Next Word Prediction

David Majercak, Grant Waldow, Colleen Klein, Ahmed Alghafri

April 2021

1 Problem Space

Our objective with this project is to create a next word prediction that relies on the word, or words, inputted previously. The user would input a single word, and our generator would continue adding words that would make logical sense based on the word given and the data it is trained on. This data could include vocabularies from various sources, including tweets, news articles, and blogs. Depending on the data used, some language could be used differently in each context. For example, a news article probably uses more formal language than a tweet. This also could be expanded to predict the next words from a specific person by using data from a specific person's Twitter account that could predict their next tweets.This also could be expanded to predict the next words from a specific person by using data from a specific person's Twitter account that could predict their next tweets.

The major challenges in this process include finding a suitable training corpus, handling large amounts of data, and improving the accuracy of our model. Finding a training corpus can be difficult because the data should reflect the types of text we wish to emulate as much as possible, and it should also be as big as possible given the constraints of our hardware. We settled on a corpus of just over 200 MB, which results in almost 2.5 GB of ngram data for unigrams through 5-grams. Our code should be as efficient as possible to be able to keep runtimes as low as possible in order to get more testing done.

Improving the accuracy of our model is difficult because speech is dynamic. Humans are always inventing new words and using old words in new ways. It is difficult to create a model that can accurately predict speech patterns when these patterns are constantly changing, so we have to strike a balance of accuracy, efficiency, and time to implement. Another challenge is cleaning the data in order to remove extraneous punctuation and spam, which can interfere with the accuracy of our model. To begin using this kind of constraint, we must limit our data to only being in one language, English in our case.

Different variations within this problem space include using more than just the previous word to predict the next word. Using multiple previous words can be used to compare accuracies, and depending on the significance of certain words in a certain order, can affect the probability of a subsequent word. Implementing this technique as well as using only the previous word can be used to help find which technique is most accurate and appropriate for the given data.

2 Techniques

The basis of our project is the use of Markov chains and n-grams to predict the next word given a training data set. This works by using the chain rule of probability to find the probability of the next predicted word given the previous words. From this idea, we are able to divide up the sentences from the given data set into n-grams of lengths of 1, 2, 3, 4, and 5. These n-grams are used in these predictions to show the words that were used previously. The n-gram model can also be represented by the probability of getting a certain n-gram, including the predicted word being considered, divided by the probability of the n-gram without.

A limitation within the use of this technique is that an input may not exist in the training corpus, as the English language is very large. In this case, our implementation will select the most common word from the corpus, however, this may lead to the resulting sentence being less coherent to the user. In this way, the predictor cannot know every word in the English language, as we have stated previously because many datasets are very large.

Creating ngram dictionaries:
The time complexity of creating ngrams is O(k*m*n*v) where n is the number of words in our corpus, m is the number of stopwords, k is the size of the ngrams we are creating (ie. 3 for trigrams), and v is the number of ngram dictionaries we wish to create. The space complexity is O(k*n) where n is the number of ngram dictionaries we wish to create, and k is the number of unique ngrams of that size. The limitations of this algorithm is that it is time consuming to generate these dictionaries. We used Pickle to save and load these dictionaries which vastly improves runtime. Loading the dictionaries should theoretically be O(n) where n is the size of the file, but it is a bit more complicated than that. Loading is still much faster than generating the dictionaries each time. One alternative we were considering was using the Google ngram dataset, which was orders of magnitude larger than our corpus, but using it to predict the next word would have been too time consuming. We also wanted to create the ngrams ourselves using any corpus we wanted, rather than using a dataset of ngrams and their number of occurrences that already existed.

Predicting next word:
The time complexity of predicting the next word is O(n*k) where n is the number of unique unigrams in the corpus, and k is the number of ngram dictionaries you have (since we use backoff).

The space complexity is O(1), since we already have the number of occurrences of each ngram stored in our dictionaries.
Limitations of the approach are that we will only guess next words if they already exist within our corpus. Since language is always evolving, our model will never guess words that it hasn't seen before, or words that it hasn't seen after the previous words.
There are many alternatives within the realm of natural language processing, however, none of these other approaches seemed accessible to us to code from scratch after looking at what these methods entail. For example, GPT-3 which is considered to be the most accurate language model uses deep learning.

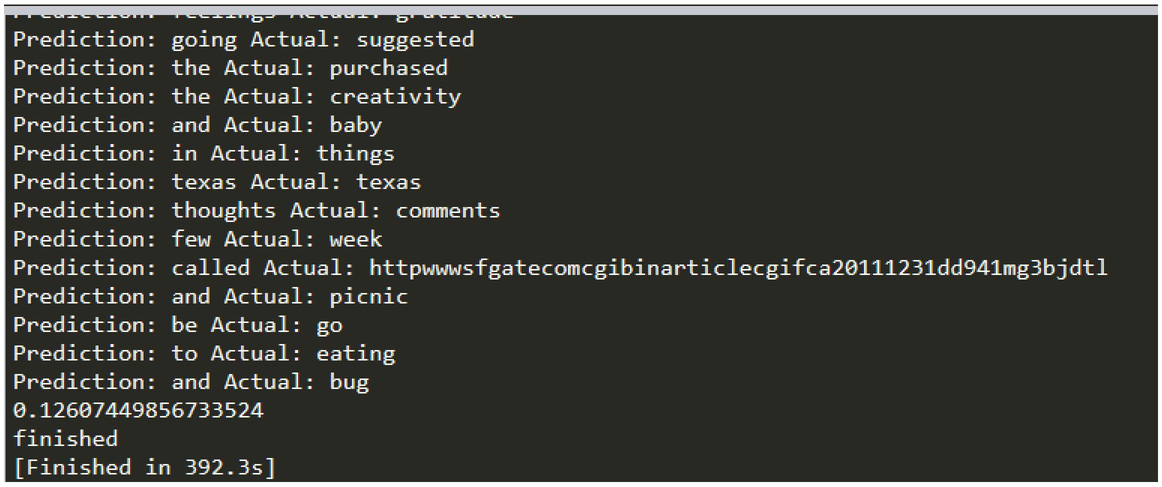
3 Modeling Human Thought Processes

This kind of predictor models human thought processes in the way such that children learn how to speak when they don't necessarily know what all of the words mean or how grammar works. For example, a child would learn that it is proper to say "She goes" rather than "She go." This process could be simplified to comparing the rates at which the child hears the two phrases. The child would most likely hear the former more frequently than the latter, and our predictor works in a similar fashion.

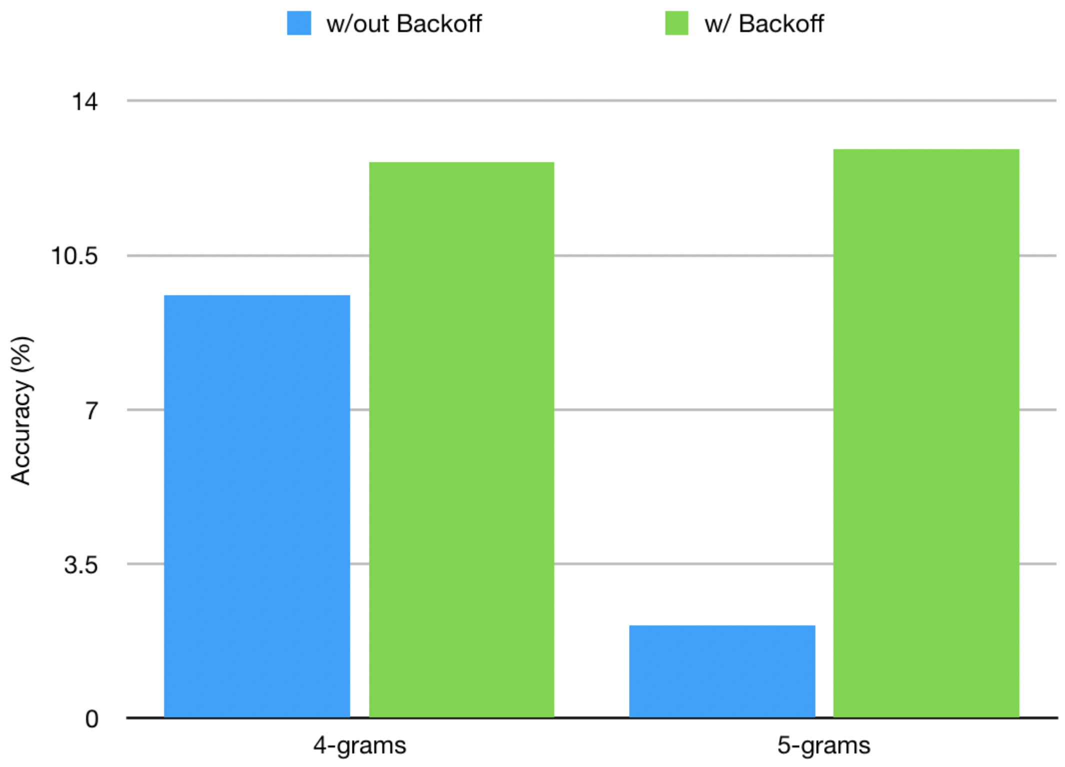
Another way to visualize this problem in terms of human thought processing is if a friend wanted you to guess the next word they would say. In order to guess this word, you would think about what their vocabulary normally includes, such as if they have any words or phrases that they use frequently, while also considering the word they just said. In this way, you would somewhat be calculating the probability of each of their most frequent and most likely words given the word they just said.

Our next word generator behaves in a similar way to both of these scenarios. It Selects the highest probability of n-grams created from adding every word in a given vocabulary to the previously recorded words. While the human modeling is much less calculated, the generator could be a more accurate representation of what humans do in this situation. It could also act much faster than the human mind, which could be a reason why these predictors have become so common in the keyboards in our phones when texting or sending emails. Over time, these predictions become more accurate the more we use them as it gathers more data on what words we use most frequently and the order in which we use them.

4 Algorithm Analysis



```
Prediction: going Actual: suggested
Prediction: the Actual: purchased
Prediction: the Actual: creativity
Prediction: and Actual: baby
Prediction: in Actual: things
Prediction: texas Actual: texas
Prediction: thoughts Actual: comments
Prediction: few Actual: week
Prediction: called Actual: httpwwwsfgatecomcgibinarticlecgifca20111231dd941mg3bjdtl
Prediction: and Actual: picnic
Prediction: be Actual: go
Prediction: to Actual: eating
Prediction: and Actual: bug
0.12607449856733524
finished
[Finished in 392.3s]
```



Our testing method was to split our corpus into training and test data, with the test data containing the first 1000 lines, and the rest being used as training data. After training our model on the training data, we used the first n-1 words of each line to see if our model could predict the nth word. We kept track of the number of times our model predicted the next word correctly, as well as the total number of predictions, and divided to get a percentage of accuracy. These results showed that our model has some potential, but needs to be rethought to get a high degree of accuracy for predicting the next word. The reason that 5-grams without backoff performed so poorly is that most 5-grams do not already exist in our corpus, so our model returned nothing.

These results were a bit higher than we had initially expected. After improving our accuracy by improving ngram size and implementing backoff, it seems that this is a fairly good accuracy for an ngram based approach to predicting the next word.

The results imply that this problem space is very nuanced with many edge cases. It is no wonder why the best approaches to NLP today use machine learning because there are far too many variables to account for using more straightforward approaches.
One of the side-effects of predicting the next word is that we can generate entire sentences with our model by using the word we predicted as input for another word. Although it would require modifications, it is still interesting to see what kinds of sentences our model generates. Our model seems to cause sentence generation to form loops, but that is to be expected since we are trying to predict the most likely next word. It might be interesting to add some randomness to our model for sentence generation, but that would decrease our accuracy for next word prediction.

Although our accuracy is not extremely high, our model often gives us a good candidate next word. Even though it might not be the next word we were thinking, our model almost always makes correct grammatical sense.

5 Outside Resources

In order to understand this concept better and how the math works behind the Markov chains and n-gram models, we used Speech and Language Processing (3rd ed.) from Dan Jurafsky and James H. Martin. This book provided us with basic details on how n-gram models work and defined many of the mathematical models that were used. It also provided us with more information about how we could further improve this model using techniques such as smoothing and interpolation. We did not receive any code from this source, but it simply provided more of a mathematical background to our project, which we could then implement on our own.

We used a dataset from Kaggle that included 4 million text entries from tweets, blogs, and news articles. This dataset was created by SWIFTKEY in collaboration with the Johns Hopkins Data Science Specialization. The set was primarily created for Natural Language Processing research. The data includes multiple languages, but we only used the English language to simplify our problem.

6 To Be Expanded or Improved

We could add interpolation to our model. Instead of using backoff alone, we could give different weights to different sized ngrams, and determine the best choice of next words based on input from multiple different sizes of ngrams. We attempted to implement this, but it requires having a held-out corpus that is used to optimize these weights after initializing the dictionaries using our training data. This could increase the accuracy of our model, but we have not yet been able to implement interpolation that increases the accuracy of our model thus far.
We could also use sentiment analysis to determine the sentiment of the sentence so far, using dictionaries generated from a different corpora depending on sentiment to increase the accuracy of our model.
We also learned about how much data is generated when using an ngram approach to word prediction. The data generated about the number of occurrences of each ngram from our input was an order of magnitude larger (from about 200 MB to 2.5 GB of data).