Emustru: User Guide

Manu Konchady

Contents

1. Introduction

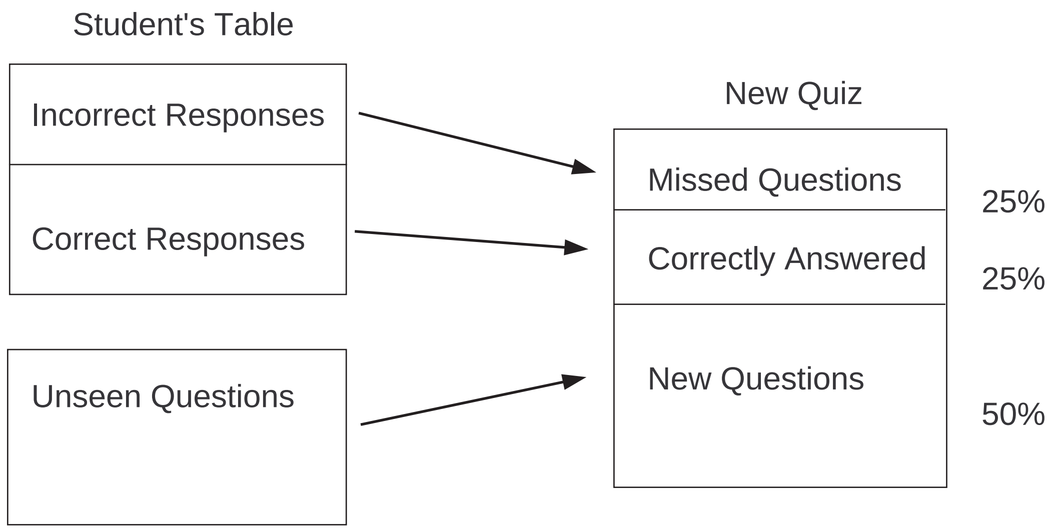
Emustru is an open source English language tutor with features to learn vocabulary, grammar, and essay writing using Computer Assisted Language Learning (CALL) [1]. CALL is a personalized approach to facilitate the language learning process. The two primary features of CALL are student-based or individualized learning and interactive learning. In a student-based lesson, the material shown to each student is adjusted based on prior performance. Students who are able to answer questions without any errors will be able to complete lessons faster than students who miss many questions. CALL is not a replacement for a teacher, but instead supplements the lessons taught in a classroom with personalized material.

CALL has several advantages. One, a student can learn at his or her own pace without the burden of keeping up with a class of students. Students from different backgrounds will have customized lessons that are adjusted to the level of each student. Two, a teacher can monitor the performance of a large number of students with CALL tools. Software can keep track of individual information such as the number of questions a student has missed as well as global information such as the most frequently missed question. Feedback indicating the type of questions that a class of 50 students or more have missed is much harder to identify manually. CALL software can keep track of a large number of parameters to monitor the performance of a class of students.

1.1. Emustru Quizzes

Emustru uses some of the philosophies behind the CALL approach to learn a language. A student learns vocabulary through dynamic quizzes that are generated based on prior performance. Sets of correct and incorrect responses per student are maintained in database tables. Emustru generates a custom quiz using some of the questions that were missed earlier, a set of new questions, and a set of questions that were answered correctly (see Figure 1.1).

Figure 1.1.: A Dynamic Student's Quiz Generated from a Database Table

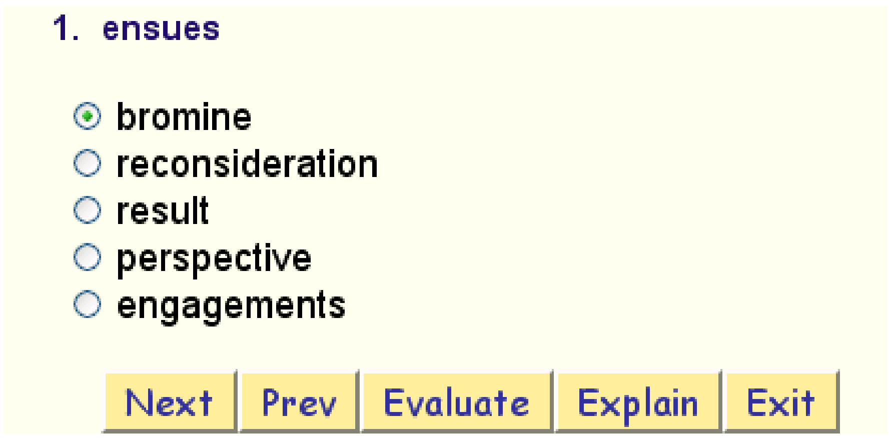


Often a student loses interest in a static quiz after the first attempt. The same questions with the same answers are repeated and it becomes relatively easy to recollect the correct answers. Dynamic quizzes have several advantages over static quizzes -

- A student uses a different set of questions in each session.
- Questions that have been missed are repeated till a student has correctly answered such questions more than once.
- Questions that were correctly answered can be repeated as many times as necessary to verify that a student did not answer a question correctly by chance.
- A new set of questions can be chosen by rank in every dynamic quiz.

The interactive question-answer format is a simple and attractive way to keep a student's attention during a session. Feedback is immediate and a student can verify answers through an Explain button (see Figure 1.2). The sample vocabulary question in Figure 1.2 includes a test word, five possible answers, and five buttons.

Figure 1.2.: A Sample Vocabulary Question



1. ensues

[✓ bromine]
reconsideration
result
perspective
engagements

One of the five options for the test word, *ensues* is correct. Emustru picks the test word by rank or at random from a given word list. The Next button will evaluate the current question and return the next question. The evaluation will indicate if the given answer was correct or not. The Prev button shows the previous question and answers. A question that was answered earlier cannot be modified. The Evaluate button is used to display the answer for the current question. Once this button has been pressed, the question is assumed to be answered and the student must continue to the next or previous question. The Explain button is only active following an evaluation of the question. This button returns a dictionary entry or a group of sentences that use the test word to describe how the word is used in context and its meanings.

1.2. Emustru Features

There are many sites [2] on the Web to learn vocabulary, grammar, writing, and reading skills for languages. The main skills a student of any new language would need include -

Listening: A student listens to an audio passage and answers questions to evaluate comprehension.

Writing: An essay prompt is provided and a student creates an essay of several hundred words in response.

Vocabulary: A good knowledge of vocabulary is important in speaking, writing, and comprehension.

Grammar: A student must understand the syntax of a language before writing sentences that are grammatically correct.

Reading: A student's grasp of the contents of a given audio passage is evaluated with a series of questions.

The evaluation of listening tests the recognition of accents, pronunciation, vocabulary, and comprehension. Reading is similarly evaluated, except that a student must know the alphabet and spellings of words. Most students learn a vocabulary of several thousand words in a language before acquiring a level of knowledge sufficient to pass competitive exams. Learning vocabulary is a fairly routine task and a computer is well-suited to make this task interactive and more attractive to a student. Two of the popular sites on the Web to learn vocabulary are Quizlet (`http://www.quizlet.com`) and FreeRice (`http://www.freerice.com`).

Learning the grammar of a language is more challenging than learning new words. Languages like English have many rules and exceptions that can only be learnt through practice. There are fewer grammar checker sites on the Web than sites for building vocabulary and other types of word games. A grammar checker examines the text of a document, one sentence at a time and returns errors and suggestions for corrections. Most popular word processors include a grammar checker that identifies syntax errors and generates potential corrections. Unfortunately, some grammar checkers miss sentences that should be marked as incorrect. This is usually an intentional feature to ensure that the percentage of flagged errors that are actual errors is high.

There are even fewer sites on the Web to evaluate writing skills. Some sites return a manual review of a text passage. This is of course more expensive than an automated evaluator and is more likely to be accurate than an automated evaluation. Many of the current automated text evaluators are proprietary of subscription-based.

1.2.1. Spelling

Emustru include features to learn some of the skills mentioned earlier. The spelling quiz selects words from a given word list, that has been optionally ordered by rank, and generates an audio file to "say" the word. The open source speech synthesizer FreeTTS (`http://freetts.sf.net`) was used to generate the audio file. In some cases, the spoken word is not easy to recognize.

1.2.2. Vocabulary

A word is selected from a pre-defined or user-provided word list. The meaning is extracted from the WordNet [3] dictionary. Some words have more than one meaning and just two of the most popular meanings are selected for a quiz. Several words that are unrelated to the given word are added to the list of answers. A student selects the meaning of a given word from a list of five options.

Emustru includes several word games including Hangman, jumbled words, and partial words. Two lesser known games are finding the most likely word before or after a given word. For example, the word *strong* is more likely to be seen before the word *tea* than the word *powerful*, even though both words have the same meaning. In another game, the student must identify the type of relationship (synonym, antonym, or hypernym) between two or more given words.

1.2.3. Sentence Analysis

The Cloze (`http://en.wikipedia.org/wiki/Cloze_test`) test is a test where some words of a sentence are removed and the student must identify the missing words from a set of given words. This test evaluates vocabulary and knowledge of words in context. For example, the following sentence has two missing words and a set of five choices.

> For his eighth grade project, Ebright tried to find the cause of a ________ disease that kills __________ all monarch caterpillars every few years.

- neighborhoods, crudeness
- viral, nearly
- dilemmas, container
- tongued, unfolding
- deceleration, maneuvered

The words that are missing in the sentence are selected from a pre-defined or custom word list. A student learns the context in which words chosen from the word list are used in sentences. This test complements the earlier vocabulary test where a student learnt the meaning of words.

1.2.4. Grammar

Most word processors include a grammar checker along with a spell checker to help the writer create a document that has correct syntax and spelling. In general, a grammar checker limits the number of *false positives*, i.e. the number of flagged errors that are not valid. A writer is more likely to be annoyed by a grammar checker that identifies errors in correct sentences and may be willing to tolerate error sentences that are not detected.

Emustru uses a statistical rule-based grammar checker to find errors. A large number of rules are constructed after observing part of speech tag and word patterns in a corpus that is known to contain sentences with valid syntax. These patterns are encoded in rules and stored in database tables. The grammar checker compares patterns extracted from a test sentence with patterns saved in tables. Any pattern that is rare or unusual is flagged as a potential error. The grammar checker in Emustru is included the essay writing evaluation function.

1.2.5. Essay Writing

The essay evaluation function in Emustru assigns a score based on a number of extracted features from a short essay of about 300-400 words. Many of the current competitive exams such as the SAT and IELTS include an essay writing question to test an examinee's vocabulary, grammar, and writing skill. Although, it is debatable whether writing an essay in a short period of half an hour or less can actually test an examinee's creativity and writing skills, the essay writing question has become popular.

Essay writing is usually the only free-form question in competitive exams, that allows unstructured text answers. Most of the other types of questions are multiple choice questions that can be machine graded. A human grader evaluates an essay and assigns a score, say from 1 (Poor) to 6 (Excellent) based on an overall impression of the essay. The human grader looks for grammatical mistakes, spelling errors, language usage, and several other features to compute an overall score. Two or more human graders may score the same essay to resolve errors that may arise during the grading process. When the score from two graders for the same essay differs by more than one, a third grader scores the essay. The Educational Testing Service (ETS) has successfully replaced one of the two human graders with an automated essay evaluator, E-rater([4]). In more than 90% of the graded essays, the absolute difference between a human grader's and the E-rater score were within one point.

The essay evaluator used in Emustru extracts features such as the number of spelling errors, number of unique words, number of grammatical mistakes and several other features to generate an overall score. The list of over 20 features is described in Chapter 5.

1.3. Why use Emustru?

A manually generated quiz will typically be superior to a similar automatically generated quiz. The questions and answers in a manual quiz are carefully selected and verified. A dynamic quiz attempts to reproduce this process using an algorithm. Test writers are known to create questions and answers in fairly standard patterns. Wrong answers are generated in a somewhat predictable manner. Emustru uses a few simple heuristics to automatically generate a quiz based on some observations from a manual quiz. Some of the advantages of Emustru and dynamic quizzes are listed below.

- Dynamic quizzes are well-suited to prepare for competitive exams such as the SAT. A list of words to prepare for these exams is fairly long and the use of dynamic quizzes makes it easy to generate questions ordered by word rank. Further, a quiz can be tweaked to repeat certain questions that the student found difficult.

- Some of the sites that have developed CALL software are subscription-based or proprietary. Emustru is open source software and the data sources can be customized to suit individual requirements.

- The Web interface of Emustru is intuitive and can be used without an Internet connection

- The Emustru essay evaluator is one of the few open source alternatives to commercial software such as Criterion (`http://criterion.ets.org`), Intelligent Essay Assessor (`http://www.knowledge-technologies.com/prodIEA.shtml`), and Intellimetric (`http://www.vantagelearning.com/school/products/intellimetric/`).

- The statistical grammar checker included with Emustru can be modified to find fewer or more errors in text.

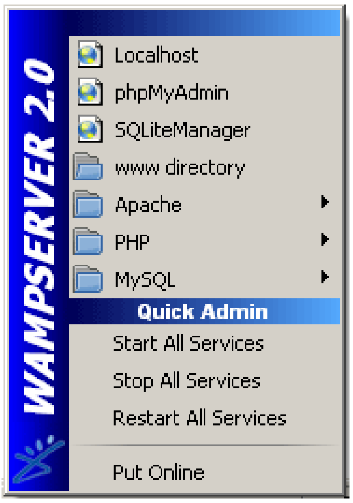
1.4. Installation

Emustru has been tested on the Windows and Linux platforms. The application is Web-based and runs on the Linux-Apache-MySQL-PhP (LAMP) or the Windows-Apache-MySQL-PhP (WAMP) stacks. Several applications, such as the open source course software Moodle, have been implemented on the LAMP/WAMP stacks.

1.4.1. Windows

This document will assume you have an existing stack on either the Windows or Linux platforms. The WAMP project (`http://www.wampserver.com/en/`) distributes the three components of the stack - Apache, MySQL, and PhP. The WAMP distribution makes it simple to install the stack without downloading and customizing each of the individual components (see Figure 1.3).

Figure 1.3.: Configuring Apache, MySQL, and PhP with WAMP



Apache and MySQL run as services and must be started before installing Emustru. The Administrative Tools of the Control Panel includes options to enable these services at startup time. The default directory for WAMP is `c:\wamp` and the `www` sub-directory under this directory is the location for Web projects. The Emustru distribution can be unzipped in the `c:/wamp/www` directory. A default `index.php` file is created in the `www` directory and can be viewed from the browser at the URL `http://localhost/index.php`.

Initially the MySQL root userid may be created without a password. This is a potential security problem and you can set a password for the root userid from the command line with the following commands.

```
C:\wamp\bin\mysql\mysql5.0.51b\bin> mysql mysql
>update mysql.user set Password=PASSWORD('MyNewPass') where
   User='root';
>flush privileges;
```

Replace `MyNewPass` with a password for the root userid. WAMP includes the `phpmyadmin` tool under the `apps` directory to manage the MySQL database tables. This is a very useful tool to troubleshoot problems with database tables and is fairly easy to use. The root MySQL password must be set in the `config.inc.php` file under the `phpmyadmin` directory.

```
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = 'MyNewPass';
```

You can verify your installation by starting a browser session at `http://localhost/index.php`. If both, MySQL and Php appear to be working, unzip the Emustru distribution in the WAMP `www` directory. Then, open a browser session at `http://localhost/emustru` and continue as shown in section 1.4.3.

1.4.2. Linux

Many of the current Linux distributions include options to install a Web server (Apache), a database server (MySQL) and PhP. If you have not installed these components, then you can either install a separate package XAMPP, use the distribution to add these components, or download each of the options separately.

The XAMPP (`http://www.apachefriends.org/en/xampp.html`) project is a multi-platform tool to build the AMP stack on Linux, Windows, MacOS, and Solaris platforms. XAMPP includes the same components as WAMP and a few others as well. On Linux, the XAMPP distribution is a gzipped tar file, that can be unzipped in an `/opt` directory. You will need to become the `root` user to complete the rest of the installation. After unzipping the distribution, you can start Apache and MySQL with the `"lampp start"` command from the top level installation directory. This command will start Apache and MySQL if existing servers are not running on the same ports.

Before starting XAMPP, you should stop any existing Web or database server to avoid conflicts. The `/etc/init.d` directory may contain the scripts to start and stop other Web and database servers. If you decide to make the XAMPP installation override any existing Apache and MySQL installation, you can modify the startup and shutdown scripts to start both servers from the XAMPP directory alone.

There are several security problems that need to be fixed before running the servers. The command `"lampp security`” sets passwords to access the Web pages, the MySQL database, a FTP server, and the Phpmyadmin tool. The script will also limit network access to the MySQL server by modifying the `my.cnf` file in the `etc` directory.

The root Web directory is the found in the `htdocs` directory under the installation directory. Emustru should be unzipped in this directory during installation. The installation can be verified by starting a browser session pointing to the URL `http://localhost`. You should see a page with an orange background and a number of menu options.

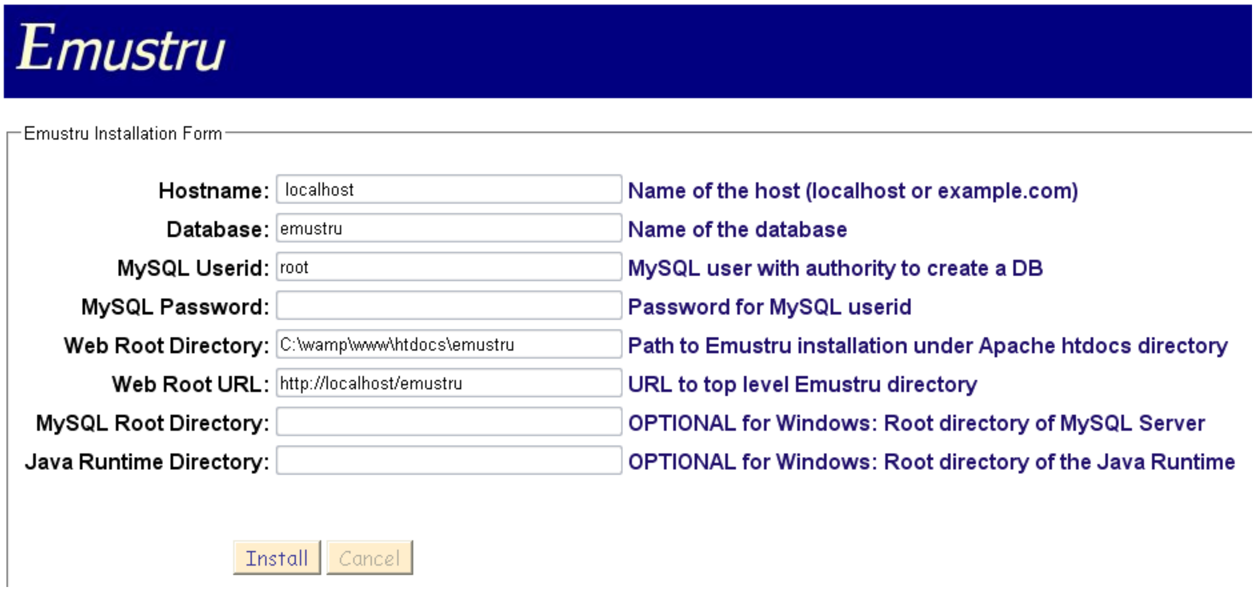
1.4.3. Emustru

The screen shown in Figure 1.4 should appear, if the Emustru distribution has been unzipped under the `htdocs` directory, from a browser session with the URL at `http://localhost/emustru/index.php`. This screen is common to Linux and Windows installations.

The installation screen in Figure 1.4 is based on a Windows installation. A Linux installation is similar with the exception of the entries for the Web root directory, MySQL root directory and Java Runtime directory. The MySQL userid should have the authority to create a database and load tables. The `mysqlimport` utility found in the `bin` directory under the root directory of the MySQL server, is used as a backup if the `load table` command fails. In Windows, the `java` executable is found from the environment variables that are set when Java is installed. However in Linux, the `java` executable may not be found in the `PATH` variable and therefore the runtime directory (i.e. the directory above the `bin` directory) may be required to run Java code.

During the installation, about 30 database tables are loaded and two configuration files - `config.php` and `config.prp` for PhP and Java respectively, are created. The configuration files should be made read only after a successful installation since these files contain a userid and password for the MySQL server. Both configuration files are first created in a temp directory. In Windows, the temp directory may be `c:/WINDOWS/TEMP/emustru` or `C:/WINNT/TEMP/emustru` and in Linux it may be `/tmp/emustru`. The temporary files are copied to the Web root installation directory.

Figure 1.4.: Emustru Installation Screen



Emustru

Emustru Installation Form

Hostname:	localhost	Name of the host (localhost or example.com)
Database:	emustru	Name of the database
MySQL Userid:	root	MySQL user with authority to create a DB
MySQL Password:		Password for MySQL userid
Web Root Directory:	C:\wamp\www\htdocs\emustru	Path to Emustru installation under Apache htdocs directory
Web Root URL:	http://localhost/emustru	URL to top level Emustru directory
MySQL Root Directory:		OPTIONAL for Windows: Root directory of MySQL Server
Java Runtime Directory:		OPTIONAL for Windows: Root directory of the Java Runtime

In Linux, this is usually a problem, since the Web user (such as `nobody`, `www`, or `apache`) does not have the authority to create files in the Web directories. The Linux installation will end with the following message -

- The `config.php` could not be moved to the `/opt/lampp/htdocs/emustru` directory because of permissions.
- To complete the installation, you will need to copy the file
- 1.`/tmp/emustru/config_temp.php` to `/opt/lampp/htdocs/emustru/config.php` and
- 2. `/tmp/emustru/config.prp` to `/opt/lampp/htdocs/emustru/java/data/config/config.prp`
- `cp /tmp/emustru/config_temp.php /opt/lampp/htdocs/emustru/config.php;`
- `cp/tmp/emustru/config.prp /opt/lampp/htdocs/emustru/java/data/config/config.prp`

This installation assumes that MySQL and Apache have been installed under the `/opt/lampp` directory. After copying the configuration files to the `/opt/lampp` directories, the login page for Emustru will be shown.

1.4.4. Customization

The default distribution comes with a word list of about 8,000 words and 6,500 sentences. Two additional sources of sentences and words can be downloaded from SourceForge.net - `brown.zip` and `sat.zip`. The `brown.zip` file contains 25,000 words and 35,000 sentences from the Brown corpus [5]. The `sat.zip` file contains 8,500 words and 120,000 sentences extracted from e-books downloaded from the Project Gutenberg [6].

Unzip both of these files in the `install/table_data` directory of the installation directory. Then login as `admin` (initial password `admin`) and press the "Load Word Table" button shown in Figure 1.5.

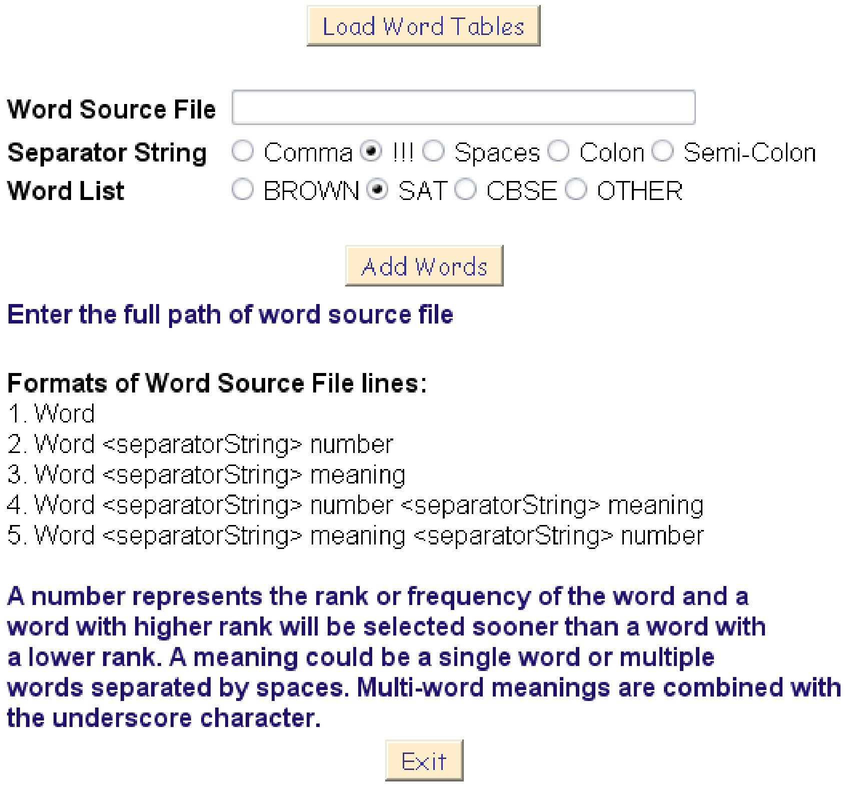
You can also add a list of words to one of the word list types. Emustru will accept a file with one word per line in several formats. An optional number accompanying the word is interpreted as a rank and words with higher ranks will be shown earlier in quizzes than other words in a generated quiz. If no ranks are provided, all words are assigned the same rank and a rank order quiz for such a word list, will fetch words in alphabetic order. The words for questions in any quiz can be selected at random or by rank order. An option to select an order type for the quiz is provided before a quiz is generated.

1.4.5. Troubleshooting

The Java code uses a JDBC connector to access the MySQL database and will not function if network access is disabled in MySQL. Network access is set through the skip-networking option in the `my.ini` file. During a fresh installation, you may need to clear out any existing log and configuration files from the temporary directory.

- There are several log files that contain messages indicating problems with the installation or running of Emustru.
  - The emustru.log file in the temporary directory contains messages from problems found in the PhP or Java code.

Figure 1.5.: Adding Words and Sentences to Emustru



Word Source File	
Separator String	○ Comma [✓ !!!] ○ Spaces ○ Colon ○ Semi-Colon
Word List	○ BROWN [✓ SAT] ○ CBSE ○ OTHER

Enter the full path of word source file

Formats of Word Source File lines:
1. Word
2. Word <separatorString> number
3. Word <separatorString> meaning
4. Word <separatorString> number <separatorString> meaning
5. Word <separatorString> meaning <separatorString> number

A number represents the rank or frequency of the word and a word with higher rank will be selected sooner than a word with a lower rank. A meaning could be a single word or multiple words separated by spaces. Multi-word meanings are combined with the underscore character.

  - Entries in the Apache error log file, the MySQL log file, and a PhP error log file may contain useful information to debug a problem.

- The essay evaluate function starts a shell script from PhP to run the Java code and can be found in the temporary directory.

- Similarly, the other Java functions are run from PhP using the `shell_exec` command which may not work if PhP is operating in safe mode.

- Finally, directory permissions in Linux are often a source of installation problems. Permissions and files left over from a previous installation may cause problems in a new installation since some files cannot be removed.

2. Learning Vocabulary

Before writing sentences, we need to know vocabulary to accurately express meaning and convey messages to the reader. The English language has a vast vocabulary (roughly half a million words) with many words adopted from other languages and a complex set of rules and exceptions to build words. There is no easy way to learn new words without studying each word separately. Some words can be deciphered from their roots, but other words are harder to understand.

Fortunately, it is not necessary to learn a large number of words to write effectively in English. Most competitive exams test from a word list of 10,000 to 20,000 root words. However, even learning a list of 10,000 words is time consuming and routine. A computer based system can make the task of learning new words more interesting.

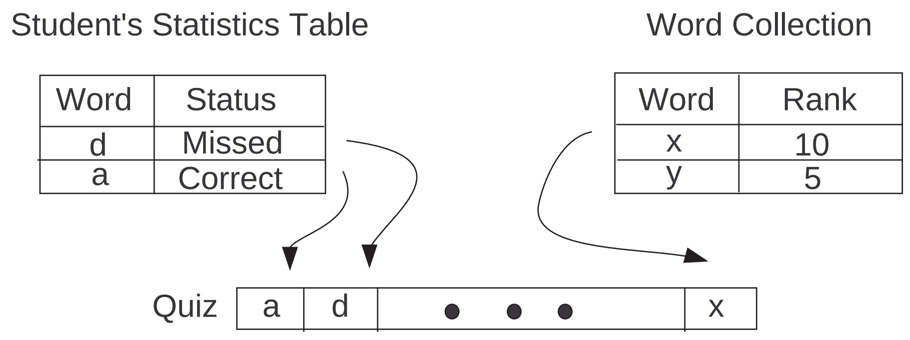
Quizlet™ [7] is a Web-based system to learn words and their associated meanings. A student uploads a list of words and associated meanings. Quizlet generates a set of quizzes with questions generated at random to test the student. The type of questions include multiple choice, true or false, and free form-based. The tedious job of learning a word list is less tiresome with Quizlet.

Emustru includes a set of word games, a spelling quiz, and other word related quizzes. Beyond making the study of words more interesting, the use of computer-based learning can dynamically generate quizzes that are customized for a student. In general, we can expect students at various skill levels in a random group. Most students may fall in a mid-level range, a few students at an advanced level, and the remaining students at a lower level. The students at an advanced level may prefer to take quizzes with a large proportion of new words, while students at a lower level may be overwhelmed with words from the same quizzes. Therefore, it makes sense to repeat words for some students who appear to be having difficulty learning words, while other students will be more motivated to take quizzes with new words.

2.1. Learning Algorithm

The learning algorithm is based on prior performance. At first, all the words in a quiz are new words. Following a student's attempt at a quiz, a statistics table is populated with words that the student answered correctly or missed. In a successive attempt, some words from the statistics table are used in a generative quiz. A fraction (upto 25%) of the quiz may contain words that the student had missed in an earlier attempt (see Figure 2.1).

Figure 2.1.: Generating a Quiz using the Statistics Table and a Word Collection



Another fraction (upto 25%) of the quiz consists of words that the student answered correctly. Sometimes, a student may have correctly guessed the answer to a question and a second attempt will confirm if the student is sure of the answer. The `config.php` file in the Emustru root directory contains parameters to limit the number of question repetitions.

```
$CFG->repeat_hangman = '1';
$CFG->repeat_spelling = '1';
```

If the parameter `$CFG->repeat_spelling` is set to 0, then a word that was correctly spelled once will not be repeated in another quiz, otherwise, the same word will be repeated n (based on the number in the `config.php` file) times in future quizzes. It may not be desirable to have the same words appear in two consecutive quizzes. The `$CFG->qinterval` parameter sets the interval between repetitions of the same word in a quiz type.

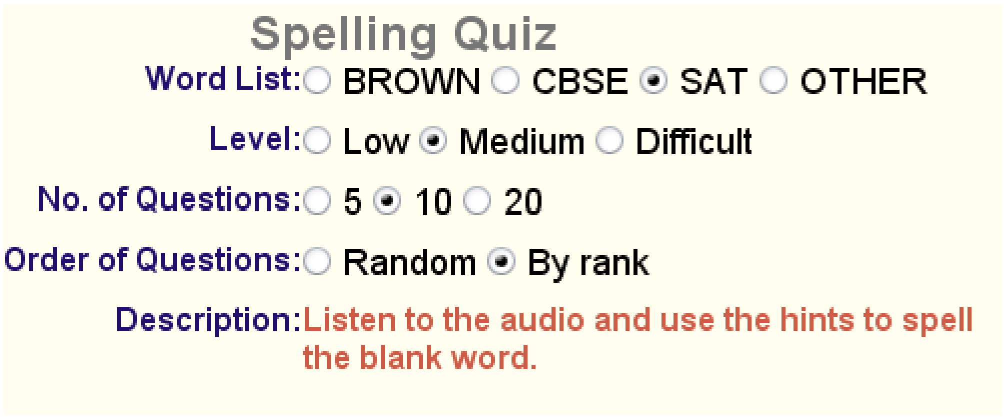
```
$CFG->qinterval = '30';
```

The default interval between word repetitions is 30 minutes. In other words, a word x that was misspelled in a quiz, will only appear in a successive quiz 30 minutes after the current quiz. The interval parameter can be adjusted in the `config.php` file.

2.2. Quiz Types

Emustru includes a few options to customize a quiz (see Figure 2.2). New words can be selected by rank order or at random from a word collection. When words are selected by rank, the same words in order will appear in other quizzes. For example, the words generated in a spelling quiz in rank order will be identical to the words of a hangman quiz also generated in rank order. It may be easier for a student to learn new words, if the same words appear in different quiz types, reinforcing the meaning and spelling of the words.

Figure 2.2.: Quiz Options



Spelling Quiz

Word List: BROWN CBSE [✓ SAT] OTHER

Level: Low [✓ Medium] Difficult

No. of Questions: 5 [✓ 10] 20

Order of Questions: Random [✓ By rank]

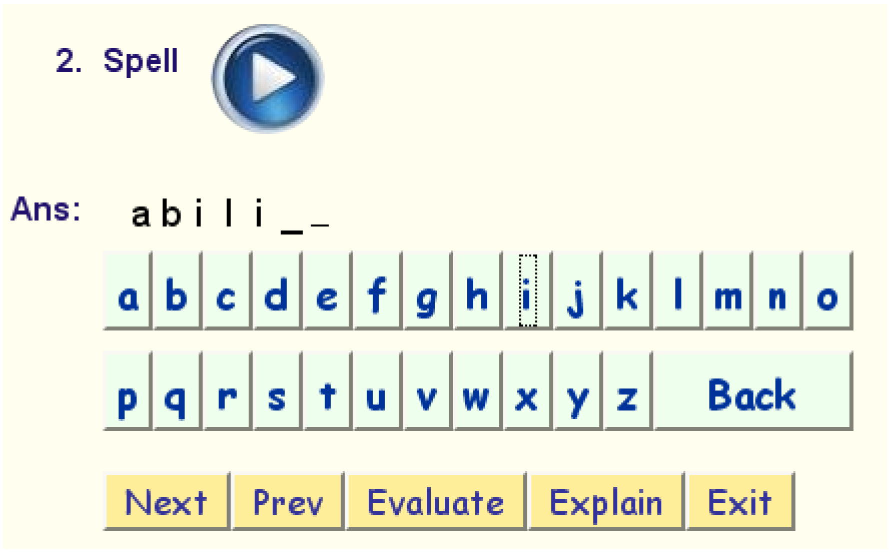
Description: Listen to the audio and use the hints to spell the blank word.

The number of questions in a quiz is also specified in the same screen. The level of difficulty defines the maximum length of a word in the quiz. A quiz of *medium* difficulty will have words less than eight characters long, while a difficult quiz will have words less than 24 characters long.

2.3. Learning Spelling

A spelling quiz is built using the specified word list and other options specified in Figure 2.2. Each question consists of a single word with a keyboard like interface to enter the characters of the word on a browser screen (see Figure 2.3).

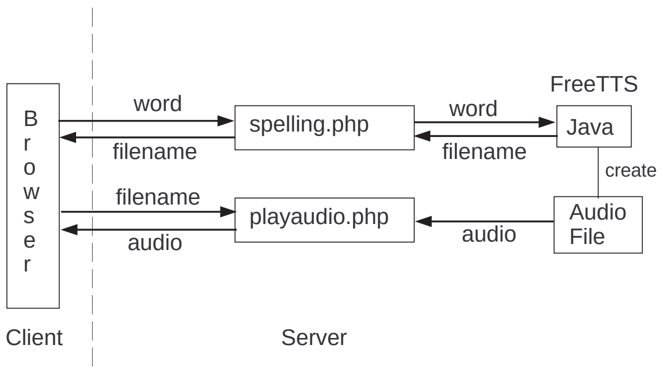
Figure 2.3.: A Spelling Question with a Link to an Audio File


2. Spell
Ans: a b i l i _ _
a b c d e f g h i j k l m n o
p q r s t u v w x y z Back
Next Prev Evaluate Explain Exit

The keyboard consists of the 26 letters of the alphabet and the Back button. Clicking on the Play button sends a generated audio file to the browser (see Figure 2.4). The FreeTTS speech synthesizer software (`http://freetts.sf.net`) is used to generate the audio file for the word. In some cases, the quality of the audio is good enough to hear a word very clearly, but in other cases, the pronunciation is difficult to decipher. A few hints are included at the bottom of the screen (not shown) Figure 2.3 to assist a student.

The audio file is generated in the WAV format, that most browsers can play with a plug-in for audio files. An alternate method is to generate a word file for each word

Figure 2.4.: Generating an Audio File for a Spelling Word



and save the collection of files in the server. This is of course faster than calling the FreeTTS Java library to generate an audio file.

The Explain button can only be pressed after a given answer has been evaluated. A pop-up window with two tabs is shown when the Explain button is pressed. The first tab returns the WordNet meanings of the word and the second tab runs a Java program to find sentences that contain the word. Every word list type has an associated searchable index of sentences. If a sentence is not found in the index for a particular word list, the indexes of other word list types will be searched in sequence. The pop-up window for word meanings and example sentences is common in several quizzes.

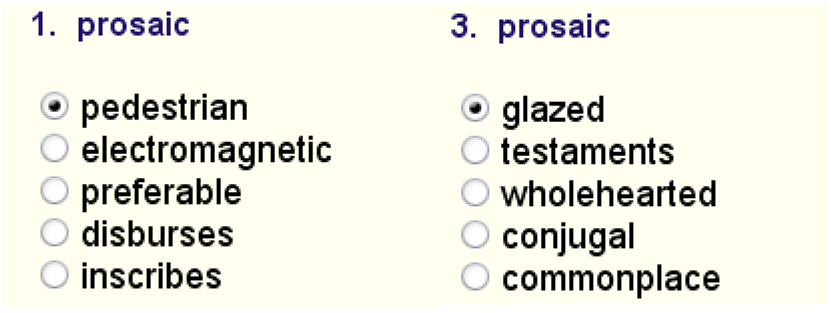
2.4. Vocabulary

An individual possessing a large vocabulary finds it easy to precisely express meaning in a sentence. Choosing the right word to convey an emotion or thought is critical in Essay writing (see Chapter 5). Besides, a large vocabulary also helps a reader understand a passage. Both, reading and writing are critical skills that an examinee needs to score well in a competitive exam.

Many books have been written on techniques to increase one's vocabulary. Most of the techniques used include a worksheet-like format with a number of questions to test the reader. In most cases the computer's role has been to primarily to show the same questions on a monitor and tabulate the results. Emustru is different in that all questions are dynamically generated and customized for a particular student.

However, word selection in a vocabulary quiz is slightly more complicated than in a spelling quiz. Some words have more than one meaning and a single question can only test for one meaning. It is common to find a less frequently used meaning of a word being tested in an exam. Emustru picks the top (most frequently interpreted) two meanings of a word in a quiz.

Figure 2.5.: One Question per Word Meaning



1. prosaic

[✓ pedestrian]
electromagnetic
preferable
disburses
inscribes

3. prosaic

[✓ glazed]
testaments
wholehearted
conjugal
commonplace

Since two questions with the same word and different meanings in the same quiz can be confusing to the student, the questions are separated by the question interval, $CFG->qinterval. The four words that are incorrect in the list of five possible meanings are selected such that there is no overlap with the correct meaning.

For example, all the meanings of the word *prosaic* in Figure 2.5 must be excluded from the list of four misleading meanings. A hyponym of the word is used as one of the four meanings to make the selection of the correct meaning non-trivial to the student. The remaining three meanings are selected at random based on the suffix or prefix of the word.

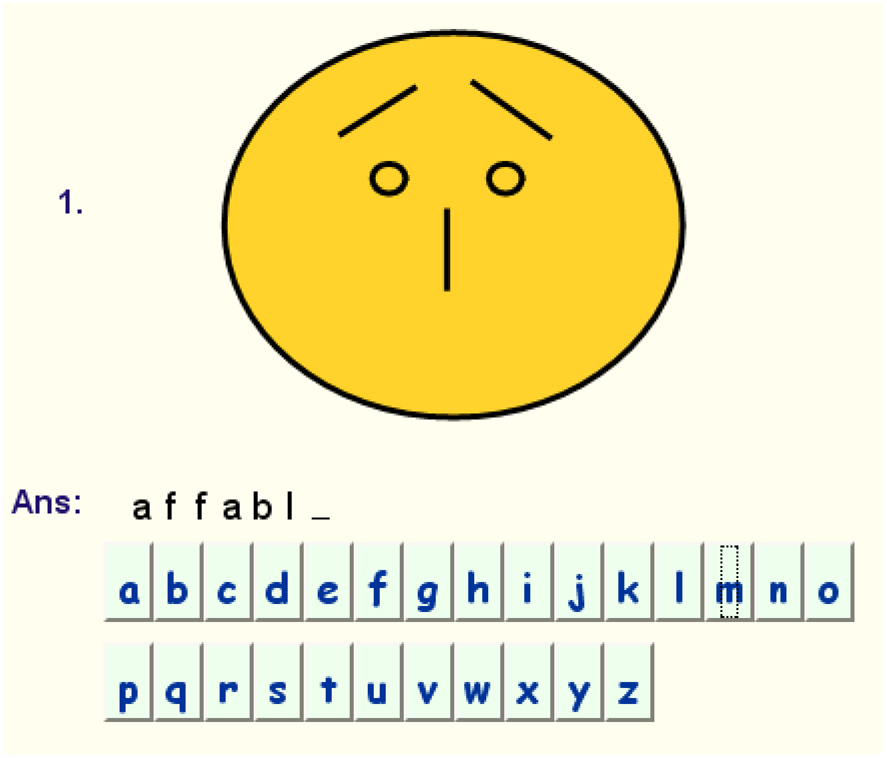
2.5. Word Games

Emustru includes several word games that use the same word lists to learn vocabulary. Words are selected as before - by rank, difficulty, and word list type. (see Figure 2.2).

2.5.1. Hangman

This is a fairly well known game to find a word within n chances. Letters are selected from a screen-based keyboard and shown in the word, if they appear in the word. Vowels and a few consonants such as r, s, t, and n are the most frequent letters in words.

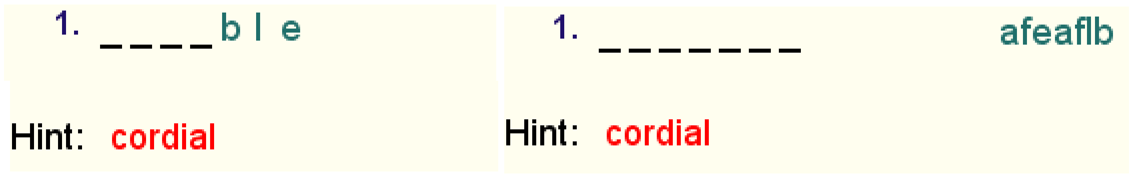
Figure 2.6.: Hangman Word Game



2.5.2. Partial and Unscramble Word Games

The partial word game is a similar game with a few letters of the word that are shown (see Figure 2.7). The letters that are shown are 2 or more consecutive letters from the beginning, middle, or end of the word. The student has to complete the remainder of the word. The meaning of the word is shown as a hint.

Figure 2.7.: Partial Word and Unscramble Questions for the word Affable

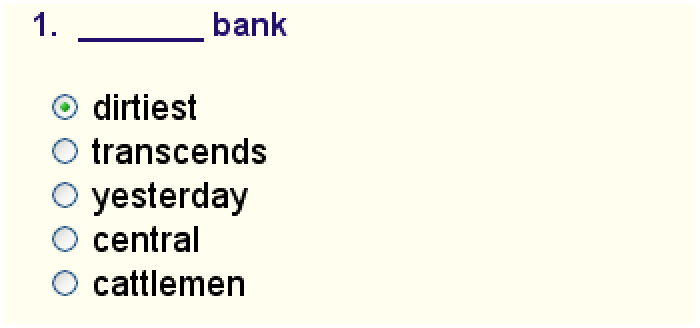


The right hand side of Figure 2.7 has the equivalent unscramble question for the same word. The letters of the word are jumbled and the student must enter the letters of the word in the correct order.

2.5.3. Following Word Game

Roughly 5700 popular two word phrases were selected from the Brown corpus and saved in the `wn_phrases` table. The first or second word of the phrase is shown in a question and the student must guess which is the most likely following or preceding word respectively (see Figure 2.8).

Figure 2.8.: Guess the Preceding Word

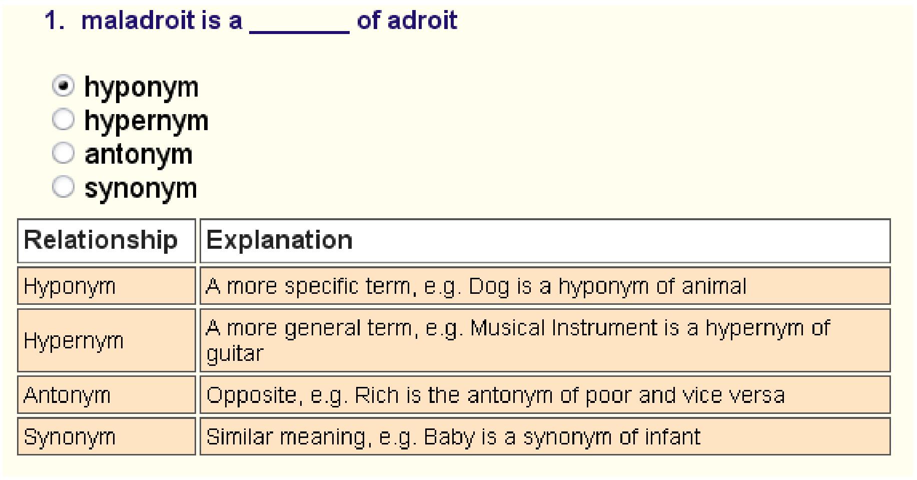


The second word *bank* of the two word phrase *central bank*, is shown with a list of possible preceding words. The purpose of this game is to learn phrases such as *strong tea* that is more likely than a similar meaning phrase, *powerful tea*. Roughly half of the questions in the quiz will show a preceding word and the remainder the following word of a phrase.

2.5.4. Word Relationships

This game is based on some of the word relationships defined in the WordNet [3] dictionary / thesaurus. Two sets of words that are related by one of four relationships are shown in a question (see Figure 2.9). The four relationship types are hyponyms, hypernyms, synonyms, and antonyms.

Figure 2.9.: A Word Relationship Question



1. maladroit is a _______ of adroit

● hyponym
○ hypernym
○ antonym
○ synonym

Relationship	Explanation
Hyponym	A more specific term, e.g. Dog is a hyponym of animal
Hypernym	A more general term, e.g. Musical Instrument is a hypernym of guitar
Antonym	Opposite, e.g. Rich is the antonym of poor and vice versa
Synonym	Similar meaning, e.g. Baby is a synonym of infant

A word x maybe related to more than one word in a single relationship. For example, the word *affable* has the following synonyms *amiable*, *cordial*, and *genial*. The antonym relationship is defined between a single word x and another word y. This game is surprisingly difficult since it requires a student to think of word meanings in terms of relationships and is more abstract than the previous games.

3. Building Sentences

A sentence is a sequence of words such that the order of words is syntactically valid and is intelligible. Consider the order of a set of seven words in the following four examples.

1. Moon beautiful there is a tonight out.
2. A beautiful moon is there out tonight.
3. Tonight there is a beautiful moon out.
4. There is beautiful moon out tonight.

The last sentence is both syntactically valid and intelligible while the first three sentences are non-sensical. In this chapter and the following chapter, the syntax of language alone is studied. It is quite simple to make non-sensical sentences such as "Colorful ideas sleep furiously" that are yet syntactically valid. Deciphering if a sentence is intelligible requires a large knowledge base to ascertain the meaning of a sentence which cannot be extracted from the sum of the meanings of the individual words.

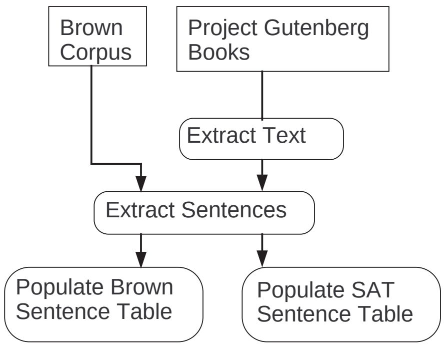
Sentences are constructed using a complex set of rules that are not described here. Instead, patterns and usage of words in sentences is learnt from a sentence collection extracted from books, articles, and other text documents. There are many sources of text documents and Emustru uses two main text sources - The Brown Corpus [5] and the Project Gutenberg [6].

3.1. Extracting Sentences

The Brown corpus is a set of roughly 35,000 sentences collected in the 1960s from various text genres. The corpus is divided into 500 files. The total number of words is 1.1 million, of which roughly 25,000 were unique. Each of the sentences in the corpus is printed on a single line of a file. This of course make sentence extraction from the Brown corpus trivial. Every word of every sentence is also part of speech (POS) tagged. The Brown corpus uses a tag set of about 100 POS tags.

The project Gutenberg is a large collection of e-books written in plain text or formatted in Adobe PDF and Microsoft Word formats. The text extracted from a plain text file for an average e-book will contain a large collection of sentences. However, there are no defined sentence boundaries to demarcate sentences. Instead, the most likely sentences are extracted using a heuristic algorithm. A period character is the most widely used sentence separator, but there are exceptions. For example, the middle initial of a name contains a period, the letters of an acronym are separated by periods, and more recent words such as *.com* and *SourceForge.net* contain embedded periods. The sentence extractor in Emustru uses a heuristic algorithm from LingPipe[8] to find sentences (see Figure 3.1).

Figure 3.1.: Extracting Sentences from the Brown Corpus and Project Gutenberg



The sentence tables in Emustru contain the sentence, the POS tags for each word of the sentence, and the source of the sentence. The Lucene search engine API was used to construct separate indexes for each of the sentence tables. The indexes are searched for sentences that contain a given word. Matching sentences are shown as examples of word usage in a sentence.

3.2. Sentence Completion

This quiz tests the skill of a student in identifying the missing word from a sentence, given a set of words. The selective removal of words in a sentence is also called a *Cloze* test to measure a student's comprehension of a sentence and vocabulary (see Figure 3.2). The removal of words could be mechanical such as the deletion of every 5th word of a sentence or selective. In a selective deletion, words for removal are chosen from a list or based on some criteria.

3.2.1. Sentence Features

A sentence is first selected for a question and then one or two words are selectively deleted. The same sentence is not used more than once in a quiz and less than twice for any particular student. The use of different sentences not only makes a quiz more interesting, but also prevents students from answering a question by memorizing the sequence of words in a sentence.

Long sentences are not desirable for beginner students, since such sentences can be difficult to comprehend. A student can set a difficulty level to limit the length of a sentence. At the easy level, the maximum length of a sentence is set to 20 words and at the medium and difficult levels the maximum sentence length is 30 and 40 words respectively.

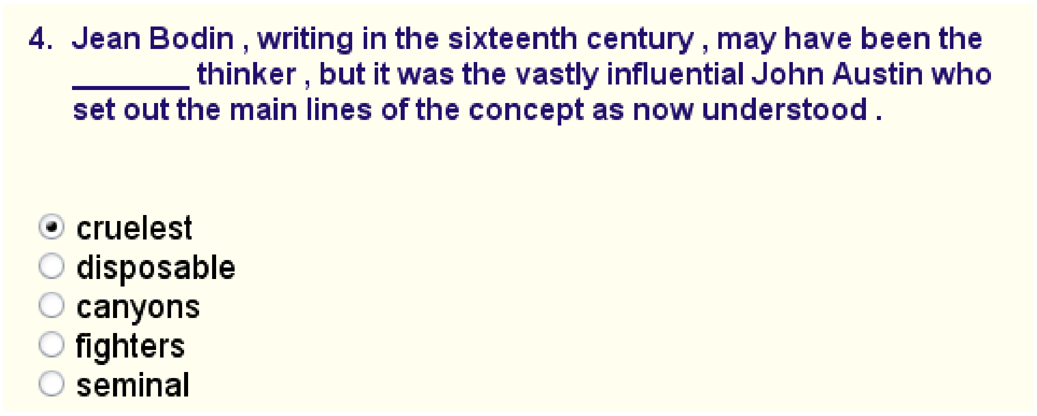
The sentences selected in a sentence must also have words from the associated word list type. Finally, a sentence with one of the following words - *not*, *but*, *although*, *however*, *despite*, *no*, *none*, *never*, *merely*, *always*, *often*, *gradually*, *sometimes*, *because*, *since*, *like*, *therefore*, and *so* is given higher priority than other sentences. These words are also known as discourse markers and are used to present information in a formal manner.

For example, the discourse markers - *regarding*, *as far as*, and *as for* may change the subject in the fragment that follows a marker. Similarly, markers like *however* and *despite* are used to present two contrasting ideas and words like *since* and *therefore* illustrate a subsequent statement that should logically follow a given statement.

3.2.2. Single Word Sentence Completion

This is a common test in language exams. A student uses the visible words of a sentence to guess the most likely word that was deleted based on the context and meaning of the sentence. On occasion, more than one word may be removed. A sentence question with two words missing may be harder to answer than a sentence with a single word missing.

Figure 3.2.: A Single Missing Word from a Sentence



4. Jean Bodin , writing in the sixteenth century , may have been the ________ thinker , but it was the vastly influential John Austin who set out the main lines of the concept as now understood .

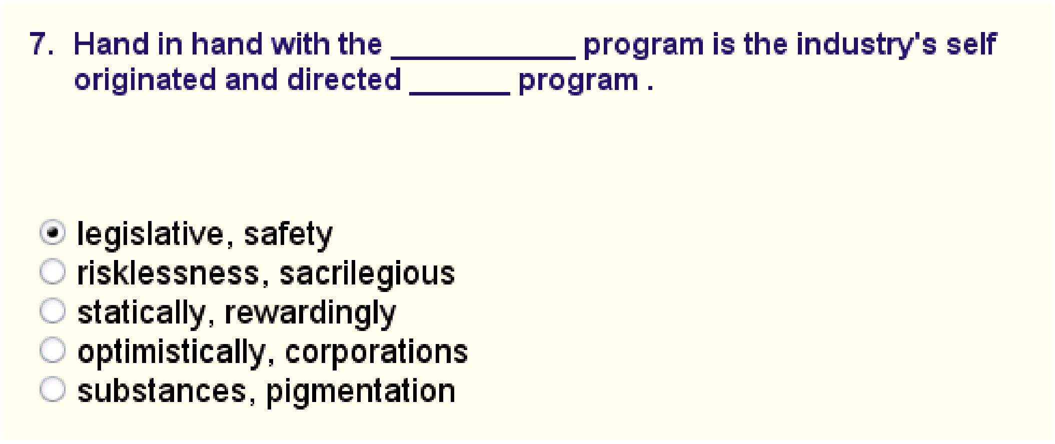
[✓ cruelest]
disposable
canyons
fighters
seminal

In Figure 3.2, the sentence presents two contrasting thoughts. The first part of the sentence describes a writer from the sixteenth century, while the second part of the sentence is about an influential person describing a concept. The correct word *seminal* is the most logical word to complete the sentence that compares two influential writers from different times. The other answer words are automatically selected by a function that first looks for a hyponym and then selects three other words that are not one of the synonyms and not *close* to the correct word. We define close in terms of the number of characters and operations needed to transform one word into another. These restrictions are necessary to avoid words that may be inflections of the correct word.

3.2.3. Double Word Sentence Completion

Two words omissions in sentences are not necessarily harder than single word omissions. However, an automatic question generator must carefully select the omitted words from a sentence (see Figure 3.3). In most cases, the two omitted words will be separated by one or more words.

Figure 3.3.: Two Missing Words from a Sentence



7. Hand in hand with the ___________ program is the industry's self originated and directed ______ program .

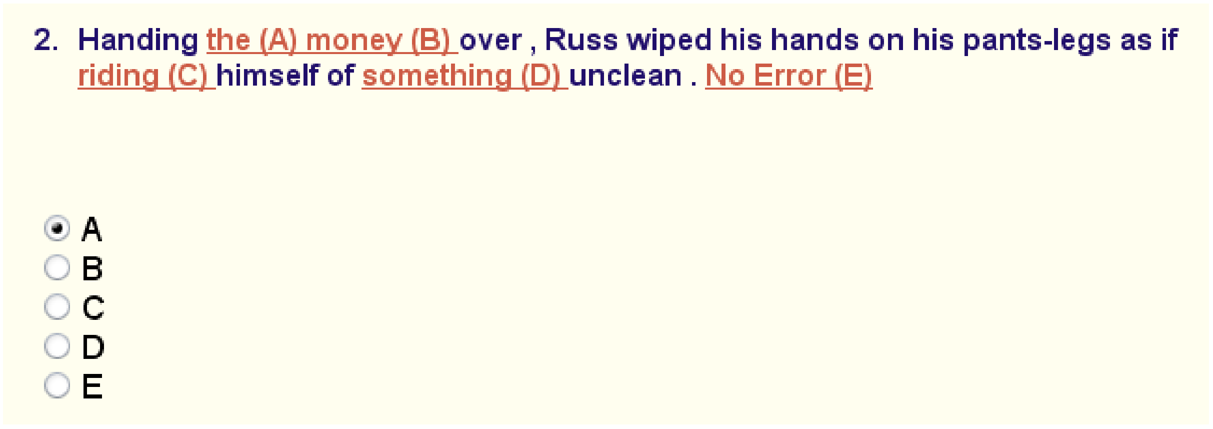
[✓ legislative, safety]
risklessness, sacrilegious
statically, rewardingly
optimistically, corporations
substances, pigmentation

The same strategy adopted for single missing words can be used here. The most likely first word is identified followed by a check for the correctness of the second word. Sometimes, the first word may be difficult to identify in the given choices and eliminating the wrong answers from the second word can lead to the correct answer.

3.3. Find the Error

These types of sentence questions present a sentence with selected fragments underlined that may be correct or incorrect. A set of four words are underlined in a sentence and one of these four words may be incorrect in the context of the sentence. The student needs to identify the incorrect word (see Figure 3.4). A few of the questions may contain zero errors, i.e. the sentence is correct as-is and does not need any modification. This is usually the 5th choice.

Figure 3.4.: Spot the Error in a Sentence



2. Handing the (A) money (B) over , Russ wiped his hands on his pants-legs as if riding (C) himself of something (D) unclean . No Error (E)

[✓ A] B C D E

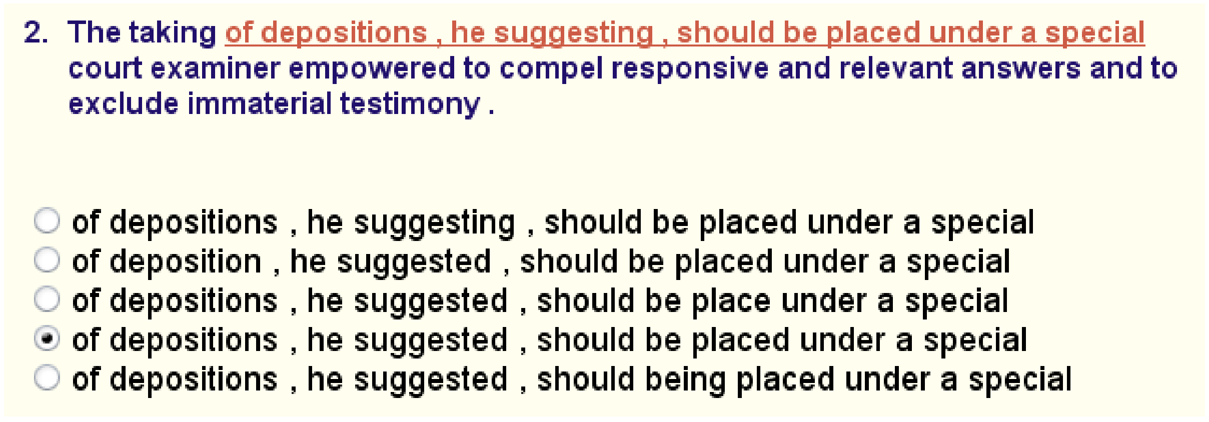
In Figure 3.4, four words have been underlined with the letters A through D. One of these words may be incorrect in the sentence. The last choice E is selected when the sentence appears to be error-free. In this example, the word *ridding* in choice C has been automatically replaced with the word *riding*. The student is not required to give the correct answer in this type of question and instead needs to merely select the word that is incorrect. Sometimes more than word may be underline in a single choice. For example, the two word phrases *himself of* or *as if* may be possible answer choices.

Emustru attempts to duplicate the human process used to generate such questions. First, a valid sentence is selected from a collection of sentences. In 20% of the generated sentences, no changes are made to the original sentence. In the remaining 80% of the sentences, a word is randomly selected and replaced with another word that shares a common stem. For example, the two words `riding` and `ridding` share the same stem `rid`. Similarly, the word *accepting* may be replaced with the past tense of the word *accepted*, since both words share the same root *accept*.

3.4. Correct the Sentence

Another type of sentence question contains a fragment of 5-10 words that is underlined (see Figure 3.5). The entire fragment may be correct as-is or incorrect.

Figure 3.5.: Select the Best Fragment to Complete the Sentence



2. The taking <u>of depositions , he suggesting , should be placed under a special</u> court examiner empowered to compel responsive and relevant answers and to exclude immaterial testimony .

- of depositions , he suggesting , should be placed under a special
- of deposition , he suggested , should be placed under a special
- of depositions , he suggested , should be place under a special
- [✓ of depositions , he suggested , should be placed under a special]
- of depositions , he suggested , should being placed under a special

The answer choices contain other sentence fragments that are corrections for the given sentence, if it appears to be incorrect in the given form. The fragment that has been underlined has been selected at random from a sliding window of 12 words. The correct fragment for the sentence is the fourth choice shown in the Figure 3.5. A few of the words in the remaining choices have been altered from singular to plural or vice versa.

4. A Grammar Checker

A grammar checker verifies free-form unstructured text for grammatical correctness. In most cases, a grammar checker is part of an application, such as a word processor. In this paper, a Web-based grammar checker is implemented to verify the correctness of short essays that are submitted by students in competitive exams. An essay can vary from a single paragraph to a medium sized document made up of several pages (~ 100 Kbytes).

The earliest grammar checkers in the 80s searched for punctuation errors and a list of common error phrases. The task of performing a full blown parse of a chunk of text was either too complex or time consuming for the processors of the early PCs. Till the early 90s, grammar checkers were sold as separate packages that were installed with a word processor. The software gradually evolved from a set of simplistic tools to fairly complex products to detect grammatical mistakes beyond a standard list of common style and punctuation errors.

While a grammar checker verifies the syntax of language, a style checker compares the use of language with patterns that are not common or deprecated. A style checker may look for excessively long sentences, out-dated phrases, or the use of double negatives. We have not considered style checking in this work and have focused on syntax verification alone. Further, there is no verification of semantics. For example, the sentence - "Colorless green ideas sleep furiously." was coined by Noam Chomsky to illustrate that sentences with no grammatical errors can be nonsensical. Identifying such sentences requires a large knowledge corpus to verify the semantics of a sentence.

4.1. Requirements

The main purpose of a grammar checker is to help create a better document that is free of syntax errors. A document can be analyzed in its entirety or one sentence at a time. In a batch mode, the entire text of a document is scanned for errors and the results of a scan is a list of all possible errors in the text. An online grammar checker will identify errors as sentences are detected in the text. Grammar checkers can be computationally intensive and often run in the background or must be explicitly invoked. One of the primary requirements for a grammar checker is speed. A practical grammar checker must be fast enough for interactive use in an application like a word processor. The time to analyze a sentence should be a few milliseconds or less following an initial startup time.

The second requirement is accuracy. A grammar checker should find all possible errors and correct sentences as well. There are two types of errors. The first type of error is a false positive or an error that is detected by the grammar checker but which is not an actual error. The second type of error is an actual error that was not detected by the grammar checker (a false negative). In general, the number of false positives are minimized to avoid annoying the user of the application.

The third requirement to limit the number of correct sentences that are flagged as errors by the grammar checker, is related to the second requirement. At first, we may assume that simply setting the threshold high enough for an error should be sufficient to satisfy this requirement. However, a grammar checker with a threshold that is too high will miss a large number of legitimate errors. Therefore, the threshold should be such that the number of false positives are minimized while simultaneously reducing the number of false negatives as well. The accuracy parameter defined in the Evaluation section combines these two attributes in a single value, making it possible to compare grammar checkers.

Since it is difficult to set an universal threshold that is appropriate for all situations, the user can select a level of “strictness” for the grammar corrector. A higher level of strictness corresponds to more rigorous error checking.

4.2. Emustru Grammar Checker

An online Web-based essay evaluator in Emustru accepts a chunk of text written in response to an essay type question, that elicits the opinion of the writer regarding an issue or topic. The essay evaluator uses the number of grammatical errors in addition to other parameters such as the use of discourse words, organization, and the number of spelling errors in the text to assign an overall evaluation score.

The essay evaluator returns a score and a category for the essay along with a list of parameters computed from the text of the essay. The grammar checker returns results for each sentence extracted from the text. A sentence that is incorrect is flagged and a description explaining the error along with a suggestion is given.

Sentence: My farther is fixing the computer.

Description: The tag an adverb, comparative is not usually followed by **is**

Suggestion: Refer to **farther** and **is**

The words in the sentence that are incorrect are highlighted. The description and suggestion are generated automatically based on the type and location of the error. The checker marks words in the sentence that is part of an error and subsequent errors due to the same words are ignored. Therefore, some sentences may need to be corrected more than once.

A ruleset is automatically generated from a tagged corpus and used to detect potential errors. This method is purely statistical and will be inaccurate when the tagged corpus does not cover all possible syntax patterns or if the tagged corpus contains mis-tagged tokens. Further, since the grammar checker uses a trained POS tagger, the accuracy of the checker is constrained by the correctness of the POS tags assigned to individual tokens of sentences.

Despite these inherent problems with a statistically-based grammar checker, the results are comparable with the grammar checker used in the popular Microsoft Word™ word processor. A sample corpus of 100 sentences made up of 70 correct and 30 incorrect sentences was used in the evaluation. The accuracy of the grammar checker can be adjusted using a likelihood parameter. The grammar checker has also been evaluated using the standard Information Retrieval recall and precision parameters.

4.3. Methods

Grammar checkers first divide a chunk of text into a set of sentences before detecting any errors. A checker then works on individual sentences from the list of sentences. Two tasks that are necessary in all grammar checkers are sentence detection and part of speech (POS) tagging. Therefore, this dependency limits the accuracy of any grammar checker to the combined accuracy of the sentence detector and POS tagger. Sentence detectors have fairly high precision rates (above 95%) for text that is well-written such as newswire articles, essays, or books. POS taggers also have high accuracy rates (above 90%), but have a dependency on the genre of text used to train the tagger.

Two methods to detect grammatical errors in a sentence have been popular. The first method is to generate a complete parse tree of a sentence to identify errors. A sentence is parsed into a tree like structure that identifies a part of speech for every word. The detector will generate parse trees from sentences that are syntactically correct. An error sentence will either fail during a parse or be parsed into an error tree.

One problem with this approach is that the parser must know the entire grammar of the language and be able to analyze all types of text written using the language. Another problem is that some sentences cannot be parsed into a single tree and there are natural ambiguities that cannot be resolved by a parser. The open source word processor AbiWord uses a parser [10] from Carnegie Mellon University to find grammatical errors.

The second method is to use a rule-based checker that detects sequences of text that do not appear to be normal. Rule-based systems have been successfully used in other NLP problems such as POS tagging [4]. Rule-based systems have some advantages over other methods to detect errors. An initial set of rules can be improved over time to cover a larger number of errors. Rules can be tweaked to find specific errors.

4.3.1. Manual Rule-based Systems

Rules that are manually added can be made very descriptive with appropriate suggestions to correct errors. LanguageTool [3] developed by Daniel Naber is a rule-based grammar checker used in OpenOffice Writer and other tools. It uses a set of XML tagged rules that are loaded in the checker and evaluated against word and tag sequence in a sentence. A rule to identify a typo is shown below.

```
<rule id="THERE_EXITS" name="Possible typo:  'There exits'
(There exists)">
```

```
<pattern mark_from="1"><token>there</token>
<token>exits</token> </pattern>
<message>Possible typo.  Did you mean <suggestion>
 exists </suggestion>?  </message>
<example correction="exists" type="incorrect"> There
<marker>exits</marker> a distinct possibility.
</example>
<example type="correct"> Then there exists a distinct
 possibility.  </example>
</rule>
```

Every rule begins with id and name attributes. The id is a short form name for the rule and the name attribute is a more descriptive text that describes the use of the rule. The pattern tags describe the sequence of tokens that the checker should find in a sentence, before firing this particular rule. In this example, the two consecutive tokens – there and exits define the pattern.

Once a rule is fired, a message and a correction is generated. Since rules are manually generated in LanguageTool, the error description and correction are very precise. The section of text from the sentence that matches the pattern can be highlighted to indicate the location of the error in the sentence.

A rule with tokens in a pattern is quite specific, since the identical tokens must occur in the matching sentence, in the same order as the tokens in the pattern. More general rules may use POS tags instead of specific tokens in the pattern. For example, a rule may define a pattern where a noun tag follows an adjective tag. This particular order of tags is rare in English and is a potential error.

LanguageTool uses many hundreds of such rules to find grammatical errors in a sentence. Some of the patterns of these rules include regular expression-like syntax to match a broader variety of tag and token sequences. Although, LanguageTool is a very precise grammar checker, there are two drawbacks. One, the manual maintenance of several hundreds of grammar rules is quite tedious. It has become a little simpler to collaboratively manage large rule sets with the use of Web-based tools. Two, the number of rules to cover a majority of the grammatical errors is much larger. Therefore, the recall of LanguageTool is relatively low. Finally, each language requires a separate set of manually generated rules.

Other rule-based checkers include EasyEnglish from IBM Inc. and a Constituent Likelihood Automatic Word-tagging System (CLAWS) probabilistic tagger to identify errors. The well known grammar checker used in Microsoft Word is closed source and many of the other grammar checkers are similarly not available to the public. The design of the Emustru grammar checker is based on a probabilistic tagger suggested by Atwell [5]. Rules are generated automatically from a tagged corpus and errors are identified when low frequency tag sequences are observed in a sentence. The assumption is that a frequent tag sequence in a tagged corpus that has been validated is correct.

4.3.2. Automatic Rule-based Systems

Grammar checkers based on automatically generated rule sets have been shown to have reasonable accuracy [6,7] to be used in applications such as Essay Evaluation. The automated grammatical error detection system called ALEK is part of a suite of tools being developed by ETS Technologies, Inc. to provide students learning writing with diagnostic feedback. A student writes an essay that is automatically evaluated and returned with a list of errors and suggestions. Among the types of errors detected are spelling and grammatical errors.

The ALEK grammar checker is built from a large training corpus of approximately 30 million words. Corpora such as CLAWS and the Brown corpus, characterize language usage that has been proofread and is presumed to be correct. The text from these corpora is viewed as positive evidence that is used to build a statistical language model. The correctness of a sentence is verified by comparing the frequencies of chunks of text from the test sentence with similar or equivalent chunks in the generated language model.

A collection of ill-formed sentences constitutes negative evidence for a language model. Text chunks from a sentence that closely match a language model built from negative evidence are strong indicators of potential errors. However, it is harder to build a corpus of all possible errors in a language. The number and types of errors that can be generated are very large. Consider a four word sentence.

My name is Ganesh.

There are 4! or 24 ways of arranging the words of this particular sentence, of which only one is legitimate. Other sources of errors include missing words or added words that make ill-formed sentences. The construction of a corpus made up of negative evidence is time consuming and expensive. Therefore like ALEK, the Emustru grammar checker uses positive evidence alone to build a language model.

Text is preprocessed (see Preprocessing section) before evaluation. A grammar check consists of comparing observed and expected frequencies of words / POS tag combinations. This same method is used to identify phrases such as “New Delhi” or “strong tea” in text. Bigram tokens such as these are seen more frequently than by chance alone and therefore have a higher likelihood of occurrence.

Consider the phrase “New York” in the Brown corpus. The probability of observing the word “York” when it is preceded by the word “New” is 0.56, while the probability of observing “York” when it is preceded by any word except “New” is 0.00001. These probabilities along with individual word counts are used to find the likelihood that two words are dependent.

The log-likelihood measure [7] is suggested when the observed data counts are sparse. An alternate mutual information measure compares the expected relative frequency of a bigram in the corpus to the expected relative frequency assuming the bigram is independent.

$$MI = log(\frac{p(name-is)}{p(name) \times p(is)})$$

where *p(name-is)* is the probability of the bigram “*name is*” and the denominator is the product of the unigram probabilities of “*name*” and “*is*”. Both the mutual information and log-likelihood measures have been used in the Emustru grammar checker. The log-likelihood measure is used when the number of occurrences of one of the words is less than 100 (in the Brown corpus).

A generated statistical language model is a large collection of word/tag pairs The occurrence of words and tags in text is not independently distributed, but instead has an inherent association built in the usage patterns that are observed in text. For example, we would expect to see the phrase “name is” more often than the phrase “is name”. A rule would assign a much higher likelihood for the phrase “name is” than the phrase “is name”. The design for the ruleset used in the Emustru grammar checker is based on a large number of these types of observations.

4.4. Design of Emustru Grammar Checker

The design of the Emustru grammar checker is made up of three steps. The first preprocessing step is common to most grammar checkers. Raw text is filtered and converted to a list of sentences. Text extracted from files often contains titles, lists, and other text segments that do not form complete sentences. The filter removes text segments that are not recognizable as sentences. The text of each extracted sentence is divided into two lists of POS tags and tokens. Every token of a sentence has a corresponding POS tag. The lists of tags and tokens for a sentence are passed to the checker.

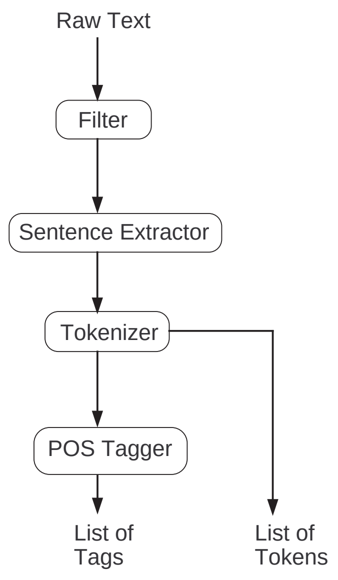
The second step is to generate a rule set that will be used by the checker. In this step, four tables consisting of several thousand rules are automatically generated from a tagged corpus and lists of stop words and stop tags. The final step is the application of the generated rules to detect errors. The lists of tokens and tags are analyzed for deviations from expected patterns seen in the corpus. Sequences of tags and tokens are evaluated against rules from four different tables for potential errors. The first error in a tag / token sequence that may have multiple errors is returned from the grammar checker. This limits the total number of errors per sentence.

4.4.1. Preprocessing

A pipeline design is used in the Emustru grammar checker. The raw text is first filtered and converted into a stream of sentences. The sentence extractor from LingPipe is used to extract sentences from the text (see Figure 4.1). The sentence extractor uses a heuristic method to locate sentence boundaries. The minimum and maximum lengths of sentences are set to 50 and 500 characters respectively. Most English sentences end with a sentence terminator character, such as a period, comma, or a exclamation. These characters are usually followed by a space and the first word of the following sentence or the end of the text.

The sentence extractor will fail to extract sentences that do not separate the sentence terminator from the first word of the next sentence. Instead a complex token such as abc.com or an abbreviation will be assumed. A POS tagger accepts a list of tokens from a sentence and assigns a POS tag to each token. The output from the preprocessing step is a list of tokens and associated tags per extracted sentence. Most of the tokens in a sentence can be extracted by simply splitting the sentence

Figure 4.1.: Extracting List of tokens and POS tags



string when a whitespace is observed. Although this works for most tokens, some tokens such as I'll or won't are converted to their expanded versions "I will" and "will not". Other tokens such as out-of-date and Sourceforge.net are not split into two or more tokens. Tokens that contain periods such Mr. or U.S. are retained as is.

4.4.2. Creating a Rule Set

The rule set used in the grammar checker is a collection of four database tables. A tagged corpus and lists of stop words and tags are used to build the set of rule database tables (see Figure 4.2). The rule set is created once before the grammar checker can be applied. A modified rule table must be re-loaded in the database to take effect.

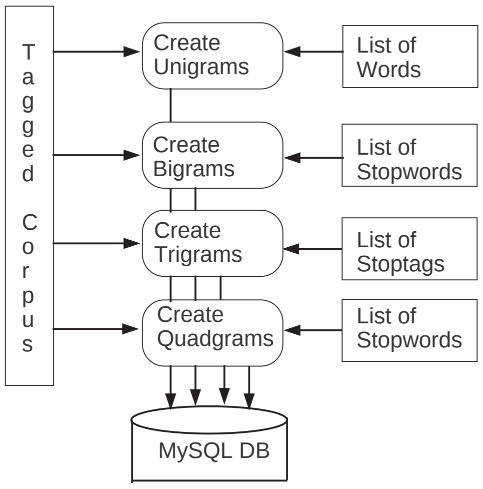
4.4.2.1. Unigrams

The first table is the unigram table. This table contains the most common tags for words in the dictionary. A POS tag y that was assigned to a word x in fewer than 5% of all cases in the tagger corpus is noted in a rule for x. Any sentence that contains the word x tagged with y is considered a potential error by the checker. The types of errors detected are pairs of words that are used incorrectly such affect and effect or then and than. For example, the probability of finding the word affect used as a noun was less than 3% in the Brown corpus. The unigram rule for the word affect will detect the erroneous use of the word in the sentence below.

We submit that this is a most desirable affect of the laws and one of its principal aims.

The grammar checker returns the following description - "The word affect is not usually used as a noun, singular, common" and the suggestion - "Refer to affect, did you mean effect". There are numerous other pairs of such words that are often mixed up, such as bare / bear, accept / except, and loose / lose.

Figure 4.2.: Create a Ruleset made up of four Database Tables.



4.4.2.2. Bigrams

The bigram tag table is constructed by observing tag sequences in the corpus and computing a likelihood measure for each tag sequence. Consider the erroneous sentence – “My father fixing the computer.”. The tag sequences extracted from this sentence and their likelihoods are shown in Table 4.1. The START and END tags are added to the beginning and the end of the sentence respectively.

Table 4.1.: Bigram Tag Sequences for an Erroneous Sentence

Token	Tag Sequence	Likelihood	Error
My	START-PP$	0.33	No
father	PP$-NN	1.93	No
fixing	NN-VBG	-1.11	Yes
the	VBG-AT	0.71	No
computer	AT-NN	1.90	No
.	NN-.	1.32	No

All the tag sequences in Table 4.1 have positive likelihoods with the exception of the NN-VBG tag sequence. The likelihood of this tag sequence is the likelihood of a verb or present participle following a noun. It is negative since a present participle is usually preceded by a present tense verb such as is. These types of errors are found in sequences of bigram tags.

Other types of bigram sequences include tag-word and word-tag sequences. Words found in text are separated into two sets – open class and closed class words. The open class set contains mainly nouns, adjectives, verbs, and adverbs. These words are numerous and together are found often in text. The individual frequency of a noun or adjective is typically small compared to the frequency of a closed class word. Conjunctions, prepositions, and articles are fewer in number but occur often in text. Golding [9] showed that it is possible to build context models for word usage to detect errors. The context of a word x that does not match the context defined in the bigram table for x is a potential error.

The words that are used most frequently in the tagged corpus are selected in a stop list that includes words such as – the, and, of, and did. Consider tag-word rules for the word the that model the context of tags before the stop word. An adjective is rarely seen before the word the. The rule with the "JJ-the" context will detect an error in the sentence – "Can you make large the photo?". Similarly in the sentence - "The goal to find was who attended." the word-tag rule for the "was-WPS" context detects an error in the word sequence "was who". All three types of sequence rules – tag-tag, tag-word, and word-tag are used to detect bigram error sequences in sentences.

4.4.2.3. Trigrams

The use of trigrams to model word / tag usage requires a very large corpus. Consider the Brown corpus with roughly 100 POS tags. The maximum number of trigram tag sequences that can be generated is one million. The number of words in the Brown corpus is one million and is clearly not sufficient to model the usage patterns of all possible trigram tag sequences.

Instead, the problem is limited to modeling a much smaller set of trigram tag sequences. The modeled set of tag sequences represents tags that are frequently found in grammatically incorrect sentences. For example in the sentence - "I did not wanted to clean the room.", the verb want is used in the wrong tense. The sentence is correct if the present tense of the word is used instead of the past tense. We can collect pairs of such tags that are interchanged in grammatically incorrect sentences. These tags form a stop tag list that have one or more replacement tags that may fix the error. The grammar checker returns the following description and suggestion for the incorrect sentence above.

Description: The fragment not wanted to is rare.

Suggestion: Possible agreement error: Replace wanted with verb, base: uninflected present ...

The detector uses the tags before and after the stop tag to build the context of the given sentence. The likelihood of the tag sequence is extracted from the database table and compared with the likelihood of another tag sequence that replaces the stop tag with a substitute tag. An error is generated when the likelihood of the tag sequence with the substitute tag is much higher than the likelihood with the original tag. Consider another incorrect sentence - "She come to college late every day.". The present tense of the verb come is used instead of the past tense. Here, the grammar checker returns -

Description: The fragment She come to is rare.

Suggestion: Possible agreement error: Replace come with verb, past tense.

The purpose of using trigrams is to identify errors that the bigram tables fail to detect. For example, in the first sentence the token sequences “not wanted” and “wanted to” are both legitimate token sequences independently. However, the combined tag sequence “not wanted to” is rare and is a potential error. The replacement of the past tense tag with the present tense tag produces a tag sequence that is more likely than the original tag sequence. The checker generates errors for cases where the replacement tag creates a more likely tag sequence.

This is not a fool-proof method, since the best possible replacement tag cannot be predicted beforehand in a stop tag list. An attempt is made to find the pairs of tags that are most often mixed up in an error corpus. A stop tag may also have more than one replacement tag. In such situations, the most likely replacement tag is selected to compare with the liklelihood of the original tag. Finally, a corpus larger than the one million word Brown corpus is needed to accurately model trigram tag sequences.

4.4.2.4. Quadgrams

An extremely large corpus would be needed to model all possible quadram tag sequences for the same reasons as the trigram tag sequences mentioned earlier. The number of possible quadgram sequences is very large and accurately modeling the usage patterns of such a huge number of sequences would require a corpus that is not currently available. The space and time required to build a quadgram model would be correspondingly large.

The quadgram model is simplified to identify specific words that are used in the wrong context. Quadgram sequences are constructed for a set of stop words. These stop words represent pairs of words like is / are, was / were, and there / their. Consider the sentence - “A herd of horses are better than a flock of sheep.”. The grammar checker returns with the following description and suggestion:

Description: The fragment better than **a** is not usually preceded by **are**

Suggestion: Possible agreement error: Replace **are** with **is**

The checker begins by constructing two quadgrams when a stop word is observed in the sentence. For example, in the sentence above, the two quadgrams - “herd of horses are” and “are better than a” are generated when the word are is seen. All the words in the quadgrams are replaced by their corresponding tags with the exception of the stop word (are). The use of tags instead of the specific words themselves, makes the quadgrams more general and easier to model with a smaller corpus. The likelihood of both quadgrams with the given stop word are, is evaluated using the Quadgram database table.

The stop word are is replaced with is in both quadgrams and the likelihood of the modified quadgrams is extracted from the database table as before. If the likelihood of the modified quadgram substantially exceeds the likelihood of the original quadgram, then the checker generates an error.

The quadgram model is subject to the same types of problems as the trigram model. We need to know beforehand words that are frequently used incorrectly and the appropriate replacement word. The type of words included in the quadgram stop word list are seen in subject-verb agreement errors. The stop word list used in the Emustru grammar checker is made up of a few of these types of words. The size of the Brown corpus may not be sufficient to build an accurate model for these quadgrams.

Long distance word dependencies in sentences are not detected by the bigram or trigram table look ups. Such dependencies occur when the subject and verb are separated by one or more words. making it difficult for the bigram or trigram checker to detect the low likelihood of a plural form of a verb used with a singular form of a noun. The quadgram table models some of these long distance word dependencies that are missed in the earlier steps.

4.4.2.5. Error Model

The spelling error model has been adapted to describe a grammar error model. In the spelling error model, four modification functions that operate at the character level are used to correct the spelling of a word. For example, the transpose function will interchange the letters i and e in the mis-spelled word recieve to create the correct word receive. One of more of these four functions can be used to correct any mis-spelled word. The edit distance is a measure of the number of times a modification function needs to be applied to transform a mis-spelled word into the correct word.

The grammar error model uses the same functions as the spelling error model, except that the modifications for the grammar error model operate at the word level (see Table 4.2).

Table 4.2.: Types of Grammar Errors

Error	Sentence
Delete	Why did the chicken cross road?
Insert	Who is the the chicken?
Transpose	The chicken's food is from made soya.
Modify	The number of chickens were large.

For example, the delete error in the first row of Table 4.2 is corrected by applying the insert function that adds the word the between the words cross and road. Similarly, the transpose function transforms the word sequence "from made" to "made from" in the third row of Table 4.2. The rules in the ngram tables that detect these errors is shown in Table 4.3.

Table 4.3.: Database Tables used to Correct Errors in Table 4.2

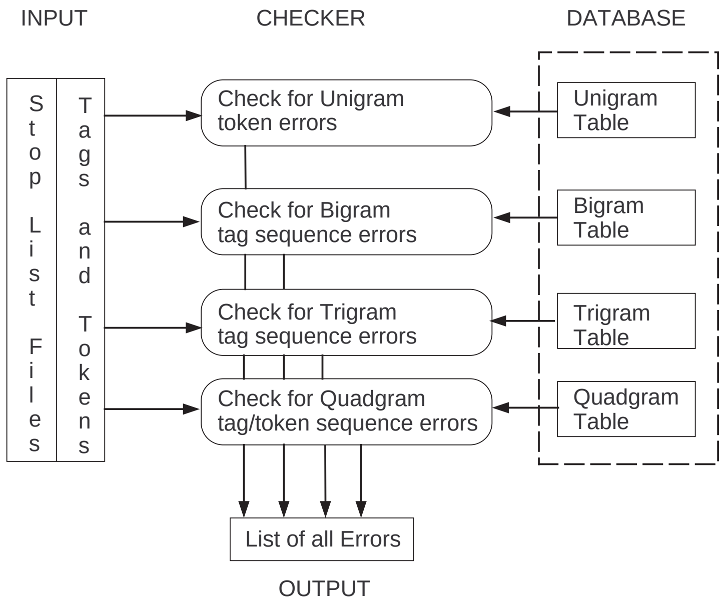
Error	DB Table	Message
Delete	Bigram (tag_tag)	A noun is not usually followed by a noun (refer to cross and road).
Insert	Bigram (word_tag)	The token the is not usually followed by an article (refer to the and the).
Transpose	Bigram (word_tag)	The token is is not usually followed by a preposition (refer to is and from).
Modify	Quadgram (tag_word)	The fragment number of chickens is not usually followed by were (Possible agreement error: Replace were with was)

This is a simplistic error model and does not make distinctions between errors such as subject-verb agreements, run-ons, and other grammatical mistakes. Although the error model is unsophisticated, the descriptions and suggestions are usually good enough for the user to correct an error. The part of the sentence that contributed to the error is highlighted and the automatically generated description explains why the tag(s) or token were not appropriate in the sentence.

4.4.3. Applying a Rule Set

The input to the grammar checker is a list of tokens and tags along with the stop lists for each of ngram checks. The four ngram database tables are checked in order starting from the Unigram to the Quadgram table (see Figure 4.3). The output from the grammar checker is a list of possible errors.

Figure 4.3.: Applying a Ngram-based Rule Set to Detect Errors


INPUT
CHECKER
DATABASE
Stop List Files
Tags and Tokens
Check for Unigram token errors
Check for Bigram tag sequence errors
Check for Trigram tag sequence errors
Check for Quadgram tag/token sequence errors
Unigram Table
Bigram Table
Trigram Table
Quadgram Table
List of all Errors
OUTPUT

Each of the errors returned contains a description and a suggestion. The text for these fields is automatically generated and therefore not as precise as a message from a manually generated rule. An unigram error states that a word is rarely used with the assigned POS. A bigram error mentions that either a word is not usually followed by a tag or two assigned tags are rarely seen together. The trigram and quadgram errors suggest some type of agreement error and propose alternate tags or alternate words to correct a sentence.

4.5. Evaluation

The Emustru grammar checker has been evaluated using a small corpus of 100 annotated sentences. A fraction (70%) of the sentences are grammatically correct and the remainder have one or more errors. This is a small corpus and a large scale evaluation would use a wider variety of sentences and errors to test the checker. The corpus to test the grammar checker is a collection of e sentences, out of which a sentences ($a < e$) are grammatically incorrect. The grammar checker was run against the set of e sentences to detect errors. The results of the test are shown in Table 4.4.

Table 4.4.: Grammar Checker Evaluation

		Actual Errors	
		Yes	No
Flagged Errors	Yes	a	b
	No	c	d

where $e = a + b + c + d$. The value of a is the number of errors that were actually errors and correctly flagged by the checker. The value of b is the number of errors that were not found by the checker. The value of c is the number of errors that were wrongly identified by the checker. Finally, d is the number of correct sentences that was assigned zero errors by the checker.

Most of the sentences in the test corpus have been selected from various news articles on the Web. News articles from reputed sources have been proofread and can be presumed to be error-free. A sample of sentences from news articles on different topics were selected for the set of correct sentences. The set of incorrect sentences were generated from the most common types of grammatical errors mentioned on the Web.

Notice, this error corpus is not annotated to mention a particular type of grammatical error. Instead, a sentence is merely defined as correct or incorrect. Therefore, there is no verification of error type detected to match a particular grammatical error. Any error detected in a sentence that is invalid is considered as a correctly flagged error and tabulated in the value of a in Table 4.4. Currently, there is no standard corpus for grammatical error detection and no standard format to define a syntax error.

Recall and precision are two standard evaluation parameters used in Information Retrieval to evaluate the performance of search engines. In this context, we can define recall as the value *a / (a + c)* and precision as *a / (a + b).* Typically, recall and precision are inversely related. i.e. high recall is associated with low precision and vice versa. Consider an extreme case where $a = e$ and every sentence in the test corpus is flagged as an error. Recall will be 1.0 or the maximum since the value of c, the number of missed errors will be zero. However, precision will be low since the value of b, the number of wrongly detected errors will be high.

Conversely, the checker may flag a very small fraction of errors that are obvious. In this case, the value of a will be very small and correspondingly the value of b will be zero leading to precision of 1.0. However the value of c, the number of actual errors that were not flagged by the checker will be high making recall low.

This problem of balancing recall and precision is fairly common in other NLP tasks such as entity extraction and sentiment analysis. The implementation in this paper attempts to maximize precision with reasonable recall. In other words, flagging an excessive number of errors is more annoying to the user than allowing a few undetected errors. We can vary the threshold higher or lower to detect fewer or more errors respectively. The threshold should be high enough to minimize the number of false positives, i.e. the sentences the checkers believes are errors, but are actually correct. In Table 4.4, this means that the checker should minimize the value of b. Recall is controlled to a lesser extent by minimizing the value of c, i.e. finding as many of the actual errors as possible. A third evaluation parameter is *accuracy.* This parameter combines all the results in Table 4.4 into a single value. It is defined as *(a + d) / (a + b + c + d).* Accuracy measures the number of error sentences and correct sentences detected relative to the total number of sentences.

Table 4.5 contains four sample sentences from the test corpus of 100 sentences, two sentences each from the correct and error categories. These sentences are similar in style to the other sentences in the corpus. There is a large number of grammatical error types and the test corpus covers some of the common errors. The types of errors covered include subject-verb agreement and the incorrect location of words in a sentence.

Table 4.5.: Sample Error and Correct Sentences

Category	Sentence
Error	I did good in this course.
Error	Their is a major problem with this paper.
Correct	It'll recover this year after the temporary adjustment.
Correct	Some people survive much longer based on the tumor's subtype, size, whether it has spread and the patient's age.

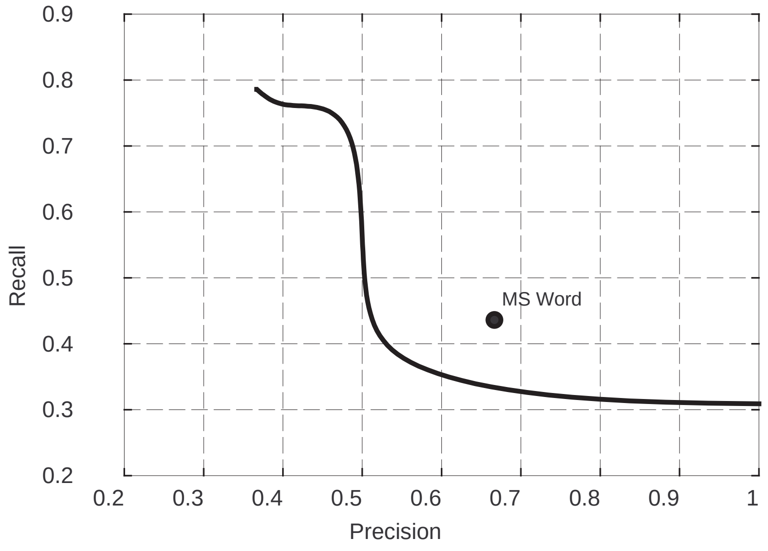
A test of the Emustru grammar checker with a likelihood threshold of 6.5 gave an accuracy measure of 0.81 with the values of 15, 4, 15, and 66 for the parameters *a, b, c*, and *d* respectively of Table 4.4. The recall and precision measures were evaluated using the same error corpus.

Figure 4.4 shows a fairly typical recall-precision plot that is observed in other information retrieval applications. When recall is high, precision is low and vice versa. The recall and precision of the MS Word grammar checker for the same error corpus was 0.433 and 0.684 respectively. The precision of the MS Word grammar checker is set high enough to ensure that very few false positives will be reported.

The accuracy of the Emustru grammar checker was varied by altering the likelihood parameter (see Figure 4.5). The likelihood was roughly proportional to the accuracy till a threshold of about 6.0 and thereafter the accuracy was relatively constant. We would expect low accuracy when the likelihood is low since a large number of false positives will be reported. The recall for low likelihoods is close to 1.0, while precision is much lower at 0.3. At higher likelihoods, the recall falls to about 0.5 and the precision rises up to about 0.79. The interface to the grammar checker on the client allows the user to control the likelihood parameter by adjusting the grammar level from very strict to liberal. The highest accuracy with the Emustru grammar checker of 0.81 was a slight improvement over the accuracy of 0.77 with MS Word.

There are several reasons why the accuracy of the grammar checker cannot be tweaked by adjusting the likelihood alone. The first possible source of errors is a wrong sentence boundary detection. Tokens from a sentence that is broken or a com-

Figure 4.4.: Recall-Precision Plot for the Corpus of 100 sentences.

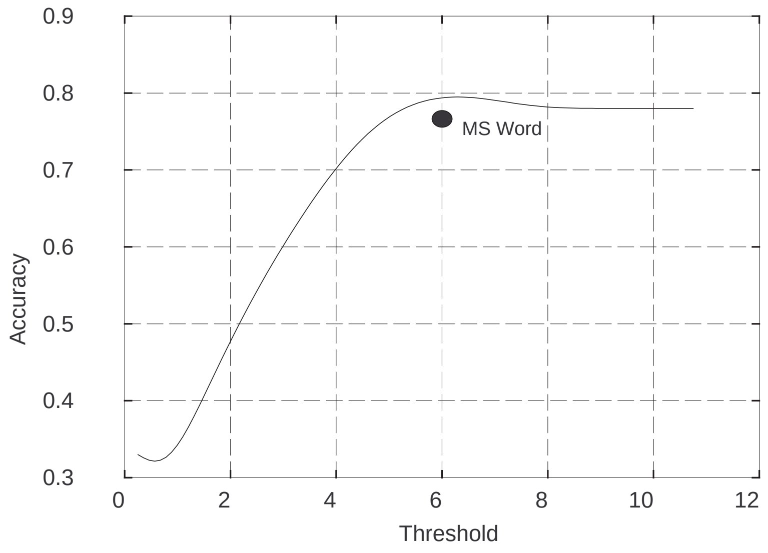


bined sentence will be harder to tag accurately. However, most sentence boundary detectors are very accurate, if the text passed is filtered to remove text fragments such as titles and table text. A second reason for low accuracy is the assignment of the wrong POS tag. This happens in roughly 5% of all tokens and therefore the grammar checker cannot possibly make an accurate judgement of a grammatical error. Finally, the tagged corpus used to build the ngram Rule sets may not accurately represent language usage patterns.

4.6. Performance

A majority of the time to detect errors is spent running SQL statements to search the four database tables. Roughly 1000 SQL select requests were required to check the error corpus of 100 sentences (2020 words). Roughly 50% and 25% of the SQL statements were lookups of the bigram and unigram tables respectively. The remaining 25% of the SQL statements were primarily lookups of the trigram table. The time to run a SQL select statement on an Intel P4 Dual-core machine with 1 Gbyte

Figure 4.5.: Accuracy-Threshold Plot for the Corpus of 100 sentences.



of RAM is a few milliseconds or less. Therefore, the time to analyze a sentence is roughly 30 milliseconds or less. The time to initially load the grammar checker is significant. A Hidden Markov Model-based POS tagger that is used to assign tags to tokens is read from a file. The time to read the 6 Mbyte POS tagger file at startup time is roughly 1.25 seconds. The size of the database is shown in Table 4.6.

Table 4.6.: Number of Rows in Ngram Rule Set Tables

Table	No. of Rows
en_unigrams	18,379
en_bigrams	10,463
en_trigrams	19,133
en_quadgrams	17,080
Total	65,055

5. Essay Evaluator

This chapter describes the use of automated methods to evaluate essays. Automated Essay Scoring (AES) [9] began in the 1960s and has since become an accepted method of grading essays and often accompanies the ratings generated by a human grader. Competitive exams such as the SAT and GMAT use a human and an automated grader to evaluate an essay. Some of the benefits of an AES include lower costs, less time to evaluate an essay, and the absence of any bias.

Prior to automated essay evaluation, all essays were manually scored, adding to the high cost of evaluation. Consider the SAT exam that a million or more students may take in any given year. Each of the million essays of roughly 100-300 words must be read and evaluated. A human grader needs to spend about 3-4 minutes to assign a score to each essay. In large scale exams, the use of an automated system can significantly reduce this cost and the time to complete the evaluation. Finally, a human grader is susceptible to fatigue or may be biased if a topic is open-ended and possibly controversial. An automated system runs the same algorithm to compute a score for all essays and is free of any bias.

The time to correct an essay is also a burden for a class teacher. Imagine a class of 30 students or more, who submit 2-3 writing assignments per week. A teacher will need to correct about one hundred essays per week; a time consuming and dull task. The use of AES can reduce this burden to some extent through an initial machine evaluation to identify some of the obvious errors in an essay. Another task that is difficult for a teacher is to identify the common errors in a collection of 30 or more essays. An AES can easily collect, maintain, and summarize global information for a collection of essays.

A common criticism of AES is that a machine never really understands the contents of an essay and instead assigns scores based on a set of features. It is true that an evaluation algorithm does not actually comprehend the essay. Yet, it has been shown that a small number of carefully selected features are sufficient for an algorithm to compute a score that is very close to the score that a human would have assigned to the same essay.

Consider the E-rater ™ [10, 11] from ETS Technologies that has been used to evaluate roughly 360,000 essays per year. Every essay is scored in the range of 1 to 6, where 1 is the lowest and 6 the highest score. In 97% of the essays, the absolute score difference between a human grader and the E-rater was less than 2. An absolute score difference of more than 1 was resolved by a second human grader.

Even though human graders and AES strongly correlate, it is possible to generate a poor essay that would score high with AES. A human grader assigns a grade to an essay based on concepts such as organization, discourse, and structure. An AES implicitly computes values for features that represent these concepts and it is possible to generate an artificial essay that scores well with an AES but is actually a poor essay. However, the effort to create such an essay is not minimal and would require a trained writer to generate text such that most of the features used in an AES are fully represented in the essay. This also implies that a human grader is still necessary to confirm the scores returned by an AES.

It is unlikely that AES will identify the next great writer given the limitations of the technology. But, automatic evaluation of an essay can simplify a teacher's job in a classroom. A student can submit an essay to a machine, make corrections based on the feedback from the AES, and then submit a second and possibly improved version of an essay to the teacher.

A teacher would like to see automated feedback to a student that is very similar to what a human would have generated. The type of feedback would include spelling mistakes, grammatical errors, and the use of discourse terms. The current AES tools cannot generate feedback of the same quality that a teacher can provide a student. However, AES uses approximations to measure features that are considered important to a teacher.

5.1. How does it Work?

First, consider how a human grader evaluates an essay. The grader reads the entire essay, forms an opinion on the quality of the essay, and assigns a score. This score is also called a *holistic* score based on the grader's overall impression of the essay. A holistic score is a single value in a range (1-6) computed from the grader's evaluation of a set of characteristics. While reading the essay, a grader looks for certain traits that characterize a good essay. The presence of such traits or features in an essay motivate the grader to assign a higher score to such essays compared to other essays in which the same traits are absent. These traits typically include content, creativity, mechanics, style, and organization. An AES scans an essay and evaluates and searches for the presence of such traits.

The difficulties in building an AES lie in identifying features that accurately represent these traits. Page and Petersen[10] use the terms *trins* and *proxes* to describe traits and features respectively. Trins represent characteristics that a human grader evaluates in an essay such as style, organization, and content. On the SAT, the types of characteristics that should be present in a high scoring essay include a well-stated and developed point of view, critical thinking, examples, supporting evidence, coherent arguments, strong vocabulary, and grammatically correct sentence. A proxe or approximation is a variable that is automatically extracted and roughly estimates a trin or characteristic. Some of the roughly 30 proxes used in Project Essay Grade (PEG [9]) include - the average sentence length, the number of paragraphs, total number of words, and average word length. A proxe may represent part of one or more trins and a trin may use multiple proxies. In other words, there is a many-to-many relationship between trins and proxes.

5.1.1. Traits and Features

Clearly, an AES is more likely to be accurate when a proxe closely represents a trait. How do we find proxes or variables that define traits? The best way is to ask human graders what they look for in an essay to evaluate a particular trait. For example, the total number of words, the number of unique words, and the presence of domain specific words are proxes to measure the content of an essay. A human grader may not actually count the number of occurrences of words, but will make judgements from an estimate of the length of the essay, the presence of some words, and the use of vocabulary. An AES can make very precise counts of words, frequencies, sentences, word types, and other parameters. Table 5.1 contains a list of traits and associated features that are measured in an essay.

The types of grammatical errors identified for the **grammar** trait can include missing punctuation, run-on sentences, subject-verb agreement, ill-formed verbs, pronoun

Table 5.1.: Traits and Associated Features

Trait	Features
Grammar	Measures of grammatical errors
Usage	Misuse of articles, wrong word forms, preposition errors, and faulty comparisons
Mechanics	Spelling mistakes and missing punctuation marks.
Style	Use of passive voice, inappropriate sentence lengths, and faulty conjunction usage.
Organization	Presence of an introduction, content paragraphs, and a conclusion
Development	The average length of a discourse element
Lexical Complexity	Average word length and number of medium-long words
Vocabulary Usage	Presence of prompt-specific terms

errors, and forms of garbled sentences. The measures of the **usage** trait are also mostly grammatical and include the misuse of articles, wrong word forms, confused words, preposition errors, and faulty comparisons. The sentence - “We don’t have *many* information on the subject”, does not use the proper article. The wrong word form of *danger* is used in the sentence - “Until recently, the Hudson river contained *danger* levels of pollutants”. The word *effect* is used in the sentence - “Lack of sleep *effects* the quality of work.”, instead of the word *affect*. The sentence - “They arrived *to* the town” contains a preposition error. A faulty comparison compares two nouns that not alike. For example, the sentence - “The weather in Germany is colder than Gabon”, makes an illogical comparison between weather and a country.

The **mechanics** of an essay includes spelling mistakes, the wrong case of a letter in a word, missing punctuation marks, and incorrect fused or compound words. The measures to evaluate the **style** of an essay look for the use of passive voice, repetition of words, and sentences that are either too long or too short. An essay for a typical prompt in an exam is expected to have an introduction and a conclusion. Between these two discourse elements, an essay should also contain main points, supporting material, and a thesis. The absence of these discourse elements in an essay will potentially lead to a lower score. A discourse element such as a main point without any supporting material is weak and possibly not fully **developed**. A completely developed main point will have at least one or more sentences to support the argument.

The measures for **lexical complexity** evaluate the usage of words. A large number of words that are more than five or six characters long may indicate a strong vocabulary. Finally, in a group of essays generated for a specific prompt, we would expect to see a similar set of words in essays with high scores. These words are prompt specific and represent the common terminology used to discuss the essay prompt.

Traits such as content and organization can be reasonably approximated using a set of variables. But, other traits such as creativity are hard to define in the form of an algorithm that can be coded in an AES. It is difficult because there is no model that an AES can precisely measure to evaluate the degree of creativity in an essay. Such traits that are based on features that cannot be estimated apriori are difficult to approximate and are potential sources of errors in an AES.

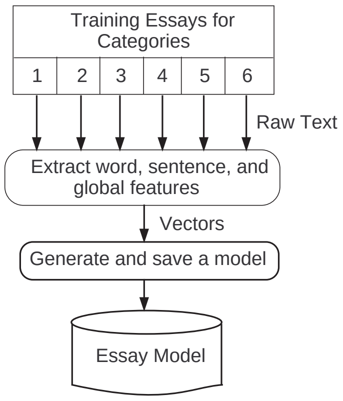
Despite these deficiencies in an AES, scores computed automatically have a strong correlation with the scores of a human grader. This is possibly because the traits that an AES can evaluate with some accuracy are sufficient to generate a precise score. The precision of an AES is not proportional to the number of features. In other words, a few good features are sufficient to accurately categorize an essay[11]. Further, an essay with a strong trait like creativity is usually accompanied with high values of other evaluated traits such as organization and vocabulary.

5.1.2. Creating a Model

An unseen essay is assigned a score using a model that was built with a set of prescored essays. Consider a set of training essays for each of the six categories. A group of features is extracted for each training essay and the collection of such features for a particular essay category define the model (see Figure 5.1).

Every training essay has a vector of values that represent the extracted features. For example, one of the values in the vector represents the number of unique words in the essay. Each vector has an associated category and each position in the vector represents an identical feature. The model is a logistic regression classifier created from the set of training vectors and associated categories.

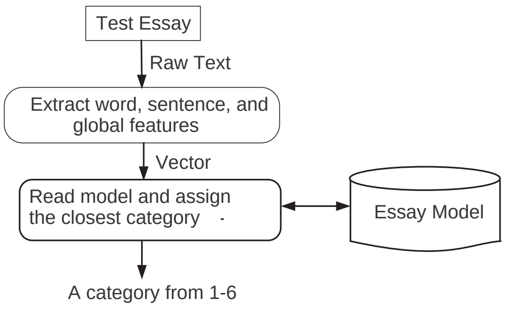
Figure 5.1.: Building an Essay Model with Training Essays


Training Essays for Categories
1
2
3
4
5
6
Raw Text
Extract word, sentence, and global features
Vectors
Generate and save a model
Essay Model

5.1.3. Using a Model

An unseen essay is first processed to create a vector of values in the same manner a vector was created for the set of training essays. The vector is a list of feature values, but excludes any category. The classifier accepts a vector and returns the closest category based on the trained model (see Figure 5.2).

Figure 5.2.: Assigning a Category to an Essay


Test Essay
Raw Text
Extract word, sentence, and global features
Vector
Read model and assign the closest category
Essay Model
A category from 1-6

5.2. Emustru Essay Evaluator

The Emustru essay evaluator is based on the design explained in Section 5.1. A set of 52 training essays for the International English Language Testing System (IELTS) were collected from the Web. In an ideal case, the number of training essays per category should be large enough to build a precise model. Unfortunately, the number of training essays per category varied from a maximum of 20 essays to a minimum of 5 essays. The category 1 was assigned in the extreme case when insufficient text was provided.

A logistic classifier assigns a weight to each of the features in the vector generated from an essay, such that the weighted vector fits a model generated from the set of training essays. In other words, a feature x that has high values for a particular category y alone, will have a higher weight in the model for y. So, if an unseen essay contains a high value for feature x, it is more likely to be assigned category y. The list of features for a given essay in Emustru are -

- Total number of words
- Total number of characters
- Number of unique words
- Fourth root of the number of words
- Number of spelling errors
- Number of grammatical errors
- Number of paragraphs
- Number of sentences
- Average word length
- Number of unique words per 100 words
- Average sentence length in words
- Number of words with more than 5 letters

- Number of words with more than 6 letters
- Number of words with more than 7 letters
- Number of words with more than 8 letters
- Number of passive voice sentences
- Standard deviation of word lengths
- Standard deviation of sentence lengths
- Number of discourse markers
- Average coherence between sentences and the entire text
- Average coherence between paragraphs and the constituent sentences
- Average coherence between consecutive sentences

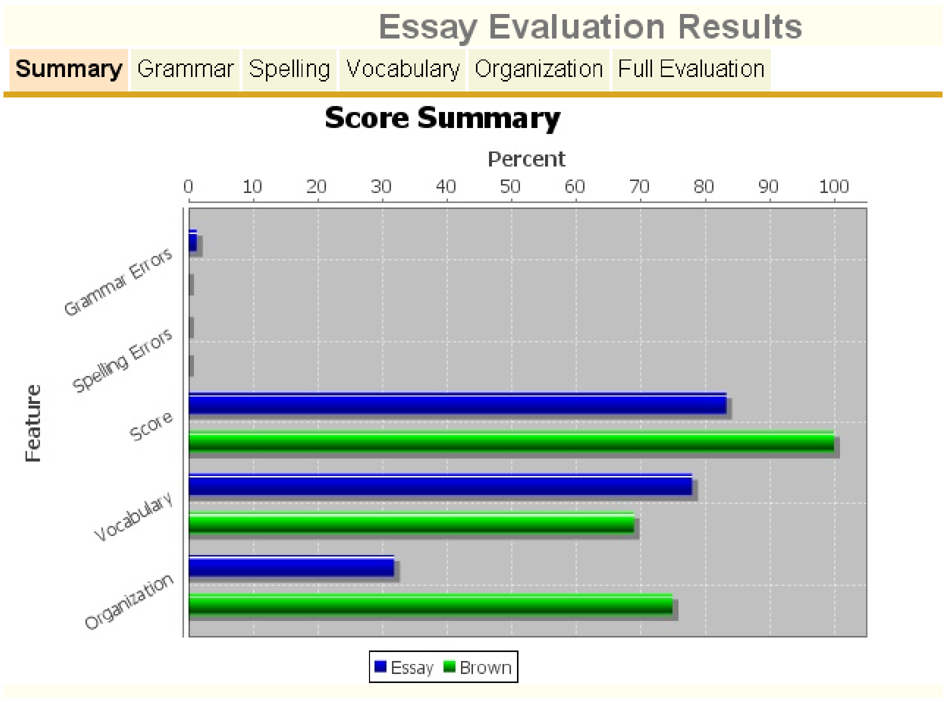
The collection of 22 features was extracted in a vector and passed to the classifier model. The values of the features are a mixture of integers and floating point numbers. The logistic classifier returns the closest category for the passed vector. The text for an essay that is to be evaluated is pasted into a textbox in a Web based form and passed to the essay evaluator. The evaluator returns a tabbed screen of results shown in Figure 5.3.

The first tab is a summary screen showing some of the scores of the essay compared to a high scoring essay called Brown. A few features including the score, grammatical errors, spelling errors, and vocabulary are shown in the summary. All values have been scaled to a 0-100 range.

The next tab contains a list of grammatical errors shown per sentence. The first sentence in Figure 5.4 contains a grammatical error. The statistical grammar checker detected an adjective (proud) following a noun (country); such an occurrence is very rare in English. The following sentence did not contain any error that could be detected.

The **spelling** tab shows the list of sentences along with any spelling errors in each of the sentences. For every spelling error, a potential suggestion is also found. The **vocabulary** tab shows a few of the word statistics such as the number of words,

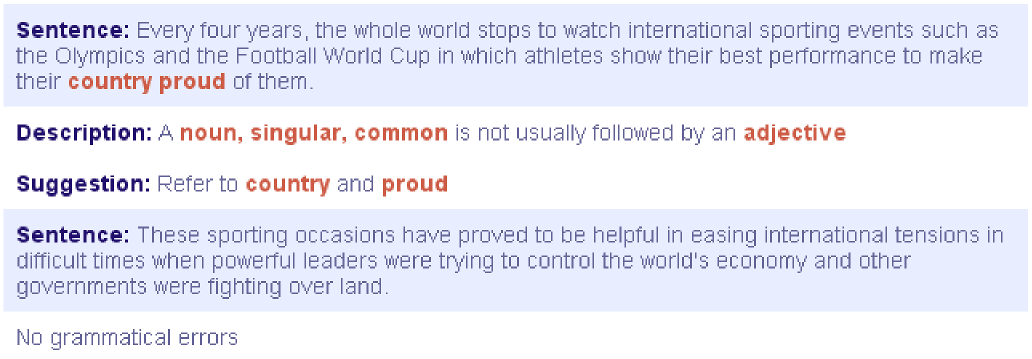
Figure 5.3.: Evaluation of an Essay in Emustru



average word length, number of unique words, and the standard deviation of the word length. The **organization** tab shows the coherence between individual sentences, sentences and their parent paragraph, and sentences with the essay text as whole. Other statistics include the counts of the use of passive voice and discourse markers. Notice, a higher count of passive voice markers may lead to a lower score while a higher count of discourse marker is usually associated with a high scoring essay. The final tab contains a list of all the features in the essay compared to an ideal high scoring essay (Figure 5.5).

The score assigned to the essay (5 in this case) is shown compared to an ideal essay. The number of grammatical errors and the category are also shown in Figure 5.5. The remaining 19 features are not shown in the figure. Any value that is not reasonable close to the ideal value is shown highlighted in the results. For example,

Figure 5.4.: The First Two Sentences of an Essay in the Grammar Tab



Sentence: Every four years, the whole world stops to watch international sporting events such as the Olympics and the Football World Cup in which athletes show their best performance to make their **country proud** of them.

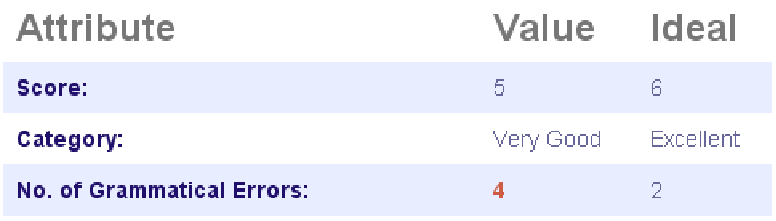
Description: A **noun, singular, common** is not usually followed by an **adjective**

Suggestion: Refer to **country** and **proud**

Sentence: These sporting occasions have proved to be helpful in easing international tensions in difficult times when powerful leaders were trying to control the world's economy and other governments were fighting over land.

No grammatical errors

Figure 5.5.: Three Attributes from the Full Evaluation of an Essay.



Attribute	Value	Ideal
Score:	5	6
Category:	Very Good	Excellent
No. of Grammatical Errors:	**4**	2

the number of grammatical errors has been highlighted in Figure 5.5 since it is double the number of grammatical errors found in an ideal essay.

5.3. Applying AES

The use of AES has become popular in schools and universities. Some of the popular commercial AES products include E-rater ™ [11], Intellimetric ™[12], and Intelligent Essay Assessor™ [13]. These products have been successfully used with a large number of essays on many different topics. However, AES is not perfect and there are valid criticisms about automated ways to evaluate writing.

5.3.1. Is AES Valid?

In several evaluations, AES products have shown a high correlation with human graders and the use of AES in competitive exams is accepted despite its weaknesses. Even though critics may claim that an AES does not understand an essay in the same way a human can appreciate an essay, there is no denial that the final outcome (score) of an AES is valid in most cases.

A secondary issue is whether a student can write a bad faith essay to fool the AES into assigning a high score. A human grader would quickly detect a bogus essay and assign a low score. But, an AES can be deceived by an essay that scores well in the features needed for a high score. For example, consider an essay that is reasonably long, uses a large vocabulary, with no grammatical errors, and is largely coherent. Such as essay would receive a high score, even though the facts mentioned and examples were completely wrong. The AES has no background knowledge to detect such errors.

The Intelligent Essay Assessor (IEA) claims to overcome this problem with a collection of pre-scored essays on a particular topic. An unscored essay would receive a high score, if it appears to be *close* to a group of essays that were assigned high scores prior to the evaluation. The assumption is that high scoring essays for a particular topic will look more similar to another unseen but well-written essay than a poor and irrelevant essay.

IEA uses a matrix factorization algorithm to simultaneously consider all words in an essay. This makes it difficult to write a bad faith essay, since there are no known features that can be artificially manipulated to generate a high score. Instead, IEA relies on the content words alone.

5.3.2. Essay Prompt

An essay prompt describes the main topic or issue that the student's essay should discuss. A few sample prompts are listed below.

- How do you feel about people using cell phones in public? Should cell phones be banned in public places ? Why or why not?
- What is your favorite time of the year? Why? What do you like about that period?

- You have passed a driving test. Your friend who does not have a driver's license would like to know the procedure. Explain how you passed the driving test.

The first prompt is an argumentative prompt. There is no right or wrong answer and an argument can be made both ways to support or disapprove a ban on cell phones in public places. Essays based on these types of argumentative prompts are a little harder to compose than other prompts. An argumentative essay must first make a thesis statement and present a well-developed list of supporting arguments.

The second prompt is a descriptive prompt. An essay for a descriptive prompt creates the background for an event or period and elaborates on the topic. A good essay for such a prompt lists the facts justifying the opinion of the writer in a logical order. The third prompt is an expository prompt. This type of essay explains a procedure step-by-step from the start to finish in order.

All prompts are not equally difficult. Prompts on a complex topic may be harder to write about than a simple topic. For example, a descriptive prompt is a more appropriate assignment for a fourth grade class than an argumentative prompt. The AES does not make distinctions between an easy or difficult prompt and treats essays for all prompts in the same manner.

An AES such as the IEA that relies on content words, uses separate models for each prompt. In other AES products, it would not be appropriate to use a model trained with essays from students of the fourth grade to evaluate essays from students of the twelfth grade. Similarly, it would not be appropriate to grade the Gettysburg address using a model generated from student essays.

The E-rater 2.0 [11] uses a model that is not based on any particular prompt to grade essays. This is very convenient, since it is not necessary to create separate models based on each topic. A single model can capture the necessary information to score any essay. One argument against the use of a single model for all prompts is that content specific words are not given any additional importance in the model. The use of content specific words in an essay is an indicator that the student has understood the prompt and the essay is relevant. Even though this feature is absent in E-rater 2.0, the results are comparable to other AES.

5.3.3. Essay Length

Products like Intellimetric use several hundred features in contrast to the much fewer number of features in E-rater. The number of features appears to play less of a role in the quality of grading results of an AES. On the other hand, one particular feature, the essay length is the most dominant feature. The score of an essay was very closely related to the essay length (number of words). It would seems as if a student could easily fake the AES into assigning a high score simply by generating a long bogus essay with a large number of words. The simplest method to generate a long essay is to repeat a sentence endlessly till the essay is sufficiently long.

However, such an essay would have very low values for other features such as the number of different words and the sentence length standard deviation. Further, a student would not risk submitting such an essay if there was a possibility that a human grader may score the essay. Yet, as long as there is a possibility that a bad faith essay may be scored incorrectly by an AES, it is unlikely that human graders can be completely replaced.

The benefits of a long essay for a competitive exam diminish beyond three main points. Increasing the number of examples from one to three will improve an essay more significantly than from three to five examples. The development or elaboration of an example is also a feature used to compute the essay score. A fully developed example with an introduction, thesis, and strong supporting material will contribute to the final score.

A. Sources

The sample code used in this manual can be downloaded from `http://emustru.sf.net`. You can use Ant (`http://ant.apache.org`) or the Eclipse platform (`http://www.eclipse.org`) to compile the Java code. Several open source products have been used in Emustru. Open source software is constantly under development and therefore there will be features in newer versions of some tools that will replace the ones mentioned in this manual. However, an effort has been made to use functions that are not likely to become obsolete in the near future.

Table A.1.: Sources and Licenses

Name	License
WordNet	`http://wordnet.princeton.edu/license`
LingPipe	`http://alias-i.com/lingpipe/web/download.html`
Lucene	`http://www.apache.org/licenses/`
Apache	`http://www.apache.org/licenses/`
MySql	`http://www.mysql.com/about/legal/`
Project Gutenberg	`http://www.gutenberg.org/wiki/Gutenberg:The_Project_Gutenberg_License`
Brown Corpus	`http://www.hit.uib.no/icame/brown/bcm.html`
FreeTTS	`http://freetts.sourceforge.net/license.terms`
Google Web Toolkit	`http://code.google.com/webtoolkit/terms.html`
Emustru	`http://www.gnu.org/licenses/gpl-2.0.html`

Source Code

The directory structure of the Emustru code is shown below in Table A.2. The root directory contains the PhP code and the java directory contains the java source code and associated configuration files.

Table A.2: Directory Structure of Sample Code

Name	Description
/	Configuration, readme, install, and startup PhP scripts.
complete/	PhP scripts for the complete word game
docs /	Documentation
essay /	PhP scripts to evaluate essays
follow/	PhP scripts to guess the following word
grammar /	PhP scripts for sentence completion and error ID identification
icons/	Images
java /	Root directory for Java code
java/classes/	Jar and class files for java code
java/data/	Configuration files for java code
java /lib/	Third party jar files
java/src/	Source Java files
lib/	Common PhP scripts
relationship/	PhP scripts for the relationship game
scramble /	PhP scripts for the scramble game
sentence/	PhP scripts for the sentence completion quiz
signon/	PhP script to signon
spelling	PhP scripts for the spelling quiz
vocabulary	PhP scripts for the vocabulary quiz
wordnet	PhP scripts to handle WordNet functions

Bibliography

[1] http://en.wikipedia.org/wiki/Computer-assisted_language_learning, Computer Assisted Language Learning (CALL).

[2] http://www.camsoftpartners.co.uk/freestuff.htm Free resources and articles on Computer Assisted Language Learning.

[3] http://wordnet.princeton.edu The WordNet lexical database for English.

[4] http://ftp.ets.org/pub/res/erater_iaai03_burstein.pdf Criterion: Online essay evaluation: An application for automated evaluation of student essays.

[5] http://en.wikipedia.org/wiki/Brown_Corpus The Brown Corpus.

[6] http://www.gutenberg.org/wiki/Main_Page The Project Gutenberg..

[7] http://www.quizlet.com Quizlet: Flashcards, vocabulary memorization, and word games.

[8] http://www.alias-i.com The LingPipe Computational Linguistics software.

[9] Page, E. B. and Petersen, N.S.: The computer move into essay grading. Upgrading the ancient test. Phi Delta Kappa, 76(7), 561-565.

[10] Burstein J., The E-rater Scoring Engine: Automated essay scoring with natural language processing, in Automated Essay Scoring: A Cross Disciplinary Perspective, Lawrence Erlbaum Associates, 2003, pp 113-121.

[11] Attali Y. and Burstein J.: Automated Essay Scoring With e-rater® V.2, The Journal of Techonology Learning and Assessment, Vol. 4, No. 3, February 2006.

[12] `http://www.vantagelearning.com/school/products/intellimetric/`, Intellimetric, Vantage Learning.

[13] `http://www.knowledge-technologies.com/prodIEA.shtml`, Intelligent Essay Assessor, Prentice Hall.