Natural Language Processing

Final Project Report

Topic: Word Auto-Suggestion System for News Articles

CS 6320.001

Submitted by:

Praful Bhosale(pbb140230)

1. Abstract

There has been many Softwares which has the Auto-correction or spelling correction of words feature (ex. MS word) but there is no software as such for Auto-suggestion of next word. So I came up with an idea to implement such an application which suggests most likely next word once user provides any valid word as an input.

This semester while studying Natural Language Processing subject as a part my course work, I have learned many interesting concepts such as N-gram models, Markov Chain Models, Hidden Markov Models (HMM) and POS tagging etc. So I decided to utilize these concepts to develop my NLP project which is word auto-suggestion system.

We can use such application for writing articles of various fields such as news, technology and also for chat-messaging applications.

2. Introduction

My project is about suggesting most likely next word based on the given previous words which in turn speed up the writing process for articles or even chat messaging on daily basis.

N-gram language models are a used in many machine translation systems. In this project, I present Bi-gram language model and Markov Chain model implementation that are both highly compact and fast to query. As my application is regarding word auto-suggestion for news articles, I have included recent news articles, blogs into my corpus.

I have developed the GUI using java swing framework which has text area where user can write an article. Whenever user enters any word and hits space bar, an application automatically suggests list of most likely next words into list section provided on UI. Once user selects the desired word from the list, that word gets added into text area of article as a selected next word.

We have to understand the fact that, corpus plays a crucial role in terms of finding the most likely next word suggestions. If the corpus has limited sentences, then my application may provide empty result set for the typed word. In order to overcome this limitation, I am allowing user to update the corpus while writing article by allowing him append his content to the existing corpus. My application learns and remembers the writing style of a user and provides more suggestions for next word, thus implements supervised learning in a very efficient manner.

3. Previous Research

There no as such research paper found related to word auto-suggestion. Few papers mentioned in references provides some information regarding what NLP concepts one should use in order to implement such application.

4. System Design

Proposed Method:

I am using N-gram (specifically Bi-gram) model along with Markov Chain Model. A common method of reducing the complexity of n-gram modeling is using the Markov Property. The Markov Property states that the probability of future states depends only on the present state, not on the sequence of events that preceded it. This concept can be elegantly implemented using a Markov Chain storing the probabilities of transitioning to a next state.

Let's look at a simple example of a Markov Chain that models text using bigrams. Consider I have included the following sentence into my corpus.

S = "I am Sam. Sam I am. I do not like green eggs and ham."

My program will create a list of bigrams from a piece of text and store it into a HashMap <Key, Value> where Key is Bigram and Value is bigram count (i.e. frequency of bigram into given corpus).

[< ('I', 'am'), 1>, < ('am', 'Sam.'), 1>, < ('Sam.', 'Sam'), 1>, < ('Sam', 'I'), 1>, < ('I', 'am.'), 1>, < ('am.', 'I'), 1>, < ('I', 'do'), 1 >, <('do', 'not'), ('not', 'like'), ('like', 'green'), ('green', 'eggs'), ('eggs', 'and'), ('and', 'ham.')]

Listing the bigrams starting with the word 'I' results in: I am, I am., and I do. If we were to use this data to predict a word that follows the word 'I' we have three choices and each of them has the same probability (1⁄3) of being a valid choice. Modeling this using a Markov Chain results in a state machine with an approximately 0.33 chance of transitioning to any one of the next states.

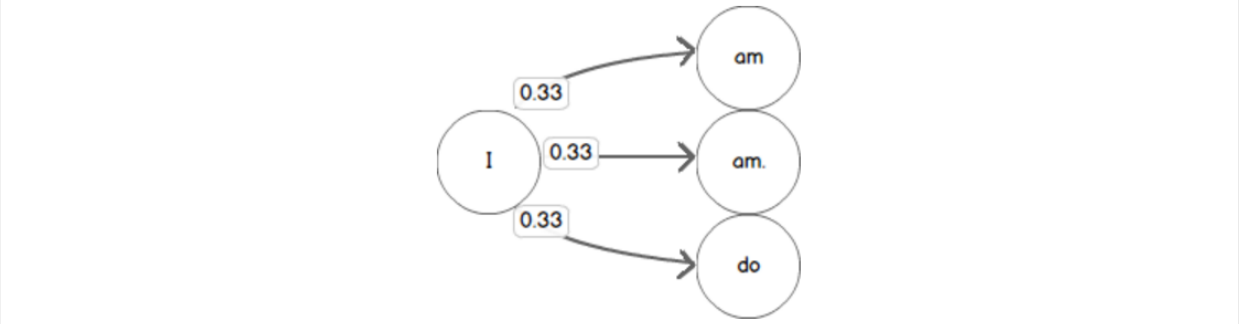
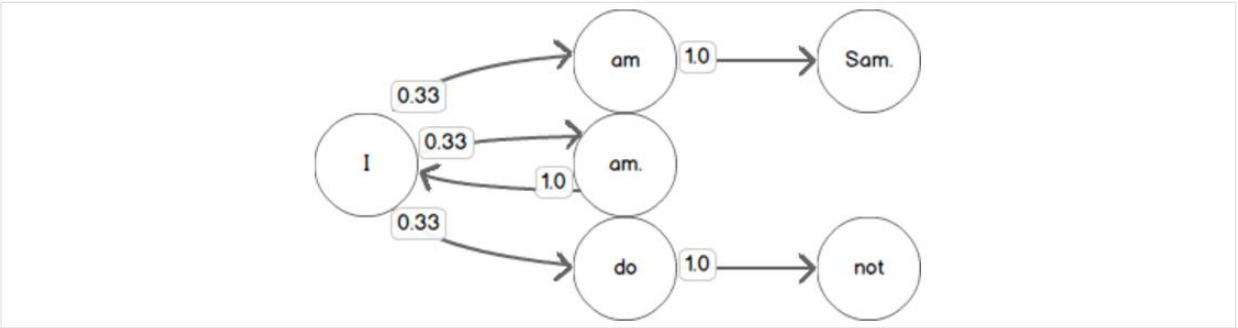


Fig 1. Transitions from 'I'

We can add additional transitions to our Chain by considering additional bigrams starting with am, am., and do. In each case, there is only one possible choice for the next state in our Markov Chain given the bigrams we know from our input text. Each transition from one of these states therefore has a 1.0 probability.



Now, given a starting point in our chain, say I, we can follow the transitions to predict a sequence of words. This sequence follows the probability distribution of the bigrams we have learned. For example, we can randomly sample from the possible transitions from I to arrive at the next possible state in the machine.

The Markov Chain implementation is a simple dictionary with each key being the current state and the value being the list of possible next states. For example, after learning the text I am Sam. our dictionary would look like this.

```
{
   'I': ['am'],
   'am': ['Sam.'],
}
```

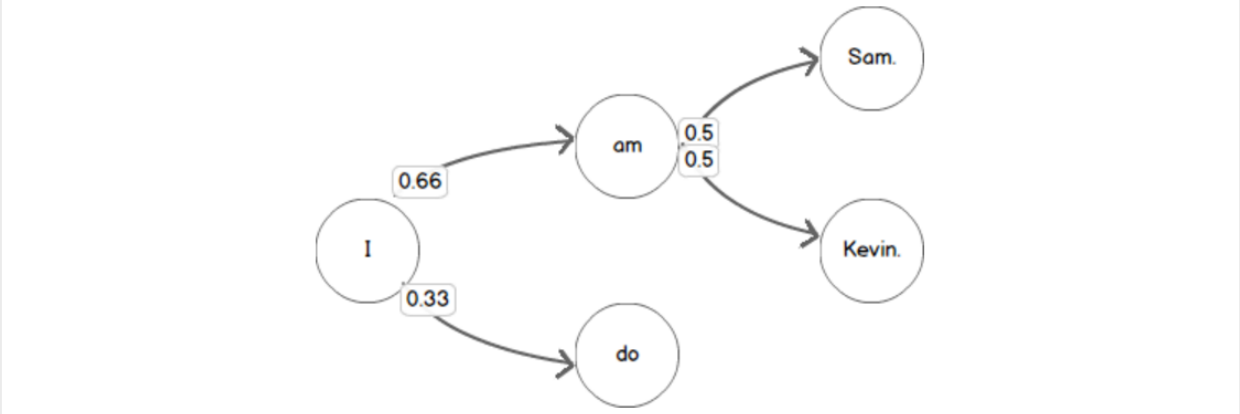
After learning entire corpus, program will implement a basic Markov Chain that creates a bigram dictionary which would look like this.

```
{
'I': ['am', 'am.', 'do'],
 'Sam': ['I'],
 'Sam.': ['Sam'],
 'am': ['Sam.'],
 'am.': ['I'],
 'and': ['ham.'],
 'do': ['not'],
 'eggs': ['and'],
 'green': ['eggs'],
 'like': ['green'],
 'not': ['like']
```

}

We can then transition to a new state in our Markov Chain by randomly choosing a next state given the current state. If we do not have any information on the current state, we can randomly pick a state to start in. The transition probabilities between states naturally become weighted as we learn more text. For example, in the following sequence we learn a few sentences with the same bigrams and in the final state we are twice as likely to choose am as the next word following I by randomly sampling from the next possible states. For e.g. consider following statements 'I am Sam.' 'I am Kevin.' and 'I do.' will generate Markov chain of bigrams as follows: {'I': ['am', 'am', 'do'], 'am': ['Sam.', 'Kevin.']}

The state machine produced by our code would have the probabilities in the following figure.



Putting it all together we have a simple Markov Chain that can learn bigrams and babble text given the probability (or count) of bigrams that it has learned. Markov Chain's are a simple way to store and query n-gram probabilities. Program uses these probabilities to suggest all the possible next words to user in descending order of bigram probabilities (or count) by keeping most likely word at the top of the list.

Concepts:

Project uses the below mentioned concepts from NLP -

Markov Chain Model

- A set of states, Q = q1, q2…qN where qt is the state at time t.
- A set of transition probabilities, A = a01 a02…an1…ann. Where each aij represents the probability of transitioning from state i to state j
- Markov Assumption: Current state only depends on previous state $P(qi \mid q1...qi\text{-}1) = P(qi \mid qi\text{-}1)$

Bigram Model

It approximates the probability of a word given all the previous word by the conditional probability of the preceding word.

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1})$$

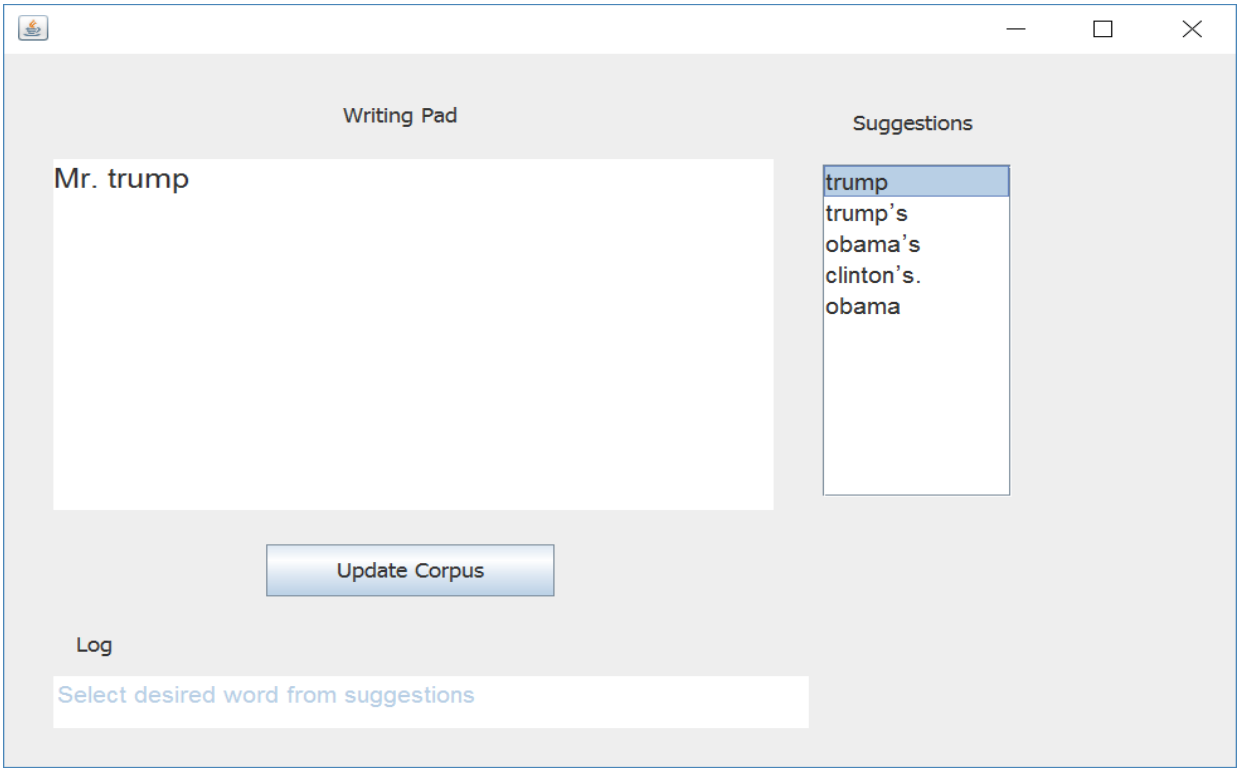
$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k | w_{k-1})$$

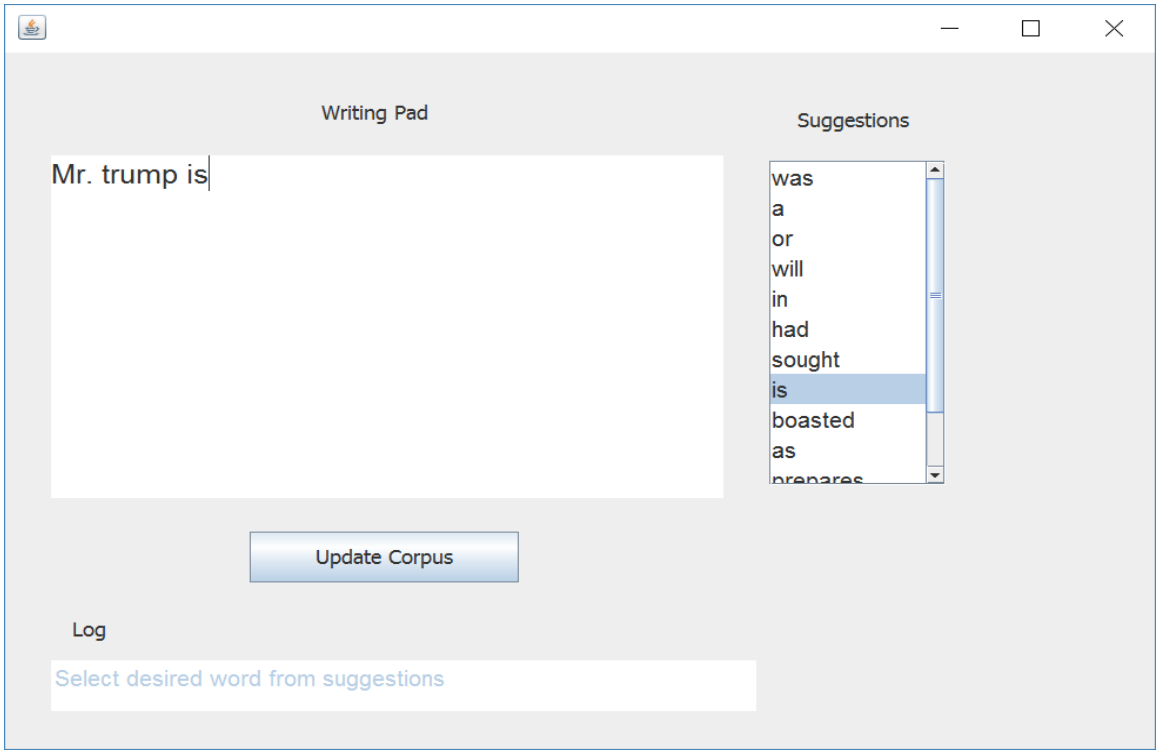
Algorithm: (steps discussed in proposed method)

1. Read corpus.txt file line by line
2. Generate dictionary of distinct tokens and count using Hashtable (tokenMap)
3. Generate dictionary of distinct bigrams and count using Hashtable (bigramMap)
4. Iterate through bigramMap to generate Hashtable which stores Markov chain for each word.
5. Whenever user types new word and hits space, application search this word and displays list of suggestions mentioned in resp. Markov chain if word exists in Hashtable.

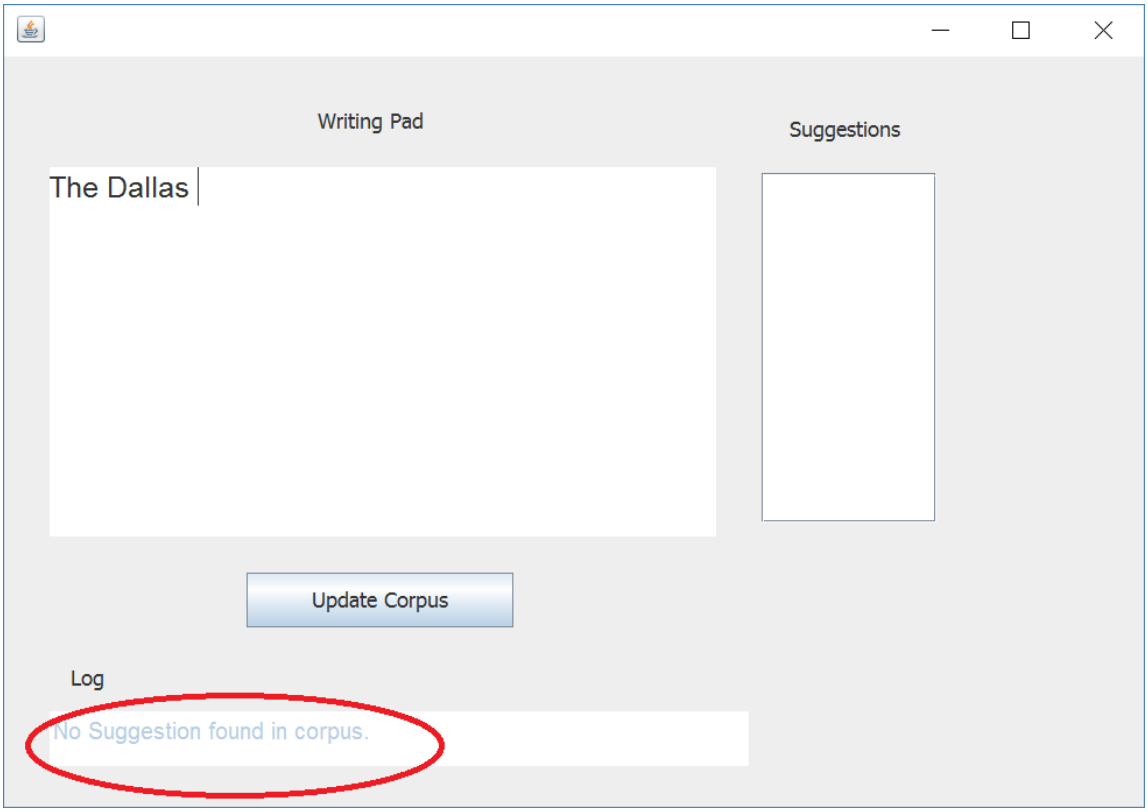
5. Input and Output

Once user types any word in the given writing pad text area and hits space, application displays next word suggestions as shown in below fig. Next if user clicks on any word from suggestions, then selected word gets added into writing pad, next to already typed word.

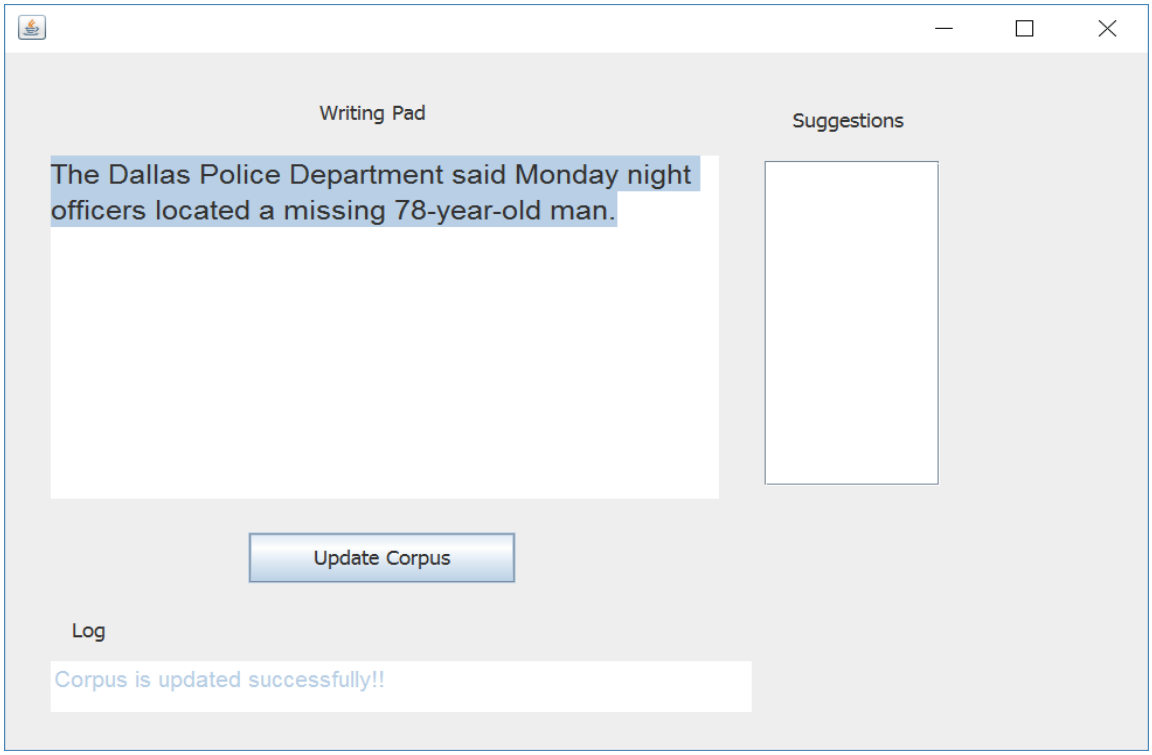




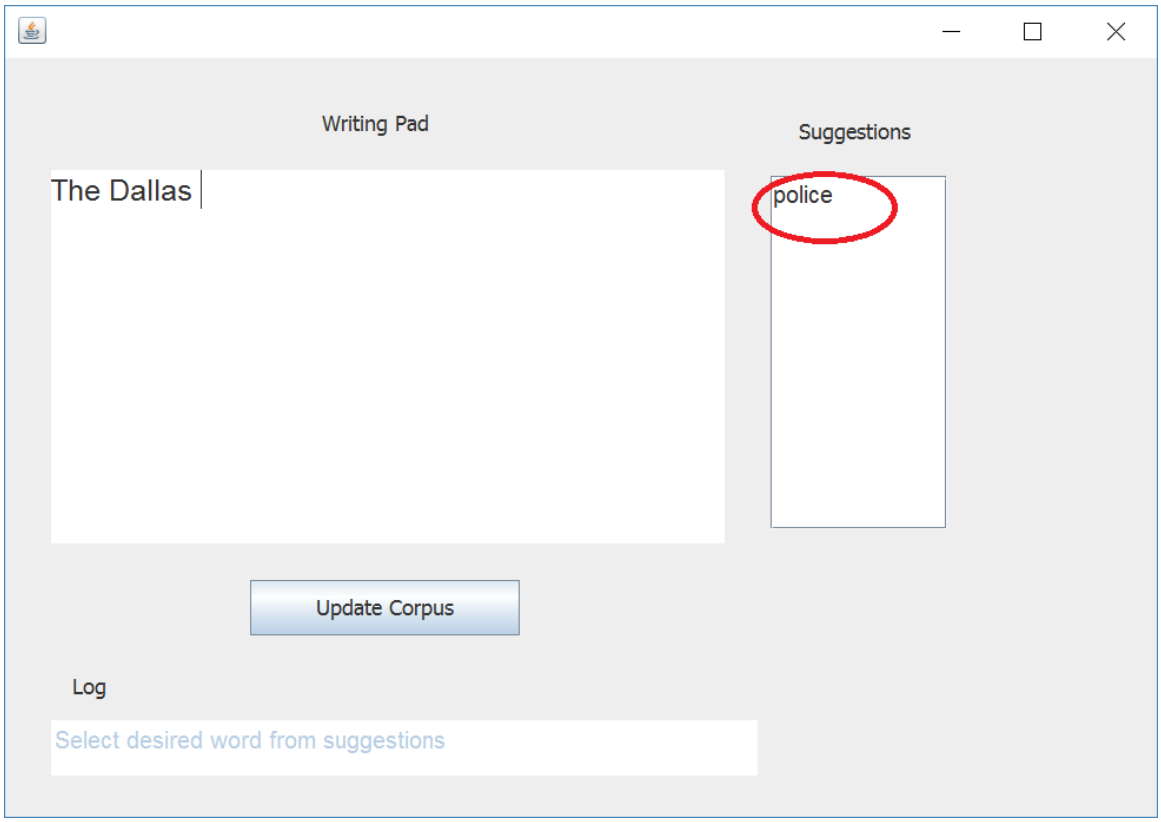
As we mentioned earlier, there is limitation for this application which is corpus content. If any bigram has zero probability (or count), then system returns empty result set as shown below.



My application overcomes this problem by allowing user to update corpus by adding the sentence into corpus. By adding this feature, I have imposed supervised learning.



So when time user enters the same word next time, system provides the suggestions as shown below:



6. Data Used

I have created corpus using some recent news articles from internet news blogs, news channel web site sources.

7. Resources

I am using eclipse IDE with Java SWT framework to implement GUI for the application.
Programming has been done in core Java.

8. Conclusion and Future Scope:

Allowing user to update corpus helps word auto-suggestion system to learn writing style of a user and improves the performance and does not reach any biased conclusion if bigram is not present. The data size, semantic analysis etc. are more important features as they increase learnability. In future, there is a scope to add other features like Trigram language model and Hidden Markov Models (HMM) which can improve performance and semantic analysis. There is a scope to incorporate grammar checking while adding any sentence into corpus, which will make sure valid next word suggestions every time when user types any word into system.

9. References

1] http://nlp.cs.berkeley.edu/pubs/Pauls-Klein_2011_LM_paper.pdf

2] https://lagunita.stanford.edu/c4x/Engineering/CS-224N/asset/slp4.pdf