Simplified Statement Extraction Using Machine Learning Techniques

Conor McGrory, Princeton University

***Abstract*—The automatic generation of basic, factual questions from a single sentence of text is a problem in the field of natural language processing (NLP) that has received a considerable amount of attention in the past five years. Some studies have suggested splitting this problem into two parts: first, decomposing the source sentence into a set of smaller, simple sentences, and then transforming each of these sentences into a question. This paper outlines a novel method for the first part, combining two techniques recently developed for related NLP problems. Our method uses a trained classifier to determine which phrases of the source sentence are potential answers to questions, and then creates different compressions of the sentence for each one.**

I. Introduction

Asking questions is one of the most fundamental ways that human beings use natural language. When someone studies a foreign language, many of the first utterances they learn are basic questions. The ability of a speaker to form a grammatical question — to request a specific piece of information from another party — is indispensable in most practical situations involving basic communication. Over the past five years, there has been a significant amount of new research towards developing computer systems that can automatically generate basic questions from input text. This is referred to in the literature as the problem of Question Generation (QG), and it has many potential applications in education, including the development of computerized tutoring systems and the generation of basic reading comprehension questions for elementary-level students. Although some studies in the past have tried to generate questions based on whole blocks of text [1], the majority of recent work done on QG has focused on the problem of generating factual questions from a single sentence of input.

Early attempts to solve this problem used complicated sets of grammatical rules to transform the input sentence directly into a question [2]. However, in 2010, Heilman and Smith [3] suggested separating the problem into two steps: first, simplifying the source sentence, and then transforming it into a question. The advantage of this approach is that grammatical rules are much better at transforming simple sentences into questions than they are at transforming complex ones. Our paper outlines a method for preforming the first step in this process, which we refer to as the problem of Simplified Statement Extraction (SSE).

Conor McGrory is participating in a National Science Foundation REU at the University of Colorado at Colorado Springs, Colorado Springs, CO, 80918. e-mail: cmcgrory@princeton.edu

II. Prior Work

In a paper also published in 2010 [4], Heilman and Smith developed a rule-based SSE algorithm that extracted multiple simple sentences from a source sentence. This algorithm recursively applied a set of transformations to the a phrase structure tree representation of the input sentence to generate the simple statements. By extracting multiple simplified statements from the source sentence, they greatly increased the number of possible questions that could be generated and the percentage of words from the input sentence that appeared in one of the output statements [4].

Two problems in NLP that are related to QG are cloze question generation and sentence compression. A cloze question is a type of question commonly used to test a student's comprehension of a text, where the student is asked, after reading the text, to complete a given sentence by filling in a blank with the correct word. One example could be the question

A ____________ is a conceptual device used in computer science as a universal model of computing processes.

In this case, the answer would be *Turing machine*. Because these questions are commonly used in testing, and require no syntactic rearrangement of the source sentence (just deletion of a specific phrase), they seem like an easy place to apply QG techniques. However, selecting which phrase or phrases in the sentence to delete is somewhat difficult. A question like

A Turing Machine ____ a conceptual device used in computer science as a universal model of computing processes.

with the verb *is* as the answer would be completely useless to a student interested in testing their knowledge of basic computer science. An automatic cloze question generator needs to have some way of distinguishing informative questions from extraneous ones. Because the quality of a cloze question can depend on complicated relationships between a large number of factors (syntax, semantics, etc.), distinguishing quality of a question is a good task for a machine learning system. Becker et al.[5] did this by training a logistic regression classifier on a corpus of questions paired with human judgements of their quality. The classifier was able to identify 83 percent of the high-quality sentences correctly and only misidentified 19 percent of low-quality questions as high quality[5].

Sentence compression is the problem of transforming an input sentence into a shorter version that is grammatical and retains the most important semantic elements of the original. This can be used to generate summaries or headlines for large blocks of text. Various methods have been developed to attack this problem. Knight and Marcu [6] used a statistical language model where the input sentence is treated as a noisy channel and the compression is the signal, while Clarke and Lapata [7] used a large set of constituency parse tree manipulation rules to generate compressions.

Filippova and Strube [8] developed a sentence compression system where the compressed sentence is generated by pruning the dependency parse tree of the input sentence. Using the Tipster corpus, they calculated the conditional probabilities of specific dependencies occurring after a given head word. These were used, in combination with data on the frequencies of the words themselves, to calculate a score for each dependency in the tree. They then formulated the problem of compressing the sentence as an integer linear program. Each variable corresponded to a dependency in the tree. A value of 1 meant the dependent word of that dependency would be preserved in the compression, and a value of 0 meant that it would be deleted. Constraints were added to the linear program to restrict the structure and length of the compression, and the objective function set to be maximized was the sum of the scores of the preserved dependencies.

The central assumption made by Fillippova and Strube's method is that the frequency with which a particular dependency occurs after a given word is a good indicator of its grammatical necessity. For example, transitive verbs like *chase* require direct objects, so the frequency of the *dobj* dependency after the head word *chase* in the corpus is very high. Although *chase* can also be the governor of a prepositional phrase, this is not grammatically necessary, so there will be many more instances in the corpus where *chase* does not govern a prepositional phrase, resulting in the frequency of the *prep* dependency after *chase* to be lower.

III. Problem Definition

In explaining our system, it will help to have a formal definition of the problem. We will define the problem of simplified statement extraction as follows:

For a source sentence S, create a set of simplified statements $\{s_i ... s_n\}$ that are semantic entailments of S. A sentence is considered to be a *simplified statement* if it is a declarative sentence (a statement) that can be directly transformed into a question-answer pair (QA pair) without any compression. Ideally, the interrogative transformations of the generated $\{s_i\}$ should include as many as possible of the set of QA pairs a human being could generate given S. We will call the ratio of computer-generated, grammatical QA pairs to human-generated QA pairs the *coverage* of the system.

IV. Solution

As Becker et al. [5] showed with their work on cloze questions, there are certain phrases in S that make sense as answers to questions and others that do not. The fundamental idea behind our SSE system is that knowledge of which phrases in S are good answers can inform the compression process, preventing us from missing important information and thereby maximizing coverage. We divide the SSE problem into two parts: first identifying potential answers, and then generating for each of these answers a compression of S where that answer is preserved. These compressions form the set $\{s_i\}$ of simplified statements. Because each one of these statements will ultimately be transformed into a question with the given answer, our goal when compressing for a particular answer is to find the *shortest grammatical compression* of S that contains the given answer. This will ensure that each selected answer is preserved in at least one of the simplified statements and that these statements will contain minimal amounts of extraneous information.

To select potential answers from the input sentence, we use a slightly modified version of Becker et al.'s cloze question generation system [5]. Because all questions are essentially requests for specific pieces of information, determining which phrases in S make good answers to a standard grammatical question is very similar to determining which phrases make good blank spaces for a cloze question. Once we have the set of possible answers, we use a more substantially modified version of Filippova and Strube's dependency tree pruning method [8] to generate the set of shortest grammatical compressions of S that contain each of the answers.

V. Answer Selection

We designed and implemented the answer selection system using the Stanford NLP Toolkit [9] and the Weka machine learning software [10]. It uses the corpus of sentences, QA pairs, and human judgments developed by Becker et al.[5] to train a classifier to find the nodes in the parse tree of the input sentence that are most likely viable answers to questions. Our implementation performs two basic functions. First, it has the ability to read in the corpus, calculate a set of features and determine a final classification for each potential answer, and output this data set as an .arff file (the standard file format used by Weka). When the program needs to find the good answers in an input sentence, it loads the classifier from the file, determines all grammatically possible answer phrases in the input sentence (this is based on a set of constraints given by Becker et al. [5]), and uses the classifier to determine which of these phrases are good answers.

A. Feature Set

The Stanford NLP Toolkit [9] provides us with two very useful tools for describing the grammatical structure of a sentence: a Penn Treebank style constituency parse tree and the Stanford dependency relations [11]. The Stanford dependency relations are a set of grammatical relations between governor and dependent words in a sentence. Some examples include verb-subject, verb-indirect object, noun-modifier, and noun-determiner. Essentially, it is a dependency grammar with more specific information than which words a given word governs and which words it depends on. The relations also have a set hierarchy. For example, the verb-subject, verb-object, and verb-adverbial modifier relations are all instances of the parent relation predicate-argument. This enables the user to work at different levels of detail. For our purposes, we used the 56 basic relations defined in the Stanford library to categorize all of our dependencies.

We used many of the same features as Becker et al.[5] did, but because we used a different NLP package to implement our system (we used Stanford's, they used a toolkit developed by Microsoft), some of our features were significantly different. At this point, we have also implemented far fewer features than they did. Our features can be divided into three basic categories: token count features, syntactic features, and semantic features.

The token count features we used were the exact same as those used by Becker et al. This category contained 5 features which had to do with the length of the answer in comparison to the length of the sentence, like the raw lengths of both and the length of the answer as a percentage of the length of the question.

The syntactic features were calculated using the constituency parse tree. Currently, our system uses three syntactic features: the Penn part-of-speech tag of the word that comes immediately before the answer in the sentence, the tag of the word that comes immediately after, and the set of tags of words contained in the answer phrase.

The semantic features use the Stanford dependencies system and are completely different than the semantic features used by Becker et al. The purpose of these is to determine the grammatical role the answer phrase plays within the sentence. We currently have four semantic features implemented: the dependency relation between the head of the answer phrase and its governor in the sentence, the set of relations between governors in the answer and dependents not in the answer, the set of relations with both governors and dependents in the answer, and the distance in the constituency tree between the answer node and its maximal projection.

B. Classifier

The classifier used in our system is the Weka Logistic classifier [12]. Because each instance is classified as either "Good" or "Bad", this is a binary logistic regression classifier, similar to the one used by Becker et al. However, Becker et. al also used L2 regularization (adding a constant multiple of the L2 norm of the regression coefficients to the error function as a penalty for overfitting), which we have not yet implemented.

C. Human Judgments

The corpus provided by Becker et al. consists of slightly over 2,000 sentences, each with a selected answer phrase and four human judgments of the quality of the answer. Human judges could rate answers as either "Good", "Okay", or "Bad". Because the classifier requires that each instance be classified in only one category, we had our program use the four judgments to calculate a score for each answer, which we then used to determine how to classify it in the data set. A "Good" rating added 0.25 to the score, an "Okay" added 0.125, and a "Bad" rating added nothing. This score is then compared to the threshold value (a pre-set constant in the program). If the score is greater than or equal to this value, the answer is classified in the data set as "Good". Otherwise, it is classified as "Bad".

VI. Results

We used the program to produce a data set from the Becker et al. corpus [5]. This data set was created using a threshold value of 1.0 (all four human judges have to rate the sentence as "Good"). Then, using Weka, a random sample of the sentences was drawn from this data to produce a subset with a comprable amount of "Good" and "Bad" sentences. This set contained a total of 582 instances, 278 of which were "Good" and 304 of which were "Bad". We tested both the Weka Logistic classifier [12] and the Weka Simple Logistic classifier on the data using 10-fold cross-validation.

The statistics we were most concerned with were the correct classification rate (the number of correctly classified instances divided by the total number of classified instances), the true positive rate (the number of correctly classified "Good" instances divided by the total number of "Good" instances), and the false positive rate (the number of incorrectly classified "Bad" instances divided by the total number of "Bad" instances). We also looked at the Weka-generated "confusion matrix," which summarizes the classifications.

For the Logistic classifier, the correct classification rate was 72.3%, the true positive rate was 78.4%, and the false positive rate was 33.2%. For the confusion matrix (which is normalized), we have:

	Classified "Good"	Classified "Bad"
"Good"	218	60
"Bad"	101	203

In total, 54.8% of the instances were labeled "Good" and 45.2% were labeled "Bad".

For the Simple Logistic classifier, the correct classification rate was 74.2%, the true positive rate was 81.3%, and the false positive rate was 32.2%. For the confusion matrix, we have:

	Classified "Good"	Classified "Bad"
"Good"	226	52
"Bad"	98	206

In total, 55.7% of the instances were labeled "Good" and 44.3% were labeled "Bad".

Becker et. al were able to get a true positive rate of 83% and a false positive rate of 19% at the equal error rate [5]. Although their false positive rate is lower, the true positive rate of our system is definitely comparable to theirs.

VII. Sentence Compression

To compress S into the different simplified statements, we used a modified version of the integer linear programming (ILP) model described by Filippova and Strube [8]. We first calculated probabilities of dependencies occurring after head words and used this as an estimate of the grammatical necessity of different dependencies given the presence of a head word. Along with all of the constraints placed on the ILP in the original model, we added an extra constraint that ensures the preservation of the answer phrase in the compression. We then used a linear program solver to solve the ILP for all length values between 0 and the length of S, generating a set of compressions of S with all possible lengths. From these compressions, we used a 3-gram model to calculate the Mean First Quartile (MFQ) grammaticality metric described by Clark et al. [13]. Compressions with an MFQ value lower than a threshold were deemed grammatical, and the shortest of these was selected as the final compression of S for the given answer.

A. Dependency Probabilities

In order to be more precise, we used a larger set of Stanford dependencies to calculate the conditional probabilities than we did for the feature set in the selection part of the system. The extra dependencies included in this set were collapsed dependencies [11], which are created when closed-class words like *and*, *of*, or *by* are made part of the grammatical relation, producing dependencies like *conj_and*, *prep_of*, and *prep_by*.

To calculate the frequencies of dependencies after certain head words, we used a pre-parsed section of the Open American National Corpus [14]. Filippova and Strube [8] used part of the TIPSTER corpus to calculate their frequencies, but we lacked the computational resources to parse the data ourselves, so we used the pre-parsed data. The frequency of a dependency in our system is defined as the the number of words in the document that are governors of at least one of these dependencies. If a dependency appears more than once for a given governor word (e.g. if a noun is modified by two prepositional phrases), our program will only increase its count by one. This prevents the frequency of a dependency following a given head word from ever exceeding the frequency of the head word itself.

To prevent rounding errors, we used a smoothing function when calculating the probabilities from the frequency data. If we let $f_{(\ell|h)}$ be the frequency with which dependency of type ℓ occurs with head word h in the corpus, and let f_h be the frequency of word h in the corpus, then we define the smoothed probability $P_{(\ell|h)}$ to be

$$P_{(\ell|h)} = \log_2(\frac{f_{(\ell|h)}}{f_h} + 1)$$

Because $f_h \geq f_{(\ell|h)}$ and $f_h, f_{(\ell|h)} \geq 0$, $\frac{f_{(\ell|h)}}{f_h} \in [0, 1]$. Therefore, because

$$\log_2(x + 1) \in [0, 1] \forall x \in [0, 1]$$

we know that $P_{(\ell|h)} \in [0, 1]$ for all possible ℓ and h.

Finally, to avoid problems that come with probability values of zero, our system linearly maps the $P_{(\ell|h)}$ values from $[0, 1]$ to $[10^{-4}, 1]$.

B. Integer Linear Program

Like Filippova and Strube [8], we formulate the compression problem as an ILP. For each dependency in the parse tree (say, the dependency with the Stanford type ℓ, holding between head word h and dependent word w), we create a variable $x^{\ell}_{h,w}$. These variables must each take on a value of 0 or 1 in the solution, where dependencies whose variables are equal to 1 are preserved in the resulting compression and dependencies whose variables are equal to 0 are deleted, along with their dependent words. The ILP maximizes the objective function

$$f(X) = \sum_x x^{\ell}_{h,w} \cdot P_{(\ell|h)} \cdot t(\ell, P_{(\ell|h)})$$

where t is the *tweak function*, which corrects discrepancies between frequency and grammatical necessity that occur with some specific types of dependencies. For example, conjunctions (*conj*) occur very frequently in written English, but they are generally not necessary for the grammaticality of a sentence. Often, deleting parts of conjunctions can actually be an effective way to compress a sentence. Multiplying a particular probability by t linearly maps the range of that value from $[0, 1]$ to $[min_\ell, max_\ell]$. The tweak function is defined as

$$t(\ell, P_{(\ell,h)}) = max_\ell - min_\ell(1 + \frac{1}{P_{(\ell,h)}})$$

where

$$min_{conj} = 0.0, max_{conj} = 0.4$$

,

$$min_{det} = 0.4, max_{det} = 1.0$$

,

$$min_{poss} = 0.5, maxposs = 1.0$$

, and

$$min_\ell = 0.0, max_\ell = 1.0$$

for all other dependencies, which means that $t(\ell, P_{(\ell,h)}) = 1$ for all dependencies besides conjunctions, determiners and possessives. Our tweak function replaces the importance function used in Filippova and Strube's objective function [8].

Filippova and Strube also used two constraints in their model to preserve tree structure and connectedness in the compression:

$$\forall w \in W, \sum_{h,\ell} x^{\ell}_{h,w} \leq 1$$

$$\forall w \in W, \sum_{h,\ell} x^{\ell}_{h,w} - \frac{1}{|W|} \sum_{u,\ell} x^{\ell}_{w,u} \geq 0$$

and one to restrict the length of the final compression to α:

$$\sum_x x^{\ell}_{h,w} \leq \alpha$$

To ensure that all of the words in the pre-selected answer A are also preserved, we include in our model the extra constraint

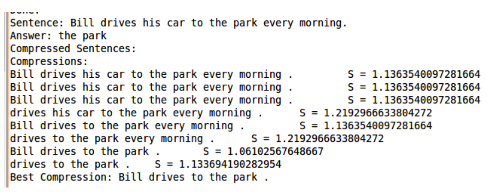
$$\forall w \in A, \sum_{h,\ell} x^{\ell}_{h,w} \geq 1$$

We solved these integer linear programs using lp_solve [15], an open-source LP and ILP solver.

C. Shortest Grammatical Compression

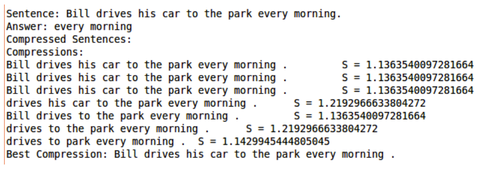
In order to find the shortest grammatical compression of S, our system first finds a solution to the ILP for S and A for every value of α (the maximum length constraint parameter) between the length of S and the length of A. Because the constraints also specify that every word in A is preserved in the compression, any model where α is less than the length of A would have no solution.

Although all solutions to the ILP are connected dependency trees, some of the actual sentences created by linearizing these trees will not be grammatical. To determine the grammaticality



```
Sentence: Bill drives his car to the park every morning.
Answer: the park
Compressed Sentences:
Compressions:
Bill drives his car to the park every morning .         S = 1.1363540097281664
Bill drives his car to the park every morning .         S = 1.1363540097281664
Bill drives his car to the park every morning .         S = 1.1363540097281664
drives his car to the park every morning .      S = 1.2192966633804272
Bill drives to the park every morning .         S = 1.1363540097281664
drives to the park every morning .      S = 1.2192966633804272
Bill drives to the park .       S = 1.06102567648667
drives to the park .    S = 1.133694190282954
Best Compression: Bill drives to the park .
```

Fig. 1. Simulation Results



```
Sentence: Bill drives his car to the park every morning.
Answer: every morning
Compressed Sentences:
Compressions:
Bill drives his car to the park every morning .         S = 1.1363540097281664
Bill drives his car to the park every morning .         S = 1.1363540097281664
Bill drives his car to the park every morning .         S = 1.1363540097281664
drives his car to the park every morning .      S = 1.2192966633804272
Bill drives to the park every morning .         S = 1.1363540097281664
drives to the park every morning .      S = 1.2192966633804272
drives to park every morning .  S = 1.1429945444805045
Best Compression: Bill drives his car to the park every morning .
```

Fig. 2. Simulation Results

of the compressions, we use the MFQ metric, which is based on a 3-gram model created using the Berkeley Language Model Toolkit [16] and trained on the OANC text. This metric was shown to work well at distinguishing grammatically well-formed sentences from ungrammatical ones by Clarke et al. [13]. It considers the log-probabilities of all of the n-grams in the given sentence, selects the first quartile (25% with the lowest values), and calculates the mean of the ratios of each n-gram log-probability over the unigram log-probability of that n-gram's head word. The larger the MFQ value is, the less likely the sentence is to be grammatical.

Our system looks through the list of different length compressions and selects the shortest compression with an MFQ value less than a specified threshold (for our 3-gram model, we used a threshold of 1.14). This compression is returned as the simplified statement extracted from S for the answer A.

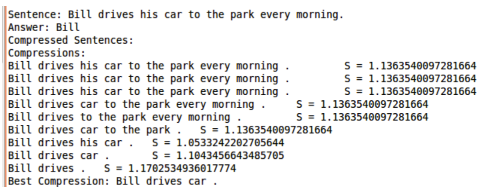
D. Results

We have not yet been able to conduct a test of the compression system, because testing the grammaticality of the generated compressions and their coverage of the set of possible simplified statements requires the use of a large number of human judges. However, the basic functionality of the compression system can at least be demonstrated with some sample outputs from the compressor. In each of the outputs, the sentence and answer are specified at the top, and then each row contains a potential compression and its MFQ value (labeled as 'S' on the readout).

Figure 1 shows a perfect compression of the sentence *Bill drives his car to the park every morning*. In the list of generated compressions, the one ultimately selected is clearly the shortest grammatical compression of the input sentence.

The output in Figure 2 is still grammatical, but there is one shorter compression in the list that is also grammatical, but was not identified by the program. This is because the MFQ value for *Bill drives to the park every morning* was 1.136, which is slightly less than the threshold of 1.14. Examples like this make it clear that tuning the gramamticality threshold is very important.

Figure 3 is not grammatical, but there is a grammatical sentence in the compression list only one word longer than the



```
Sentence: Bill drives his car to the park every morning.
Answer: Bill
Compressed Sentences:
Compressions:
Bill drives his car to the park every morning .         S = 1.1363540097281664
Bill drives his car to the park every morning .         S = 1.1363540097281664
Bill drives his car to the park every morning .         S = 1.1363540097281664
Bill drives car to the park every morning .     S = 1.1363540097281664
Bill drives to the park every morning .         S = 1.1363540097281664
Bill drives car to the park .   S = 1.1363540097281664
Bill drives his car .   S = 1.0533242202705644
Bill drives car .       S = 1.1043456643485705
Bill drives .   S = 1.1702534936017774
Best Compression: Bill drives car .
```

Fig. 3. Simulation Results

compression that was selected. The chosen compression had a higher MFQ score than the true shortest grammatical sentence, but because it was shorter, it was chosen nonetheless.

VIII. Conclusion

The key principle around which our system is built is that selecting the answer at the beginning of the QG process and using them to guide SSE can improve the coverage of the system. We implemented the machine learning-based approach for answer selection used by Becker et al. [5] and developed a way to compress a sentence while leaving a specified answer phrase intact. Although we have not yet been able to perform large scale tests on this system where the output is rated by human judges, we have generated some good output sentences. Once our implementation is perfected and tuned, we will perform more powerful and complete tests.

This system will soon be integrated with Jacob Zerr's Part-of-Speech Pattern Matching system for direct declarative-to-interrogative transformation to produce a full, functional, QG system.

References

[1] Kunichika, Hidenobu, Tomoki Katayama, Tsukasa Hirashima, and Akira Takeuchi. "Automated question generation methods for intelligent English learning systems and its evaluation." In Proceedings of ICCE2004, pp. 2-5. 2003.

[2] Wolfe, John H. "Automatic question generation from text-an aid to independent study." In ACM SIGCUE Outlook, vol. 10, no. SI, pp. 104-112. ACM, 1976.

[3] Heilman, Michael, and Noah A. Smith. "Good question! statistical ranking for question generation." In Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp. 609-617. Association for Computational Linguistics, 2010.

[4] Heilman, Michael, and Noah A. Smith. "Extracting simplified statements for factual question generation." In Proceedings of QG2010: The Third Workshop on Ques-tion Generation, p. 11. 2010.

[5] Becker, Lee, Sumit Basu, and Lucy Vanderwende. "Mind the gap: learning to choose gaps for question generation." In Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 742-751. Association for Computational Linguistics, 2012.

[6] Knight, Kevin, and Daniel Marcu. "Statistics-based summarization-step one: Sentence compression." In AAAI/IAAI, pp. 703-710. 2000.

[7] Cohn, Trevor, and Mirella Lapata. "Sentence Compression as Tree Transduction." Journal of Artificial Intelligence Research 34 (2009): 637-674.

[8] Filippova, Katja, and Michael Strube. "Dependency tree based sentence compression." In Proceedings of the Fifth International Natural Language Generation Conference, pp. 25-32. Association for Computational Linguistics, 2008.

[9] Stanford NLP Toolkits, http://nlp.stanford.edu/software.

[10] Holmes, Geoffrey, Andrew Donkin, and Ian H. Witten. "Weka: A machine learning workbench." In Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on, pp. 357-361. IEEE, 1994.

[11] De Marneffe, Marie-Catherine, and Christopher D. Manning. ”Stanford typed dependencies manual.” URL http://nlp. stanford. edu/software/dependencies manual. pdf (2008).
[12] Le Cessie, Saskia, and J. C. Van Houwelingen. ”Ridge estimators in logistic regression.” Applied statistics (1992): 191-201.
[13] Clark, Alexander, Gianluca Giorgolo, and Shalom Lappin. ”Statistical representation of grammaticality judgements: the limits of n-gram models.” CMCL 2013 (2013): 28.
[14] Ide, Nancy, and Catherine Macleod. ”The american national corpus: A standardized resource of american english.” In Proceedings of Corpus Linguistics 2001, vol. 3. 2001.
[15] Berkelaar, Michel. ”lpSolve: Interface to Lp solve v. 5.5 to solve linear/integer programs.” R package version 5, no. 4 (2008).
[16] Pauls, Adam, and Dan Klein. ”Faster and smaller n-gram language models.” In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1, pp. 258-267. Association for Computational Linguistics, 2011.