

---

# BBM 497: NATURAL LANGUAGE PROCESSING

## LABORATORY ASSIGNMENT 1

---

**Dilşad Ergün**

March 21, 2019

### **ABSTRACT**

I address the problem of predicting a word from previous words in a sample of text. In particular, I discuss n-gram models based on classes of words. I also discuss several statistical algorithms for assigning words to classes based on the frequency of their co-occurrence with other words. I found that I am able to extract classes that have the flavor of either syntactically based groupings or semantically based groupings, depending on the nature of the underlying statistics. The quality of the representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on N-gram models. Furthermore, I tried to show that these representations provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

## **1 Introduction**

Many current NLP systems and techniques treat words as atomic units - there is no notion of similarity between words, as these are represented as indices in a vocabulary. This choice has several good reasons - simplicity, robustness and the observation that simple models trained on huge amounts of data outperform complex systems trained on less data. (1)

In a number of natural language processing tasks, we face the problem of recovering a string of English words after it has been garbled by passage through a noisy channel. To tackle this problem successfully, we must be able to estimate the probability with which any particular string of English words will be presented as input to the noisy channel. In this assignment, I tried to discuss a method for making such estimates. I also discuss the related topic of assigning words to classes according to statistical behavior in a large body of text. In following sections, I look at the subset of n-gram models in which the words are divided into classes. I show that for  $n = 2$  the maximum likelihood assignment of words to classes is equivalent to the assignment for which the average mutual information of adjacent classes is greatest. Finding an optimal assignment of words to classes is computationally hard.

After modeling N-grams truly, I have described in the following. The essay generation process is described in details including the algorithm with its pseudo code that I have used in the assignment. I have mentioned the evaluation of my essays in results and discussions section.

In the following sections, I apply mutual information to two other forms of word clustering. First, I use it to find pairs of words that function together as a single lexical entity. Then, by examining the probability that two words will appear within a reasonable distance of one another, I use it to find classes that have some loose semantic coherence.

In general, I have tried to evaluate results with comparing 3 n-gram models separately and my algorithm in details. I have also mentioned the mistakes and adjustments for them in the implementation stage of my assignment. Lastly I have mentioned my offers which I could not implement clearly and also can make results more correct.

## 2 Related Work

After my researches I have seen that N-grams find use in several areas of computer science, computational linguistics, and applied mathematics. I could not imagine some areas with N-grams so I have viewed them in detail. They have been used to: design kernels that allow machine learning algorithms such as support vector machines to learn from string data, find likely candidates for the correct spelling of a misspelled word, improve compression in compression algorithms where a small area of data requires n-grams of greater length assess the probability of a given word sequence appearing in text of a language of interest in pattern recognition systems, speech recognition, OCR (optical character recognition), Intelligent Character Recognition (ICR), machine translation and similar applications, improve retrieval in information retrieval systems when it is hoped to find similar "documents" (a term for which the conventional meaning is sometimes stretched, depending on the data set) given a single query document and a database of reference documents, improve retrieval performance in genetic sequence analysis as in the BLAST family of programs, identify the language a text is in or the species a small sequence of DNA was taken from, predict letters or words at random in order to create text, as in the dissociated press algorithm, crypt analysis.

## 3 Implementation

### 3.1 Pre-Process Stage

I have started my implementation with the pre-processing stage for my separate train sets. In the following stages of my implementation I saw that the most basic key part was the pre-process stage. I needed to make a lot of changes on that part while developing other stages. I also I saw that any change in the pre-process stage, creates huge effects on the results. So I can say that best processed data creates the best N-gram model. But this stage can differ in different projects. It is a dependent stage as others for N-gram modeling.

On pre-processing stage I had mainly used regular expression structure. A regular expression (regex or regexp for short) is a special text string for describing a search pattern. For this purpose I have used the "re" module of Python language. This module provides regular expression matching operations similar to those found in Perl. I can say that it is really a practical module and really helped my pre-process stage.

Key parts of my pre-process stage

- Reading each train file and appending each of them to other
- Removing certain punctuation characters
- Collapsing multiples of certain chars
- Replacing end punctuation characters with end of sentence symbol which I have defined, </s>
- Replacing middle punctuation characters with empty characters
- Turning all characters in lower
- Removing all extra spaces for avoiding ambiguity
- Replacing negation annotations with separate forms as replacing can't with can not or won't with will not

At the beginning I could not be sure about punctuation characters about removing or not. For my decision, I had firstly make researches about different algorithms. I have read that Google does not use punctuation characters in its algorithms. But before making a certain decision, after implementing the perplexity calculation I have tried the pre-process stage in both ways. As I mentioned, little changes are making big effects. The correction grew a lot so I preferred to remove punctuation characters.

While implementing the test stage I have directly used the pre-process stage that I have implemented for training data. For getting best results they need to be exactly in the same form.

### 3.2 N-Gram Modeling

We needed to introduce clever ways of estimating the probability of a word  $w$  given a history  $h$ , or the probability of an entire word sequence  $W$  for basic natural language processing applications. We have started with the most basic to spelling, with the chain rule. Chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words. (2)

As we followed in our lecture I have started with a basic test applying chain rule to sequences and hold my records for further comparisons including implementation complexity and result correctness. We can't just estimate by counting

the number of 1 times every word occurs following every long string, because language is creative and any particular context might have never occurred before. The intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words.

Firstly, I started my design with creating data structure set of the project. The most basic point is to hold these N-Grams. In my designs, I have considered two different types of data structure: table and dictionary. Since I am more familiar with it, I have chosen the dictionary type. The structure is seen as different for different N-Grams but I have implemented the modeling stage in a single method for three of them. The modeling algorithm that I have implemented is described with the pseudo code below.

```

ngramDictionary ← emptyDictionary
ngramArray ← each word in train data

for word in ngramArray do
    sequence ← leader
    end ← follower

    if sequence not in ngramDictionary then
        add sequence to dictionary, create empty dictionary as sequence's value
    end if

    if end not in ngramDictionary[sequence] then
        assign count value to 0
    end if

    increase count by one
    end for

return final nested dictionary as model

```

The first part of modeling is to collect the splitted words in a list. The dictionary is deviated from the list. Differently in the list, n-grams are repetitive. While deriving the dictionary these repeats return into the counts which are the values of the keys. As it can be understood from the algorithm the structure only holds counts, not the possibilities. This provides the algorithm both space and complexity efficiency. The possibilities are not needed until calculating perplexities.

### 3.2.1 1-Gram

Unigram models are structured as 1 dimensional dictionaries. Dictionary keys consists of unique words which are contained in the whole data set. Unigram models were the most basic form of all models which are implemented by me. After implementation unigram dictionary is shown as in the Figure 1

Figure 1: Unigram Dictionary

### 3.2.2 2-Gram

In bigram representation the inner dictionary structure comes into scene. Now the dictionary is defined as a nested dictionary. In my 2-gram representation, the keys are again the unique words in the train set but this time the values are dictionaries. Sometimes with 1 element, sometimes with 10 elements. The inner dictionary represents the follower words and how many times did they follow, actually bigram counts. For example for the set only contains “to the”, the outer key becomes the “to” and the inner key becomes “the”, since the bigram only occurs 1 time the inner value becomes 1. The visualization of my bigram dictionary is shown in the Figure 2

```
assignment1-dilsadargunn --bash --130x24
'chords': {'and': 1}, 'creature': {'of': 1}, 'habit': {'</s>': 1}, 'strikes': {'his': 1}, 'senses': {'will': 1}, 'sensations': {'o
f': 1}, 'weakened': {'by': 1}, 'concern': {'and': 1}, 'recur': {'to': 2}, 'familiarity': {'and': 1}, 'comprehensiveness': {'of': 1
}, 'circulates': {'through': 1}, 'channls': {'and': 1}, 'currents': {'in': 1}, 'using': {'force': 1}, 'securing': {'a': 1}, 'enjoy
ed': {'by': 1}, 'officers': {'legislative': 1}, 'and': 1, 'who': 1, '</s>': 1, 'being': 1}, 'sanctity': {'of': 1}, 'oath': {'</s>':
1}, 'magistrates': {'of': 1}, 'incorporated': {'into': 1}, 'extends': {'and': 1}, 'enforcement': {'of': 1}, 'w': {'any': 1}, 'ref
lections': {'the': 1}, 'calculate': {'upon': 1}, 'arbitrarily': {'suppose': 1}, 'deduce': {'any': 1}, 'please': {'from': 1}, 'prov
oke': {'and': 1}, 'precipitate': {'the': 1}, 'wildest': {'excesses': 1}, 'encroachment': {'can': 1}, 'promoted': {'by': 1}, 'denie
d': {'</s>': 1}, 'corroborated': {'the': 1}, 'seditions': {'and': 1}, 'unhappily': {'maladies': 1}, 'maladies': {'as': 1}, 'tumors
': {'and': 1}, 'eruptions': {'from': 1}, 'governing': {'at': 1}, 'such': 1}, 'admissible': {'principle': 1}, 'doctors': {'whose': 1
}, 'sagacity': {'disdains': 1}, 'experimental': {'instruction': 1}, 'mischief': {'</s>': 1}, 'slight': {'commotion': 1}, 'commotio
n': {'in': 1}, 'residue': {'would': 1}, 'eventually': {'endangers': 1}, 'endangers': {'all': 1}, 'communicated': {'itself': 1}, 'i
nsurgents': {'and': 1}, 'conductive': {'to': 1}, 'irrational': {'to': 1}, 'pervade': {'a': 1}, 'employment': {'of': 2}, 'repressing
': {'the': 1}, 're': {'establish': 1}, 'recurring': {'to': 1}, 'extremities': {'be': 1}, 'surprising': {'that': 1}, 'applies': {'w
ith': 1}, 'tenfold': {'weight': 1}, 'unceasing': {'agitations': 1}, 'agitations': {'and': 1}, 'scourges': {'of': 1}, 'petty': {'re
publics': 1}, 'lieu': {'of': 1}, 'casualties': {'and': 1}, 'upholding': {'its': 1}, 'parcels': {'of': 1}, 'subdivisions': 1}, 'diff
erently': {'from': 2}, 'peremptory': {'provision': 1}, 'efficacious': {'security': 1}, 'power': 1}, 'betray': {'their': 1}, 'usurpe
rs': {'the': 1}, 'clothed': 1}, 'can': 1, 'stride': 1}, 'consists': {'having': 1}, 'rush': {'tumultuously': 1}, 'tumultuously': {'to
': 1}, 'courage': {'and': 1}, 'despair': {'</s>': 1}, 'embryo': {'</s>': 1}, 'systematic': {'plan': 1}, 'preparations': {'and': 1}
, 'rapidly': {'directed': 1}, 'coincidence': {'of': 1}, 'facilities': {'of': 1}, 'defend': {'them': 1}, 'their': 1}, 'exaggeration'
: {'may': 1}, 'check': {'the': 1}, 'throwing': {'themselves': 1}, 'preponderate': {'</s>': 1}, 'redress': {'</s>': 1}, 'cherishing
': {'the': 1}, 'prized': {'</s>': 1}, 'contingencies': {'afford': 1}, 'projects': {'of': 1}, 'masked': {'under': 1}, 'escape': {'the':
2}, 'penetration': {'of': 1}, 'combine': {'all': 1}, 'communicate': {'with': 1}, 'fresh': {'forces': 1}, 'subdue': {'the': 2}
, 'renewed': {'and': 1}, 'arrive': {'that': 1}, 'erecting': {'a': 1}, 'celerity': {'regularity': 1}, 'regularity': {'and': 1}, 'co
mmanding': {'its': 1}, 'incidents': {'to': 1}, 'watching': {'over': 1}, 'skill': {'in': 1}, 'like': 1}, 'discern': {'that': 1}, 'fi
```

Figure 2: Bigram Dictionary

### 3.2.3 3-Gram

Trigram modeling can be thought as harder than others but the implementation part is so similar to bigram modeling. The only different part is the outer keys. Differently from bigrams now the keys become word sequences instead of words. For example for the set that only contains “to the people” the outer key becomes “to the” and the inner key becomes “people” with the value 1. The visualization of my trigram dictionary is shown in the Figure 3.

```
assignment1-dilsadargunn --bash --130x24
ts': {'who': 1}, 'tyrants who': {'had': 1}, 'had meditated': {'so': 1}, 'meditated so': {'foolish': 1}, 'so foolish': {'as': 1}, '
foolish as': {'well': 1}, 'as so': {'wicked': 1}, 'so wicked': {'a': 1}, 'wicked a': {'project': 1}, 'crush them': {'in': 1}, 'the
ir imagined': {'intrenchments': 1}, 'imagined intrenchments': {'of': 1}, 'intrenchments of': {'power': 1}, 'make them': {'an': 1},
'them an': {'example': 1}, 'just vengeance': {'of': 1}, 'vengeance of': {'an': 1}, 'an abused': {'and': 1}, 'abused and': {'incen
sed': 1}, 'and incensed': {'people': 1}, 'incensed people': {'</s>': 1}, 'this the': {'way': 1}, 'way in': {'which': 1}, 'which u
rsurpers': {'stride': 1}, 'usurpers stride': {'to': 1}, 'stride to': {'dominion': 1}, 'to dominion': {'over': 1}, 'dominion over': {
'a': 1}, 'over a': {'numerous': 1}, 'numerous and': {'enlightened': 1}, 'and enlightened': {'nation': 1}, 'enlightened nation': {
</s>': 1}, 'nation </s>': {'do': 1}, 'do they': {'begin': 1}, 'usually': 1}, 'they begin': {'by': 1}, 'by exciting': {'the': 1}, 'e
xciting the': {'detestation': 1}, 'the detestation': {'of': 1}, 'detestation of': {'the': 1}, 'very instruments': {'of': 1}, 'inst
ruments of': {'their': 1}, 'their intended': {'usurpations': 1}, 'intended usurpations': {'</s>': 1}, 'usurpations </s>': {'do': 1
}, 'they usually': {'commence': 1}, 'usually commence': {'their': 1}, 'commence their': {'career': 1}, 'by wanton': {'and': 1}, 'w
anton and': {'disgustful': 1}, 'and disgustful': {'acts': 1}, 'disgustful acts': {'of': 1}, 'power calculated': {'to': 1}, 'answer
no': {'end': 1}, 'no end': {'but': 1}, 'end but': {'to': 1}, 'to draw': {'upon': 1}, 'draw upon': {'themselves': 1}, 'upon themse
lves': {'universal': 1}, 'themselves universal': {'hatred': 1}, 'universal hatred': {'and': 1}, 'hatred and': {'execration': 1}, '
and execration': {'</s>': 1}, 'execration </s>': {'are': 1}, 'are suppositions': {'of': 1}, 'suppositions of': {'this': 1}, 'sort
the': {'sober': 1}, 'the sober': {'admonitions': 1}, 'sober admonitions': {'of': 1}, 'of discerning': {'patriots': 1}, 'discerning
patriots': {'to': 1}, 'patriots to': {'a': 1}, 'a discerning': {'people': 1}, 'discerning people': {'</s>': 1}, 'or are': {'they'
: 1}, 'are they': {'the': 1}, 'they the': {'inflammatory': 1}, 'the inflammatory': {'ravings': 1}, 'inflammatory ravings': {'of':
1}, 'ravings of': {'incendiaries': 1}, 'of incendiaries': {'or': 1}, 'incendiaries or': {'distempered': 1}, 'or distempered': {'en
thusiasts': 1}, 'distempered enthusiasts': {'</s>': 1}, 'enthusiasts </s>': {'if': 1}, 'rulers actuated': {'by': 1}, 'most ungove
rnable': {'ambition': 1}, 'ungovernable ambition': {'it': 1}, 'ambition it': {'is': 1}, 'would employ': {'such': 1}, 'employ such':
{'preposterous': 1}, 'such preposterous': {'means': 1}, 'preposterous means': {'to': 1}, 'accomplish their': {'designs': 1}, 'des
igns </s>': {'in': 1}, 'or invasion': {'it': 1}, 'invasion it': {'would': 1}, 'be natural': {'and': 1}, 'proper that': {'the': 1},
'neighboring state': {'should': 1}, 'marched into': {'another': 1}, 'into another': {'to': 1}, 'another to': {'resist': 1}, 'comm
```

Figure 3: Trigram Dictionary

## 3.3 Essay Generation

The following operation was to generate 2 essays using each N-Gram model(unigram, bigram, trigram) totally 6 essays for each author. We were expecting to see more meaningful essays from unigrams to trigrams. I have divided the generation process into two parts: Generating the raw text and the post process part to turn the text into essay as style. The evaluation of the essay correctness are discussed in the 4th section. I have described the processes and steps separately of my implementation for essay generation. (3)

### 3.3.1 Generation Algorithm

Generating a text from a N-Gram model is actually equivalent to sampling a distribution. The process actually grow out of randomness. But not a random randomness. Of course the start is a random word or a word sequence chosen by related N-gram dictionary. At first step the sample space is the whole dictionary but in the following steps this sample space becomes narrow excluding unigram dictionaries. Because unigrams are not dependent to each other the sample space remains the same for each step. The pseudocode of my essay generation algorithm is described below.

```

    number of words  $\leftarrow$  0
     $N \leftarrow N - \text{gram}$ 
    model  $\leftarrow N - \text{gram model}$ 
    choose random start from model
    while number of words  $\leq$  30 do
        if sentence ends with end of sentence symbol then
            essay is finished
        else
            sentence += findNextWord(sentence, N)
        end if
    end while

```

The essay's stop criteria is either finding the end of sentence symbol or reaching the 30 words limit. So the sentence can contain both 1 word and 30 words. The text grows with finding the next word. The possibility calculations' key part is finding the next word. I have implemented a weighted version of random.choice by using the probability distribution mechanism. It could either be implemented using numpy's inside weighted choice too. To be able to understand clearly I have chosen this way. The logic of finding next word is described below.

```

    opportunities  $\leftarrow$  last  $N - 1$  word of sentence
    total  $\leftarrow$  sum of counts in opportunities
     $r \leftarrow$  create a uniform random probability distribution
    upto  $\leftarrow$  0
    for choice, weight in opportunities do
        upto  $\leftarrow$  upto + weight
        if upto  $\geq$   $r$  then
            nextWord  $\leftarrow$  choice
        end if
    end for

```

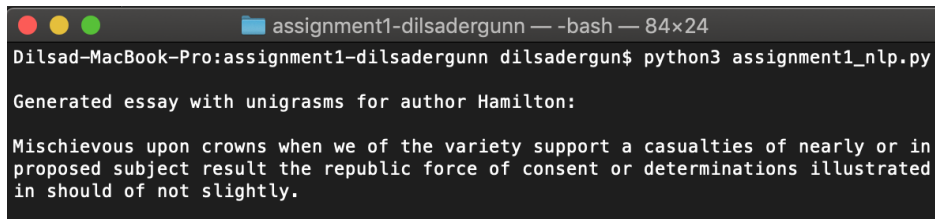
As it can be seen in the pseudocode the opportunities list represents the sample space of the current step. Sample space consists of words which have a possibility to come after the word or word sequence in the given sentence. There is an abstract interval divided into n sub intervals. The following word is chosen from this not random interval randomly. Full randomness only applies to unigram sentence generation.

### 3.3.2 Post-Process Stage

Since the essay only contains 30 words or less the post process stage becomes quite simple. Post process actually turns the raw text into a meaningful style. To look like a sentence the beginning letter of the essay is turned to upper. The text is either stopped with end of sentence symbol or with the word limit. So if the text is stopped with the end of sentence symbol, it should contain the symbol in it. The symbol is replaced with a punctuation mark for a clear look. In the second option there is no symbol but the sentence finished so rather than replacing, the punctuation mark is directly inserted to the text. The essay classification is done before post processing because it should be turned into raw data to calculate probabilities. Before post processing, the data is still raw.

### 3.3.3 Examples of Generated Essays

3 essays derived from author Hamilton's training set is provided with examples. As we can see in Figure 4 which is generated with the 1-gram model is meaningless.



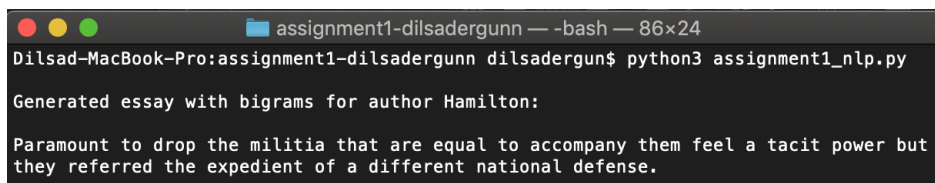
```

assignment1-dilsadergunn — -bash — 84x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py
Generated essay with unigrams for author Hamilton:
Mischievous upon crowns when we of the variety support a casualties of nearly or in
proposed subject result the republic force of consent or determinations illustrated
in should of not slightly.

```

Figure 4: Essay Generated with 1-Gram Model

When we go 2-gram model, we see that we can catch more meaningful sequences as we can see in Figure 5.



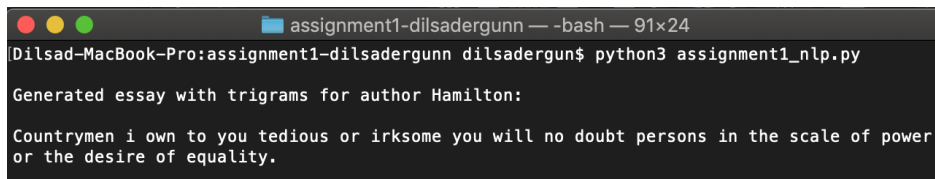
```

assignment1-dilsadergunn — -bash — 86x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py
Generated essay with bigrams for author Hamilton:
Paramount to drop the militia that are equal to accompany them feel a tacit power but
they referred the expedient of a different national defense.

```

Figure 5: Essay Generated with 2-Gram Model

When we come to 3-gram model the essay becomes like a written paper, the meaning success gets higher as seen in Figure 6. We can clearly see the differences between models. As we have waited, we get the best results with 3-gram models on essay generation task.



```

assignment1-dilsadergunn — -bash — 91x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py
Generated essay with trigrams for author Hamilton:
Countrymen i own to you tedious or irksome you will no doubt persons in the scale of power
or the desire of equality.

```

Figure 6: Essay Generated with 3-Gram Model

## 3.4 Probability and Perplexity

The last step is to create an environment to be able to evaluate our modeling and essay generation correctness. For this purpose we need to find “how possible this essay can be written by author Hamilton or author Madison?” or “how possible this whole paper can be written by author Madison or Hamilton?”. To be able to answer these questions we need to calculate these possibilities using self generated models and also the inverse of the probability, perplexity.

### 3.4.1 Smoothing

On each calculation there could be some words or word sequences which are not in my defined models. There are too many words and also too many word sequences related to them. A train set can not handle all of them and also our train set is relatively small. So we needed to handle the zero probability problem. As it is wanted I have implemented Laplace smoothing on my assignment. The equation of Laplace smoothing is defined as below. (4)

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + V} \quad (1)$$

Since we add 1 to each count there is no probability to get a value 0 from our probability calculations. A key point is the V number in the equation. V refers the unique words in the train set. But the V number differs according to the n-gram model that we use. For unigram and bigram models, V refers the number of unigrams which are calculated by us. Differently in trigram, now V refers to the number of bigrams. So, I needed to calculate 2 different V values for each author and these values are used separately according to the calculation. I have applied smoothing directly before returning the probabilities from related method so the method takes V values with the other parameters.

### 3.4.2 Calculating Probability

Each probability is calculated according to current n-gram separately. I have implemented the probability calculation by sending words one by one to the related method. The method applies the related calculation for each word. The smoothing algorithm is applied to the calculated probability before returning the value. (5)

$$P(w_1 w_2 \dots w_n) = P(w_1) P(w_2) \dots P(w_n) \quad (2)$$

The final probability is calculated by applying logarithm applications. Single probability values is returned in logarithm on base 2. In general method these probabilities are summed to get the whole probability. Logarithm is used to avoid underflow instead of multiplying.

$$\log_2 P(w_1 w_2 \dots w_n) = \log_2 P(w_1) + \log_2 P(w_2) + \dots \log_2 P(w_n) \quad (3)$$

### 3.4.3 Calculating Perplexity

The perplexity measure actually arises from the information-theoretic concept of cross-entropy, which explains otherwise mysterious properties of perplexity and its relationship to entropy. The perplexity of a model P on a sequence of words W is now formally defined as the exp of this cross-entropy. I have calculated the perplexity with the equation below. Since I have used  $\log_2 x$  while calculating perplexity logarithm should be destroyed.

$$PP(W) = 2^{-\frac{1}{N} \sum_{i=1}^N \log_2 P(w_1 w_2 \dots w_n)} \quad (4)$$

The perplexities which are found for each test is used to calculate the classification rates. Smaller perplexity means more probability. Perplexity differs both between n-gram models and different test data.

## 3.5 Classification

I have implemented classification operations for both test data, 9.txt-11.txt-12.txt-47.txt-48.txt-58.txt, and also the essays which are generated from all models. Within these I also classified the unknown author essays for classify tests. Classification results give the percentage of probabilities between 2 authors. The percentages differs relatively to models. Classification implementation is actually only the comparisons of found perplexities.

Generated essay classification is a bit different from test data classification. It uses probabilities to classify. But this time it actually checks if the generated essay from an author's train set is more probable to be written by that author. So actually it is a self checking system of the essay generation implementations.

## 4 Results and Evaluation

### 4.1 Classification Results

In this section I have provided some of my program outputs on different test data. Perplexities and author percentages are displayed on the screen for the related input. Test data consists of 6 papers, 3 of them is written by Author Hamilton, 9-11-12 and 3 of them is written by Author Madison, 47-48-58. Regarding to my test, using 2-gram and 3-gram my program correct classification rate is 83,3%, 5 of the essays out of 6 is classified correctly. The perplexity values may seem because the data set is not wide enough to get too high probabilities. By looking to difference rate, the percentages are gained. The whole process is repeated for the unknown author essays too but since we do not know the author we can not check the classification's correctness.

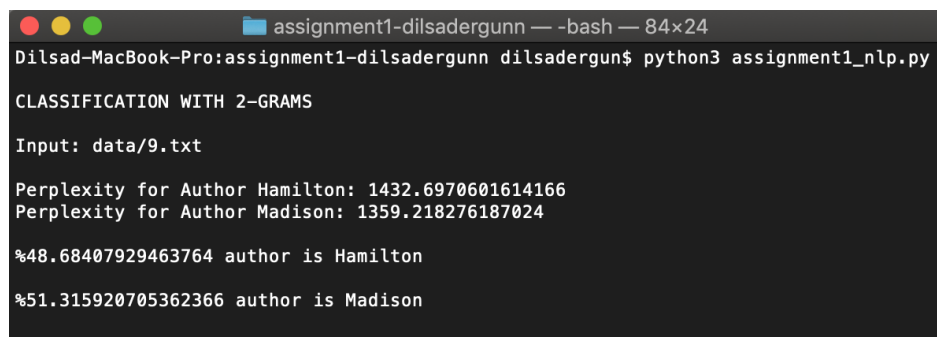
#### 4.1.1 Comparisons and Discussions

By perplexity calculations the reverse logic is applied to calculate percentages. Percentages may seem close but actually the perplexity results gives the real difference between bigrams and trigrams.

Since the train set contains maximum 15 papers for each author, we can not define our set as huge. As we know from computer science logic your accuracy gets higher. Big data set can mean millions of paper. Data set is limited for the assignment so perplexity results differ too much between 2-gram models and 3-gram models. In a full model I was expecting to see better perplexities in 3-gram models, I was expecting less perplexities than 2-grams. But the reverse became true.

As I have mentioned, the probabilities come higher in smaller models because of the data set size. So for this assignment we can define 2-gram model is more successful than 3-gram. Maybe If we had millions of papers we could see more successful 3-gram estimations. On 2-gram probability calculations the V which is used is the distinct words in the train set. On 3-gram probability calculations, this time, V refers to distinct bigrams on train set. This creates difference in probabilities also.

First test file is the 9th paper in the whole papers. We know that paper 9 is written by Author Hamilton. As we can see in the Figure 7 and Figure 8, for both bigram and trigram models the file is unclassified. The only wrong classification is the 9th file. But we can see the that difference is not too wide. It can be considered as tolerance. Although we get similar results in both 2-gram and 3-gram, we can see a wide difference between perplexities. 2-gram perplexities is smaller than 3-gram's. So we can conclude that 2-gram is more successful for this assignment.



```

assignment1-dilsadergunn — -bash — 84x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

CLASSIFICATION WITH 2-GRAMS

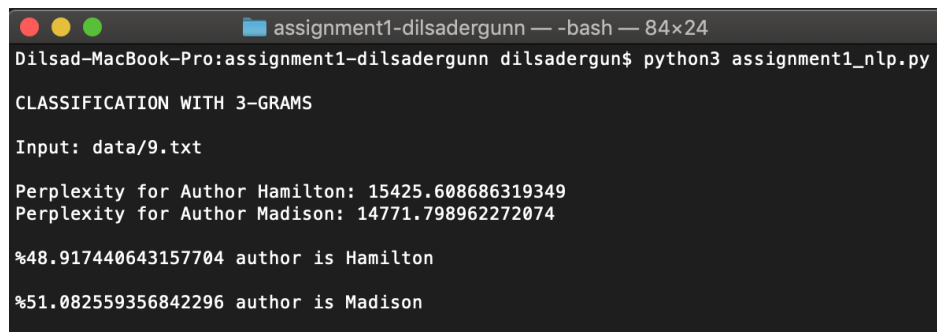
Input: data/9.txt

Perplexity for Author Hamilton: 1432.6970601614166
Perplexity for Author Madison: 1359.218276187024

%48.68407929463764 author is Hamilton
%51.315920705362366 author is Madison

```

Figure 7: File 9 with 2-Gram Model



```

assignment1-dilsadergunn — -bash — 84x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

CLASSIFICATION WITH 3-GRAMS

Input: data/9.txt

Perplexity for Author Hamilton: 15425.608686319349
Perplexity for Author Madison: 14771.798962272074

%48.917440643157704 author is Hamilton
%51.082559356842296 author is Madison

```

Figure 8: File 9 with 3-Gram Model

Second test file is the 11th and third test file was 12th, they are both written by author Hamilton. As we can see in the figures, Figure 9 - Figure 10 - Figure 11 - Figure 12, the classification is completed successfully. The results are Hamilton for both n-gram models. Again we can see the perplexity differences between 2 models and again we see that 2-gram's success over 3-gram for this purpose.



```

assignment1-dilsadergunn — -bash — 84x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

CLASSIFICATION WITH 2-GRAMS

Input: data/11.txt

Perplexity for Author Hamilton: 1556.1642681304863
Perplexity for Author Madison: 1603.294728733833

%50.74586283047386 author is Hamilton
%49.25413716952614 author is Madison

```

Figure 9: File 11 with 2-Gram Model

```

assignment1-dilsadergunn — -bash — 84x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

CLASSIFICATION WITH 3-GRAMS

Input: data/11.txt

Perplexity for Author Hamilton: 15938.680983354498
Perplexity for Author Madison: 15976.76625626873

%50.059665892550846 author is Hamilton
%49.940334107449154 author is Madison

```

Figure 10: File 11 with 3-Gram Model

```

assignment1-dilsadergunn — -bash — 84x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

CLASSIFICATION WITH 2-GRAMS

Input: data/12.txt

Perplexity for Author Hamilton: 1539.592224969356
Perplexity for Author Madison: 1574.919832487889

%50.56714513970087 author is Hamilton
%49.43285486029912 author is Madison

```

Figure 11: File 12 with 2-Gram Model

```

assignment1-dilsadergunn — -bash — 84x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

CLASSIFICATION WITH 3-GRAMS

Input: data/12.txt

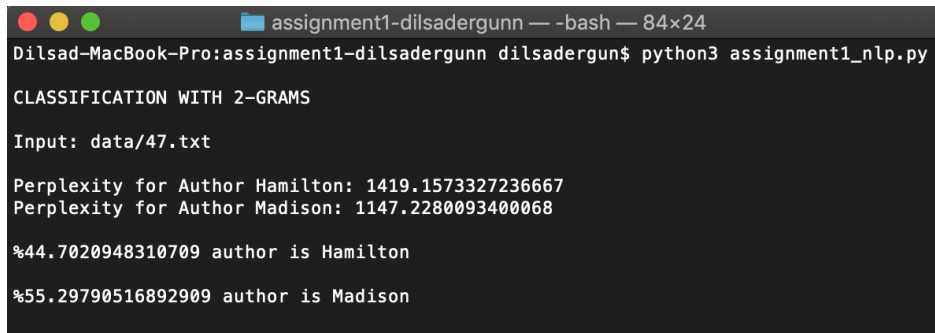
Perplexity for Author Hamilton: 15947.652487487816
Perplexity for Author Madison: 15980.453688957039

%50.051367283245604 author is Hamilton
%49.948632716754396 author is Madison

```

Figure 12: File 12 with 3-Gram Model

The last 3 test files are 47th, 48th and 58th files. All of them are written by author Madison. In these tests we can see more clear results. The gap widens between the wrong one and the true one. Again the perplexities for 2-gram is smaller than 3-gram. So we see again 2-gram is optimal for classification on the assignment.



```

assignment1-dilsadergunn — -bash — 84x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

CLASSIFICATION WITH 2-GRAMS

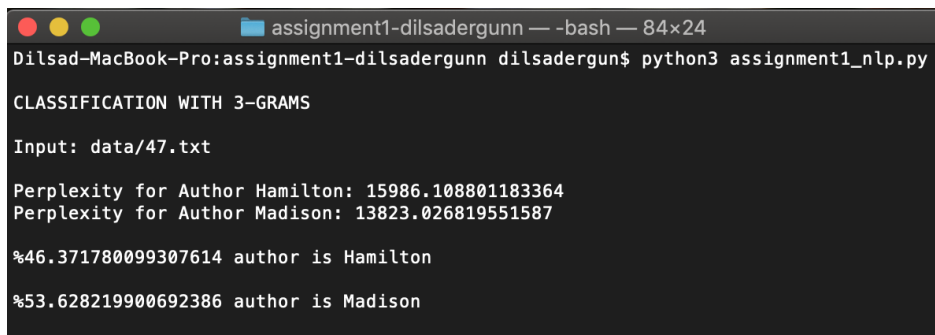
Input: data/47.txt

Perplexity for Author Hamilton: 1419.1573327236667
Perplexity for Author Madison: 1147.2280093400068

%44.7020948310709 author is Hamilton
%55.29790516892909 author is Madison

```

Figure 13: File 47 with 2-Gram Model



```

assignment1-dilsadergunn — -bash — 84x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

CLASSIFICATION WITH 3-GRAMS

Input: data/47.txt

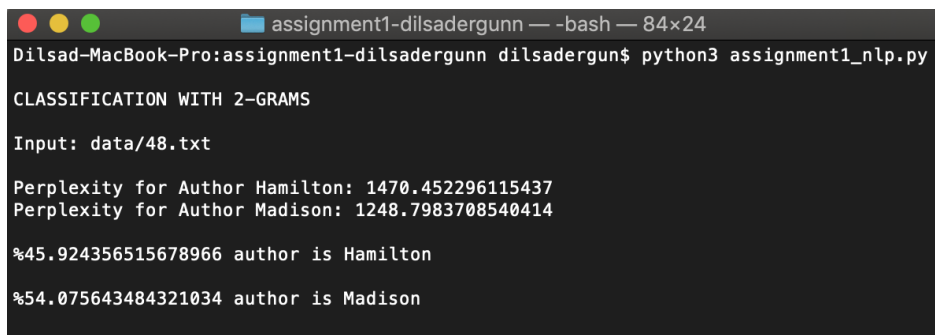
Perplexity for Author Hamilton: 15986.108801183364
Perplexity for Author Madison: 13823.026819551587

%46.371780099307614 author is Hamilton
%53.628219900692386 author is Madison

```

Figure 14: File 47 with 3-Gram Model

All of the tests are classified correctly and the true percentages gets high up to 55% percentages. We can see that the project is more successful on Madison's papers. The difference between wrong and true increases.



```

assignment1-dilsadergunn — -bash — 84x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

CLASSIFICATION WITH 2-GRAMS

Input: data/48.txt

Perplexity for Author Hamilton: 1470.452296115437
Perplexity for Author Madison: 1248.7983708540414

%45.924356515678966 author is Hamilton
%54.075643484321034 author is Madison

```

Figure 15: File 48 with 2-Gram Model

```

assignment1-dilsadergunn — -bash — 84x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

CLASSIFICATION WITH 3-GRAMS

Input: data/48.txt

Perplexity for Author Hamilton: 15865.459602082405
Perplexity for Author Madison: 14455.866387623473

%47.67557458579237 author is Hamilton
%52.324425414207624 author is Madison

```

Figure 16: File 48 with 3-Gram Model

Again the perplexities for 2-gram is smaller than 3-gram. So we see again 2-gram is optimal for classification on the assignment.

```

assignment1-dilsadergunn — -bash — 84x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

CLASSIFICATION WITH 2-GRAMS

Input: data/58.txt

Perplexity for Author Hamilton: 1393.204274855527
Perplexity for Author Madison: 1157.7452279551114

%45.3848744038056 author is Hamilton
%54.61512559619441 author is Madison

```

Figure 17: File 58 with 2-Gram Model

```

assignment1-dilsadergunn — -bash — 84x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

CLASSIFICATION WITH 3-GRAMS

Input: data/58.txt

Perplexity for Author Hamilton: 15753.316148944987
Perplexity for Author Madison: 13968.464398665508

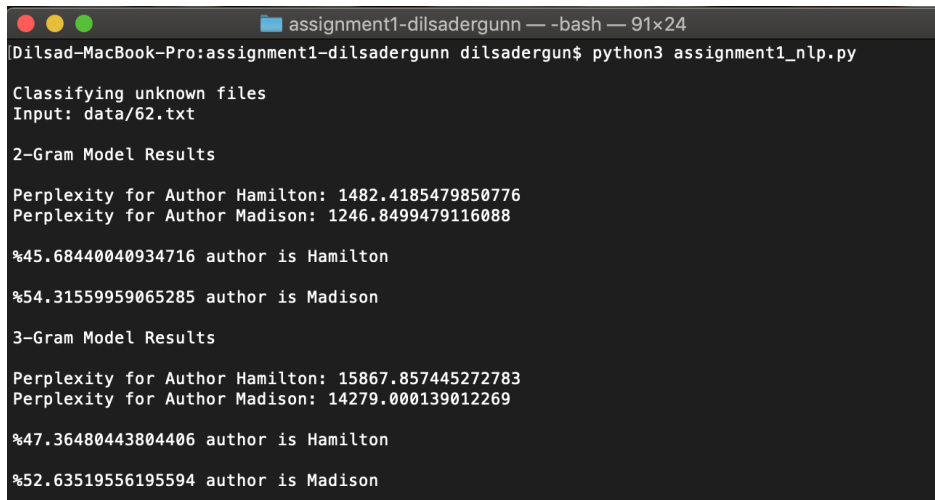
%46.99740103487344 author is Hamilton
%53.00259896512656 author is Madison

```

Figure 18: File 58 with 3-Gram Model

As we can conclude from the test set, the n-gram model which I have created can be defined as successful according to results. From my observations, I think that the size of the data set creates an environment for 2-grams to win the success race. Both 3-grams and 2-grams did not fail on authorship detection. But when we compare the perplexities, we can see that 2-gram models are more useful for this purpose.

For the unknown papers I used the same implementation directly. On Internet I have read about that the unknown titled papers were commonly written by author Madison. I have provided an example of a paper results with an unknown author in Figure 19. The results are similar to the results of papers written by Author Madison.



```

assignment1-dilsadergunn — -bash — 91x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

Classifying unknown files
Input: data/62.txt

2-Gram Model Results

Perplexity for Author Hamilton: 1482.4185479850776
Perplexity for Author Madison: 1246.8499479116088

%45.68440040934716 author is Hamilton
%54.31559959065285 author is Madison

3-Gram Model Results

Perplexity for Author Hamilton: 15867.857445272783
Perplexity for Author Madison: 14279.000139012269

%47.36480443804406 author is Hamilton
%52.63519556195594 author is Madison

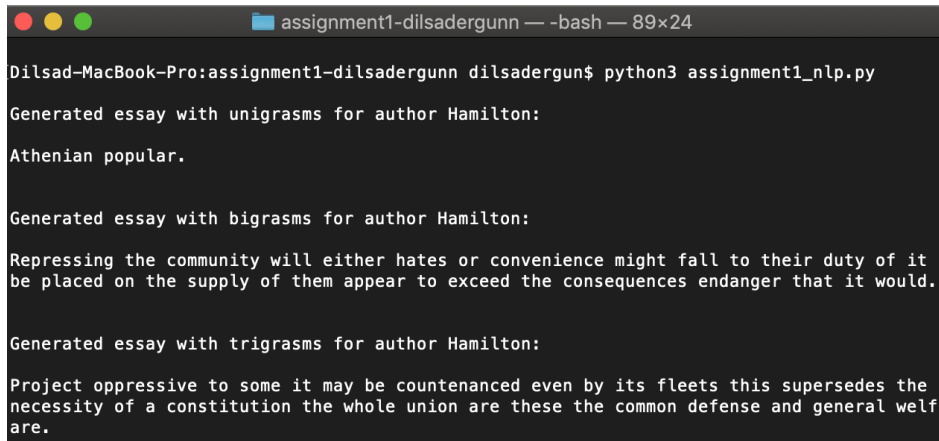
```

Figure 19: Unknown Author Classification

## 4.2 Essay Generation Results and Discussions

### 4.2.1 Essay Generation Results

Multiple essays from different authors are generated in test stages with the implementation described in essay generation algorithm. In Figure 20, essays derived from all models is shown. As it is expected, essays gets more meaning while walking through the 3-gram model.



```

assignment1-dilsadergunn — -bash — 89x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

Generated essay with unigrams for author Hamilton:

Athenian popular.

Generated essay with bigrams for author Hamilton:

Repressing the community will either hates or convenience might fall to their duty of it
be placed on the supply of them appear to exceed the consequences endanger that it would.

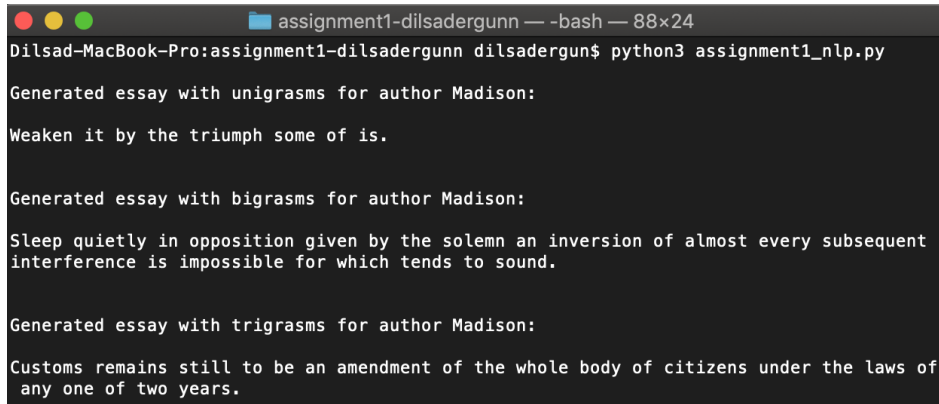
Generated essay with trigrams for author Hamilton:

Project oppressive to some it may be countenanced even by its fleets this supersedes the
necessity of a constitution the whole union are these the common defense and general welf
are.

```

Figure 20: Generated Essays From Hamilton

Also in Figure 21, the essays derived from the author Madison's data set is shown. We can see similar results with a different vocabulary.



```

assignment1-dilsadergunn — -bash — 88x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

Generated essay with unigrams for author Madison:

Weaken it by the triumph some of is.

Generated essay with bigrams for author Madison:

Sleep quietly in opposition given by the solemn an inversion of almost every subsequent
interference is impossible for which tends to sound.

Generated essay with trigrams for author Madison:

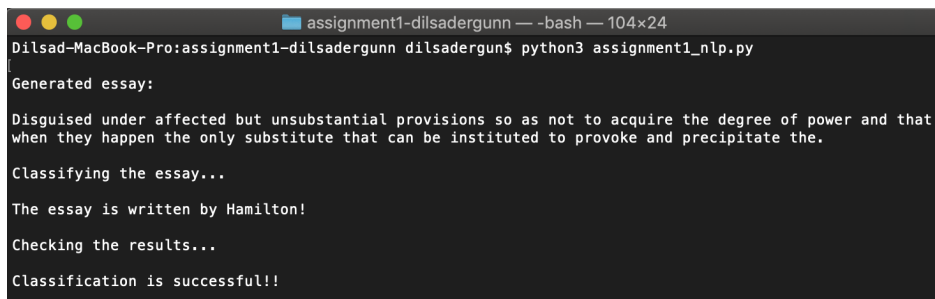
Customs remains still to be an amendment of the whole body of citizens under the laws of
any one of two years.

```

Figure 21: Generated Essays From Madison

#### 4.2.2 Comparisons and Discussions

After the essay generation process, I have applied some tests to be able to evaluate generated essays clearly. First test that I have done was the authorship detection as I have done at the previous stages. After a long test stage all essays generated by an author by using 2-gram and 3-gram models, classifications reached to success with high percentages. As we can guess, the 1-gram derived essays do not give stable results for classification. An example is shown in the Figure 22.



```

assignment1-dilsadergunn — -bash — 104x24
Dilsad-MacBook-Pro:assignment1-dilsadergunn dilsadergun$ python3 assignment1_nlp.py

Generated essay:

Disguised under affected but unsubstantial provisions so as not to acquire the degree of power and that
when they happen the only substitute that can be instituted to provoke and precipitate the.

Classifying the essay...

The essay is written by Hamilton!

Checking the results...

Classification is successful!!

```

Figure 22: Essay Classification

The second tests' aims that were applied to generated essays, was to make comparisons between n-gram models. For generated essays from different models are tested with probability calculations. Each essay's probability is calculated for each n-gram model separately. I have divided test process into 3 sub-processes and divided sub-processes into 3 sub-sub-processes again. So at last I have taken 9 possible probabilities. These results are gained with multiple tests applied on randomly generated essays.

Since only the comparisons are enough for evaluation at this stage I have continued to use  $\log_2$  probability form. So in tables the probabilities is formed in negative forms. The table is used for the comparisons. We check the highest probability for the best model on the current essay.

The results came so similar so I have generalized the results on evaluating stage. Table 1 shows the results evaluated with all n-gram models comparatively. The biggest probabilities are shown in bold text format in table.

Table 1: Probabilities for Generated Essays

| Generated Essay | 1-Gram Model | 2-Gram Model | 3-Gram Model |
|-----------------|--------------|--------------|--------------|
| 1-Gram Essay    | <b>-182</b>  | -185         | -217         |
| 2-Gram Essay    | -248         | <b>-194</b>  | -271         |
| 3-Gram Essay    | -149         | <b>-112</b>  | -128         |

As we can see in the comparisons, generated essays from 1-gram models give the highest probability as expected. Since 1-gram generates a totally random essay, the words gave the highest probability with independent probabilities. When we come to essay evaluation with the 2-gram generated essays the 1-gram probability becomes much less. And the difference between 2-gram and 1-gram probabilities widens too much. The highest probability comes in 2-gram model with a huge difference for the 2-gram generated essay. In first 2 cases, as expected, 3-gram is too far to win.

Lastly we evaluate the essays which are generated with 3-gram models. These essays as we see in the previous sections are the most meaningful essays comparing to others. So we can infer that it is like a real written paper by a real author. We have taken 2-gram model winning results in real papers so I expected to see similar results for 3-gram generated essays. I get the same results as I have expected. 2-grams won the race over 3-grams because of reasons like data set sizes. But we see that the 3-gram probability gets higher in these tests. Since the essays are more meaningful, it is an expected result. We can define the tests are successful according to results.

## 5 Limitations and Future Work

After the evaluation of several results, I have taken a success rate 83,3%. Within this success I believe that this rate can be increased with different approximations. Actually n-grams are the most basic form of language models. Comparing to simplicity they are so successful but there are also disadvantages of n-grams too. One of the disadvantage that I realized is long range dependencies are not captured. Actually we forget about the history while applying evaluation with n-gram models. This creates some tolerance. For example, in this assignment we only kept the history of 2 words before which is not enough for more sensitive applications.

Another limitation that I can comment is n-gram models are dependent on having a corpus of data to train from it. 10 papers are not enough to analyze a structure. There can be too many different approximations in the corpus. Sparse data for low frequency affect tags adversely affects the quality of the n-gram model. For example in this assignment because of the data we have too high V values which effect the probabilities directly in a negative way. We can not calculate probabilities because of zero problem and we get too low probabilities as a result. In the following assignments, we will create more successful models with different approximations.

## 6 Conclusion

As a result I have completed the assignment with a success rate 83,3%. With these experiments I firstly had a chance to understand language modeling structures with the first approximation of language models. Creating a model from scratch helped me to understand the back stage of the applications which I use commonly in daily life. As we see theoretically in our lecture, in practice n-grams are much different from the one that I understand.

Another point in the assignment was the comparisons. With a real structure, I could clearly see the differences between different n-gram models. At first, I was expecting to see 3-gram models as a winner in all of the evaluations. But when I got the results, except the essay generation task in the meaning look, I saw that I got better results in 2-gram models. When I discussed the results I realized that the train data size also effects the language model evaluations. I gained a different view to the problem.

At the end, I could understand the n-gram structure clearly in all purposes. After my experiments, now I am able to develop applications based on n-grams. I believe that n-grams can be useful for not too complex structure with a basic implementation. And also the most basic advantage that I can conclude is the speed of creating and using n-grams.

## References

- [1] S. Uni., “Natural language processing lecture.” [Online]. Available: <https://web.stanford.edu/class/cs124/lec/>
- [2] E. K. Steven Bird and E. Loper, “N-gram models,” in *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O’Reilly Media, July 10, 2009, pp. 220–250.
- [3] G. Seif, “An easy introduction to natural language processing.” [Online]. Available: <https://towardsdatascience.com/an-easy-introduction-to-natural-language-processing-b1e2801291c1>
- [4] V. Saurus, “Laplace smoothing.” [Online]. Available: <https://vsoch.github.io/2013/laplace-smoothing/>
- [5] E. K. Steven Bird and E. Loper, “Probability space,” in *Taming Text*. Manning, April 29, 2013, pp. 120–130.