University of Texas at Dallas
CS 6320 : Natural Language Processing
Fall 2014
Instructor: Yang Liu

Spelling Corrector

• Team members:

1.Ameya Parab
UTD ID: 2021166954
email: axp132530@utdallas.edu

2. BinitaSinha
UTD ID: 2021196721
email:bxs133130@utdallas.edu

• Abstract

Spelling Correction and Prediction is a very important component of any system processing text.Creation of the text is prone to errors, people make typing errors or they sometimesignore the correct spelling of the word. Therefore they need spelling correctors. Anotherissue where spelling correction is highly necessary is optical character recognitionsystems. These systems often make errors during recognition due to bad quality ofinput, change of fonts or insufficient training. Today, many spelling correction functionsare available in many languages like English.

• Introduction

How does it work? In this project we have used Edit distance to correct the incorrect word from the user. In order to know the best use of the word in the context say "should lates be corrected to "late" or "latest"?, we use Bayes' Theorem. We have also combined the concept of unigram and bigram to predict the next correct word from the context .

- ## Edit Distance and Bayes' Theorem

Given a word, we are trying to choose the most likely spelling correction for that word (the "correction" may be the original word itself). There is no way to know for sure (for example, should "lates" be corrected to "late" or "latest"?), which suggests we use probabilities. We will say that we are trying to find the correction *c*, out of all possible corrections, that maximizes the probability of *c* given the original word *w*:

$\text{argmax}_c\ P(c|w)$

By Bayes' Theorem this is equivalent to:

$\text{argmax}_c\ P(w|c)\ P(c)\ /\ P(w)$Since $P(w)$ is the same for every possible *c*, we can ignore it, giving:

$\text{argmax}_c\ P(w|c)\ P(c)$

There are three parts of this expression. From right to left, we have:

1. $P(c)$, the probability that a proposed correction *c* stands on its own. This is called the **language model**.
2. $P(w|c)$, the probability that *w* would be typed in a text when the author meant *c*. This is the **error model**"
3. argmax_c, the control mechanism, which says to enumerate all feasible values of *c*, and then choose the one that gives the best combined probability score.

One obvious question is: why take a simple expression like $P(c|w)$ and replace it with a more complex expression involving two models rather than one? The answer is that $P(c|w)$ is *already* conflating two factors, and it is easier to separate the two out and deal with them explicitly. Consider the misspelled word *w*="thew" and the two candidate corrections *c*="the" and *c*="thaw". Which has a higher $P(c|w)$? Well, "thaw" seems good because the only change is "a" to "e", which is a small change. On the other hand, "the" seems good because "the" is a very common word, and perhaps the typist's finger slipped off the "e" onto the "w". The point is that to estimate P(*c*|*w*) we have to consider both the probability of *c* and the probability of the change from *c* to *w* anyway, so it is cleaner to formally separate the two factors.

Now we are ready to show how the program works. We will read a text file,**correction.txt** which consists of about a million words. We then extract the individual words from the file (using the function words, which converts everything to lowercase, so that "the" and "The" will be the same and then defines a word as a sequence of alphabetic characters, so "don't" will be seen as the two words "don" and "t"). Next we train a probability model, which is a fancy way of saying we count how many times each word occurs, using the function train. It looks like this:

At this point, NWORDS[w] holds a count of how many times the word w has been seen. There is one complication: novel words. What happens with a perfectly good word of English that wasn't seen in our training data? It would be bad form to say the probability of a word is zero just because we haven't seen it yet. There are several standard approaches to this problem; we take the easiest one, which is to treat novel words as if we had seen them once. This general process is called **smoothing**, because we are smoothing over the parts of the probability distribution that would have been zero. Now let's look at the problem of enumerating the possible corrections *c* of a given word *w*. It is common to talk of the **Edit Distance** between two words: the number of edits it would take to turn one into the other. An edit can be a deletion (remove one letter), a transposition (swap adjacent letters), an alteration (change one letter to another) or an insertion (add a letter). Here's a function that returns a set of all words *c* that are one edit away from *w*:

This can be a big set. For a word of length *n*, there will be *n* deletions, *n*-1 transpositions, 26*n* alterations, and 26(*n*+1) insertions, for a total of 54*n*+25 (of which a few are typically duplicates). The literature on spelling correction claims that 80 to 95% of spelling errors are an edit distance of 1 from the target.

We had some intuitions: mistaking one vowel for another is more probable than mistaking two consonants; making an error on the first letter of a word is less probable, etc. By "known word" we mean a word that we have seen in the language model training data -- a word in the dictionary. We can implement this strategy as follows:

The function correct chooses as the set of candidate words the set with the shortest edit distance to the original word, as long as the set has some known words. Once it identifies the candidate set to consider, it chooses the element with the highest P(*c*) value, as estimated by the NWORDS model.

• Unigram Bigram Language Model

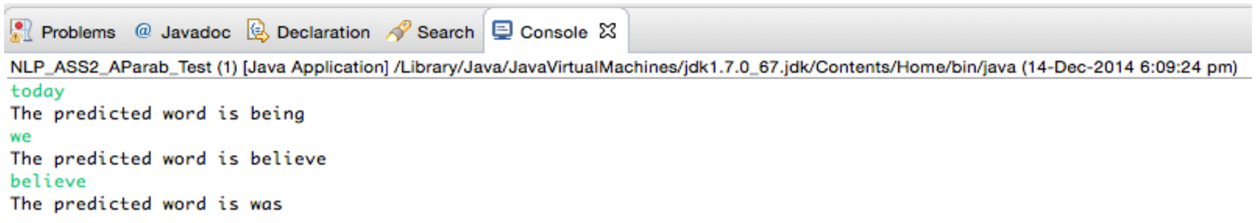
Guessing the next word (or word prediction) is an essential subtask of speech recognition, hand-writing recognition, augmentative communication for the disabled, and spelling error detection. In such tasks, word-identification is difficult because the input is very noisy and ambiguous.

Guessing the next word turns out to be closely related to another problem :computing the probability of a sequence of words . Algorithms that assign probability to a sentence can also be used to assign a probability to the next word in an incomplete sentence, and vice- versa. The models of word sequences we will consider are probabilistic models ; ways to assign probabilities to strings of words, whether for computing the probability of an entire sentence or forgiving a probabilistic prediction of what the next word will be in a sequence. We can use these relative frequencies to assign a probability distribution across following words. So if we've just seen the string, we can use the probability .07 for the and .00001 for rabbit to guess the next word. But suppose we've just seen the following string: Just then, the white. In this context rabbit seems like a more reasonable word to followwhite than the does. This suggests that instead of just looking at the individual relative frequencies of words, we should look at the conditional probability of a word given the previous words. That is, the probability of seeing rabbit given that we just saw white (which we will represent as P(rabbit white)) is higher than the probability of rabbit otherwise.Given this intuition, let's look at how to compute the probability Simple (Unsmoothed) N-grams 195complete string of words (which we can represent either as w1 : : : wn or w If we consider each word occurring in its correct location as an independent event, we might represent this probability as follows:P(w1; w2 : : : ; wn We can use the chain rule of probability to decompose this probability:1; wn) (6.4)) = P(w1)P(w2 jw1)P(w3jwnP(wkjwk=12) : : : P(wnjw11n1k1) (6.5)1But how can we compute probabilities like P(wnjwknow any easy way to compute the probability of a word given a long sequence of preceding words. (For example, we can't just count the number of times every word occurs following every long string; we would need far too .We solve this problem by making a useful simplification: we approximate the probability of a word given all the previous words. The approximation e will use is very simple: the probability of the word given the singleprevious word! The bigram model approximates the probability of a word BIGRAM given all the previous words P(preceding word P(P(rabbitjJust the other I day I saw a) (6.6)we approximate it with the probabilityP(rabbitja) (6.7)This assumption that the probability of a word depends only on theprevious word is called a Markov assumption. Markov models are the class MARKOV of probabilistic models that assume that we can predict the probability ofsome future model without looking too far into the past.

• Evaluation

1. Only Prediction

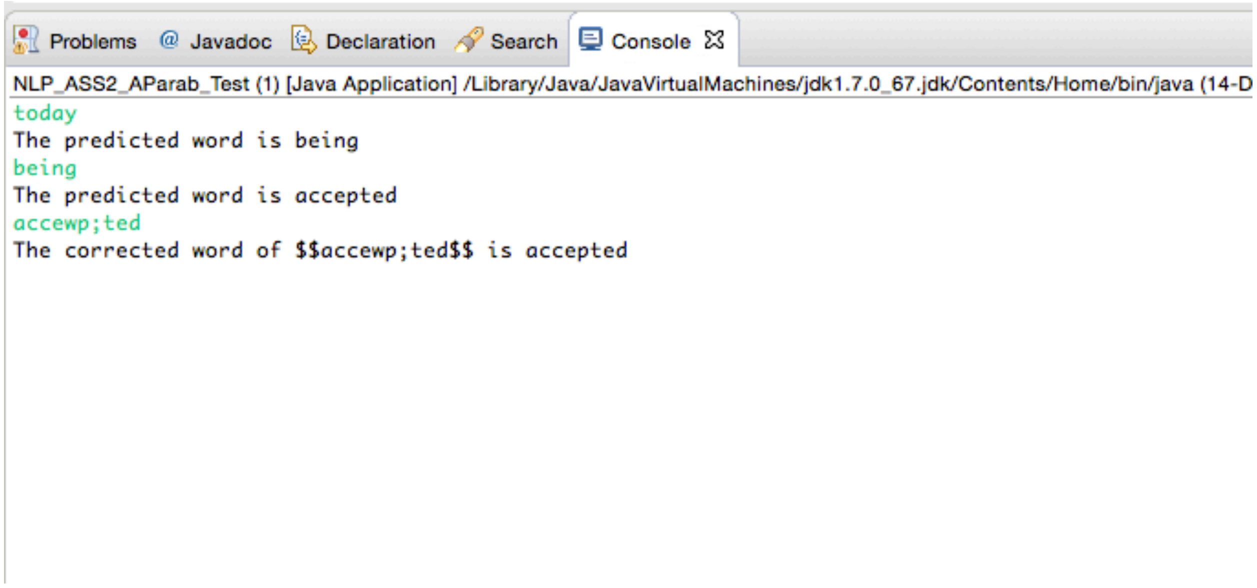
Here in this case the words are correct and it doesn't need any correction, so only we get the prediction of the next best word.



```
NLP_ASS2_AParab_Test (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (14-Dec-2014 6:09:24 pm)
today
The predicted word is being
we
The predicted word is believe
believe
The predicted word is was
```

2. Prediction & Correction

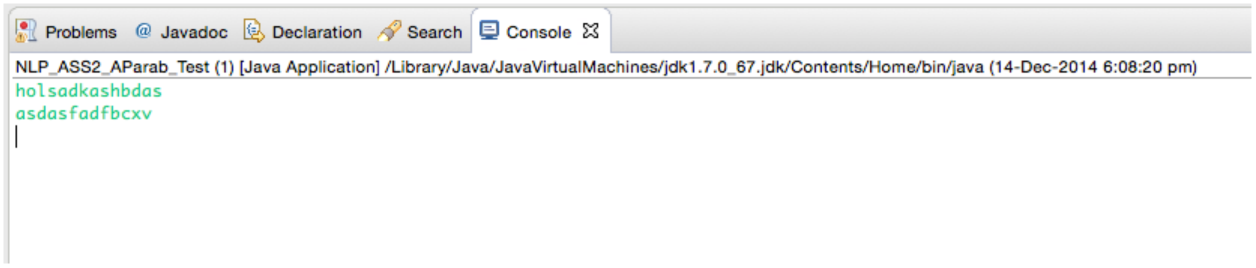
Here in this case we get both correction and predication and also the word is in the corpus in correction.txt


Problems @ Javadoc Declaration Search Console

```
NLP_ASS2_AParab_Test (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (14-D
today
The predicted word is being
being
The predicted word is accepted
accewp;ted
The corrected word of $$accewp;ted$$ is accepted
```

3. Nothing When there is a new word out of corpus

Here in this case both the word isn't in correction and training set so we don't get anything which means the word is new and is added to the corpus with some backoff probability sp when it comes next time it is predicted and the machine is learning.


Problems @ Javadoc Declaration Search Console

```
NLP_ASS2_AParab_Test (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (14-Dec-2014 6:08:20 pm)
holsadkashbdas
asdasfadfbcxv
```

- **Problems Encountered**
  1. The main problem, which was encountered, was to get the best prediction for the next word.

2. Also for spelling correction we had to use multiple theorems to get the best corrected spelling

- **Pending Issues**
  1. The main issues is to improve the training of the corpus. bIgger the corpus bigger is the execution time. Need to write more efficient algorithms to reduce the execution time.
  2. Also the improve the prediction by using trigram models.

- **Future Improvement**
  1. One of the pending issues is to improve the accuracy of the project. Currently the accuracy is 70%. Using some more complex algorithms we can definitely increase the accuracy of the project.