of Canada.

REFERENCES

1. Bigert, J., Kann, V., Knutsson, O., and Sjobergh, J. Grammar checking for Swedish second language learners. CALL for the Nordic Languages: Tools and Methods for Computer Assisted Language Learning, 33–47.
2. Brockett, C., Dolan, W., and Gamon, M. Correcting ESL errors using phrasal SMT techniques. Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL (2006), 249–256.
3. Buttcher, S., and Clarke, C. Indexing time vs. query time: trade-offs in dynamic information retrieval systems. Proceedings of the 14th ACM international conference on Information and knowledge management (2005), 317–318.
4. Crystal, D. English as a Global Language. Cambridge University Press, 2003.
5. Eeg-Olofsson, J., and Knutsson, O. Automatic grammar checking for second language learners-the use of prepositions, 2003.
6. Ellis, R. The Study of Second Language Acquisition. Oxford University Press, 1994.
7. Gamon, M., Gao, J., Brockett, C., Klementiev, A., Dolan, W., Belenko, D., and Vanderwende, L. Using contextual speller techniques and language modeling for ESL error correction. Urbana 51, 61801.
8. Granger, S. Prefabricated patterns in advanced EFL writing: Collocations and formulae. Phraseology: Theory, Analysis, and Applications (1998), 145–160.
9. Guo, S., and Zhang, G. Building a customized Google-based collocation collector to enhance language learning. BJET 38 (2007), 747–750.
10. Hill, J., and Lewis, M. Dictionary of selected collocations. Language Teaching Publications, 1997.
11. Kann, V., Domeij, R., Hollman, J., and Tillenius, M. Implementation aspects and applications of a spelling correction algorithm. Text as a Linguistic Paradigm: Levels, Constituents, Constructs. Festschrift in honour of Ludek Hrebicek 60 (2001), 108–123.
12. Knutsson, O., Pargman, T., and Eklundh, K. Transforming grammar checking technology into a learning environment for second language writing. Proceedings of the HLT-NAACL 03 workshop on Building educational applications using natural language processing-Volume 2 (2003), 38–45.
13. Knuttson, O., Pargman, T., Eklundh, K., and Westlund, S. Designing and developing a language environment for second language writers. Computers & Education 49, 4 (2007), 1122–1146.
14. Leed, R., and Nakhimovsky, A. Lexical functions and language learning. The Slavic and East European Journal 23, 1 (1979), 104–113.
15. Lewis, M. Teaching Collocation: Further Developments in the Lexical Approach. Language Teaching Publications, 2000.
16. Nesselhauf, N. The use of collocations by advanced learners of English and some implications for teaching. Applied Linguistics 24, 2 (2003), 223–242.
17. Shei, C., "Combining Translation into the Second Language and Second Language Learning", Ph.D. Thesis, University of Edinburgh, 2002.
18. Shei, C., and Pain, H. An ESL writer's collocational aid. Computer Assisted Language Learning 13, 2 (2000), 167–182.
19. Yi, X., Gao, J., and Dolan, W. A web-based English proofing system for English as a Second Language users. At IJCNLP 2008 (2008).
20. Vernon, A. Computerized grammar checkers 2000: capabilities, limitations, and pedagogical possibilities. Computers and Composition 17, 3 (2000), 329–349.
21. White, L. Second language acquisition: From initial to final state. Second Language Acquisition and Linguistic Theory (2000).