Bayesian method This method was used to do a context based spelling correction for the sentences and phrases. This method was combined along with the trigrams based method, to obtain the final the suggestion. Also, the edit distance was included in the final score calculation for the suggestion.

This method was split into two parts. The first part is the training phase, where we used the Brown corpus to train the context and collocation words. Context, is the set of consecutive words that occurs before the given word. In this approach we used a k-context, that is the k previous words for a given word. After some experimenting, we found for k=20, the performance was the best. We also noted the k previous parts of speech that occured along in the context. Hence, the context feature had the context word and the parts of speech for various words occuring in the context. As for collocation, we used the k words that occured before the given word and the k words that occured after the given word. We found that for more than k=20, the collocation words no longer made sense. This also is a reasonable observation since, an average English sentence is usually about 15-17 words long. Hence, a window of 40 words accounts to approximately 2.5-3 sentences.

Once we trained our code on the Brown corpus, we serialised the various hash maps so that consequent runs of the program will be faster. Now, for an input, we tagged the parts of speech by using the most frequently seen parts of speech for that word in the corpus. This is an approximation since, some words have multiple parts of speech, and the parts of speech used in the sentence may not necessarily be the same as the one most frequently used. But, we found this to be a fair enough approximation in most cases. Once the parts of speech was tagged to the input, we then found the probability of $P(w_i|w_{i-1}.w_{i-2}..)$ and approximated this using the Naive Bayesian assumption (Here w_i is the mispelt word). We performed a similar thing on the parts of speech for the mispelt word. We then found the collocation words of the mispelt word in the given input and found an intersection with the confusion set's collocation in the corpus. The confusion set has been defined as the words with edit distance of atmost 3 from the mispelt word with words having lesser edit distance at a higher priority. We then assigned scores to these words in the confusion sets based on the amount on intersection and the number of times they were found around the particular word in the corpus. We then combined the above three scores in a ratio of 1:1:1, to obtain a final score for the candidate words.

One assumption our code makes is that only one word in the given sentence is mispelt. Hence, if two words are mispelt in the input sentence, then a new parts of speech will be aassigned to the other mispelt word and then the first mispelt word will be given suggestions. Hence, if the two mispelt words form critical words for giving sense to the sentence, our code will give almost irrelevant suggestions. For example, “Hitt the rod” is the input the words “Hit” and “road” form the critical parts in giving sense and both are mispelt. On the other hand, suppose we have “Hit thee rod”, in this case the code performs slightly better, although since “the” is an article and this part of speech plays an important role; this case is better than the previous case.

Context free spelling correction - Use of Confusion Matrix
We used the confusion matrix based approach for correcting spelling errors without a context. In this approach we first hardcoded a confusion matrix for substitution,deletion addition of words. We populated a dictionary of English words from the UNIX dictionary. This contained about 65000 words. We then maintaned a Hash Table of length to the words list of words having that length. This structure helped us in pruning the search space when we needed to look at candidate words. We limited the search space to an edit distance of atmost 3 since, beyond this we found it unreasonable to make a good suggestion anyways.

Once we read the input word, we get the ArrayList of words from dictionary which differ in length by atmost 3. Beyond this the edit distance anyway would be greater than 3 and hence, its useless to fetch it. Once we have this list, we now simulate the edit distance algorithm. And we assign scores based on the confusion matrix. For example, suppose the input was “rainin”, while performing edit distance algorithm we identify that the optimal way to convert this to “raining” is by inserting a “g” after the “n”. Hence, we use this value from the addition matrix and return that as the score. Note that, all the confusion matrix are δ smoothed using Laplace smoothing.

This was the idea taken from the paper “A spelling correction program based on a noisy channel model” by Kerningham , et. al. However, we made a small tweak to the scoring system to accomodate for slightly better results. For example, given the input “rainin”, the word “rain” receives a higher score over “raining”, since the values in confusion matrix suggests that adding the two respective characters occurs more frequently than deleting the “g”. Hence, we added an offset to the score by adding the edit distance value to the score proportionally. Hence, words at edit distance 1 will receive more weigthage than words with edit distance 2. The proportion that was added was decided based on experimentation.

The key element to the code which increased the speed considerably was maintaining the Hash table structure from the word length to the list of words. This got down the search factor from asymptotic $O(|V|)$ to $O(log|V|)$, where $|V|$ is the size of the vocabulary. Though for the number of words in the dictionary, this is not a significant improvement, this still gives the code more time on finding the edit distance on an average case.