Algorithmic Explanation:

For many NLP problems, data is just as important as how you implement your word correction algorithm. A common theme is utilizing word frequency and n-gram frequency in the English language. For my submission I extracted a word-to-count .txt file and a two-word-bigram-to-count .txt file from the URL given to me and created two dictionaries from them. Henceforth, I will refer to the word-to-count dictionary and the two-word-bigram-to-count dictionary as word count and bigram count respectively.

The algorithm itself can be characterized as a brute force word search. First turn the error string into a list of words and punctuation. Then iterate through the list two terms at a time. The two terms will fall under two cases:

The first case is the main one: when we run into two words consecutively. In such a case, we will create a list of replacements for each word, make sure each replacement exists in our dictionaries, and then create bigrams out of word one replacements and word two replacements. Finally, we will generate a score for each two word set defined by [C1*word1 count + C2*word2 count + C*association score] and find the max.

To find replacements, we will utilize four operations: add, remove, substitute, and neighbor swapping. Add has 27 options, the alphabet plus the space character. Note adding the space character can create valid bigrams out of single words so we need to check the bigram dictionary whenever we get our replacement list. If we do not find any words on the first edit distance i.e. one swap, removal, add, or substitute, we will find words two edit distances away, e.g. one swap, then one add. This will continue until we hit the max edit distance.

There are some complexities and salient things to note about our first case formula. One, the association score is calculated based the bigram (word1 + " " + word2) existing in the bigram count; the algorithm punishes the score heavily if the bigram does not exist. C1 and C2 are multipliers if the original word shows up for word1 or word2 as we always consider the original word itself as a valid replacement. Let me explain the existence of C1 and C2 with an example. A bigram like "the the" has a very high score and might surpass the penalty for not being a bigram because the individual word count scores are so high. Thus, we want to find replacements if the two words exist in word count and have high counts (above a threshold) but form a bigram that does not exist (or is below a count threshold). And it is likely the typist meant to say one of the high count words since they are so common. Since we do not know which one we will give a reward multiplier to word1 or word2 if their individual originals appear again. In practice I found the multipliers to work quite effectively. I would imagine there are better ways to deal with the "the the" problem. Finally C is a multiplier for the association score because the counts of two word bigrams will naturally be smaller than single words. The mean value in word count is 1764374 which is about 150 times larger than the mean value in bigram count which is 12711.

The second case happens when we run into punctuation marks. If both terms are punctuation marks skip forward two, else if the first term is a punctuation mark skip forward one, else if the second term is a punctuation mark we will simply find the best replacement for the first term which must be a word.

A lot of my algorithm is centered around big error strings. However, there are a few options I added in that will give more accurate results for smaller error strings.

One is changing the STEP_SIZE from two (default) to one. This means that we will go through the list with overlap: 0 1, 1 2, 2 3, 3 4... You can change the STEP_SIZE for larger error strings but the benefit is marginal and not worth the reduction in runtime. That being said the reader is free to do so. Two is setting IMPLEMENT_SMALLER_TEXT to True which will find all the replacements within the max edit distance even if the edit distances before give valid replacements. Setting this to true for larger texts will cause the runtime to spike exponentially and it is definitely not recommended.

Problem Areas:

Although the data was extensive, it was not perfect by any means. The threshold mentioned above, in the explanation of the first case, is 400,000. Often words below this threshold in both dictionaries were rare or invalid English. For example, "howlanguage" does not correct to "how language" simply because "how language" does not exist in the bigram count. Instead it corrects to "to language". Ultimately, the sheer quantity of garbage values in the word count actually makes the relationship between mean word count value and mean bigram count value closer to the scale of 1000, which is why C is 4000 by default as we want to prioritize the relationship.

The max edit distance is defaulted to 2. This is because an edit distance of 3 would essentially stall the algorithm looking to replace a word like "diffefentapproacjz". This was a huge trouble area for me because each increase to the edit distance increased the run time exponentially. I think a Bloom Filter, memoization, and better data might help here. But looking around I did not find a definitive answer to this problem so very misspelled words could not be fixed.

Areas for Improvement and Exploration:

As mentioned a Bloom Filter would help a lot here. It would help the space and time complexity on the scale of about 2.5x. However, dealing with false positives requires us to put everything in a dictionary anyway which is why I omitted it.

The score for word or word set is entirely probabilistic and dependent on a word or a bigram showing up in the data set. Thus, the more data sets we glean from the stronger our algorithm

becomes. More specifically the domain knowledge of our algorithm will expand, allowing us to tackle more unique terms like "fused participles" which are not contained in the one data set we used. To better handle our test string, the content of which discusses grammar and writing, we should gather pertinent terms from a high school English textbook or an SAT verbal textbook.

Furthermore, if our test string is large enough, it would be worth it to add memoization to catch errors like "writimgnis" the next time they appear in the text.

Another method to try that would yield better run time results is a meet in the middle approach for the initial word count and bigram count hash table. This will increase fetching time but may give substantially faster albeit less accurate results. We would first create terms within edit distance 2 away (deletes only) from each dictionary term and pair them together with the original term to the dictionary. Then we would create terms within edit distance 2 away, again deletes only, and search for them in the dictionary. In my view, although the method is faster when we the initial pre-fetching, the accuracy of the end result is dubious.

Time and Space Complexity:

Although a Bloom Filter would reduce the run time by quite a bit, the big O would remain the same as coefficients and constants would be eschewed.

Time Complexity (Ignoring Dictionary Creation): O(m*((k*a)^e) with m as the number of terms in the error string, k as the max length of string, e as the edit distance, and a as the length of add options. This algorithm is very expensive but will get smarter as you analyze more text with memoization. With a Bloom Filter it will be the same big O.

Space Complexity: O(max(n,m,(g)^e)): with m, e and a being the same, n is the size of the data set, g is the amount of replacements added to the list in the first edit distance loop. The g value can vary substantially depending on the quality of word count and bigram count. Realistically it will always be O(n) unless you are doing something strange.

Examples:

MAX_EDIT_D = 2 STEP_SIZE = 2

Among the many challenges of writing is dealing with rules of correct usage: whether to worry about split infinitives, fused articles, and the meanings of words such as fortuitous, decorate and comprise. Supposedly a writer has to choose between two radically diffefentapproacjz to these rules. Prescropgivists describe how language ought to be used. They uphold standards or excellence and a respect for the best of our civilization, and are a bulwark against relativism, vulgar populism and the dumbinfdown of literatur culture. Descriptovists describe to language

actually issued. They bekuevethat the rules of correct usage are nothing more than the secret handshakes of the ruling clods, designed to ker the masses intheir place. Language is an organicptoduct of human creativity, say the fescriptkvists, as a people should be allowed to write however the please.

MAX_EDIT_D = 2 STEP_SIZE = 1

Among the many challenges of writing is dealing with rules of correct usage: whether to worry about split infinitives, fused articles, and the meanings of words such as fortuitous, decorate and comprise. Supposedly a writer has to choose between two radically diffefentapproacjz to these rules. Prescropgivists describe how language or to be used. They uphold standards or excellence and a respect for the best of our civilization, and are a bulwark against relativism, vulgar populism and the dumbinfdown of literate culture. Descriptovists describe to language actually is used. They bekuevethat the rules of correct usage are nothing more than the secret handshakes of the ruling clods, designed to her the masses in their place. Language is an organicptoduct of human creativity, say the fescriptkvists, as a people should be allowed to write however the please.