

Answering Reading Comprehension Tests:N-gram and Multi-View Regression

Dept. of CIS - Senior Design 2009-2010

Andrew Pak
apak2@wharton.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Daniel Seunghyun Kim
opcx328@gmail.com
Univ. of Pennsylvania
Philadelphia, PA

Lyle H. Ungar
ungar@cis.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

ABSTRACT

Computer “reading” of natural languages is a growing research topic with numerous applications, ranging from areas of artificial intelligence to linguistics. A successful “reading” of natural languages can be interpreted in two ways: 1) extracting facts out of a passage and 2) capturing “meaning” of a text. We focus on the latter interpretation, defining “meaning” as is done in humans, namely, can the system answer the same reading comprehension questions that are given to early learners of language.

We present two approaches to answering cloze questions, which are fill-in-the blank, multiple-choice reading comprehension questions. The first approach utilizes n-gram, a basic statistical methodology that captures the local context of a passage to predict an answer. The second approach uses Multi-View Regression (“MVR”), based on canonical correlation analysis (“CCA”), which captures both the local and broader context. In this statistical approach, the “meaning” of each word is represented as a real-valued state vector associated with each word occurrence. MVR is a dynamical belief net which, like a Kalman filter or Hidden Markov Model (“HMM”), estimates the state based on the preceding and following sequence of words and predicts the word to be “emitted” based on that state.

While there has been much research on these statistical methodologies in the field of machine learning and on generation and evaluation of cloze questions in the field of natural language processing, the connection between the two hasn’t been explored much. Prior research in cloze questions suggests that proper context capturing leads to correct answers. We create a system that reads in a set of cloze questions and predicts two sets of answers with the two statistical methodologies that have different context capturing schemes, and then analyze the results. Because MVR captures both the local and broader context, it answers cloze questions better than n-gram, which only captures the local context.

1. INTRODUCTION

If computers can “read” natural languages, the benefit will be enormous to various fields and industries. We will be one step closer to creating self-learning robots, which has been the goal of artificial intelligence researchers for many years. Also, there can be numerous applications to the field of linguistics and natural language processing. For example, word sense disambiguation - the process of identifying which “meaning” of a word is used in any given sentence, when the

word has a number of distinct “meanings” -which has been an open problem for a long time can be addressed.

How can one test if machine is correctly “reading” natural languages? Successfully “reading” a passage can be interpreted in two ways, as extracting facts from the text or as comprehending the “meaning” of the text. Obtaining facts from a passage can be thought of as building a knowledge database using tuples that contain information. For example, (Bob, birthday, 6/28/1965) can be a tuple extracted from a passage about Bob. The task of extracting facts from passages is heavily researched by fact engines, which will be introduced in the “Related Work” section. Construction and maintenance of such knowledge database is valuable, but it provides no context for the passage while being costly and expensive. We focus on comprehending the “meaning” of passages by capturing contexts.

There are several approaches that can be utilized to capture context. HMM and n-gram approach captures only the local context, while Latent Semantic Indexing (“LSI”) and Latent Dirichlet Allocation (“LDA”) captures only the document level context, or topic. MVR approach captures both the local and broader context. We build a system that uses n-gram and MVR approach to answer a set of cloze questions and compare the results between the two.

In order to see how well an approach captures the “meaning” of passages, each approach is tested against a set of cloze questions. Cloze questions are a type of reading comprehension questions which can normally be answered using “yes” or “no”, a specific piece of information, or a selection from multiple choices. The cloze question answering system will focus on those fill-in-the-blank questions with multiple choices. Example of such question is:

Anna and her mother boiled water in a big pot and put the berries into it. The water turned into a deep red. Anna’s mother dipped the pale yarn into it. Soon red yarn was hanging up to dry on a clothesline strung across the kitchen. Anna and her mother changed the blank of the yarn.

1. shape
2. color
3. size
4. taste

In the above example, the last sentence is the cloze question and the sentences prior to the question are considered as the story related to that question.

Stories and questions similar to the example are obtained from Lexile, a company that sells language learning related products. Eight different levels of reading comprehension tests composed of about 50 questions each are used. With these cloze questions from Lexile, for which the student response data set and correct answers are given, the system analyzes the predictions from two approaches.

The MVR approach requires a training corpus before a question can be fed in. Project Gutenberg is an on-line digital library that contains around 30,000 novels and documents. From this library, novels and documents that have similar reading level and vocabulary as the stories and questions from Lexile are filtered to be used as a training corpus.

2. RELATED WORK

As our project incorporates statistical algorithms and cloze question answering, related work consists of two fields: statistical methodologies and question answering. In terms of statistical methodologies, we look at two approaches, n-gram and MVR. For question answering, related work includes fact-based question answering and cloze question answering.

2.1 Statistical Methodologies

2.1.1 N-gram

The task of predicting the next word can be stated as attempting to estimate the probability of the next word given previous words. In such a stochastic problem, a classification of the previous words, the history, can be used to predict the next word. On the basis of having looked at a lot of text, one can identify which words tend to follow other words.

For this task, one cannot possibly consider each textual history separately: most of the time one will be listening to a sentence that one has never heard before, and so there is no previous identical textual history on which to base predictions, and even if one had heard the beginning of the sentence before, it might end differently this time. And so one needs a method of grouping histories that are similar in some way so as to give reasonable predictions as to which words one can expect to come next. By constructing a model where all histories that have the same last n-1 words in the same equivalence class and grouping them as a phrase, one can predict the next word by looking at the last word of such n-grams [6]. Hence, n-gram algorithm maps the n word phrase to the occurrence of that phrase. In probability terms, n-gram finds the probability of a word being in a certain location given the n-1 words prior to that word.

Google N-gram is a specific product of this algorithm created by training the algorithm on the Internet. The data from Google N-gram are stored as group of text files and each text file has around million lines, where each line contains n word phrase followed by the number of occurrence of that phrase on the Internet. An example of a line in Google 3-gram files is "I went to 24"; "I went to" is the 3 word phrase and 24 is the number of occurrence. 2-gram, 3-gram, 4-gram and 5-gram data from the first version of Google N-gram database are used in this project.

The proposed cloze question answering system uses the Google N-gram to find multiple-choice answers with the highest probability of filling the blank. Only the question part, not the story, is used for the n-gram approach. For example, using the example in the previous paragraph, the question "I went &&", where "&&" denotes the blank, is fed into the

system. The system searches the Google 3-gram and finds the words that go into the blank and the number of such occurrences. Among the possible multiple choices, the word with the highest occurrence in Google 3-gram is outputted as the prediction.

2.1.2 Multi-View Regression ("MVR")

MVR is a slightly different approach to the traditional Hidden Markov Model ("HMM") that incorporates Canonical Correlation Analysis ("CCA") and logistic regression. In traditional HMM, a statistical model based on Expectation-Maximization algorithm in which the system being modeled is assumed to be a Markov process with unobserved states, the states are not directly visible, but the outputs are, and each state has a probability distribution over the possible output tokens. Thus, given some output tokens, HMM gathers information regarding the hidden states that produced such output. MVR is similar to HMM in that it also assumes there exists some set of hidden states that produced the given output tokens and that these states have a probability distribution over the possible output token. MVR, like HMM, is based on the assumption that each entry in the data series can be completely explained by two views: the entries preceding and following the target entry. Under this assumption, MVR uses CCA to learn an optimal mapping from previous or following entries to a latent state space. The hidden states found by CCA is then used as features for supervised learning to predict which entry will be seen as a function of the state.

MVR differs from HMM in that HMM assumes each state already contains all past information that led to that state and so when estimating the next state, HMM looks only at the state prior to the target state. MVR, on the other hand, looks at the sequence of states prior to the target state by running a regression on the hidden states. Thus for cloze question answering, MVR is more effective than HMM since MVR looks at all past information to make predictions while HMM does not.

Further, MVR has significant advantage over HMM in that MVR is a linear method, which makes MVR faster, more robust and more scalable to larger data set than HMM. Also, the Expectation-Maximization algorithm utilized by HMM suffers from convergence problems, which MVR does not.

Numerous techniques regarding the utilization of labeled and unlabeled data were proposed over the years in order to improve understanding and application of machine learning. Multi-task methods predict multiple labels for a given data and are divided into main and auxiliary tasks. Caruana, Ando and Zhang, and Zhang et al. assume that the same latent structure that predicts the auxiliary task labels will also be predictive of the main task. Rather than assuming the features are divided into complementary sets, they assume that the same features can be applied to different tasks.

Canonical Correlation Analysis ("CCA")

Canonical Correlation Analysis, a generalization of Principle Component Analysis ("PCA"), is a method of correlating linear relationships between two multidimensional variables. Whereas PCA computes the directions of maximum covariance between elements in a single matrix, CCA computes

the directions of maximal correlation between a pair of matrices.

PCA is a multivariate data analysis procedure that involves a transformation of a number of possibly correlated variables into a smaller number of uncorrelated variables known as principal components. PCA only makes use of the training inputs while making no use of the labels [5].

CCA makes use of two views of the same semantic object to extract the representation of the semantics. This is the method best suited for this project because it captures mathematical relationship between the views, which we need to answer questions. CCA is used instead of PCA because CCA is invariant to scale which makes it much more efficient in dealing with large corpus than PCA [5]. Ando and Zhang [2] showed that CCA can be used to project the views down to a lower dimensional space without losing predictive information about the target.

There have been other numerous researches on CCA, because of its advantages. CCA is invariant to scale and affine transformations. Furthermore, if the views of the data contain a large number of features that are unnecessary for classification of the object, but correlated with each other, CCA will detect the common features of the different views and reject extraneous dimensions.

2.2 Question Answering System

2.2.1 Fact-based Question Answering

In general, fact-based question answering system takes in a database of knowledge along with the input questions to answer that question. These database, however, are costly and often time-consuming if there is large amount of data in the database.

Many projects have attempted to create question answering system using such database and here are some examples:

1. True Knowledge Ltd. is a company in United Kingdom that specializes in knowledge base and semantic search engine software. Its first product, True Knowledge Answer, was launched in November 2007. The program was an answer engine that aimed to directly answer questions posed in plain English test, which is accomplished using a database of discrete facts.
2. Wolfarm Alpha, released in May 2009, is an answer engine developed by Wolfarm Research. It is an online service that answers factual queries directly by computing the answer from structured data. The service computes and infers answers and relevant visualizations, and thus differs from semantic search engines.
3. Cyc is an artificial intelligence project that attempts to assemble a comprehensive knowledge base of everyday common sense knowledge with the goal of enabling AI applications to perform human-like reasoning. The project was initiated in 1984 by Douglas Lenat and is developed by company Cycorp. The company's project includes researches in natural language understanding tool, which includes the lexicon, the syntactic parser and the semantic interpreter. The semantic interpreter uses commonsense knowledge to guide the interpretation process, which could be of use to our project. For example, regarding the sentence "The man saw the light with the telescope", the semantic interpreter will consult the knowledge base to

find out whether telescopes are typically used as instruments in seeing and whether lights are the kind of things that usually have telescopes.

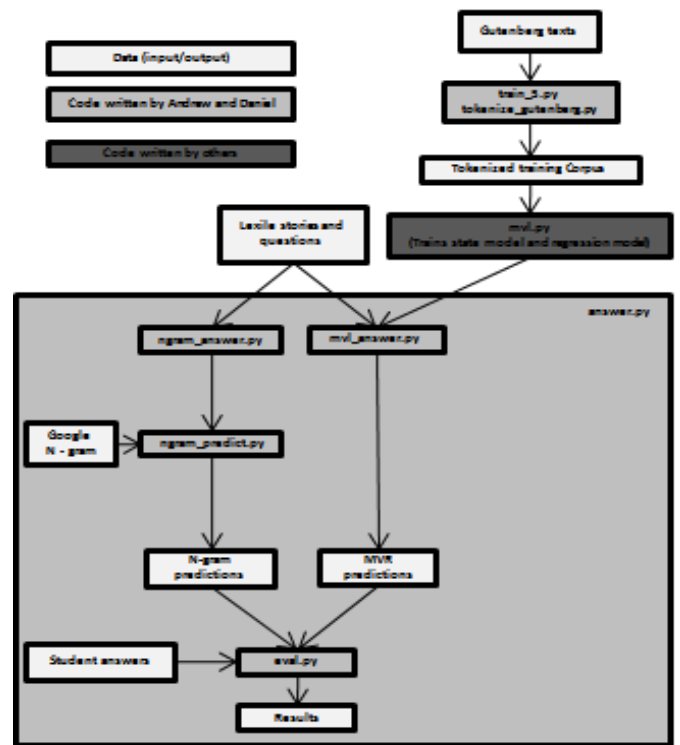
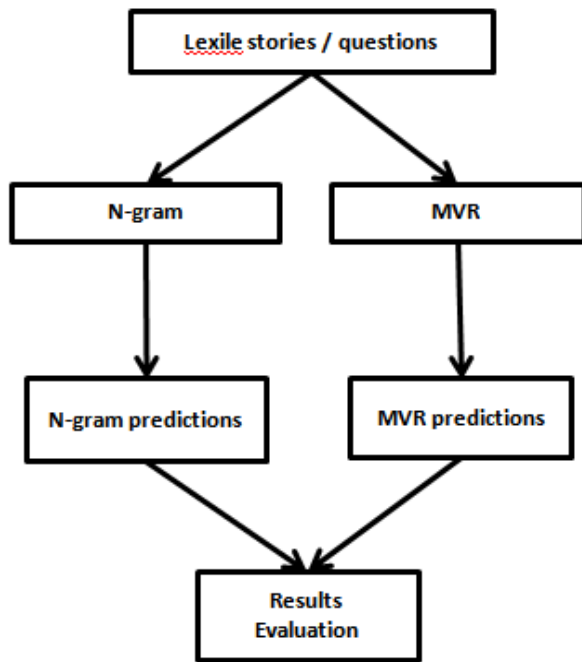
While these projects have relevance to our project in that they answer a question presented in natural language, the answering process has little relevance. Our proposed system will be different from these other systems in that it does not consult the knowledge base to find the answer but rather gathers the information by itself (Machine Learning) to predict the answer.

2.2.2 Cloze Question Answering

Cloze question has been commonly used to evaluate the difficulty of reading and comprehension level of students. There are numerous research on reading comprehension, and specifically, on the effectiveness of cloze question in assessing level of reading comprehension. For example: Project LISTEN's Reading Tutor [7], a cloze question generating program, suggests ideal properties for comprehension questions in intelligent tutors - serving tutorial functions, generating questions automatically, finding correct answers automatically, generating incorrect answers automatically, scoring students' answers automatically, predicting question difficulty automatically, targeting specific skills or knowledge, being psychometrically valid and reliable and maximizing informativeness. Reading Tutor inserts cloze questions by deleting a random word in the next sentence and choosing three random, similarly difficult distracters from the text. The student had to identify the original word, and then the tutor showed the original sentence. [Mostow, Beck, Bey, et al 2004] established that such automatically generated cloze questions were valid measures of comprehension - students' performance on these questions, weighted by word and text difficulty, correlated significantly with a standard measure of reading comprehension [7]. [Beck 2005] also found that student response times on cloze questions were a reliable indicator of student task engagement.[3] The project led to much research in quality question generation, specifically dealing with assessing comprehension, vocabulary and interventions [1] [4] [7] [8]. Project LISTEN also entailed detection of disengagement and assistance in comprehension [3]. Similar to how Project LISTEN uses generated cloze questions to evaluate the reading comprehension level of students; we attempt to use similar questions to evaluate the reading comprehension level of our system. Furthermore, these projects allow us to not only compare results between human and machine, but also allow us to compare the process of making predictions. That is, we are able to understand how human learns so that we can compare, incorporate, and enhance the machine learning process.

3. SYSTEM MODEL

The cloze question answering system takes in as inputs a cloze question and some text associated with that question and predicts the answer. The text is mainly a story related to the question and the question has a blank to fill with multiple choices. A system based on n-gram and MVR approach is built and used to compare the performance of each.



The diagram above depicts the overall design structure of the suggested cloze question answering system. The system will make two predictions, one using n-gram method and the other using MVR. Further details on how each method makes a prediction can be found under the “System Implementation” section.

If a proposed system is developed, the system itself will be a proof that it is possible for a machine to have a reading comprehension level that matches that of students for whom the questions are intended. Considering its effects, we believe a development of a consistent and efficient question answering system will be a significant contribution to the seemingly overwhelming development of artificial intelligence.

4. SYSTEM IMPLEMENTAION

Because of clarity and ease in use, the language Python is used to implement the system. Python language is convenient for parsing strings, which our work consists mostly of. Further, Python’s Natural Language Tool Kit (“NLTK”) module very much alleviates our work as the same module was used by Google to create the Google N-grams; Google N-grams utilizes NLTK module’s tokenizer to parse strings, and hence a separate tokenizer is not needed to be built, if Python is used. Other languages C and C++ were also discussed as candidates as these languages can yield higher speed performance, but since the system being built focuses more on how well it predicts the correct answer than how fast, the C and C++ was not used for the system.

The first to be implemented was the evaluation. Once the answers have been predicted using n-gram and MVR, they need to be compared and analyzed with the correct answer and student responses. Since without the comparisons, there is no way to analyze the results of the predictions, evaluation was implemented first in code ‘eval.py’. Then, the n-gram approach was implemented, followed by the MVR approach. ‘answer.py’ was written at last as a wrapper for both predictions.

Further, we pass outputs from one part of system to another by using text files as inputs and outputs. That way, the information from each part of the system is stored separately and is prevented from any memory malfunctioning as the system sometimes handles large texts. No database decisions are made for the system, as it does not employ any data other than input/output text files and the given Google N-gram database, Gutenberg library and Lexile stories and questions.

4.1 Implementation of System Functions

Following are the detailed descriptions of main system functions currently implemented.

4.1.1 eval.py

‘eval.py’ is the code used for evaluating the predictions. The code takes in three files as inputs: the predictions using n-gram or MVR algorithm, the correct answer file, and a student response file. The prediction file and correct answer file simply contain a single line of number string, where each number represents the answer choice to a questions. The student response file contains multiple lines of number strings, where each line represents a single students’ answer to questions.

‘eval.py’ analyzes the answers predicted by the system and

compares with the correct answers as well as with student responses. The code outputs the correctness of prediction per question, the percentage of right/wrong predictions, the percentage of questions the system was able to predict an answer, and the percentage of right/wrong student response per question.

4.1.2 *answer.py*

'answer.py' is the overall wrapper for the system. Calling on 'answer.py' makes the system to make two different predictions using n-gram and MVR on all available tests and evaluates the predictions.

4.1.3 *tokenizer.py*

'tokenizer.py' parses a input string using the NLTK module, the same module employed by Google to create Google N-gram. Using the same module removes conflicts when searching the Google N-gram because now the input string is parsed the same way as in Google N-gram. For example, a string "I couldn't" is tokenized in Google N-gram as "I", "could", and "t" but normally one would tokenize it as "I" and "couldn't".

4.2 Implementation of N-gram

N-gram was implemented in two separate parts. Because methods for searching Google n-gram has other uses outside the system being built, they were separated out from methods for processing the questions. The methods for searching Google n-gram are in 'ngrams.py' and the methods for processing the question to be searched in Google n-gram are in 'ngram_answer.py' and 'ngram_predict.py'.

While implementing n-gram, some important decisions had to be made. Having 5-gram as the longest gram available, we decided to use maximum number of n-gram possible to answer each question. That is, if the question contained more than 5 words, we used the 5-gram to answer the question. If the question was composed of 4 words, 4-gram was used to answer. Also, if the blank appeared in the middle of question and the question was longer than 5 words, we summed the counts of possible 5-gram searches. For example, if the question is "I went to the && to eat", then the system will search Google 5-grams for "I went to the &&", "went to the && to", and "to the && to eat" and compile the words that appear in the blank and their counts.

The system does not account for n-gram prediction with both the question and the text related to that question. After having implemented n-gram prediction using only the question, it became apparent that giving the text as additional input will have little or no effect on the result. The largest available google ngram is five, which is already far smaller than length of the question. Thus, in most cases, additional text will provide no additional information for predicting the answer.

Additionally, to obtain more plausible statistic using n-gram, the names in the questions, like "Alice", are resolved to pronouns, like "She".

4.2.1 *ngram_answer.py*

'ngram_answer.py' takes in a question file that has the following format: a line of question is followed by lines of multiple choices, which is followed by the next question. Blanks in the questions are denoted as "&&". After taking in the question file, 'ngram_answer.py' parses out each

question and calls n-gram functions to obtain the possible words that can fill the blank.

To make predictions using n-gram method, 'ngram_answer.py' calls on 'ngram_predict.py' to obtain the n-gram statistics - map of each word that can fill the blank and numbers of that words occurrence (found in Google n-gram). Then, it compares them to multiple-choice answers to make the prediction and outputs a prediction file in the format that can be used by 'eval.py'.

4.2.2 *ngram_predict.py*

'ngram_predict.py' takes in a question with a blank, denoted by "&&", changes the question into regular expression, and calls 'ngram.py' with the regular expression with correct n-gram.

'ngram_predict.py' also handles special cases such as existence of "a(an)" in the question and names. For "a(an)", we run a ngram search with both "a" and "an". For names, we replaced the name with "he", "she", "him", or "her" after looking up a list of male and female names in the name database we created specifically for the available tests.

4.2.3 *ngram.py*

'ngram.py' is called by 'ngram_predict.py'. It initializes the directory list of Google N-gram files and grep for right Google N-gram given a question string with regular expressions. It also indexes Google N-gram files for faster search.

The dataset of Google N-gram is divided into each gram category, and per gram category, there are over 150 files. Also, each file contains approximately 10 million grams. Because Google N-gram was sorted alphabetically, we decided to make the system read only the first line of Google N-gram file and compare it to the question. Google N-gram database distinguishes gram that starts with an upper case letter and a lower case letter. So a gram that starts with an upper case "A" is in different file from one that contains grams that starts with a lower case "a". Thus, the search for the right file has to be conducted twice to look for both upper case file and lower case file. Because the question always started with an upper case when the dataset for lower case was much larger, it was best to look up both files, not distinguishing the letter cases within the gram. If the question has relevance to the file, then the system takes the file to search the question sentence, using regular expression for the blank. If not, the system moves on to the next file. Such methodology allowed the system to efficiently search for the right n-gram file.

4.3 Implementation of Multi-View Regression

Provided generalized main source code for the Multi-View Regression ("MVR"), all we needed to do was modify it to fit our purpose. It was important that we limit the number of states and labels, so that the system does not run out of memory trying to handle too many variables. Especially for language processing, because number of states and labels are virtually infinite given infinite number of words, forcing a limit on states and labels was necessary for MVR to run at reasonable speed and with reasonable amount of memory. Using the exponential smoothes of the word count, the system can scale down the vocabulary and state space sizes.

To use MVR to predict answers to cloze questions, CCA and regression must be first trained on a corpus. Training

the CCA and regression on a corpus generates a model with states associated with each output words and regression coefficients, which are later used to make predictions. Further, in order for MVR to make valid predictions to cloze questions, the corpus on which MVR is trained must also contain words in the questions and multiple choices. Thus, subsets of Project Gutenberg texts which contain those words were used to train MVR.

Once the state model and regression coefficients have been generated, they will be stored as a text file so that they do not need to be trained every time the system tries to make a prediction using MVR. The generated state model contains the state transition functions and word emission probabilities for each state. And the regression coefficients, obtained by running a regression on the built state model, explain the relationship between all states, weighted by their distances from the each other. The captured relationship between all states allows the system to consider all information in the past - for the suggested system, the stories that come before the questions, in making the predictions. The cloze question answering system will take in as inputs the state model generated by CCA, the regression coefficients, and the story/question set to predict the answer to the question.

For compliance, the training corpus and the story/question set are both tokenized using the same tokenizer.

4.3.1 *train_5.py*

'train_5.py' extracts out from Project Gutenberg texts two different subsets of texts, which will later be used to train MVR. One subset contains Project Gutenberg texts which collectively contain more than 5 occurrences of all multiple-choice words. That is, each multiple-choice answer will appear more than 5 times in this subset. The other subset contains set of 100 words-long texts from Project Gutenberg that have a multiple-choice word in the middle. So, each 100 words-long text has a multiple-choice word in the middle with 50 words prior to and following that word.

'train_5.py' allows the MVR to be trained on smaller but more relevant set of texts.

4.3.2 *tokenize_gutenberg.py*

'tokenize_gutenberg.py' simply reads in the Project Gutenberg text originally in zip file format, tokenizes them using NLTK module, and saves the tokenized text in text file format. The tokenized text files are then ran through 'train_5.py' to find the appropriate subset.

4.3.3 *mvr_answer.py*

'mvr_answer.py' first reads in the story/question set and tokenizes them to be given as inputs to MVR. Then, it reads in the generated state model and regression coefficients, and uses them to make predictions on the story/question set. The MVR, upon estimating the state that produces the word in blank, returns the emission probability of all possible words. From the emission probabilities, 'mvr_answer.py' extracts out the only the emission probabilities of given multiple choice words and compares them to find the multiple choice word with the highest probability.

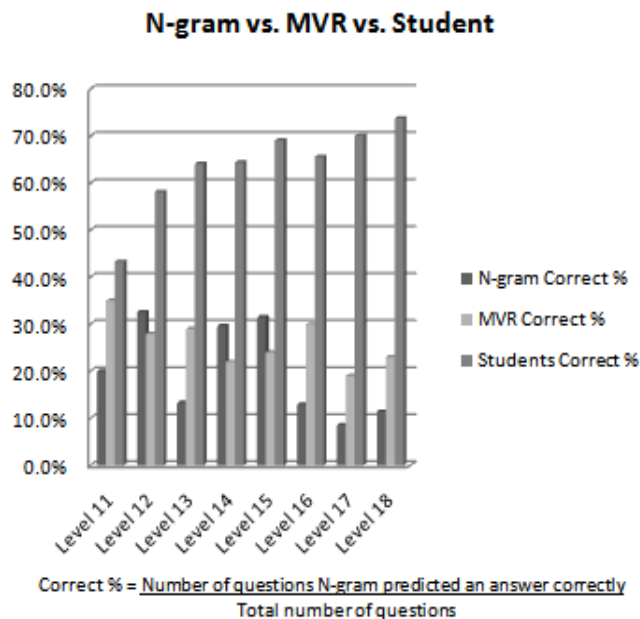
4.3.4 *MVR Source Code (mvr.py)*

Mainly written by Professor Lyle Ungar and Professor Dean Foster at the University of Pennsylvania, the MVR source code has two different parts: one for training the

MVR and one for testing MVR. Training part of MVR takes in a corpus and generates the state model and regression coefficients. For the purpose of our system, MVR was trained on subset of Project Gutenberg texts found using 'train_5.py'. CCA used the corpus to train a state model and a regression was run on that state model to find the regression coefficients. Testing part of MVR uses the generated state model and regression coefficients to estimate the state at some target location and that state's emission probabilities of words.

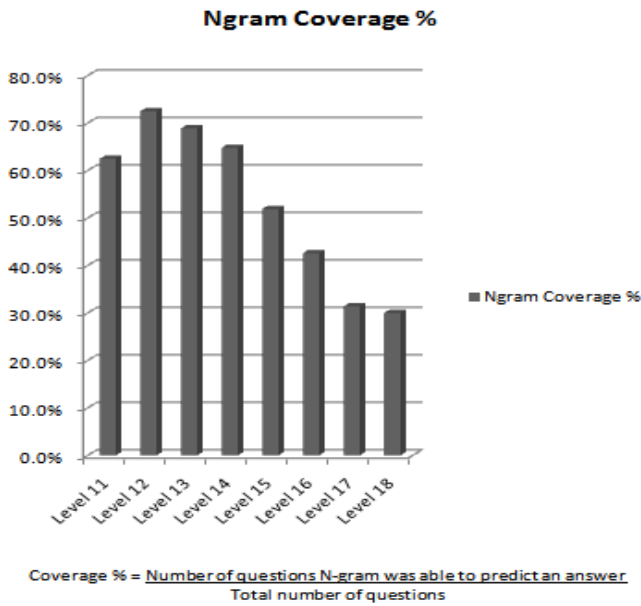
5. SYSTEM PERFORMANCE / RESULTS

The following bar graph shows correct percentage for both n-gram and MVR method, as well as the students.

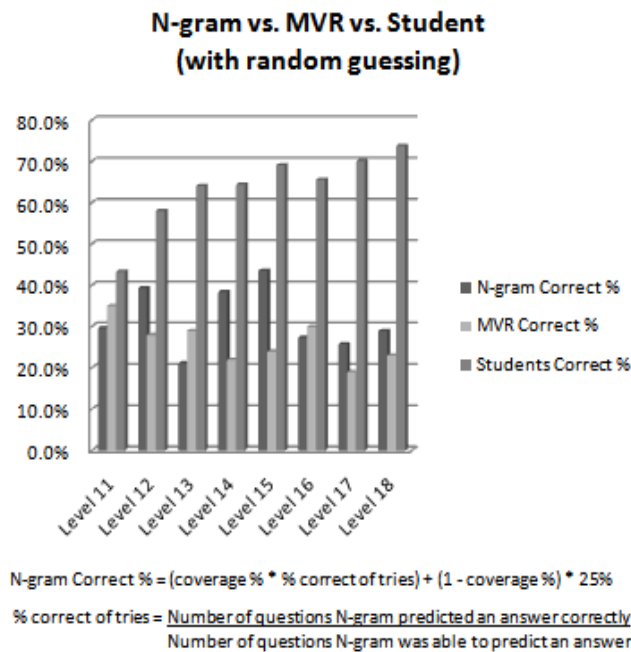


The graph shows that MVR yields higher percentage of correct answers than n-gram in higher level tests. In general, as level of tests gets higher, it requires capturing more contexts to answer the cloze question. Thus, MVR yielding higher percentage of correct answers than n-gram suggests that MVR captures broader context (or simply more context) which N-gram cannot.

Further, the similar results for level 11 tests, from n-gram, MVR, and students, suggests that the proposed cloze question answering system, to some degree, has level of reading comprehension closer to that of those students. Note that for higher level tests, the students' correct percentage is much higher than that of both n-gram and MVR. As level 11 tests are intended for second graders, we conclude that our cloze question answering system has reading comprehension level of a typical second grader.



The above bar graph shows the coverage percentage of n-gram. Because, for some questions, exact n-gram of the question may not be in Google N-gram, n-gram method is sometimes unable to predict an answer, leaving some choices blank. If we modify the n-gram algorithm to make a random guess whenever a prediction is not possible, the result looks as following. (MVR always yield 100% coverage if trained on an appropriate corpus with correct labels.)



	Average Correct %
N-gram	20.0%
N-gram with Random Guessing	31.8%
MVR	26.3%
Students	63.6%

The table above shows the average correct percentages for all levels of tests. N-gram yielded 20% correct percentage, which is lower than that of random guessing, while MVR yielded 26.3%, which is slightly better than random guessing. The N-gram with random guessing method yielded the highest average correct percentage of 31.8%, even which is not much of an improvement from random guessing.

The training of MVR on Project Gutenberg texts took approximately 10 to 12 hours. Once the MVR was trained, the average time our system took to answer each question was around 3 seconds for both N-gram and MVR methods. Provided that the main purpose of our system is to answer a question as accurately as possible, we deem the speed of our system to be sufficiently efficient.

6. FUTURE WORKS

Google has recently released the second version of Google N-gram, which contains more data and efficient indexing mechanism. Our system is based on the first version of Google N-gram, but updating to the second version can produce better results.

Another possible change to the n-gram methodology is using a different database. Building an n-gram database on Gutenberg library and using it to make predictions would align the training corpus between the two methodologies and produce results that are more appropriate for comparison.

As mentioned in the "System Performance" section, the training set for MVR compared to Google N-gram is much smaller. We believe that this is one of the reasons why MVR results are not much of an improvement from those of n-gram. Our system used a small subset of the Project Gutenberg library, not having access to the whole library and because of concerns regarding training time and network capacity. By training MVR on a bigger set of data, which is comparable to the size of Google N-gram, the MVR prediction would show much improvement and possibly perform significantly better than the n-gram method. The best approach would be to train MVR on the web, but that is unreasonable as well as implausible. Hence, possible future works in terms of MVR include using a larger set of documents as a training corpus. Such corpus can be comprised of a larger set of Project Gutenberg texts, on-line news articles, Wikipedia, or a combination of data mentioned.

Further, the set of tests currently available to test n-gram and MVR are too small to obtain significant prediction rates for both n-gram and MVR methods. Thus, it would be beneficial to find and test the system on more set of questions.

Finally, only two of many methodologies regarding context capturing are incorporated in our system. Adding method-

ologies such as HMM, LSI and LDA to the system may lead to better understanding of successful context capturing for reading comprehension tests.

7. CONCLUSION

Context capturing is key in answering cloze questions and other reading comprehension questions, which require the understanding of the “meaning” of passages. Because MVR captures both the local and broader context, while n-gram captures only the local context, MVR is a more attractive methodology for reading comprehension tests. Other methodologies not included in our system appears less attractive than MVR - HMM captures only the local context, while LSI and LDA captures only the document-level context, which in other words is the topic of the document.

The n-gram method led to mediocre results as the coverage was too low. The n-gram database used did not cover a significant portion of the questions tested, forcing n-gram to fail making predictions. The low coverage resulted from two aspects: the n-gram database was not extensive and higher gram data were even less extensive. Such aspects are understandable, however, as creating an n-gram database that covers most existing texts is extremely costly and time-consuming, if not impossible. Also, acquiring data for higher grams is even more difficult as there can be more variations involved in the sequence of words.

The MVR method was better than the n-gram method overall, but the result was not satisfying, slightly beating random guessing and n-gram. However, as mentioned in the “Future Works” section, improvement can be made with a larger training corpus. As MVR performed as well as, if not better than, n-gram with a less extensive training corpus, a better training corpus will lead to much better result over n-gram. Also, MVR showed stronger results than n-gram in higher level texts, which require more context capturing. Thus, we conclude that MVR is successful and is better than n-gram in capturing contexts.

Representing state as a real vector, as in MVR, is powerful, viewing “meaning” as a mixture of local and broader context. Also, linear models can be estimated quickly, which makes MVR more attractive. Thus, we believe that MVR is the most appropriate approach for answering reading comprehension tests and “reading” natural languages, which has numerous applications in artificial intelligence and linguistics.

8. REFERENCES

- [1] G. Aist. Towards automatic glossarization: Automatically constructing and administering vocabulary assistance factoids and multiple-choice assessment, 2001.
- [2] Rie Kubota Ando and Tong Zhang. Two-view feature generation model for semi-supervised learning., 2007.
- [3] J.E. Beck, J. Mostow, and J. Bey. Can automated questions scaffold children’s reading comprehension., 2004.
- [4] B. S. Hansler and J. Beck. Better student assessing by finding difficulty factors in a fully automated comprehension measure., 2001.
- [5] David R. Hardoon, Sandor Szedmak, and John Shawe-Taylor. Canonical correlation analysis: An overview with application to learning methods, 2003.

- [6] Christopher D. Manning and Hinrich Schutze. Foundations of statistical natural language processing, 1999.
- [7] J. Mostow and G. Aist. Evaluating tutors that listen: An overview of project listen, 2001.
- [8] X. Zhang, J. Mostow, and J.E. Beck. Can a computer listen for fluctuations in reading comprehension., 2007.