Text Prediction Using Model of N-Grams

Neelkumar S Bhuva
Yashwanth kumar Pamidimukkala

Acknowledgement

We would like to acknowledge Professor Jason Van Hulse for giving us an opportunity to work on this project and Michael Burkhardt for encouraging us and for evaluating and providing feedback on the report. We would also like to thank Dhrubojyoti Roy for letting us use his computer for testing our code on huge dataset.

1. Introduction

In few years mobile phone can be your only computer which is able to do most of the work that laptops and CPUs can do. But lacking a full-sized keyboard, text entry on touchscreen devices can be quite cumbersome. Automated text entry will help us to solve this problem by predicting the next word as we type. Understanding its importance, we decided to perform this task by building a model which will predict/produce a word given the previous word/words. The simplest model that we can implement in the field of computational linguistics is the model of n-gram.

A language model defines a probability distribution over sequences of tokens in a natural language. Depending on how the model is designed, a token may be a word, a character, or even a byte. Tokens are always discrete entities. The earliest successful language models were based on models of fixed-length sequences of tokens called n-grams. An n-gram is a sequence of n tokens.

Models based on n-grams define the conditional probability of the n-th token given the preceding n − 1 tokens. The model uses products of these conditional distributions to define the probability distribution over longer sequences:

$$P(x_1,\ldots,x_\tau) = P(x_1,\ldots,x_{n-1}) \prod_{t=n}^{\tau} P(x_t \mid x_{t-n+1},\ldots,x_{t-1}).$$

This decomposition is justified by the chain rule of probability. However, in n-gram, we truncate the history to length n - 1 and it is so-named because it characterizes a sequence of n variables.

$P(x_i \mid x_1,\ldots,x_{i-1}) = P(x_i \mid x_{i-n+1},\ldots,x_{i-1}).$

- Unigram $P(x_i)$ $(i.i.d\ process)$
- Bigram $P(x_i|x_{i-1})$ $(Markov\ chain\ model)$
- Trigram $P(x_i|x_{i-2},x_{i-1})$

Markov Assumption: Probability is approximated by last few words. For Example: In a bigram model: P(sports **|** I like to play) can be approximated as P(sports **|** the). For a Four gram model it can be approximated as P(sports | I like to). This is the assumption we make on chain rule of probability and approximate the probability calculations to just predicting the nth given n-1th word.

2. Approach

We implemented the model in Python by using customized functions where the important functions include tokenize which breaks the documents into tokens, cleaning the text, inserting tokens to identify a sentence, counting the n grams, calculating probabilities, performing Laplace smoothing, performing evaluation metrics.

3. Data Ingestion and Preprocessing

Table 1: Table showing some examples of transformation i.e. after cleaning the dataset.

Input	**Input after cleaning**
Mr. Brown	Mr. Brown
cashed in his $$$	cashed in his money
sister made, but it seemed	sister made but it seemed
Nov 9 Dec 6	Nov <D> Dec <D>
(p.1-140)	Space (text inside parentheses is replaced by spaces/ignored)
let's hear it! :)	let's hear it
3. Winter is the time	Winter is the time (3 here indicates point number 3 which is irrelevant to us)
end to years of waiting.	end to years of waiting </s> (</s> is the end of sentence token)
Firstly, I finally	<s> <s> Firstly I finally (for trigram model)
Firstly, I finally	<s> <s> <s> Firstly I finally (for four-gram model)
“gods”	Gods
Obama/congress 3/4	Obama or congress 3 or 4

We chose a Dataset which was available online which is used widely for language sequence modelling and downloaded it into our PC. We cleaned the textual content for any unnecessary symbols or textual content like smileys, exclamation marks and many other things which are included in the Python code.

Sample text data from corpus**:**
In the years thereafter, most of the Oil fields and platforms were named after pagan "gods".
We love you Mr. Brown.
Chad has been awesome with the kids and holding down the fort while I work later than usual! The kids have been busy together playing Sky lander on the Xbox together, after Kyan cashed in his $$$ from his piggy bank. He wanted that game so bad and used his gift card from his birthday he has been saving and the money to get it (he never taps into that thing either, that is how we know he wanted it so bad).

The input file contains data like smileys [:),:P,:( ], question marks, currency symbols, two character words like st, Mr. , Dr. , sq., which need to be transformed in order to learn better from the training data. This is achieved using regular expressions in python to match a particular format of text and replace it.

We insert start token/s '<s>' at the beginning of a sentence in order to calculate the conditional probabilities at the beginning of a sentence. For example: suppose our sentence is **"I like playing Volleyball".**

The conditional probabilities for trigram model:

- P (playing | I like) = Count(I like playing) / Count(I like)
- P (Volleyball | like playing) = Count(like playing Volleyball)/Count(like playing)

But, what about the words 'like' and 'I'? To calculate conditional probabilities (to get trigram context) for "like" and "I" we need two words before "like" and before "I". So, we insert start tokens <s>. We also insert end token </s> at the end of each sentence.
Our sentence is now <s> <s> I like playing Volleyball </s>

Test set: 10 percent of the dataset is held out for testing.

Output Data Format: Output of part A is the cleaned input text with start and end tokens inserted and a similar test set which is 10 percent of original dataset.

We first used matrix to store the n-gram probabilities. But, matrices are extremely slow and the code does not finish execution even on a machine with 8GB RAM intel core i5. We re-implemented our code using hash data structures. This significantly increases the speed but still does not complete execution for large training data. Fortunately, we were able to get access to a intel core i7 machine with 64GB RAM for few hours and the code executed just fine.

4. Exploratory Data Analysis

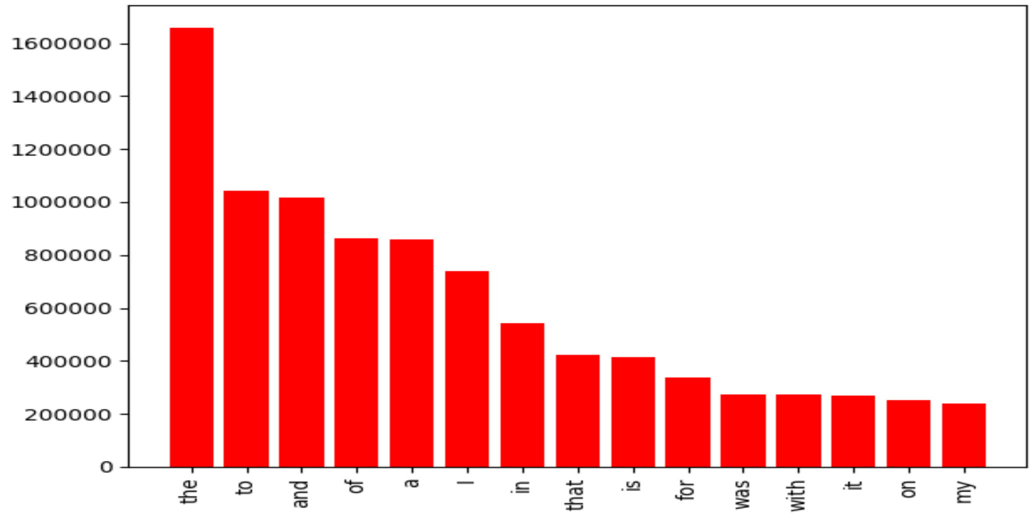
Input Data**:** Three text files from Corpus of Contemporary American English (COCA)[1]**.**

Table 2: Summary of the text documents which were used for the model

Dataset	**Lines**	**Words**	**Unique Words**
Blogs	899,288	37,334,117	362,331
News	1,010,242	34,365,936	260,258
Twitter	2,360,148	30,373,559	384,823

Unigram Counts:

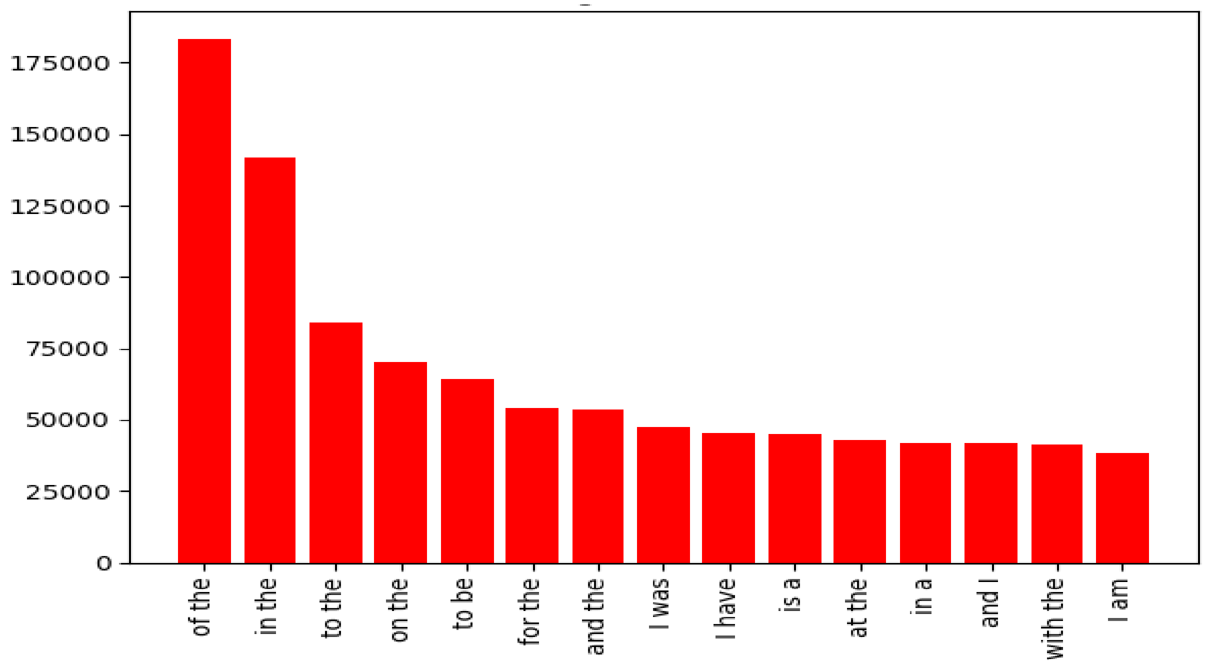
Figure 1: Plot showing frequency of unigram counts



From the unigram plots, what we noticed is the high count of words like *to, the and, of* which in text mining are considered as stop words. These words in text mining are removed from the model, but the same cannot be done in text prediction as we need every word to predict the next word. Our project is not an endpoint for developing a text prediction model, developing a model while carefully choosing the stop words involves having a deeper understanding of language modelling and language semantics. This can be one of the improvement for our project. For our model, we did not filter these words in preprocessing and included them as they help us estimate initial probabilities for word prediction.

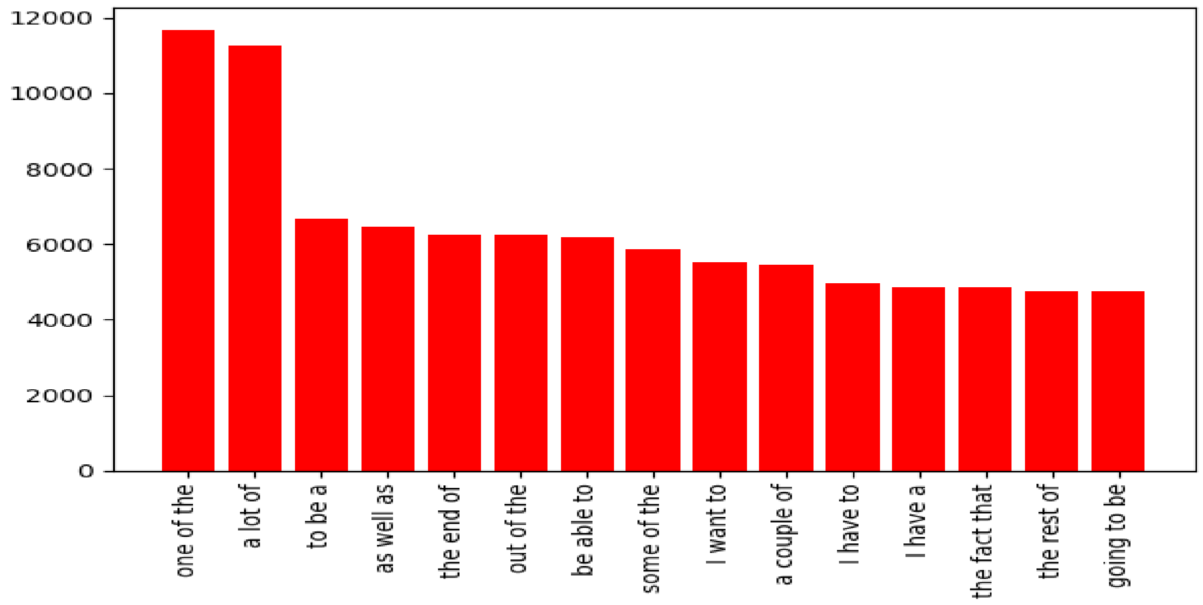
Bigram Counts:

Figure 2: Plot showing the frequency of most popular bigrams



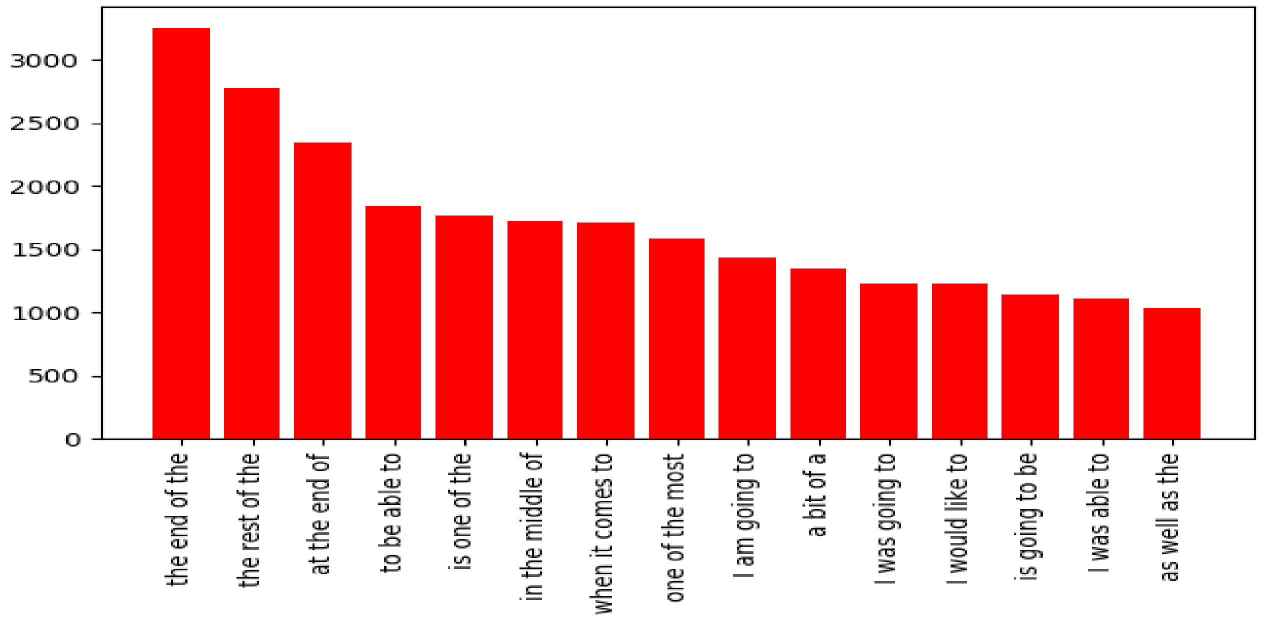
Trigram Counts:

Figure 3: Plot showing the frequency of most popular trigrams



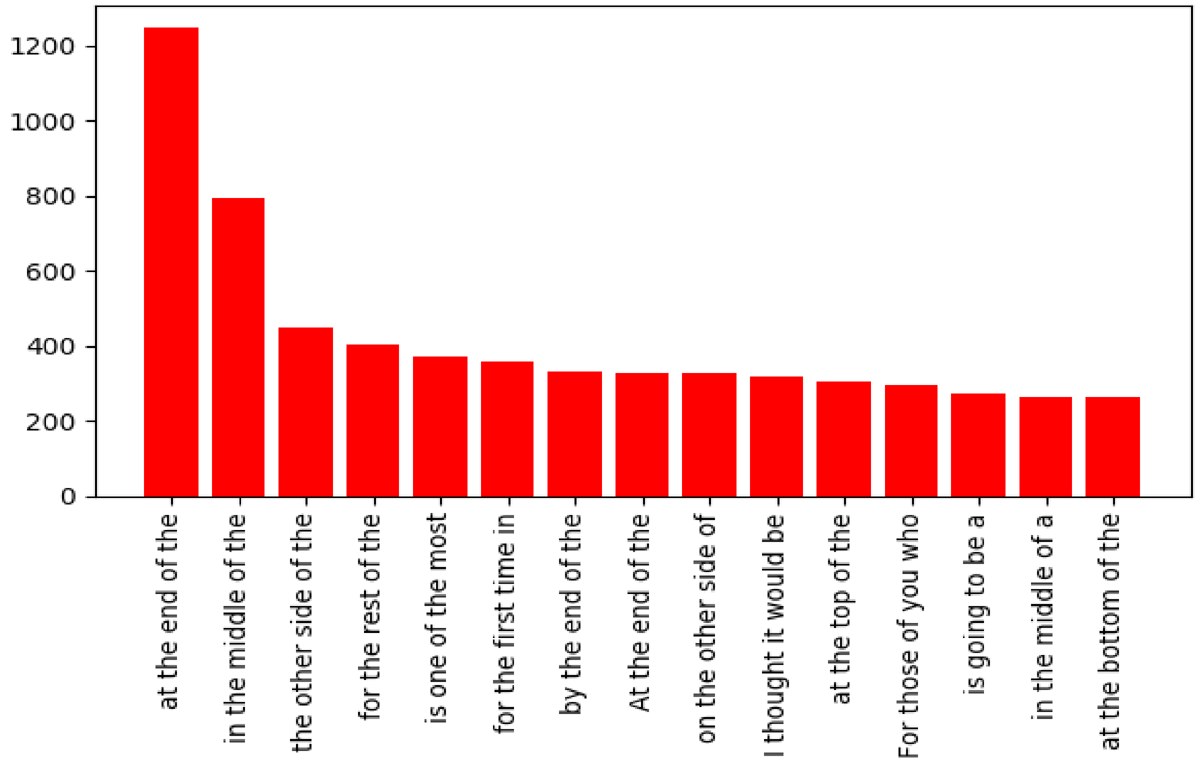
Tetragram/ Four gram counts:

Figure 4: Plot showing the frequency of most popular tetragrams



Pentagram Counts:

Figure 5: Plot showing the frequency of most popular pentagrams



From all the plots above, we notice that sentences become more meaningful and grammatically correct as we increase the size of the n grams. For bi-grams we will mostly end up with the stop words, but as we increase n to 5 or 6 we might have better prediction. However, to make full advantage of text prediction our model should perform well even for a bigram i.e. if you type in a word it should suggest words appropriately.

5. Design Choices

Testing Approach: 10 percent of data is held out for testing. This is the most common approach for testing a language model. Cross validation for testing is. **Unknown Words/ Out of Vocabulary Words (OOV):** First let's look at two types of vocabulary:

Closed Vocabulary: Fixed set of grams are in vocabulary. OOV words in training data can be replaced by token <UNK> and then can be treated as any other word.

Open Vocabulary: Entire training data is the Vocabulary.

6. Model

a. Implementing N-gram:

Once our cleaning is done, we will model the data by dividing them into N-grams

Input Data: Cleaned text.

Step 1: Tokenize - get the words in the order they appear in the input and store them as a list. We call this list as wordlist.

Step 2: Get the frequency of all unigrams i.e count the number of occurrences of each word in the input.

Step 3: Get all bigrams from the wordlist and count the number of occurrences of each bigram.

Step 4: Get all trigrams from the wordlist and count the number of occurrences of each trigram.

Step 5: Repeat Step 3 to get all four grams and five gram counts.

Step 6: Calculate the maximum likelihood estimation (MLE is estimation of n-gram conditional probabilities). MLE is nothing but normalized counts of n-grams.

For Example: Bigram probability P(of | world) = C(world of) / C(of)

In general: $P(w_n \mid w_1\, w_2 \ldots w_{n-1}) = C(w_1 w_2 \ldots\ldots w_n) \,/\, C(w_1 w_2 \ldots\ldots w_{n-1})$

Step 7: Calculating probability of a sentence $w_1, w_2, \ldots\ldots w_n$. Using chain rule,

$P(w_1 w_2 \ldots w_n) = P(w_1)\, P(w_2|w_1)\, P(w_3|w_2) \ldots. P(w_n|w_{n-1})$for bigram model.

It is possible to encounter words that appear in test set in a context that was unseen in train set. If one of the probabilities in the chain rule is 0 then the entire expression results in zero. So, even if the count of an n-gram is zero we need to use some technique to assign nonzero probabilities to the unseen context. We use Laplace Smoothing to achieve this.

b. Laplace Smoothing

Smoothing/Discounting is a technique to take a bit of probability mass from frequent words and assign to unseen words.
In this technique we add one to count of n-gram and normalize the counts as follows [2] :
For Unigram probabilities:

$$P(Wi) = \frac{C_i}{N}$$

$$P_{Laplace}(W_i) = \frac{C_i + 1}{N + V}$$

Where N is the total count of words in wordlist and V is number of unique counts in wordlist.
For Bigram probabilities:

$$P_{Laplace}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}\, w_i)+1}{C(w_{i-1}) + v}$$

For Example**:**
We extracted a part of our training data. In this part, Bigram "build network" does not appear. But "build" appears 612 times.
Earlier: C (build network) = 0 which implies P (network | build) = 0.
Laplace: P (network | build) = 0 + 1 / (612 + V) = 6.4 e-06 where V = 155169 is number of words in vocabulary.

Now, we calculate Perplexity of the test set. Perplexity is explained in Language Model Evaluation section.

7. Language Model Evaluation

Perplexity is used as a metric to evaluate a Language model. Perplexity of a test set using an n-gram language model,

$$PP(W) = P(w_1 w_2 \ldots\ldots\ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots\ldots w_N)}}$$

Where N is the total number of words in the test set. Using chain rule,

$$PP(w) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots\ldots w_{i-1})}}$$

Higher probability of the test set using the learnt model indicates a good language model. From the above formula minimizing perplexity is equivalent to maximizing test probability. So, we want models with lower perplexity. Since probabilities are between 0 and 1, product of all the n-gram probabilities becomes very small resulting in numeric underflow. We learnt this the hard way. So, we use log(probability) instead.

$$Entropy: H = -1/ \frac{1}{N\,(log(P(w_1\, w_2 \ldots w_N))}$$

$$PP(w) = 2^H$$

8. Experiment Results

Perplexity PP(W)**:** PP(W) is calculated on held out data set
Table 3: Perplexity on blogs test set using different models

Model	**Blogs Dataset**	**News Dataset**	**Twitter Dataset**
Bigram	721.1	805.78	647.14
Trigram	173.849	166.58	266.96
Four gram	38.5	49.7	181.46
Five gram	5.76	8.67	47.87

Lower the perplexity, better the language model as we saw in section 7. From Table 3, in all three datasets we observe that higher the gram model better is the test probability. Twitter dataset has relatively high perplexity. One of the reason is it requires additional data cleaning. We did not work on this due to lack of time.

In this part, we try to predict the top 3 next words in a sentence using model learnt from blogs dataset. Probability of each word is also observed. We assume that sentences do not have OOV words. Goal is to predict top 3 meaningful probable words.

Sentence 1: “He has been living in the world of”
Predicted completion of the sentence:

Table 4: Bigram Model sentence prediction with probabilities

He has been living in the world of the	0.145
He has been living in the world of a	0.026

Because there are high number of stop words in bigrams, the predicted words here are not what we would expect it to be.

Table 4: Trigram Model sentence prediction with probabilities

He has been living in the world of the	0.00033
He has been living in the world of warcraft	0.00012
He has been living in the world of publishing	0.00006

Table 5: Five-gram Model sentence prediction with probabilities

He has been living in the world of the	3.62 e-07
He has been living in the world of fashion	2.81 e-07
He has been living in the world of publishing	2.40 e-07

Trigram and five-gram model predicts words other than stop words unless bigram.

Sentence 2: "he is one of the"
Completions from Google search engine: “best”, “most”, “kind”

Table 6: Bigram Model sentence prediction with probabilities

He is one of the first	0.0095
He is one of the same	0.0092
He is one of the world	0.0064

The words “same” and “world” are less likely to follow here, than say the word “most”, “best”, so we’d like our bigram model to prefer the latter. But in fact, the former is selected because “the same” and “the world” are more frequent in the training set.

Table 7: Trigram Model sentence prediction with probabilities

He is one of the most	0.0044
He is one of the world	0.0037
He is one of the day	0.0035

Similar to bigram model “world” and “day” are preferred because “of the world” and “of the day” are more frequent.

Table 8: Four-gram Model sentence prediction with probabilities

He is one of the most	9.9 e-05
He is one of the best	4.6 e-05
He is one of the first	2.3 e-05

Table 9: Five-gram Model sentence prediction with probabilities

He is one of the most	1.37 e-05
He is one of the best	6.04 e-06
He is one of the few	2.94 e-06

Both four and five gram models predict words that are expected to follow the sentence and hence are better than bigram and trigram.

Sentence 3: "whether we will do the"

Table 10: Bigram Model sentence prediction with probabilities

whether we will do the first	0.0095
whether we will do the same	0.0092
whether we will do the world	0.0064

Table 11: Trigram Model sentence prediction with probabilities

whether we will do the same	0.0012
whether we will do the right	0.0003
whether we will do the job	0.0002

Sentence 4: "i did not come to condemn the world but to save the world"
Completions from Google search engine: “from”, “or”.

Table 13: Bigram Model sentence prediction with probabilities

But to save the world </s>	0.0128
But to save the world of	0.0040
But to save the world and	0.0034

Table 14: Trigram Model sentence prediction with probabilities

But to save the world </s>	0.0075

But to save the world of	0.0020
But to save the world and	0.0019

The word “from” is more likely to occur in this context compared to “of”, but is not selected by bigram or trigram model.

Table 15: Four-gram Model sentence prediction with probabilities

But to save the world </s>	1.48 e-06
But to save the world from	5.95 e-07
But to save the world and	5.95 e-07

Table 16: Five-gram Model sentence prediction with probabilities

But to save the world </s>	4.43 e-07
But to save the world from	3.22 e-07
But to save the world and	2.01 e-07

Both “from” and “and” are expected to follow the sentence. Also, note how end of sentence has highest probability (it is likely that the above sentence is complete) which is exactly why end of sentence tokens are used. In all the cases end of sentence is predicted with highest probability which is likely.

Sentence 5: "i embrace them and try to keep them happy and"

Table 17: Bigram Model sentence prediction with probabilities

Try to keep them happy and the	0.038
Try to keep them happy and i	0.032
Try to keep them happy and a	0.015

Table 18: Trigram Model sentence prediction with probabilities

Try to keep them happy and healthy	8.54 e-05
Try to keep them happy and i	7.71 e-05

Try to keep them happy and sad	5.78 e-05

Table 19: Four-gram Model sentence prediction with probabilities

Try to keep them happy and healthy	1.78 e-07
Try to keep them happy and i	7.71 e-05
Try to keep them happy and sad	5.78 e-05

Table 8.18: Five-gram Model sentence prediction with probabilities

Try to keep them happy and healthy	1.20 e-07
Try to keep them happy and busy	8.05 e-08

All models except bigram predict the word “healthy” with highest probability. But, trigram and four-gram also predict the word “sad” which is contradictory in this context. Five-gram model emits only two possible words (both are expected).

Observation**:** From the above tables and from perplexity, bigrams tend to predict stop words and perform poorly. Trigram model works well on occasions. Four and five gram models outperforms other models.

9. Challenges:

One of the main challenge we faced with the project was handling huge data. It was difficult for us to perform evaluation such as Perplexity on smaller data sets by sampling from the larger data set as the results would not apply to the whole data. Improving our knowledge in Python’s Data Structures and knowledge in handling Big Data would have saved a lot of time to implement this project.

Few other challenges within our model were, we couldn’t limit our probability calculations to just MLE method. Doing so, our model will produce 0 probability for words that do not occur in training set or if we had any out of dictionary words. To avoid this, we performed Laplace smoothing which was discussed above.

10. Improvements

There is a lot of room for improvement in our project, important one would be to implement Bigdata implementation to perform tasks faster and use better Data Structures. Next would be to perform better smoothing techniques such as Kneser-ney which are far better than Laplace in smoothing our Probability mass function.

All the modelling that we have done can be used to build a User Interface or an application where we user enters a text and gets a list of probable words that appear in sequence. If we had more time in our project, this was our next step and it can be achieved by creating databases of the N-grams which helps in faster prediction.

11. Individual Contributions

Data Acquisition and Cleaning: Neel
Data Preprocessing: Yashwanth
Modelling, Prediction and Testing: Together

12. References

[1] Davies, Mark. (2011) N-grams data from the Corpus of Contemporary American English (COCA).
https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip
[2] https://lagunita.stanford.edu/c4x/Engineering/CS-224N/asset/slp4.pdf
[3] www.google.com
[4] www.wikipedia.org