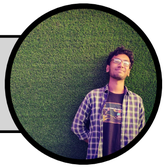
Probabilistic Language Modeling 4 Code

Dr. Antonio Mastropaolo

Instructor

Mr. Alvi Haque



Teaching Assistant

WILLIAM & MARY

CHARTERED 1693

Spring 2026

Probabilistic Language Models

Probabilistic Language Models

- PLM is a statistical approach to model the likelihood of the occurring words in a sentence

Probabilistic Language Models

- PLM is a statistical approach to model the likelihood of the occurring words in a sentence

- It estimates how likely a given word or sequence of words is to occur in a particular context, based on statistical patterns observed in large ***corpora of text***.

Probabilistic Language Models

- PLM is a statistical appr likelihood of the occurring words in a sentence

- It estimates how likely a uence of words is to occur in a particular cor istical patterns observed in large ***corpo***

Probabilistic Language Models

$$s = (w_1, w_2, \ldots, w_N)$$

Probabilistic Language Models

$$s = (w_1, w_2, \ldots, w_N)$$

$$P(s) =$$

Probabilistic Language Models

$$s = (w_1, w_2, ..., w_N)$$

$$P(s) = P(w_1, w_2, ..., w_N)$$

Probabilistic Language Models

$$s = (w_1, w_2, ..., w_N)$$

$$P(s) = P(w_1, w_2, ..., w_N)$$

Conditional Probabilities

- Event **A**
- Event **B**

Probabilistic Language Models

$$s = (w_1, w_2, ..., w_N)$$

$$P(s) = P(w_1, w_2, ..., w_N)$$

Conditional Probabilities

- Event **A**
- Event **B**

P(B|A) = P(A,B)/P(A)

Probabilistic Language Models

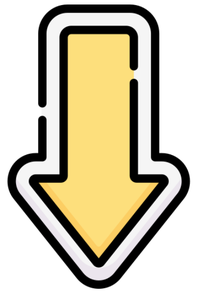
$$s = (w_1, w_2, ..., w_N)$$

$$P(s) = P(w_1, w_2, ..., w_N)$$

Conditional Probabilities

- Event **A**
- Event **B**

P(B|A) = P(A,B)/P(A)



P(A,B) = P(A)P(B|A)

Probabilistic Language Models

$$s = (w_1, w_2, ..., w_N)$$

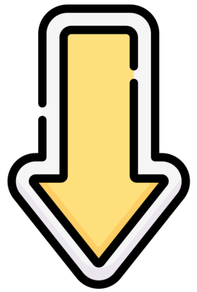
$$P(s) = P(w_1, w_2, ..., w_N)$$

$$P(w_1, ..., w_N) = P(w_1)P(w_2|w_1)...$$
$$P(w_n|w_1, ..., w_{N-1})$$

Conditional Probabilities

- Event **A**
- Event **B**

P(B|A) = P(A,B)/P(A)



P(A,B) = P(A)P(B|A)

Probabilistic Language Models

$$s = (w_1, w_2, ..., w_N)$$

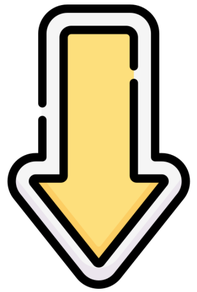
$$P(s) = P(w_1, w_2, ..., w_N)$$

$$P(w_1, ..., w_N) = P(w_1)P(w_2|w_1)...$$
$$P(w_n|w_1, ..., w_{N-1})$$
$$= \prod_{i=1}^{N} P(w_i|w_1, ...w_{i-1})$$

Conditional Probabilities

- Event **A**
- Event **B**

P(B|A) = P(A,B)/P(A)

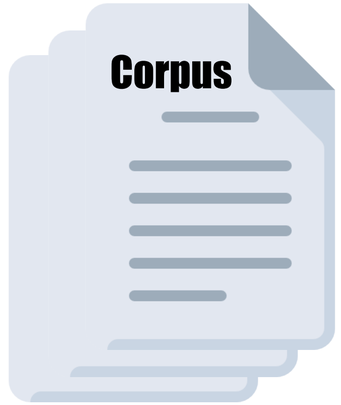


P(A,B) = P(A)P(B|A)

Probabilistic Language Models

$$P(w_1, ..., w_N) = P(w_1)P(w_2|w_1)...P(w_n|w_1, ..., w_{N-1})$$

$$= \prod_{i=1}^{N} P(w_i|w_1, ...w_{i-1})$$

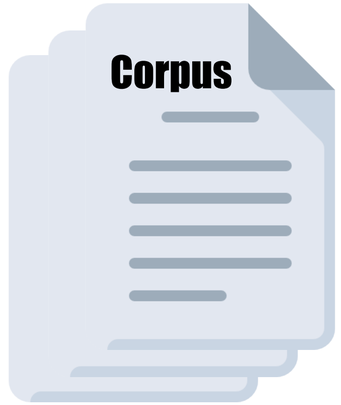


P(“its water is so transparent”) =

Probabilistic Language Models

$$P(w_1, ..., w_N) = P(w_1)P(w_2|w_1)...P(w_n|w_1, ..., w_{N-1})$$

$$= \prod_{i=1}^{N} P(w_i|w_1, ...w_{i-1})$$

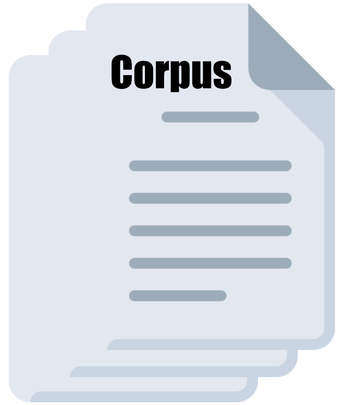


P(“its water is so transparent”) =
P(its)

Probabilistic Language Models

$$P(w_1, ..., w_N) = P(w_1)P(w_2|w_1)...P(w_n|w_1, ..., w_{N-1})$$

$$= \prod_{i=1}^{N} P(w_i|w_1, ...w_{i-1})$$

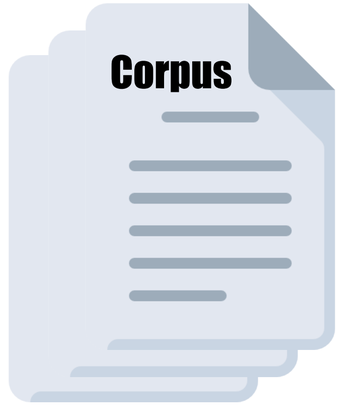


P(“its water is so transparent”) =
P(its) × P(water|its)

Probabilistic Language Models

$$P(w_1, ..., w_N) = P(w_1)P(w_2|w_1)...P(w_n|w_1, ..., w_{N-1})$$

$$= \prod_{i=1}^{N} P(w_i|w_1, ...w_{i-1})$$

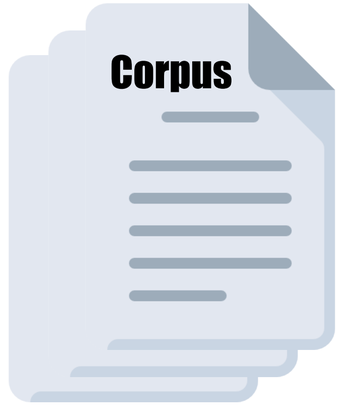


P(“its water is so transparent”) =
P(its) × P(water|its) × P(is|its water)

Probabilistic Language Models

$$P(w_1, ..., w_N) = P(w_1)P(w_2|w_1)...P(w_n|w_1, ..., w_{N-1})$$

$$= \prod_{i=1}^{N} P(w_i|w_1, ...w_{i-1})$$



P(“its water is so transparent”) =
P(its) × P(water|its) × P(is|its water) × P(so|its water is) × P(transparent|its water is so)

Probabilistic Language Models

$$P(w_1, ..., w_N) = P(w_1)P(w_2|w_1)...P(w_n|w_1, ..., w_{N-1})$$

$$= \prod_{i=1}^{N} P(w_i|w_1, ...w_{i-1})$$

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

Probabilistic Language Models

$$P(w_1, ..., w_N) = P(w_1)P(w_2|w_1)...P(w_n|w_1, ..., w_{N-1})$$

$$= \prod_{i=1}^{N} P(w_i|w_1, ...w_{i-1})$$

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

Probabilistic Language Models

The **joint probability** of a sentence like “its water is so transparent that” means we want to know how likely it is for this *exact sequence of words* to appear together in that specific order. To compute we need to:

1. Count how many times this exact sentence ("its water is so transparent that") appears in a large text collection (corpus).

2. Divide that number by the total number of possible five-word sequences in the corpus.

The problem is that this approach requires a *huge* amount of data because there are so many possible five-word combinations. That’s why it’s not practical, and we use simpler methods such as **N-GRAMS.**

Probabilistic Language Models

1. Count how many times this exact sentence ("its water is so transparent that") appears in a large text collection (corpus).

2. Divide that number by the total number of possible five-word sequences in the corpus.

Probabilistic Language Models

1. Count how many times this exact sentence ("its water is so transparent that") appears in a large text collection (corpus).

2. Divide that number by the total number of possible five-word sequences in the corpus.

Probabilistic Language Models

1. Count how many times this exact sentence ("its water is so transparent that") appears in a large text collection (corpus).

2. Divide that number by the total number of possible five-word sequences in the corpus.

Probabilistic Language Models

1. Cour hat")
appe

2. Divi s in the
corp

k-permutations of n elements.

$P(n, k)$ is the number of ***k-permutations of*** n ***elements***, the number of ways to *arrange* k objects chosen from n distinct objects.

$$P(n,k) = \frac{n!}{(n-k)!} = n(n-1)(n-2)\cdots(n-(k-1)).$$

https://discrete.openmathbooks.org/dmoi3/sec_counting-combperm.html

Probabilistic Language Models

VOCABULARY SIZE = 1K unique words

Count all the combinations of 5 words

Probabilistic Language Models

VOCABULARY SIZE = 1K unique words

Count all the combinations of 5 words

$$C(n,r)=?$$

$$C(n,r)=C(1000,5)$$

$$=$$

$$\frac{1000!}{(5!(1000-5)!)}$$

$$=$$

$$\frac{1000!}{5!\times 995!}$$

$$=8.25029125E+12$$

Probabilistic Language Models

Markov Assumption

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

Probabilistic Language Models

Markov Assumption

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-\text{N}} \ldots w_{i-1})$$

N = Context or Window

Probabilistic Language Models

Markov Assumption

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

UNIGRAM

Probabilistic Language Models

Markov Assumption

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

BIGRAM

Probabilistic Language Models

Markov Assumption

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

BIGRAM

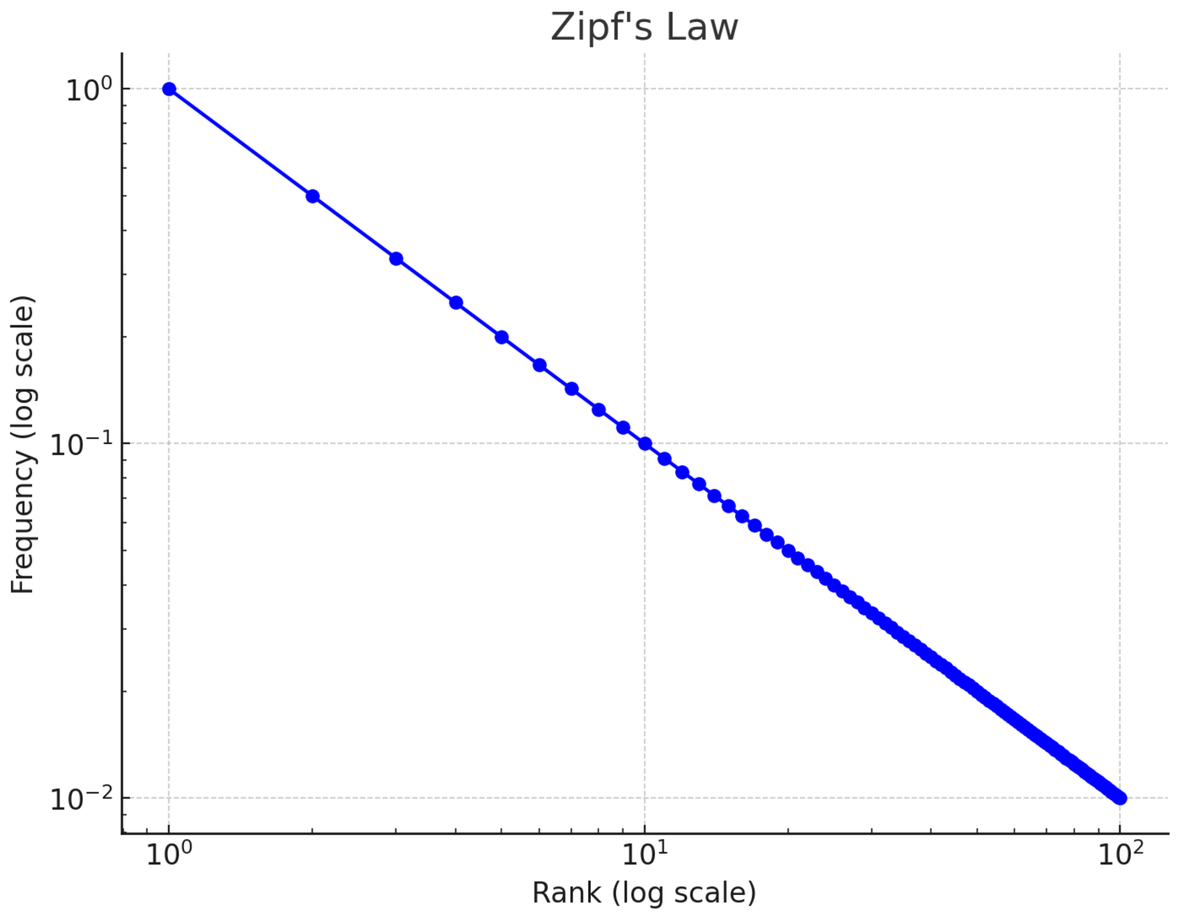
We can extend to 3-grams, 4-grams, 5-grams

Probabilistic Language Models

Zipf’s law: the long tail

- A **small** number words occur with **high frequency**

- A **large** number of words occur with **very low frequency**

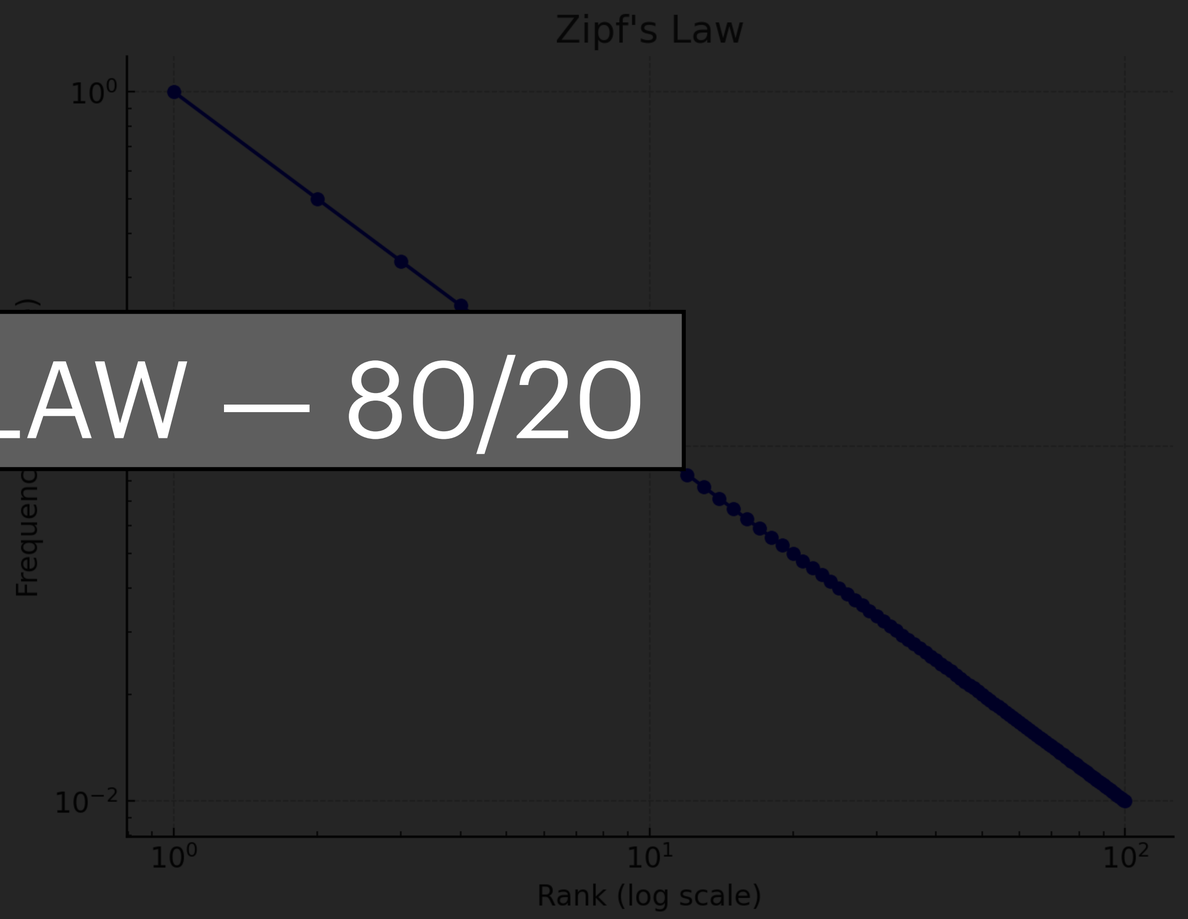


Probabilistic Language Models

Zipf’s law: the long tail

- A **small** number of words occur with **high frequency**

- A **large** number of words occur with **very low frequency**



PARETO’S LAW — 80/20

Probabilistic Language Models

Even for source code...

On the Naturalness of Software

Abram Hindle, Earl T. Barr, Zhendong Su
Dept. of Computer Science
University of California at Davis
Davis, CA 95616 USA
{ajhindle,barr,su}@cs.ucdavis.edu

Mark Gabel
Dept. of Computer Science
The University of Texas at Dallas
Richardson, TX 75080 USA
mark.gabel@utdallas.edu

Premkumar Devanbu
Dept. of Computer Science
University of California at Davis
Davis, CA 95616 USA
devanbu@cs.ucdavis.edu

***Abstract*—Natural languages like English are rich, complex, and powerful. The highly creative and graceful use of languages like English and Tamil, by masters like Shakespeare and Avvaiyar, can certainly delight and inspire. But in practice, given cognitive constraints and the exigencies of daily life, most human utterances are far simpler and much more repetitive and predictable. In fact, these utterances can be very usefully modeled using modern statistical methods. This fact has led to the phenomenal success of statistical approaches to speech recognition, natural language translation, question-answering, and text mining and comprehension.**

We begin with the conjecture that most software is also natural, in the sense that it is created by humans at work, with all the attendant constraints and limitations—and thus

efforts in the 1960s. In the ’70s and ’80s, the field was re-animated with ideas from logic and formal semantics, which still proved too cumbersome to perform practical tasks at scale. Both these approaches essentially dealt with NLP from first principles—addressing *language*, in all its rich theoretical glory, rather than examining corpora of actual *utterances*, *i.e.*, what people actually write or say. In the 1980s, a fundamental shift to *corpus-based, statistically rigorous* methods occurred. The availability of large, on-line corpora of natural language text, including “aligned” text with translations in multiple languages,[1] along with the computational muscle (CPU speed,

Probabilistic Language Models

Even for source code...

On the Naturalness of Software

"Programming languages, in theory, are complex, flexible and powerful, but the programs that real people actually write are mostly simple and rather repetitive, and thus they have usefully predictable statistical properties that can be captured in statistical language models and leveraged for software engineering tasks."

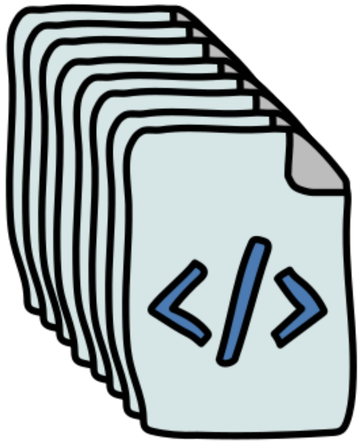
and powerful. The highly creative and graceful use of languages like English and Tamil, by masters like Shakespeare and Avvaiyar, can certainly delight and inspire. But in practice, given cognitive constraints and the exigencies of daily life, most human utterances are far simpler and much more repetitive and predictable. In fact, these utterances can be very usefully modeled using modern statistical methods. This fact has led to the phenomenal success of statistical approaches to speech recognition, natural language translation, question-answering, and text mining and comprehension.

We begin with the conjecture that most software is also natural, in the sense that it is created by humans at work,

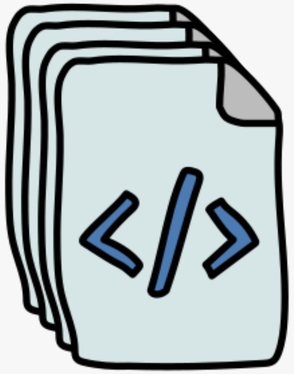
animated with ideas from logic and formal semantics, which still proved too cumbersome to perform practical tasks at scale. Both these approaches essentially dealt with NLP from first principles—addressing *language*, in all its rich theoretical glory, rather than examining corpora of actual *utterances*, *i.e.*, what people actually write or say. In the 1980s, a fundamental shift to *corpus-based, statistically rigorous* methods occurred. The availability of large, on-line corpora of natural language text, including "aligned" text with translations in multiple languages,[1] along with the computational muscle (CPU speed,

Probabilistic Language Models

1,500

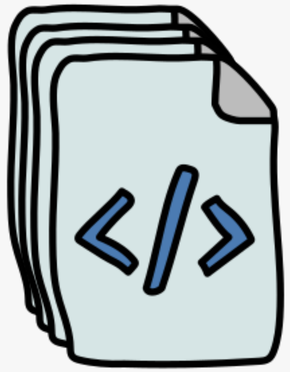


Probabilistic Language Models



1300

Training (80%)

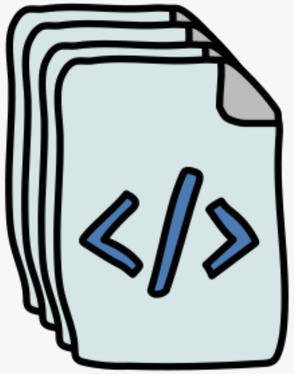


300

Testing (20%)

Probabilistic Language Models

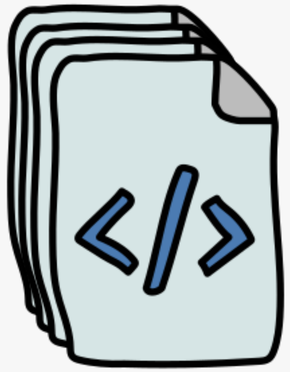
Training Corpus



1,200

Training (70%)

Held-Out Corpus

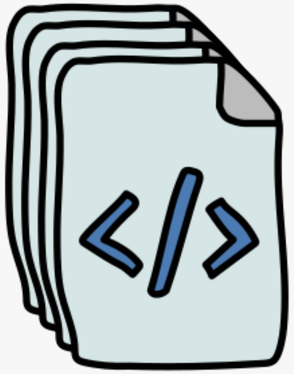


300

Testing (20%)

Probabilistic Language Models

Training Corpus

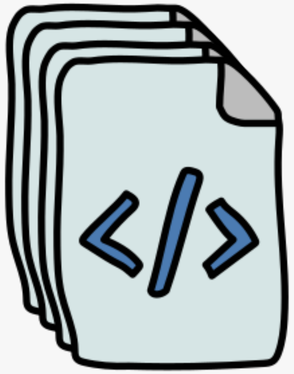


BIGRAM

N=2 $$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

Probabilistic Language Models

Training Corpus



BIGRAM

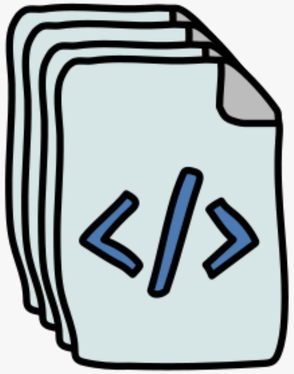
N=2 $P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$

```
public static Inet4Address getEmbeddedIPv4CA
                    (Inet6Address ip) {
      ip1 = New IP("127.0.0.1")
      if (isCompatIPv4Address(ip)) {
```

	public	static	Inet4Ad..
public	0	1800	1
static	0	10	0
Inet4Ad..	...	15	0
getEmb..	...	...	15

Probabilistic Language Models

Training Corpus



BIGRAM

N=2 $P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$

```
public static Inet4Address getEmbeddedIPv4CA
                    (Inet6Address ip) {
     ip1 = New IP("127.0.0.1")
     if (isCompatIPv4Address(ip)) {
```

	public	static	Inet4Ad..
public	0	1800	1
static	0	10	0
Inet4Ad..	...	15	0
getEmb..	...	...	15

Probabilistic Language Models

```
public static Inet4Address getEmbeddedIPv4CA
                    (Inet6Address ip) {
      ip1 = New IP("127.0.0.1")
      if (isCompatIPv4Address(ip)) {
```

	public	static	Inet4Ad..
public	0	1800	1
static	0	10	0
Inet4Ad..	...	15	0
getEmb..	...	...	15

P(getEmbeddedIPv4CA|Inet4Address) =

Probabilistic Language Models

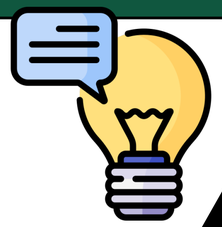
```
public static Inet4Address getEmbeddedIPv4CA
                    (Inet6Address ip) {
      ip1 = New IP("127.0.0.1")
      if (isCompatIPv4Address(ip)) {
```

	public	static	Inet4Ad..
public	0	1800	1
static	0	10	0
Inet4Ad..	...	15	0
getEmb..	...	...	15

P(getEmbeddedIPv4CA|Inet4Address) =
Count(Inet4Address,getEmbeddedIPv4CA) /
Count(Inet4Address) = 15 / 300 = 0.05

Probabilistic Language Models

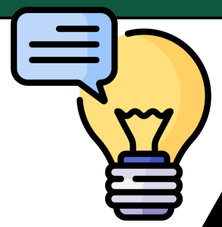
PROS



- No need for a specialized hardware

Probabilistic Language Models

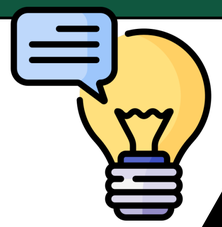
PROS



- No need for a specialized hardware
- CPU-based computation

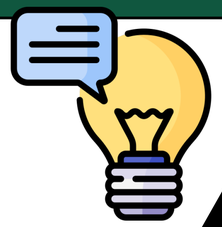
Probabilistic Language Models

PROS



- No need for a specialized hardware
- CPU-based computation
- Easy to implement

Probabilistic Language Models



PROS

- No need for a specialized hardware
- CPU-based computation
- Easy to implement

CONS

- Long distance relationship cannot be captured properly

Probabilistic Language Models

```
public static Inet4Address getEmbeddedIPv4ClientAddress
                  (Inet6Address ip) {
      ip1 = New IP(“127.0.0.1”)
      if (isCompatIPv4Address(ip)) {
          return getCompatIPv4Address(ip);
      }if (is6to4Address(ip)) {
          return get6to4IPv4Address(ip);
      }
      if (isTeredoAddress(ip)) {
          return getTeredoInfo(ip).getClient();
      }
      return ip1.getDefault();
}
```

Probabilistic Language Models

```
public static Inet4Address getEmbeddedIPv4ClientAddress
                  (Inet6Address ip) {
      ip1 = New IP("127.0.0.1")
      if (isCompatIPv4Address(ip)) {
          return getCompatIPv4Address(ip);
      }if (is6to4Address(ip)) {
          return get6to4IPv4Address(ip);
      }
      if (isTeredoAddress(ip)) {
          return getTeredoInfo(ip).getClient();
      }
      return ip1.getDefault();
}
```

NNs can easily handle longer context/windows

Probabilistic Language Models

Model Evaluation: How good is our model?

We test the model's performance on data we haven't seen.

-A test set is an unseen dataset that is different from our training set, totally unused.

An evaluation metric tells us how well our model does on the test set.

Probabilistic Language Models

Model Evaluation: Extrinsic Evaluation

Best evaluation for comparing models A and B

Test each model on a given a task ***e.g., code completion***

Run the task, get an accuracy for A and for B

How many (mis)-predicted tokens?

How many tokens have been correctly predicted?

Probabilistic Language Models

Model Evaluation: Extrinsic Evaluation

Best evaluation for comparing models A and B

Test each model on a given a task ***e.g., code generation***

Run the task, get an accuracy for A and for B

How many (mis)-predicted tokens?

How many tokens have been correctly predicted?

Time-consuming
Can take weeks

Probabilistic Language Models

Model Evaluation: Intrinsic Evaluation

We use a metric that measures the quality of a model independent of any application

Probabilistic Language Models

Model Evaluation: Intrinsic Evaluation

We use a metric that measures the quality of a model independent of any application

Perplexity

Probabilistic Language Models

Model Evaluation: Intrinsic Evaluation

We use a metric that measures the quality of a model independent of any application

Perplexity

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

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

Probabilistic Language Models

Model Evaluation: Intrinsic Evaluation

We use a metric that measures the quality of a model independent of any application

Perplexity

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

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

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

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

Probabilistic Language Models

Model Evaluation: Intrinsic Evaluation

We use a metric that measures the quality of a model independent of any application

Perplexity

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

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

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=}^{N}}$$

$$\mathrm{PP}(W) = \sqrt[N]{}$$

Probabilistic Language Models

Perplexity can be thought of as a measure of the confidence of the model when predicting the next word. **A lower perplexity** score indicates that the model is more confident in its predictions and better at capturing the patterns in the data. If model A has perplexity of 20 and model B 5, **the number of choices the model B has to make at each step in predicting the next word is considerably less**

Probabilistic Language Models

Model Evaluation: Perplexity and zero prob. n-grams

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

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

N-grams with 0 prob are not allowed, otherwise by construction we reach an absurd dividing by 0

Probabilistic Language Models

Smoothing and Interpolation Techniques

Laplace Smoothing

Simple as add 1 to all n-gram counts

$$P(w_i) = \frac{C(w_i)}{\sum_j C(w_j)} = \frac{C(w_i)}{N} \qquad P(w_i) = \frac{C(w_i)+\mathbf{1}}{\sum_j (C(w_j)+\mathbf{1})} = \frac{C(w_i)+\mathbf{1}}{N+\mathbf{V}}$$

Probabilistic Language Models

Smoothing and Interpolation Techniques

Add-k Smoothing

$$P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV}$$

Probabilistic Language Models

Smoothing and Interpolation Techniques

Add-k Smoothing

$$P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV}$$

k is an hyper-parameter that has to be optimized

Probabilistic Language Models

Smoothing and Interpolation Techniques

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n) \\ &\quad + \lambda_2 P(w_n|w_{n-1}) \\ &\quad + \lambda_3 P(w_n|w_{n-2}w_{n-1})\end{aligned}$$

Probabilistic Language Models

Smoothing and Interpolation Techniques

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n|w_{n-2}w_{n-1})$$

λ values that maximize the likelihood of the held-out corpus. In other words, we fix the n-gram probabilities and then search for the λ values that, when applied, yield the highest probability for the held-out dataset.

Probabilistic Language Models

Stupid Backoff

Probabilistic Language Models

Stupid Backoff

- In an N-gram backoff model, if the n-gram we aim to compute is 0, what we can do is approximate this computation, by **“backing off”** to the (n-1)-gram

Probabilistic Language Models

Stupid Backoff

- In an N-gram backoff model, if the n-gram we aim to compute is 0, what we can do is approximate this computation, by **“backing off”** to the (n-1)-gram

- Because we are **backing off** if we do not adjust the higher orders n-gram, our n-gram model will not be following *strictly speaking* a probability distribution

Probabilistic Language Models

Stupid Backoff

- In an N-gram backoff model, if the n-gram we aim to compute is 0, what we can do is approximate this computation, by **"backing off"** to the (n-1)-gram

$$S(w_i|w_{i-N+1:i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-N+1:i})}{\text{count}(w_{i-N+1:i-1})} & \text{if count}(w_{i-N+1:i}) > 0 \\ \lambda S(w_i|w_{i-N+2:i-1}) & \text{otherwise} \end{cases}$$

Large Language Models in Machine Translation

Thorsten Brants **Ashok C. Popat** **Peng Xu** **Franz J. Och** **Jeffrey Dean**

Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94303, USA
{brants,popat,xp,och,jeff}@google.com

Probabilistic Language Models

Visualizing Knowledge Through Sampling

- In statistic, **sampling** from a distribution means to choose random points according to their likelihood

Probabilistic Language Models

Visualizing Knowledge Through Sampling

- In statistic, **sampling** from a distribution means to choose random points according to their likelihood

- When **sampling** from a language model, we generate sentences based on their likelihood as determined by the model's probability distribution.

Probabilistic Language Models

Visualizing Knowledge Through Sampling

- In statistic, **sampling** from a distribution means to choose random points according to their likelihood

- When **sampling** from a language model, we generate sentences based on their likelihood as determined by the model's probability distribution.

- We are more likely to generate sentences that the model thinks have a high probability and less likely to generate sentences that the model thinks have a low probability.

Probabilistic Language Models

Large Language Models and N-Grams

Large Language Models (LLMs) are built on neural networks rather than n-grams, allowing them to address two significant issues associated with n-grams

Probabilistic Language Models

Large Language Models and N-Grams

Large Language Models (LLMs) are built on neural networks rather than n-grams, allowing them to address two significant issues associated with n-grams

1) the number of parameters grows exponentially as the n-gram order increases

Probabilistic Language Models

Large Language Models and N-Grams

Large Language Models (LLMs) are built on neural networks rather than n-grams, allowing them to address two significant issues associated with n-grams

1) the number of parameters grows exponentially as the n-gram order increases
2) n-grams cannot generalize from training examples to test examples unless the exact same words are used

Probabilistic Language Models

Language Models (LM) present an easy way to assign probabilities to words and sequence of words/tokens

N-gram models are LMs that make use of the Markov's property to estimate words probabilities from a fixed window

N-gram models can be evaluated via Perplexity (Intrinsic Evaluation) or by using an Extrinsic Evaluation

Probabilistic Language Models

0 count N-grams can be addressed using smoothing and interpolation

N-gram models are only as effective as the training corpus, leading to limited generalization capability.

Large Language Models, are LM that rely on Deep Learning models to learn the distribution of sequence of words

Probabilistic Language Models

https://dl.acm.org/doi/pdf/10.1145/2902362

research highlights

DOI:10.1145/2902362

On the Naturalness of Software

By Abram Hindle, Earl T. Barr, Mark Gabel, Zhendong Su, and Premkumar Devanbu

Hindle et al

Abstract

Natural languages like English are rich, complex, and powerful. The highly creative and graceful use of languages like English and Tamil, by masters like Shakespeare and Avvaiyar, can certainly delight and inspire. But in practice, given cognitive constraints and the exigencies of daily life, most human utterances are far simpler and much more repetitive and predictable. In fact, these utterances can be very usefully modeled using modern statistical methods. This fact has led to the phenomenal success of statistical approaches to speech recognition, natural language translation, question-answering, and text mining and comprehension.

We begin with the conjecture that *most software is also natural*, in the sense that it is created by humans at work, with all the attendant constraints and limitations—and thus, like natural language, it is also likely to be repetitive and predictable. We then proceed to ask whether (a) code can be usefully modeled by statistical language models and (b) such models can be leveraged to support software engineers. Using the widely adopted *n*-gram model, we provide empirical evidence supportive of a positive answer to both these questions. We show that code is also very regular, and, in fact, even more so than natural languages. As an example use of the model, we have developed a simple code completion engine for Java that, despite its simplicity, already improves Eclipse's completion capability. We conclude the paper by laying out a vision for future research in this area.

1. INTRODUCTION

Of all that we do, *communicating* is the one action that is too cumbersome to perform practical tasks at scale. Both these approaches essentially dealt with NLP from first principles—addressing *language*, in all its rich theoretical glory, rather than examining corpora of actual *utterances*, that is, what people actually write or say. In the 1980s, a fundamental shift to *corpus-based*, *statistically rigorous* methods occurred. The availability of large, on-line corpora of natural language text, including "aligned" text with translations in multiple languages,[a] along with the computational muscle (CPU speed, primary and secondary storage) to estimate robust statistical models over very large data sets has led to stunning progress and widely available practical applications, such as statistical translation used by translate.google.com.[b]

Can we apply these techniques *directly* to software, with its strange syntax, awash with punctuation, and replicate this success? The funny thing about programming is that it is as much *an act of communication*, from one human to another, as it is a way to tell computers what to do. Knuth said as much, 30 years ago:

> *Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.*[23]

If one, then, were to view programming as an act of communication, is it driven by the "language instinct"? Do we program as we speak? Is our code largely simple, repetitive, and predictable? *Is code natural?*

This, precisely, is our central hypothesis:

> *Programming languages, in theory, are complex, flexible and powerful, but, "natural" programs, the ones that real people*

Probabilistic Language Models

Recommending code tokens via N-gram models