Project 1 for CS 421 - University of Illinois at Chicago
Emily Austin ejausti2@uic.edu
Nigel Flower nflowe3@uic.edu

In the main directory, there is a python file named auto_grader.py. Simply run the run the file with the python interpreter or command line. The output of the program is stored into a text file named result.txt. The program goes through a directory that contains all the files in it. Tested under Python 2.7.

Technique:

Spelling:

Our program utilized wordnet, going through every single word and checking to see if wordnet returned a definition for each word. If not, an empty list was returned, and in these cases we marked the word to be misspelled. There were a number of words that did not appear in wordnet such as function words, pronouns, clitics and neologisms which are in fact English words. In order to prevent these words being marked as misspelled, we checked to see if a word was in that set if our call to wordnet returned an empty list. The percentage of correct words was computed by dividing the total number of correctly spelled words over the total number of words. Percentages scoring 98% were given a 5, 97% a 4, 96% a 3, 93 - 95% a 2 and anything below 93% a 1.

Verb Agreement:

Before doing anything, we created two sets which each include the pos tags for singular nouns and plural nouns respectively. We also make a set containing a list of commonly used English modals because they behave similarly to verbs but are instead tagged as adverbs by the pos tagger. We included common misspellings as well such as “cant” and “dont” because the pos tagger would commonly tag them as nouns, so they would not show up as we iterate through a list and check of all the pos tags.
We first read the text from a file and tokenize it using the NLTK’s word_tokenize function which gives us a list containing tuples of every word and its corresponding pos tag. After that, we create extract the pos tags from the list of tuples using one of Python’s list comprehensions. Then we iterated through the list and checked if the improper verb tag followed a noun of either number. In the case of modal verbs, we checked whether there was a verb in the base form, “VB” or an adverb following it; if there was, then we continue in the loop, otherwise we mark it as an error. Finally, the final number of errors were divided by the total number of verbs and then subtracted by one to yield the percentage of correct verbs.

Verb Tense:

We used the data from the pos tags to check whether each individual sentence had consistent verb tenses. If there was a present and a past tense verb, then we marked it as an inconsistency. At the end all the inconsistencies were added up and then divided by the number of sentences, which gave us our correctness percentage.

Sentences:
The number of sentences were calculated using sent_tokenize, and the total number of words was counted using a regular expression filtering out all the alphanumeric strings. The lexical richness of the text (average number of words per sentence) was calculated by dividing the number of words by the number of sentences.

The score breakdown is as follows:

17+ : high
15+ : medium
14- : low

TODO:

Look at parsed syntax trees to judge how well the sentences are formed.
Mark sentence fragments as errors.
We can use the information from the syntax trees to improve our verb agreement scores; for example, “The job of syntacticians is to describe the grammars of language” gives us an error because “syntacticians” is followed by “is,” and even though job is the head of the NP, our pos tagging data says that this is an error.