**Report on**

**Spell Checker**

**Problem Statement**

Implement a part of a noisy-channel model for spelling correction. Here we are training the model to implement the spell Checker.

I have implemented models as mentioned below for respective problems.

Problem1.

Laplace Unigram Language Model

Problem2.

Laplace Bigram Language Model  
 My Custom Language Model

**Data Used**

We are using the writings of secondary-school children, collected by David Holbrook.

holbrook-tagged-train.dat: the corpus to train your language models

**Challenges Faced**

Understanding the training of the model and how to use it for spell errors. Explore different models to understand their functionality. The training set provided is not satisfactory huge to accurately correct the spelling errors. Initially I had implemented a trigram model but the accuracy was low as the probability of finding a phrase with exactly 3 tokes was less likely. But once I in-corporated the features of Laplace Unigram and Laplace bi-gram language model the accuracy was better that Laplace unigram and bigram language models.

**Methodology**

I have built 3 models

Laplace Unigram Language Model, Laplace Bigram Language Model, My Custom Language Model

**Laplace Unigram Language Model**

A unigram model with add-one smoothing. The major problem with N-gram language modeling is that it depends on finite training corpus. Therefore, some words will be missed from the corpus which are called unknown words. If any Ngramis missing, then the language model will assign a probability of 0(zero) to it. Smoothing is helps us to

prevent assigning zero probability for missing N-gram in the training corpus.

The pseudocode to build the training model and calculate the probability values is as below.

#Get data from teh training corpus

for wordDictionary in corpus.corpus:

for datum in wordDictionary.data:

word = datum.word

#Increment the count if teh word is found in dataum and building a unigram count that would be helpful to evaluate teh score in the scocre function.

self.unigramLaplaceDict[word] = self.unigramLaplaceDict[word] + 1

self.total\_val += 1

for word in wordDictionary:

num\_count = self.unigramLaplaceDict[word]

if num\_count > 0:

#Calculate the value

value += math.log(num\_count+1)

value -= math.log(self.total\_val + len(self.unigramLaplaceDict))

else:

value -= math.log(self.total\_val + len(self.unigramLaplaceDict))

**Laplace Bigram Language Model**

Let's say we have a text document with N unique words making up a vocabulary V, |V|=N|V|=N. For a bigram language model with add-one smoothing,

The probability of bigram model can be calculated as below:

P(wi|wi−1)=count(wi−1wi)+1/count(wi−1)+|V|

The pseudocode for bi-gram laplace language model is explained below:

for wordDictionary in corpus.corpus:

word\_1 = '<s>'

self.words[word\_1] = self.words[word\_1] + 1

word\_2 = ''

for datum in wordDictionary.data:

word\_2 = datum.word

self.total += 1

self.bigramLaplaceDict[(word\_1,word\_2)] = self.bigramLaplaceDict[(word\_1,word\_2)]+1

self.words[word\_2] = self.words[word\_2] + 1

word\_1 = datum.word

word\_2 = '</s>'

self.bigramLaplaceDict[(word\_1,word\_2)] = self.bigramLaplaceDict[(word\_1,word\_2)]+1

self.words[word\_2] = self.words[word\_2] + 1

For add-one smoothed bigram counts, we need to augment the unigram count by

the number of total word types in the vocabulary V:

**My Custom Language Model**

In custom model I have used all the features of Laplace Unigram and Bigram language model along with the tri-gram language model. The motivation is to use the best possible feature of different model to increase the accuracy of the spell checker algorithm. The logic of the custom model is explained below.

The probability of tri- gram model is calculated as below:

P(wi|wi−2wi−1)=count(wi−2wi−1wi)+1/count(wi−2wi−1)+|V2|

if count3 > 0: #trigram exists

value += math.log(count3)

value -= math.log(self.bigramDict[(word\_1,word\_2)])

elif count2 > 0: # no trigram, but bigram exists

value += math.log(0.4) + math.log(count2)

value -= math.log(self.unigramDict[word\_2])

else: # no trigram or bigram

value += math.log(0.4) + math.log(self.unigramDict[word\_3]+1)

value -= math.log(self.total + (len(self.unigramDict)))

**Results:**

The results of the accuracy for different models are as follows. Accuracy is calculated by , the number of valid corrections, divided by the number of test sentences.

Laplace Unigram Language Model:

Words corrected: 52

Total number of Words: 471

Accuracy of Model: 0.110403

Laplace Bigram Language Model:

Words corrected: 64

Total number of Words: 471

Accuracy of Model: 0.135881

Custom Language Model:

Words corrected: 91

Total number of Words: 471

Accuracy of Model: 0.193206

**Conclusion:**

The above chart clearly concludes that the laplace Language models with add one smoothing provide better accuracy when compared to unsmoothed models. The accuracy of the models for bi-gram laplace model is better than uni-gram language model. In the custom language model when we in-corporate the features of tri-gram model along with the bi-gram and uni-gram laplace models the accuracy further increases than the uni-gram and bi-gram language models.