# PROBABILISTIC LANGUAGE MODELS

Ramaseshan Ramachandran

INTRODUCTION TO LANGUAGE MODELS

July 17-21, 2023

A Brief Introduction to probability

Probabilistic Language Model - Definition

Chain Rule

Markov Assumption

Target and Context words

Language Modeling using Unigrams

Generative Model

Maximum Likelihood Estimate

Bigram Language Model

Bigram Language Model - Example

Perplexity

Curse of dimensions

Ramaseshan

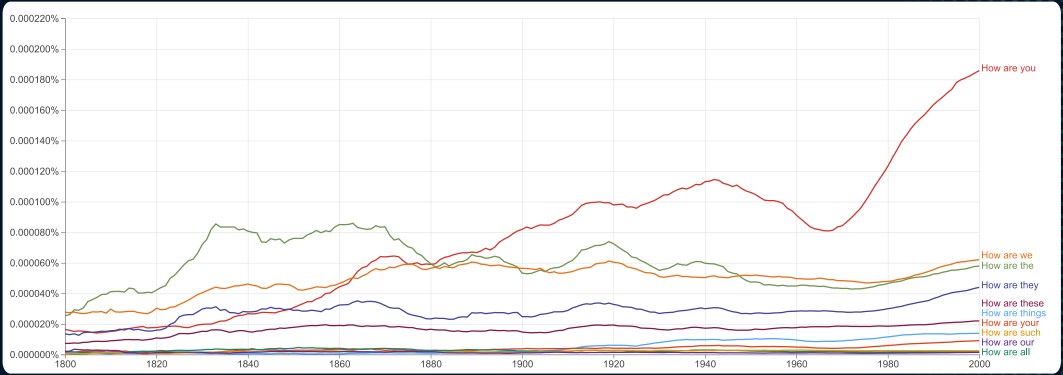
# INTRODUCTION

How are _____? Can you guess the missing word?

Ramaseshan

# INTRODUCTION

How are _____? Can you guess the missing word?



Source:Google NGram Viewer

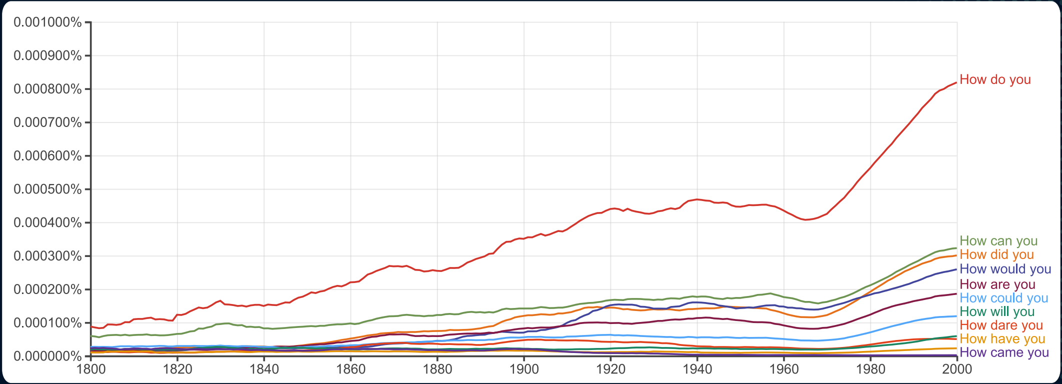
# INTRODUCTION

How _____ you? Can you guess the missing word?

Ramaseshan

# INTRODUCTION

How _____ you? Can you guess the missing word?



Source:Google NGram Viewer

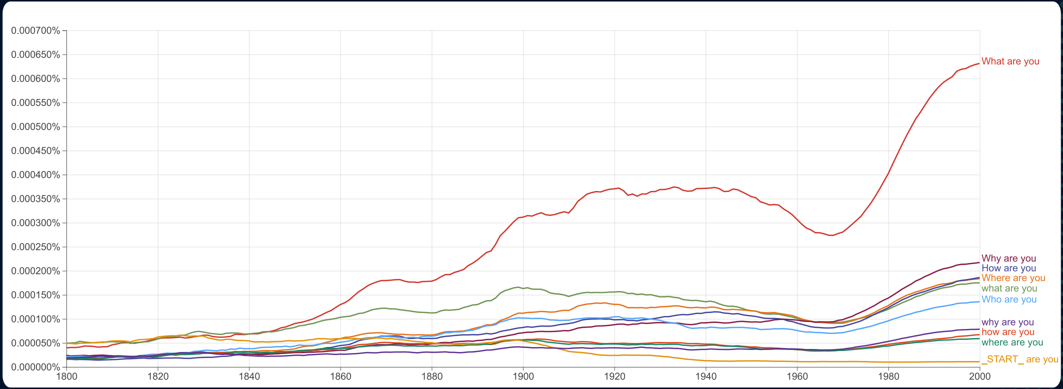
# INTRODUCTION

_____ are you?

Ramaseshan

# INTRODUCTION

_____ are you?



Source:Google NGram Viewer

# INTRODUCTION

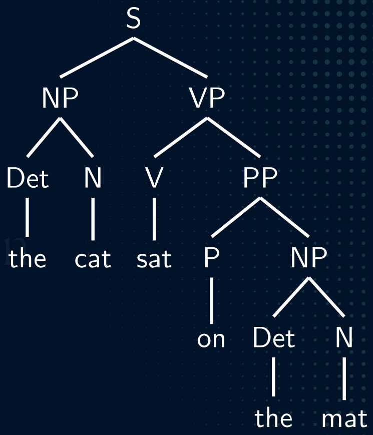
How do humans predict the next word?

- Domain knowledge
- Syntactic knowledge
- Lexical knowledge
- Knowledge about the sentence structure
- Some words are hard to find. Why?
- Natural language is not deterministic in general
- Some sentences are familiar or had been heard/seen/used several times
- They are more likely to happen than others, hence we could guess

Ramaseshan

# THE LANGUAGE MODEL

- Natural language sentences can be described by parse trees which use the morphology of words, syntax and semantics
- Probabilistic thinking - finding how likely a sentence occurs or formed, given the word sequence.
- In probabilistic world, the Language model is used to assign a probability $P(W)$ to every possible word sequence $W$.



Ramaseshan

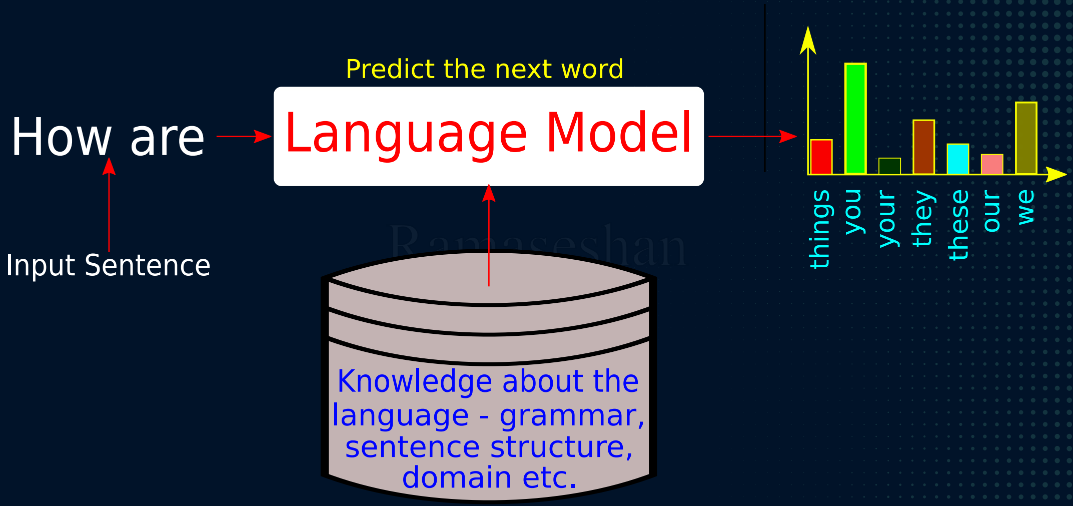
The current research in Language models focuses more on building the model from the huge corpus of text

# APPLICATIONS

| Application | Sample Sentences |
|---|---|
| Speech Recognition | Did you hear ***Recognize speech*** or Wreck a nice beach? |
| Context sensitive Spelling | One upon a ***tie***, ***Their*** lived aking |
| Machine translation | artwork is good → l'oeuvre est bonne |
| Sentence Completion | Complete a sentence as the previous word is given - GMail |

Ramaseshan

# A SIMPLE LANGUAGE MODEL IMPLEMENTATION



Predict the next word

How are

Input Sentence

Language Model

Knowledge about the language - grammar, sentence structure, domain etc.

Ramaseshan

# WHY PROBABILISTIC MODEL

- Speech recognition systems cannot depend on the processed speech signals. It may require the help of a language model and context recognizer to convert a speech to correct text format.
- As there are multiple combinations for a word to be in the next slot in a sentence, it is important for language modeling to be probabilistic in nature - judgment about the fluency of a sequence of words returns the probability of the sequence
- The probability of the next word in a sequence is real number $[0, 1]$
- The combination of words with high-probability in a sentence are more likely to occur than low-probability ones
- A probabilistic model continuously estimates the rank of the words in a sequence or phrase or sentence in terms of frequency of occurrence

Ramaseshan

# FORMAL DEFINITION

Let $\mathscr{V}$ be the vocabulary, a finite set of symbols or words. Let us use $\triangleleft$ and $\triangleright$ as the start and stop symbols and let them be the part of $\mathscr{V}$. Let $|\mathscr{V}|$ denote the size of $\mathscr{V}$.

Let $W$ be infinite sequences of words from the collection of $\mathscr{V}$. Every sequence in $W$ starts with $\triangleleft$ and ends with $\triangleright$. Then a language model is a probability distribution of a random variable $\mathscr{X}$ which takes values from $W$. Or p: $W \rightarrow \mathbb{R}$ such that

Ramaseshan

$$\forall x \in W, p(x) \geq 0 \text{ and} \tag{1}$$

$$\sum_{x \in W} p(X = x) = 1 \tag{2}$$

# PROBABILISTIC LANGUAGE MODEL

**Goal**: Compute the probability of a sequence of words

$$P(W) = P(w_1, w_2, w_3, ...w_n) \tag{3}$$

**Task**: To predict the next word using probability. Given the context, find the next word using

$$P(w_n | w_1, w_2, w_3, \ldots, w_{n-1}) \tag{4}$$

Ramaseshan

A model which computes the probability for (3) or predicting the next word (4) or complete the partial sentence is called as Probabilistic Language Model.

The goal is to learn the joint probability function of sequences of words in a language. The probability of $P(\text{The cat roars})$ is less likely to happen than $P(\text{The cat meows})$

# CHAIN RULE

Is it difficult to compute the probability of the entire sequence $P(w_1, w_2, w_3, \ldots, w_n)$?
***Chain rule*** is used to decompose the joint probability of a sequence into a product of conditional probability

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n) = P(w_1^n) \tag{5}$$

$$= P(w_1)P(w2|w_1)P(w3|w_2, w_1)\ldots P(w_n|w_{n-1}, w_{n-2}, w_{n-3}, \ldots, w_1) \tag{6}$$

$$= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \tag{7}$$

Ramaseshan

- It is possible to $P(w|h)$, but it does not really help in reducing the computational complexity
- We use innovative ways to string words to form new sentences
- Finding the probability for a long sentence may not yield good outcome as the context may never occur in the corpus
- Short sequences may provide better results

# MARKOV ASSUMPTION

**Markov Assumption**: The future behavior of a dynamic system depends on its recent history and not on the entire history

The product of the conditional probabilities can be written approximately for a bigram as

$$P(w_k|w_1^{k-1}) \approx P(w_k|w_{k-1}) \tag{8}$$

Equation (8) can be generalized for an *n-gram* as

Ramaseshan

$$P(w_k|w_1^{k-1}) \approx P(w_k|w_{k-K+1}^{k-1}) \tag{9}$$

Now, the joint probability of a sequence can be re-written as

$$\begin{align} P(W) &= P(w_1, w_2, w_3, \ldots, w_n) = P(w_1^n) \tag{10} \\ &= P(w_1)P(w2|w_1)P(w3|w_2, w_1)\ldots P(w_n|w_{n-1}, w_{n-2}, w_{n-3}, \ldots, w_1) \tag{11} \\ &= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \tag{12} \\ &\approx \prod_{k=1}^{n} P(w_k|w_{k-K+1}^{k-1}) \tag{13} \end{align}$$

# TARGET AND CONTEXT WORDS

Next word in the sentence depends on its immediate past words, known as context words

$$P(w_{k+1} | \underbrace{w_{i-k}, w_{i-k+1}, \ldots, w_k}_{\text{Context words}})$$

Ramaseshan

n-grams

- unigram - $P(w_{k+1})$
- bigram - $P(w_{k+1} | w_k)$
- trigram - $P(w_{k+1} | w_{k-1}, w_k)$
- 4-gram - $P(w_{k+1} | w_{k-2}, w_{k-1}, w_k)$

# LANGUAGE MODELING USING UNIGRAMS

- All words are generated independent of its history $W_{,}W_2, W_3, \ldots..W_n$ and none of them depend on the other
- Not a good model for language generation
- It will have $|V|$ parameters
- $\theta_i = p(w_i) = \frac{c_{w_i}}{N}$, where $c_{w_i}$ if the count of the word $w_i$ and $N$ is the total number of words in the vocabulary
- It may not be able to pick up regularities present tin the corpus
- It is more likely to generate ***the the the the*** as a sentence than a grammatically valid sentence

Ramaseshan

# GENERATIVE MODEL

- Generates a document containing $N$ words using n-gram
- A good model assigns higher probability to the word that actually occurs

$$P(\mathbf{W}) = P(N)\prod_{i=1}^{N} P(W_i) \tag{14}$$

Ramaseshan

- The location of the word in the document is not important
- P(N) is the distribution over $N$ and is same for all documents. Hence it may be ignored
- $W_i$, to be estimated in this model is $P(W_i)$ and it must satisfy $\sum_{i=1}^{N} P(w_i) = 1$

# MAXIMUM LIKELIHOOD ESTIMATE

- One of the methods to find the unknown parameter(s) is the use of Maximum Likelihood Estimate
- Estimate the parameter value for which the observed data has the highest probability
- Training data may not have all the words in the vocabulary
- If a sentence with an unknown word is presented, then the MLE is zero.
- Add a smoothing parameter to the equation without affecting the overall probability requirements

Ramaseshan

$$P(\mathbf{W}) = \frac{C_{w_i} + \alpha}{C_W + \alpha|V|} \tag{15}$$

$$\text{If } \alpha = 1, \text{then it is called as Laplace smoothing} \tag{16}$$

$$P(\mathbf{W}) = \frac{C_{w_i} + 1}{C_W + |V|} \tag{17}$$

# BIGRAM LANGUAGE MODEL

- This model generates a sequence one word at a time, starting with the first word and then generating each succeeding word conditioned on the previous one or its predecessor
- A bigram language model or the Markov model (first order)is defined as follows:

Ramaseshan

$$P(\mathbf{W}) = \prod_{i=1}^{n+1} P(w_i|w_{i-1}) \tag{18}$$

where $\mathbf{W} = w_1, w_2, w_3, \ldots, w_n$

# BIGRAM LANGUAGE MODEL

- Estimate the parameter $P(w_i|w_{i-1})$ for all bigrams
- The parameter estimation does not depend on the location of the word
- If we consider the sentence as a sequence in time, they are time-invariant MLE picks up the word that is $\frac{n_{w,w'}}{n_{w,o}}$ where $nw, w'$ is the number of times the words $w_1, w'$ occur together and $n_{w,o}$ is the number of times the word $w$ appears in the bigram sequence with any other word
- The number of parameters to be estimated $= |V| \times (|V| + 1)$

Ramaseshan

# PROBABILISTIC LANGUAGE MODEL - EXAMPLE

Peter Piper picked a peck of pickled peppers
A peck of pickled peppers Peter Piper picked
If Peter Piper picked a peck of pickled peppers
Where's the peck of pickled peppers Peter Piper picked?

—

The joint probability of a sentence formed with $n$ words can be expressed as a product conditional probabilities - we use immediate context and not the entire history

$$P(w_1|\langle\triangleleft\rangle) \times P(w_2|w_1) \times ...P(\langle E\rangle|w_n)$$

and $P(w_{i+1}|w_i) = \frac{C(w_i.w_{i+1})}{C(w_i)}$

—

What is the probability of these sentences?
P(Peter Piper picked)
P(Peter Piper picked peppers)

| Bigram | Frequency |
|---|---|
| ◁peter | 1 |
| peter piper | 4 |
| piper picked | 4 |
| picked a | 2 |
| a peck | 2 |
| peck of | 4 |
| pickled peppers | 4 |
| peppers ▷ | 1 |
| ◁a | 1 |
| a peck<br>peck of<br>of pickled | 1<br>1<br>4 |
| peppers peter | 2 |
| ... | .. |
| ◁... | 1 |

Ramaseshan

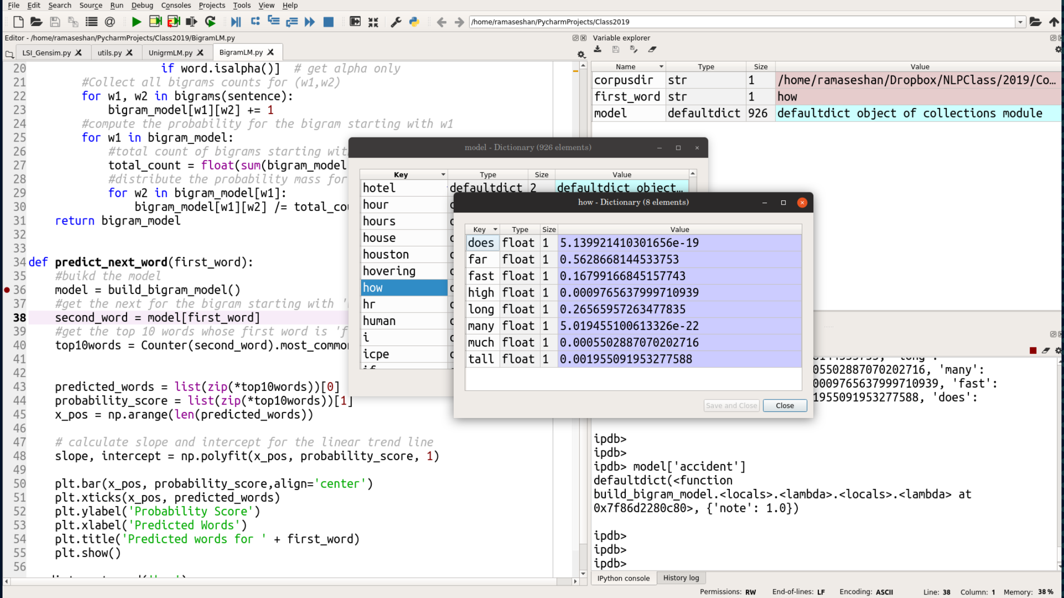
# BUILDING A BIGRAM MODEL - CODE

```python
#compute the bigram model
def build_bigram_model():
    bigram_model = collections.defaultdict(
        lambda: collections.defaultdict(lambda: 0))
    for sentence in kinematics_corpus.sents():
        sentence = [word.lower() for word in sentence
                    if word.isalpha()]  # get alpha only
        #Collect all bigrams counts for (w1,w2)
        for w1, w2 in bigrams(sentence):
            bigram_model[w1][w2] += 1
        #compute the probability for the bigram containing w1
        for w1 in bigram_model:
            #total count of bigrams conaining w1
            total_count = float(sum(bigram_model[w1].values()))
            #distribute the probability mass for all bigrams starting with w1
            for w2 in bigram_model[w1]:
                bigram_model[w1][w2] /= total_count
    return bigram_model
```

# BUILDING A BIGRAM MODEL - CODE

```python
def predict_next_word(first_word):
    #build the model
    model = build_bigram_model()
    #get the next for the bigram starting with ’word’
    second_word = model[first_word]
    #get the top 10 words whose first word is ’first_word’
    top10words = Counter(second_word).most_common(10)

    predicted_words = list(zip(*top10words))[0]
    probability_score = list(zip(*top10words))[1]
    x_pos = np.arange(len(predicted_words))

    plt.bar(x_pos, probability_score,align=’center’)
    plt.xticks(x_pos, predicted_words)
    plt.ylabel(’Probability Score’)
    plt.xlabel(’Predicted Words’)
    plt.title(’Predicted words for ’ + first_word)
    plt.show()

predict_next_word(’how’)
```

# MODEL PARAMETERS - BIGRAM EXAMPLE


File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/ramaseshan/PycharmProjects/Class2019
Editor - /home/ramaseshan/PycharmProjects/Class2019/BigramLM.py
LSI_Gensim.py
utils.py
UnigrmLM.py
BigramLM.py
Variable explorer

```python
                if word.isalpha()]  # get alpha only
        #Collect all bigrams counts for (w1,w2)
        for w1, w2 in bigrams(sentence):
            bigram_model[w1][w2] += 1
        #compute the probability for the bigram starting with w1
        for w1 in bigram_model:
            #total count of bigrams starting wit
            total_count = float(sum(bigram_model
            #distribute the probability mass for
            for w2 in bigram_model[w1]:
                bigram_model[w1][w2] /= total_co
    return bigram_model


def predict_next_word(first_word):
    #buikd the model
    model = build_bigram_model()
    #get the next for the bigram starting with '
    second_word = model[first_word]
    #get the top 10 words whose first word is 'f
    top10words = Counter(second_word).most_commo


    predicted_words = list(zip(*top10words))[0]
    probability_score = list(zip(*top10words))[1]
    x_pos = np.arange(len(predicted_words))

    # calculate slope and intercept for the linear trend line
    slope, intercept = np.polyfit(x_pos, probability_score, 1)

    plt.bar(x_pos, probability_score,align='center')
    plt.xticks(x_pos, predicted_words)
    plt.ylabel('Probability Score')
    plt.xlabel('Predicted Words')
    plt.title('Predicted words for ' + first_word)
    plt.show()
```

| Name | Type | Size | Value |
|---|---|---|---|
| corpusdir | str | 1 | /home/ramaseshan/Dropbox/NLPClass/2019/Co… |
| first_word | str | 1 | how |
| model | defaultdict | 926 | defaultdict object of collections module |

model - Dictionary (926 elements)

| Key | Type | Size | Value |
|---|---|---|---|
| hotel | defaultdict | 2 | defaultdict object… |
| hour | d | | |
| hours | d | | |
| house | d | | |
| houston | d | | |
| hovering | d | | |
| how | d | | |
| hr | d | | |
| human | d | | |
| i | d | | |
| icpe | d | | |

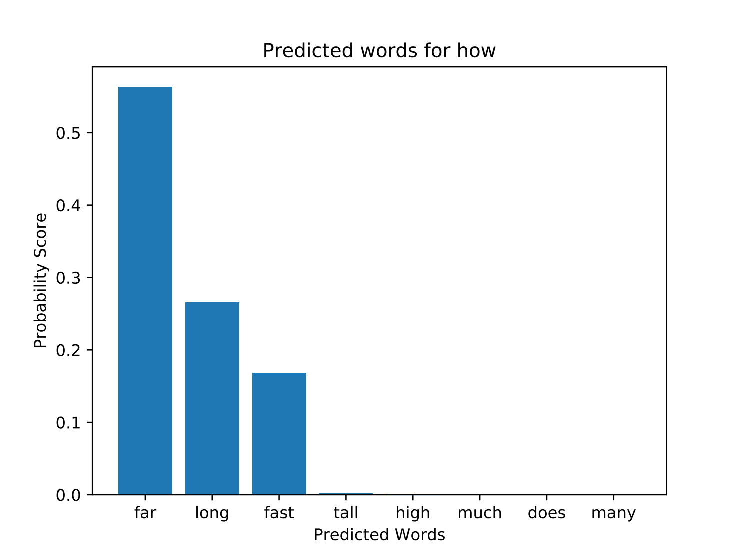
how - Dictionary (8 elements)

| Key | Type | Size | Value |
|---|---|---|---|
| does | float | 1 | 5.139921410301656e-19 |
| far | float | 1 | 0.5628668144533753 |
| fast | float | 1 | 0.16799166845157743 |
| high | float | 1 | 0.0009765637999710939 |
| long | float | 1 | 0.26565957263477835 |
| many | float | 1 | 5.019455100613326e-22 |
| much | float | 1 | 0.0005502887070202716 |
| tall | float | 1 | 0.001955091953277588 |

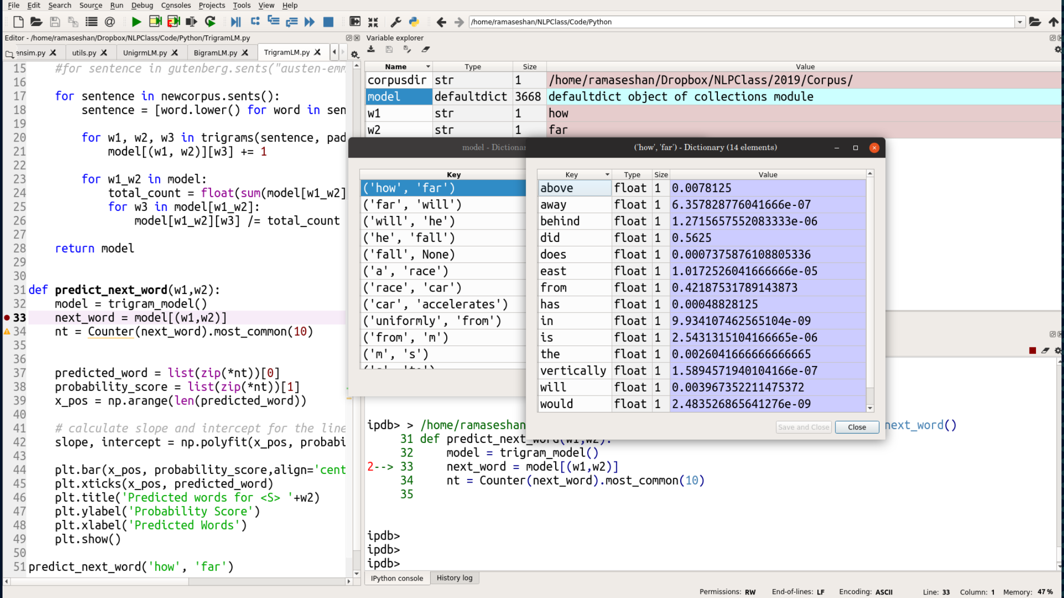
```
05502887070202716, 'many':
0009765637999710939, 'fast':
1955091953277588, 'does':

ipdb>
ipdb>
ipdb> model['accident']
defaultdict(<function
build_bigram_model.<locals>.<lambda>.<locals>.<lambda> at
0x7f86d2280c80>, {'note': 1.0})

ipdb>
ipdb>
ipdb>
```

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 38 Column: 1 Memory: 38 %

# BIGRAM MODEL - NEXT WORD PREDICTION



# MODEL PARAMETERS - TRIGRAM EXAMPLE


File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/ramaseshan/NLPClass/Code/Python
Editor - /home/ramaseshan/Dropbox/NLPClass/Code/Python/TrigramLM.py
ensim.py
utils.py
UnigrmLM.py
BigramLM.py
TrigramLM.py
Variable explorer
model - Dictionar
('how', 'far') - Dictionary (14 elements)
Save and Close
Close
IPython console
History log

```python
    #for sentence in gutenberg.sents("austen-emm

    for sentence in newcorpus.sents():
        sentence = [word.lower() for word in sen

        for w1, w2, w3 in trigrams(sentence, pad
            model[(w1, w2)][w3] += 1

        for w1_w2 in model:
            total_count = float(sum(model[w1_w2]
            for w3 in model[w1_w2]:
                model[w1_w2][w3] /= total_count

    return model

def predict_next_word(w1,w2):
    model = trigram_model()
    next_word = model[(w1,w2)]
    nt = Counter(next_word).most_common(10)


    predicted_word = list(zip(*nt))[0]
    probability_score = list(zip(*nt))[1]
    x_pos = np.arange(len(predicted_word))

    # calculate slope and intercept for the line
    slope, intercept = np.polyfit(x_pos, probabi

    plt.bar(x_pos, probability_score,align='cent
    plt.xticks(x_pos, predicted_word)
    plt.title('Predicted words for <S> '+w2)
    plt.ylabel('Probability Score')
    plt.xlabel('Predicted Words')
    plt.show()

predict_next_word('how', 'far')
```

| Name | Type | Size | Value |
|---|---|---|---|
| corpusdir | str | 1 | /home/ramaseshan/Dropbox/NLPClass/2019/Corpus/ |
| model | defaultdict | 3668 | defaultdict object of collections module |
| w1 | str | 1 | how |
| w2 | str | 1 | far |

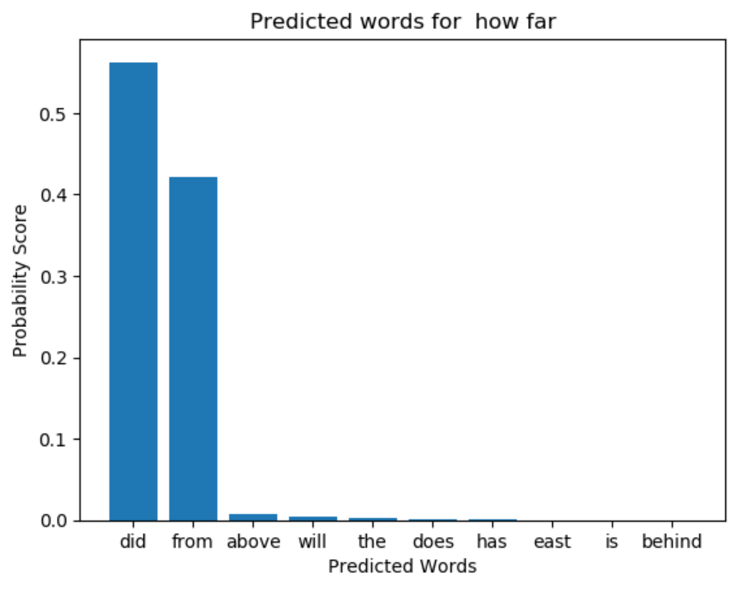
| Key |
|---|
| ('how', 'far') |
| ('far', 'will') |
| ('will', 'he') |
| ('he', 'fall') |
| ('fall', None) |
| ('a', 'race') |
| ('race', 'car') |
| ('car', 'accelerates') |
| ('uniformly', 'from') |
| ('from', 'm') |
| ('m', 's') |

| Key | Type | Size | Value |
|---|---|---|---|
| above | float | 1 | 0.0078125 |
| away | float | 1 | 6.357828776041666e-07 |
| behind | float | 1 | 1.2715657552083333e-06 |
| did | float | 1 | 0.5625 |
| does | float | 1 | 0.0007375876108805336 |
| east | float | 1 | 1.0172526041666666e-05 |
| from | float | 1 | 0.42187531789143873 |
| has | float | 1 | 0.00048828125 |
| in | float | 1 | 9.934107462565104e-09 |
| is | float | 1 | 2.5431315104166665e-06 |
| the | float | 1 | 0.0026041666666666665 |
| vertically | float | 1 | 1.5894571940104166e-07 |
| will | float | 1 | 0.003967352211475372 |
| would | float | 1 | 2.483526865641276e-09 |

```
ipdb> > /home/ramaseshan                                  next_word()
     31 def predict_next_
     32     model = trigram_model()
2--> 33     next_word = model[(w1,w2)]
     34     nt = Counter(next_word).most_common(10)
     35


ipdb>
ipdb>
ipdb>
```

Permissions: RW   End-of-lines: LF   Encoding: ASCII   Line: 33   Column: 1   Memory: 47 %

# TRIGRAM MODEL - NEXT WORD PREDICTION



# PERPLEXITY

Perplexity is a measurement of how well a probability model predicts a sample. Perplexity is defined as

$$\text{For bigram model, } PP(W_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \tag{19}$$

Ramaseshan

$$\text{For trigram model } PP(W_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1}w_{i-2})}} \tag{20}$$

A good model gives maximum probability to a sentence or minimum perplexity to a sentence

# UNKNOWN WORDS

- In a closed vocabulary language model, there is no unknown words or ***out of vocabulary words (OOV)***
- In an open vocabulary system, you will find new words that are not present in the trained model
- Pick words below certain frequency and replace them as OOV.
- Treat every OOV as a regular word
- During testing, the new words would be treated as OOV and the corresponding frequency will be used for computation
- This eliminates zero probability for sentences containing OOV

Ramaseshan

# CURSE OF DIMENSIONALITY

- A fundamental problem that makes language modeling and other learning problems difficult is the curse of dimensionality
- It is particularly obvious in the case when one wants to model the joint distribution between many discrete random variable
- If one wants to estimate the joint probability distribution of $10$ words in a language with a million words as vocabulary, then we need to estimate $10000000^9 \cdot (1000000 - 1) \approx 10^{60}$ parameters

Ramaseshan

# Thank you

Ramaseshan Ramachandran

Ramaseshan