



PROBABILISTIC LANGUAGE MODELS

INTRODUCTION TO LANGUAGE MODELS

July 17-21, 2023

Ramaseshan Ramachandran

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

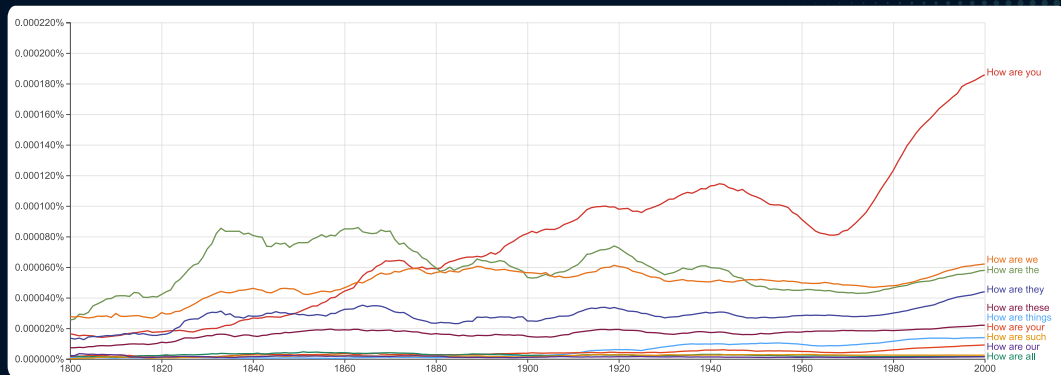
INTRODUCTION

How are ____? Can you guess the missing word?

Ramaseshan

INTRODUCTION

How are ____? Can you guess the missing word?



Source: Google NGram Viewer

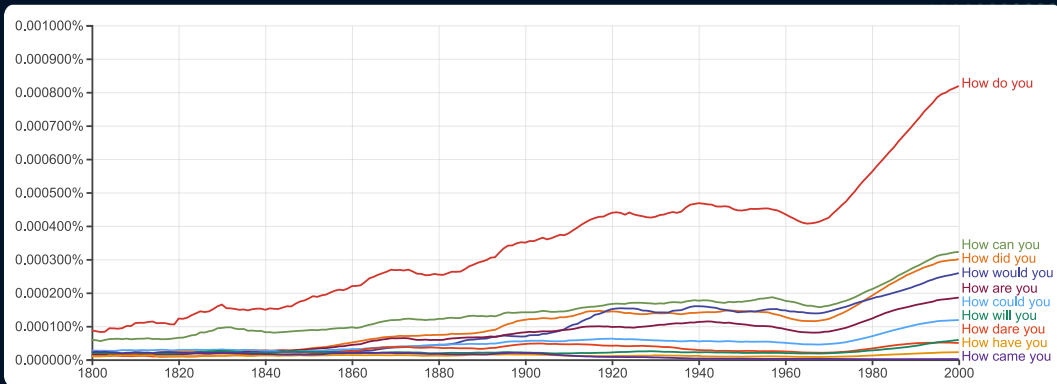
INTRODUCTION

How ____ you? Can you guess the missing word?

Ramaseshan

INTRODUCTION

How ____ you? Can you guess the missing word?



Source: Google NGram Viewer

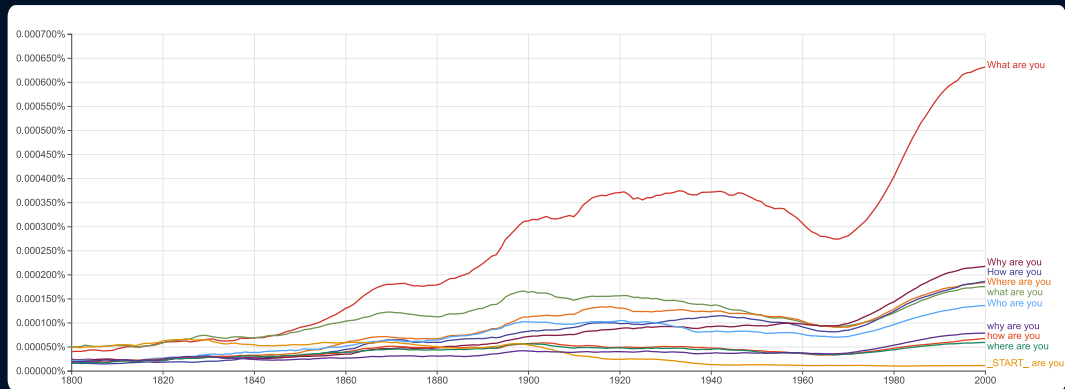
INTRODUCTION

_____ are you?

Ramaseshan

INTRODUCTION

_____ are you?



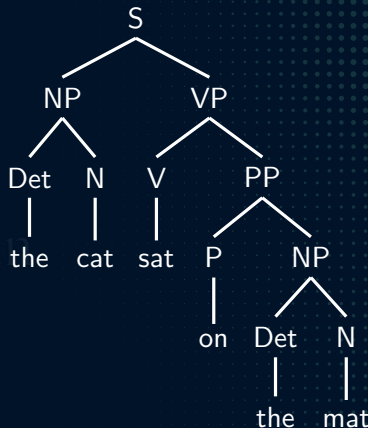
Source: Google NGram Viewer

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

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 .

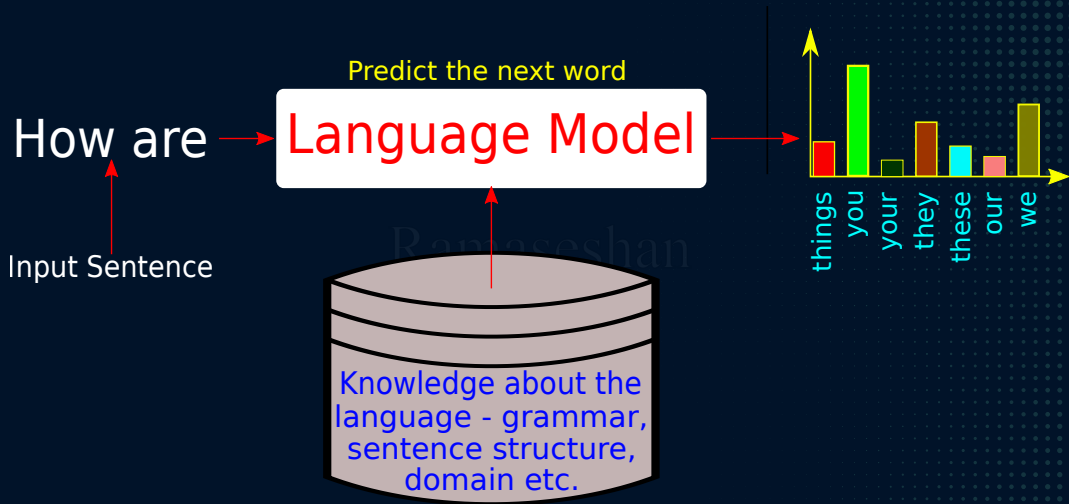


The current research in Language models focuses more on building the model from the huge corpus of text

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 |

A SIMPLE LANGUAGE MODEL IMPLEMENTATION



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

FORMAL DEFINITION

Let \mathcal{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 \mathcal{V} . Let $|\mathcal{V}|$ denote the size of \mathcal{V} .

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

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

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

PROBABILISTIC LANGUAGE MODEL

Goal: Compute the probability of a sequence of words

$$P(W) = P(w_1, w_2, w_3, \dots, w_n) \quad (3)$$

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

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

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 $P(w_1, w_2, w_3, \dots, w_n)$?

Chain rule is used to decompose the joint probability of a sequence into a product of conditional probability

$$P(W) = P(w_1, w_2, w_3, \dots, w_n) = P(w_1^n) \quad (5)$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_2, w_1) \dots P(w_n|w_{n-1}, w_{n-2}, w_{n-3}, \dots, w_1) \quad (6)$$

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

- ▶ 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

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

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

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

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

$$P(W) = P(w_1, w_2, w_3, \dots, w_n) = P(w_1^n) \quad (10)$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_2, w_1) \dots P(w_n|w_{n-1}, w_{n-2}, w_{n-3}, \dots, w_1) \quad (11)$$

$$= \prod_{k=1}^n P(w_k | w_1^{k-1}) \quad (12)$$

$$\approx \prod_{k=1}^n P(w_k | w_{k-K+1}^{k-1}) \quad (13)$$

TARGET AND CONTEXT WORDS

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

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

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 - $P(w_{k+1} | w_{k-2}, w_{k-1}, w_k)$

Ramaseshan

LANGUAGE MODELING USING UNIGRAMS

- ▶ All words are generated independent of its history $W_1, W_2, W_3, \dots, W_n$ 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} is 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 in the corpus
- ▶ It is more likely to generate ***the the the the*** as a sentence than a grammatically valid sentence

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

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

- ▶ 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

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

If $\alpha = 1$, then it is called as Laplace smoothing (16)

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

- ▶ 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:

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

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

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 $n_{w,w'}$ 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 $= |V| \times (|V| + 1)$

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

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

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

—

What is the probability of these sentences?

$P(\text{Peter Piper picked})$

$P(\text{Peter Piper picked peppers})$

| Bigram | Frequency |
|------------------------------|-----------|
| $\triangleleft \text{peter}$ | 1 |
| peter piper | 4 |
| piper picked | 4 |
| picked a | 2 |
| a peck | 2 |
| peck of | 4 |
| pickled peppers | 4 |
| peppers \triangleright | 1 |
| $\triangleleft \text{a}$ | 1 |
| a peck | 1 |
| peck of | 1 |
| of pickled | 4 |
| peppers peter | 2 |
| ... | .. |
| $\triangleleft \dots$ | 1 |

BUILDING A BIGRAM MODEL - CODE

```
1 #compute the bigram model
2 def build_bigram_model():
3     bigram_model = collections.defaultdict(
4         lambda: collections.defaultdict(lambda: 0))
5     for sentence in kinematics_corpus.sents():
6         sentence = [word.lower() for word in sentence
7                     if word.isalpha()] # get alpha only
8     #Collect all bigrams counts for (w1,w2)
9     for w1, w2 in bigrams(sentence):
10         bigram_model[w1][w2] += 1
11    #compute the probability for the bigram containing w1
12    for w1 in bigram_model:
13        #total count of bigrams conaining w1
14        total_count = float(sum(bigram_model[w1].values()))
15        #distribute the probability mass for all bigrams starting with w1
16        for w2 in bigram_model[w1]:
17            bigram_model[w1][w2] /= total_count
18    return bigram_model
```

BUILDING A BIGRAM MODEL - CODE

```
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

The screenshot displays a Jupyter Notebook environment with a code editor, a variable explorer, and several dictionary windows.

Code Editor: The code defines a bigram model and a function to predict the next word. It includes comments for each step, from collecting bigram counts to plotting the probability scores.

```
20 if word.isalpha() # get alpha only
21 #Collect all bigrams counts for (w1,w2)
22 for w1, w2 in bigrams(sentence):
23     bigram_model[w1][w2] += 1
24 #compute the probability for the bigram starting with w1
25 for w1 in bigram_model:
26     #total count of bigrams starting with w1
27     total_count = float(sum(bigram_model[w1].values()))
28     #distribute the probability mass for w1
29     for w2 in bigram_model[w1]:
30         bigram_model[w1][w2] /= total_count
31 return bigram_model
32
33
34 def predict_next_word(first_word):
35     #build the model
36     model = build_bigram_model()
37     #get the next for the bigram starting with '
38     second_word = model[first_word]
39     #get the top 10 words whose first word is 'f
40     top10words = Counter(second_word).most_common(10)
41
42     predicted_words = list(zip(*top10words))[0]
43     probability_score = list(zip(*top10words))[1]
44     x_pos = np.arange(len(predicted_words))
45
46     # calculate slope and intercept for the linear trend line
47     slope, intercept = np.polyfit(x_pos, probability_score, 1)
48
49     plt.bar(x_pos, probability_score, align='center')
50     plt.xticks(x_pos, predicted_words)
51     plt.ylabel('Probability Score')
52     plt.xlabel('Predicted Words')
53     plt.title('Predicted words for ' + first_word)
54     plt.show()
55
56
```

Variable Explorer: Shows the state of variables in the current scope.

| 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 of collections module |
| hour | defaultdict | 2 | defaultdict object of collections module |
| hours | defaultdict | 2 | defaultdict object of collections module |
| house | defaultdict | 2 | defaultdict object of collections module |
| houston | defaultdict | 2 | defaultdict object of collections module |
| hovering | defaultdict | 2 | defaultdict object of collections module |
| how | defaultdict | 2 | defaultdict object of collections module |
| hr | defaultdict | 2 | defaultdict object of collections module |
| human | defaultdict | 2 | defaultdict object of collections module |
| i | defaultdict | 2 | defaultdict object of collections module |
| icpe | defaultdict | 2 | defaultdict object of collections module |

how - Dictionary (8 elements):

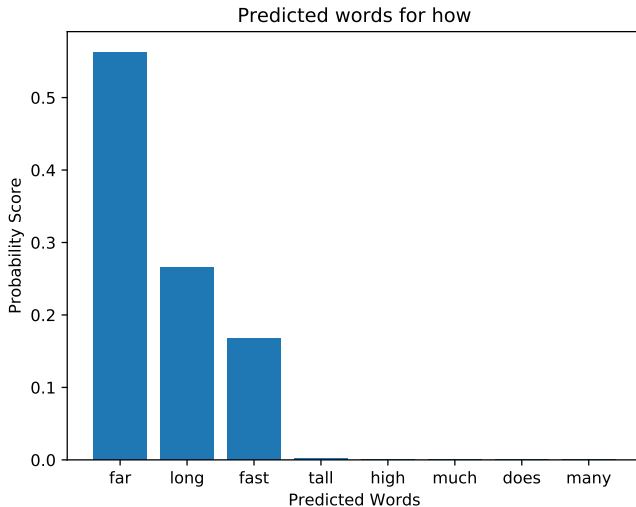
| Key | Type | Size | Value |
|------|-------|------|-----------------------|
| does | float | 1 | 5.139921410301656e-19 |
| far | float | 1 | 0.5628668144533753 |
| fast | float | 1 | 0.16799166845157743 |
| how | float | 1 | 0.0009765637999710939 |
| high | float | 1 | 0.0009765637999710939 |
| long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

IPython console:

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

Figure: A bar chart titled "Predicted words for 'how'" showing the probability scores for the top 10 predicted words. The x-axis labels are "does", "far", "fast", "how", "high", "long", "many", "much", and "tall". The y-axis represents the "Probability Score". The bars are centered and colored in a light blue shade.

BIGRAM MODEL - NEXT WORD PREDICTION



MODEL PARAMETERS - TRIGRAM EXAMPLE

The screenshot shows a Python IDE with a file explorer on the left, a code editor in the center, and two variable explorer windows on the right.

Code Editor: The code defines a trigram model and a function to predict the next word. The model is trained on the Gutenberg corpus. The function `predict_next_word` takes two words (`w1`, `w2`) and returns the most common next word and its probability score.

```
#for sentence in gutenbergsents("austen-em")
16
17 for sentence in newcorpus.sents():
18     sentence = [word.lower() for word in sen
19
20     for w1, w2, w3 in trigrams(sentence, pad
21         model[(w1, w2)][w3] += 1
22
23     for w1_w2 in model:
24         total_count = float(sum(model[w1_w2]
25         for w3 in model[w1_w2]:
26             model[w1_w2][w3] /= total_count
27
28     return model
29
30
31 def predict_next_word(w1,w2):
32     model = trigram_model()
33     next_word = model[(w1,w2)]
34     nt = Counter(next_word).most_common(10)
35
36
37     predicted_word = list(zip(*nt))[0]
38     probability_score = list(zip(*nt))[1]
39     x_pos = np.arange(len(predicted_word))
40
41     # calculate slope and intercept for the line
42     slope, intercept = np.polyfit(x_pos, probabi
43
44     plt.bar(x_pos, probability_score,align='cent
45     plt.xticks(x_pos, predicted_word)
46     plt.title('Predicted words for <S> ' +w2)
47     plt.ylabel('Probability Score')
48     plt.xlabel('Predicted Words')
49     plt.show()
50
51 predict_next_word('how', 'far')
```

Variable Explorer (Top): Shows the state of the model and corpus.

| 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 |

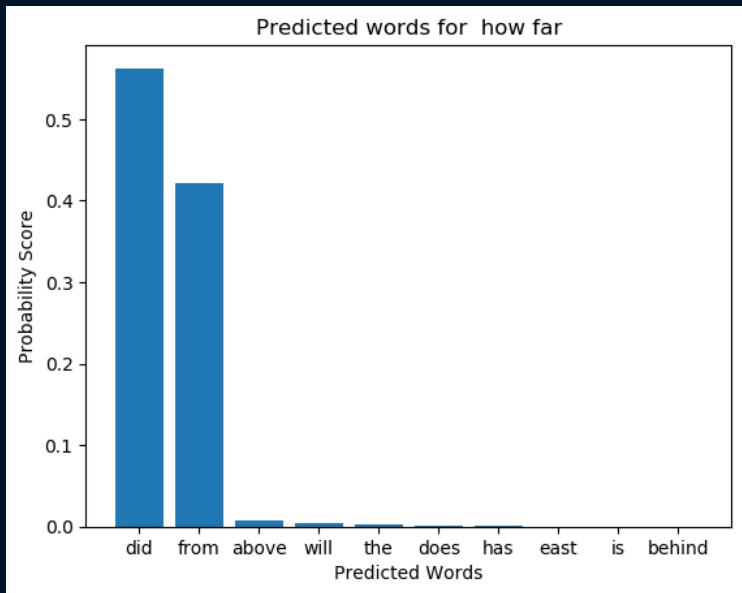
Variable Explorer (Bottom): Shows the parameters of the trigram model for the key ('how', 'far').

| Key | Type | Size | Value |
|------------------------|-------|------|------------------------|
| ('how', 'far') | float | 1 | 0.0078125 |
| ('far', 'will') | float | 1 | 6.357828776041666e-07 |
| ('will', 'he') | float | 1 | 1.2715657552083333e-06 |
| ('he', 'fall') | float | 1 | 0.5625 |
| ('fall', None) | float | 1 | 0.0007375876108805336 |
| ('a', 'race') | float | 1 | 1.0172526041666666e-05 |
| ('race', 'car') | float | 1 | 0.42187531789143873 |
| ('car', 'accelerates') | float | 1 | 0.00048828125 |
| ('uniformly', 'from') | float | 1 | 9.934107462565104e-09 |
| ('from', 'm') | float | 1 | 2.5431315104166665e-06 |
| ('m', 's') | float | 1 | 0.0026041666666666665 |
| vertically | float | 1 | 1.5894571940104166e-07 |
| will | float | 1 | 0.003967352211475372 |
| would | float | 1 | 2.483526865641276e-09 |

IPython console: Shows the execution of the `predict_next_word` function.

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

TRIGRAM MODEL - NEXT WORD PREDICTION



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

Perplexity is defined as

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

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

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

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 $1000000^9 \cdot (1000000 - 1) \approx 10^{60}$ parameters