BILL
RECORD THIS LECTURE

September 16, 2020

Revisit GCD and Math Notation

September 16, 2020

Revisit GCD Briefly

Two things about GCD I want to clarify.

- Why is $\mathrm{GCD}(x, 0) = x$ for $x \geq 1$?
- When does the algorithm stop?

GCD(404,192): I Now Supply Last Step

$$404 = 2 \times 192 + 20$$

GCD(404,192): I Now Supply Last Step

$$404 = 2 \times 192 + 20$$
$$192 = 9 \times 20 + 12$$

GCD(404,192): I Now Supply Last Step

$$404 = 2 \times 192 + 20$$
$$192 = 9 \times 20 + 12$$
$$20 = 1 \times 12 + 8$$

GCD(404,192): I Now Supply Last Step

$$404 = 2 \times 192 + 20$$
$$192 = 9 \times 20 + 12$$
$$20 = 1 \times 12 + 8$$
$$12 = 1 \times 8 + 4$$

GCD(404,192): I Now Supply Last Step

$404 = 2 \times 192 + 20$
$192 = 9 \times 20 + 12$
$20 = 1 \times 12 + 8$
$12 = 1 \times 8 + 4$
$8 = 4 \times 2 + \mathbf{0}$ STOP WHEN GET 0. Go back one: 4 is GCD.

GCD(404,192): I Now Supply Last Step

$404 = 2 \times 192 + 20$
$192 = 9 \times 20 + 12$
$20 = 1 \times 12 + 8$
$12 = 1 \times 8 + 4$
$8 = 4 \times 2 + \mathbf{0}$ STOP WHEN GET 0. Go back one: 4 is GCD.

Lets look at what the algorithm actually does:

GCD(404,192): I Now Supply Last Step

$404 = 2 \times 192 + 20$
$192 = 9 \times 20 + 12$
$20 = 1 \times 12 + 8$
$12 = 1 \times 8 + 4$
$8 = 4 \times 2 + \mathbf{0}$ STOP WHEN GET 0. Go back one: 4 is GCD.

Lets look at what the algorithm actually does:
$\mathrm{GCD}(404, 192) = \mathrm{GCD}(404 - 2 \times 192, 192) = \mathrm{GCD}(20, 192) =$

GCD(404,192): I Now Supply Last Step

$404 = 2 \times 192 + 20$
$192 = 9 \times 20 + 12$
$20 = 1 \times 12 + 8$
$12 = 1 \times 8 + 4$
$8 = 4 \times 2 + \mathbf{0}$ STOP WHEN GET 0. Go back one: 4 is GCD.

Lets look at what the algorithm actually does:
$\mathrm{GCD}(404, 192) = \mathrm{GCD}(404 - 2 \times 192, 192) = \mathrm{GCD}(20, 192) =$
$\mathrm{GCD}(20, 192 - 9 \times 20) = \mathrm{GCD}(20, 12) = \mathrm{GCD}(20 - 1 \times 12, 12) =$

GCD(404,192): I Now Supply Last Step

$404 = 2 \times 192 + 20$
$192 = 9 \times 20 + 12$
$20 = 1 \times 12 + 8$
$12 = 1 \times 8 + 4$
$8 = 4 \times 2 + \mathbf{0}$ STOP WHEN GET 0. Go back one: 4 is GCD.

Lets look at what the algorithm actually does:
$\mathrm{GCD}(404, 192) = \mathrm{GCD}(404 - 2 \times 192, 192) = \mathrm{GCD}(20, 192) =$
$\mathrm{GCD}(20, 192 - 9 \times 20) = \mathrm{GCD}(20, 12) = \mathrm{GCD}(20 - 1 \times 12, 12) =$
$\mathrm{GCD}(8, 12) = \mathrm{GCD}(8, 12 - 8) = \mathrm{GCD}(8, 4) =$

GCD(404,192): I Now Supply Last Step

$404 = 2 \times 192 + 20$
$192 = 9 \times 20 + 12$
$20 = 1 \times 12 + 8$
$12 = 1 \times 8 + 4$
$8 = 4 \times 2 + \mathbf{0}$ STOP WHEN GET 0. Go back one: 4 is GCD.

Lets look at what the algorithm actually does:
$\mathrm{GCD}(404, 192) = \mathrm{GCD}(404 - 2 \times 192, 192) = \mathrm{GCD}(20, 192) =$
$\mathrm{GCD}(20, 192 - 9 \times 20) = \mathrm{GCD}(20, 12) = \mathrm{GCD}(20 - 1 \times 12, 12) =$
$\mathrm{GCD}(8, 12) = \mathrm{GCD}(8, 12 - 8) = \mathrm{GCD}(8, 4) =$
$\mathrm{GCD}(8 - 2 \times 4, 4) = \mathrm{GCD}(0, 4)$

GCD(404,192): I Now Supply Last Step

$404 = 2 \times 192 + 20$
$192 = 9 \times 20 + 12$
$20 = 1 \times 12 + 8$
$12 = 1 \times 8 + 4$
$8 = 4 \times 2 + \mathbf{0}$ STOP WHEN GET 0. Go back one: 4 is GCD.

Lets look at what the algorithm actually does:
$\mathrm{GCD}(404, 192) = \mathrm{GCD}(404 - 2 \times 192, 192) = \mathrm{GCD}(20, 192) =$
$\mathrm{GCD}(20, 192 - 9 \times 20) = \mathrm{GCD}(20, 12) = \mathrm{GCD}(20 - 1 \times 12, 12) =$
$\mathrm{GCD}(8, 12) = \mathrm{GCD}(8, 12 - 8) = \mathrm{GCD}(8, 4) =$
$\mathrm{GCD}(8 - 2 \times 4, 4) = \mathrm{GCD}(0, 4)$

To make our formula $\mathrm{GCD}(x, y) = \mathrm{GCD}(x - ky, x)$ work all the way to 0, we **define** $\mathrm{GCD}(0, x) = x$.

Why is $5^{1/2} = \sqrt{5}$?

Why is

$$5^{1/2} = \sqrt{5}?$$

Are we multiplying a number by itself half a time?

Why is $5^{1/2} = \sqrt{5}$?

Why is

$$5^{1/2} = \sqrt{5}?$$

Are we multiplying a number by itself half a time? Discuss.

Why is $5^{1/2} = \sqrt{5}$?

Why is

$$5^{1/2} = \sqrt{5}?$$

Are we multiplying a number by itself half a time? Discuss.**No**.

Why is $5^{1/2} = \sqrt{5}$?

Why is

$$5^{1/2} = \sqrt{5}?$$

Are we multiplying a number by itself half a time? Discuss.**No**.

For $a, b \in \mathbb{N}$ we have

$$5^a \times 5^b = 5^{a+b}.$$

Why is $5^{1/2} = \sqrt{5}$?

Why is

$$5^{1/2} = \sqrt{5}?$$

Are we multiplying a number by itself half a time? Discuss.**No**.

For $a, b \in \mathbb{N}$ we have

$$5^a \times 5^b = 5^{a+b}.$$

We want this rule to still apply when $a, b \in \mathbb{Q}$.

Why is $5^{1/2} = \sqrt{5}$?

Why is

$$5^{1/2} = \sqrt{5}?$$

Are we multiplying a number by itself half a time? Discuss.**No**.

For $a, b \in \mathbb{N}$ we have

$$5^a \times 5^b = 5^{a+b}.$$

We want this rule to still apply when $a, b \in \mathbb{Q}$. So we want

$$5^{1/2} \times 5^{1/2} = 5^{1/2+1/2} = 5$$

Why is $5^{1/2} = \sqrt{5}$?

Why is

$$5^{1/2} = \sqrt{5}?$$

Are we multiplying a number by itself half a time? Discuss.**No**.

For $a, b \in \mathbb{N}$ we have

$$5^a \times 5^b = 5^{a+b}.$$

We want this rule to still apply when $a, b \in \mathbb{Q}$. So we want

$$5^{1/2} \times 5^{1/2} = 5^{1/2+1/2} = 5$$

Hence we **define** $5^{1/2} = \sqrt{5}$ to make that rule work out.

Why is $5^{1/2} = \sqrt{5}$?

Why is

$$5^{1/2} = \sqrt{5}?$$

Are we multiplying a number by itself half a time? Discuss.**No**.

For $a, b \in \mathbb{N}$ we have

$$5^a \times 5^b = 5^{a+b}.$$

We want this rule to still apply when $a, b \in \mathbb{Q}$. So we want

$$5^{1/2} \times 5^{1/2} = 5^{1/2+1/2} = 5$$

Hence we **define** $5^{1/2} = \sqrt{5}$ to make that rule work out.

Similar for 5^0 and 5^{-a}.

Why is $5^{1/2} = \sqrt{5}$?

Why is

$$5^{1/2} = \sqrt{5}?$$

Are we multiplying a number by itself half a time? Discuss.**No**.

For $a, b \in \mathbb{N}$ we have

$$5^a \times 5^b = 5^{a+b}.$$

We want this rule to still apply when $a, b \in \mathbb{Q}$. So we want

$$5^{1/2} \times 5^{1/2} = 5^{1/2+1/2} = 5$$

Hence we **define** $5^{1/2} = \sqrt{5}$ to make that rule work out.

Similar for 5^0 and 5^{-a}.

How is 5^π defined?

Why is $5^{1/2} = \sqrt{5}$?

Why is

$$5^{1/2} = \sqrt{5}?$$

Are we multiplying a number by itself half a time? Discuss.**No**.

For $a, b \in \mathbb{N}$ we have

$$5^a \times 5^b = 5^{a+b}.$$

We want this rule to still apply when $a, b \in \mathbb{Q}$. So we want

$$5^{1/2} \times 5^{1/2} = 5^{1/2+1/2} = 5$$

Hence we **define** $5^{1/2} = \sqrt{5}$ to make that rule work out.

Similar for 5^0 and 5^{-a}.

How is 5^π defined? Discuss.

What is 5^π?

We want

$$5^{3.14159} < 5^{\pi} < 5^{3.141593}.$$

What is 5^π?

We want

$$5^{3.14159} < 5^\pi < 5^{3.141593}.$$

We can replace with approximations to π that are lower and that are higher.

What is 5^π?

We want

$$5^{3.14159} < 5^\pi < 5^{3.141593}.$$

We can replace with approximations to π that are lower and that are higher.

So, with this in mind, how do we define 5^π?

What is 5^π?

We want

$$5^{3.14159} < 5^\pi < 5^{3.141593}.$$

We can replace with approximations to π that are lower and that are higher.

So, with this in mind, how do we define 5^π?

Let $\alpha_1, \alpha_2, \ldots,$ be an infinite sequence of rationals that cvg to π.

What is 5^π?

We want

$$5^{3.14159} < 5^{\pi} < 5^{3.141593}.$$

We can replace with approximations to π that are lower and that are higher.

So, with this in mind, how do we define 5^π?

Let $\alpha_1, \alpha_2, \ldots,$ be an infinite sequence of rationals that cvg to π. 5^π is **defined** to be $\lim_{i\to\infty} 5^{\alpha_i}$.

What is 5^{π}?

We want

$$5^{3.14159} < 5^{\pi} < 5^{3.141593}.$$

We can replace with approximations to π that are lower and that are higher.

So, with this in mind, how do we define 5^{π}?

Let $\alpha_1, \alpha_2, \ldots,$ be an infinite sequence of rationals that cvg to π.
5^{π} is **defined** to be $\lim_{i \to \infty} 5^{\alpha_i}$.
Need to prove that all choices of sequences yield the same result.
We won't do that here

Upshot

Sometimes functions are defined on certain values **not** because its the most natural way to do it, but because it makes prior rules work out.

Upshot

Sometimes functions are defined on certain values **not** because its the most natural way to do it, but because it makes prior rules work out.

This is the case for

Upshot

Sometimes functions are defined on certain values **not** because its the most natural way to do it, but because it makes prior rules work out.

This is the case for

- $\mathrm{GCD}(x, 0) = x$.

Upshot

Sometimes functions are defined on certain values **not** because its the most natural way to do it, but because it makes prior rules work out.

This is the case for

- $\mathrm{GCD}(x, 0) = x$.
- $5^{1/2} = \sqrt{5}$.

Upshot

Sometimes functions are defined on certain values **not** because its the most natural way to do it, but because it makes prior rules work out.

This is the case for

- $\mathrm{GCD}(x, 0) = x$.
- $5^{1/2} = \sqrt{5}$.
- $\frac{1}{2}! = \sqrt{\pi}$.

Upshot

Sometimes functions are defined on certain values **not** because its the most natural way to do it, but because it makes prior rules work out.

This is the case for

- $\mathrm{GCD}(x, 0) = x$.
- $5^{1/2} = \sqrt{5}$.
- $\frac{1}{2}! = \sqrt{\pi}$. Don't ask me why.

Upshot

Sometimes functions are defined on certain values **not** because its the most natural way to do it, but because it makes prior rules work out.

This is the case for

- $\mathrm{GCD}(x, 0) = x$.
- $5^{1/2} = \sqrt{5}$.
- $\frac{1}{2}! = \sqrt{\pi}$. Don't ask me why. The answer **it's the Γ function** is

Upshot

Sometimes functions are defined on certain values **not** because its the most natural way to do it, but because it makes prior rules work out.

This is the case for

- $\mathrm{GCD}(x, 0) = x$.
- $5^{1/2} = \sqrt{5}$.
- $\frac{1}{2}! = \sqrt{\pi}$. Don't ask me why. The answer **it's the Γ function** is (a) true, and

Upshot

Sometimes functions are defined on certain values **not** because its the most natural way to do it, but because it makes prior rules work out.

This is the case for

- $\mathrm{GCD}(x, 0) = x$.
- $5^{1/2} = \sqrt{5}$.
- $\frac{1}{2}! = \sqrt{\pi}$. Don't ask me why. The answer **it's the Γ function** is (a) true, and (b) truly UNenlightening.

Gen Sub Cipher: How to Really Crack

September 16, 2020

General Substitution Cipher

Def **Gen Sub Cipher** with perm f on $\{0, \ldots, 25\}$.

1. Encrypt via $x \to f(x)$.
2. Decrypt via $x \to f^{-1}(x)$.

Terminology: 1-Gram, 2-Gram, 3-Gram

Notation Let T be a text.

Terminology: 1-Gram, 2-Gram, 3-Gram

Notation Let T be a text.

1. The **1-grams** of T are just the letters in T, counting repeats.

Terminology: 1-Gram, 2-Gram, 3-Gram

Notation Let T be a text.

1. The **1-grams** of T are just the letters in T, counting repeats.
2. The **2-grams** of T are just the contiguous pairs of letters in T, counting repeats. Also called **bigrams**.

Terminology: 1-Gram, 2-Gram, 3-Gram

Notation Let T be a text.

1. The **1-grams** of T are just the letters in T, counting repeats.
2. The **2-grams** of T are just the contiguous pairs of letters in T, counting repeats. Also called **bigrams**.
3. The **3-grams** of T you can guess. Also called **trigrams**.

Terminology: 1-Gram, 2-Gram, 3-Gram

Notation Let T be a text.

1. The **1-grams** of T are just the letters in T, counting repeats.
2. The **2-grams** of T are just the contiguous pairs of letters in T, counting repeats. Also called **bigrams**.
3. The **3-grams** of T you can guess. Also called **trigrams**.
4. One usually talks about the freq of n-grams.

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 1-grams occur 1 time: a,d,u,v,y.

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 1-grams occur 1 time: a,d,u,v,y.

The following 1-grams occur 2 times: h,l,n,p,w.

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 1-grams occur 1 time: a,d,u,v,y.

The following 1-grams occur 2 times: h,l,n,p,w.

The following 1-grams occur 3 times: c,i,m.

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 1-grams occur 1 time: a,d,u,v,y.

The following 1-grams occur 2 times: h,l,n,p,w.

The following 1-grams occur 3 times: c,i,m.

The following 1-grams occur 4 times: r,s,t.

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 1-grams occur 1 time: a,d,u,v,y.

The following 1-grams occur 2 times: h,l,n,p,w.

The following 1-grams occur 3 times: c,i,m.

The following 1-grams occur 4 times: r,s,t.

The following 1-gram occurs 6 times: o.

Example of 1-Grams

Let the text be:
Ever notice how sometimes people use math words incorrectly?

The following 1-grams occur 1 time: a,d,u,v,y.

The following 1-grams occur 2 times: h,l,n,p,w.

The following 1-grams occur 3 times: c,i,m.

The following 1-grams occur 4 times: r,s,t.

The following 1-gram occurs 6 times: o.

The following 1-gram occurs 9 times: e.

Example of 2-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

Example of 2-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 2-grams occur 2 times: me, or.

Example of 2-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 2-grams occur 2 times: me, or.

The following 2-grams occur 1 time: ev, ve, er, rn, no, ot, ti, ic, eh, ho, ow, ws, so, et, ti, im, es, sp, pe, eo, op, pl, le, eu, us, se, em, ma, at, th, hw, wo, ds, in, nc, co, rr, re, ec, ct, tl, ly.

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

1. f_E is freq of n-grams. It is a 26^n long vector. (Formally we should use $f_E(n)$. We omit the n. The value of n will be clear from context.)

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

1. f_E is freq of n-grams. It is a 26^n long vector. (Formally we should use $f_E(n)$. We omit the n. The value of n will be clear from context.)
2. $\sigma(T)$ is taking T and applying σ to it. If σ^{-1} was used to encrypt, then $\sigma(T)$ will be English!

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

1. f_E is freq of n-grams. It is a 26^n long vector. (Formally we should use $f_E(n)$. We omit the n. The value of n will be clear from context.)
2. $\sigma(T)$ is taking T and applying σ to it. If σ^{-1} was used to encrypt, then $\sigma(T)$ will be English!
3. $f_{\sigma(T)}$ is the 26^n-long vector of freq's of n-grams in $\sigma(T)$.

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

1. f_E is freq of n-grams. It is a 26^n long vector. (Formally we should use $f_E(n)$. We omit the n. The value of n will be clear from context.)
2. $\sigma(T)$ is taking T and applying σ to it. If σ^{-1} was used to encrypt, then $\sigma(T)$ will be English!
3. $f_{\sigma(T)}$ is the 26^n-long vector of freq's of n-grams in $\sigma(T)$.
4. I and R will be parameters we discuss later.

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

1. f_E is freq of n-grams. It is a 26^n long vector. (Formally we should use $f_E(n)$. We omit the n. The value of n will be clear from context.)
2. $\sigma(T)$ is taking T and applying σ to it. If σ^{-1} was used to encrypt, then $\sigma(T)$ will be English!
3. $f_{\sigma(T)}$ is the 26^n-long vector of freq's of n-grams in $\sigma(T)$.
4. I and R will be parameters we discuss later.
   I stands for $\mathrm{Iterations}$ and will be large (like 2000).

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

1. f_E is freq of n-grams. It is a 26^n long vector. (Formally we should use $f_E(n)$. We omit the n. The value of n will be clear from context.)
2. $\sigma(T)$ is taking T and applying σ to it. If σ^{-1} was used to encrypt, then $\sigma(T)$ will be English!
3. $f_{\sigma(T)}$ is the 26^n-long vector of freq's of n-grams in $\sigma(T)$.
4. I and R will be parameters we discuss later.
   I stands for $\mathrm{Iterations}$ and will be large (like 2000).
   R stands for Redos and will be small (like 5).

Stats for 1-Gram, 2-Gram, 3-Gram, 4-Gram

Stats for 1-Gram, 2-Gram, 3-Gram, 4-Gram

1. 1-grams: $f_E \cdot f_E \sim 0.065$.

Stats for 1-Gram, 2-Gram, 3-Gram, 4-Gram

1. 1-grams: $f_E \cdot f_E \sim 0.065$.
2. 2-grams: $f_E \cdot f_E \sim 0.0067$.

Stats for 1-Gram, 2-Gram, 3-Gram, 4-Gram

1. 1-grams: $f_E \cdot f_E \sim 0.065$.
2. 2-grams: $f_E \cdot f_E \sim 0.0067$.
3. 3-grams: $f_E \cdot f_E \sim 0.0011$.

Stats for 1-Gram, 2-Gram, 3-Gram, 4-Gram

1. 1-grams: $f_E \cdot f_E \sim 0.065$.
2. 2-grams: $f_E \cdot f_E \sim 0.0067$.
3. 3-grams: $f_E \cdot f_E \sim 0.0011$.
4. 4-grams: $f_E \cdot f_E \sim 0.00023$.

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T)} \cdot f_E$ is large then σ is correct shift. Large ~ 0.065.

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T)} \cdot f_E$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T)} \cdot f_E$ is small then σ is incorrect shift. Small ~ 0.035.

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T)} \cdot f_E$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T)} \cdot f_E$ is small then σ is incorrect shift. Small ~ 0.035.
3. **Important** Will always be large or small. So we have **a gap**.

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T)} \cdot f_E$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T)} \cdot f_E$ is small then σ is incorrect shift. Small ~ 0.035.
3. **Important** Will always be large or small. So we have **a gap**.

Lets try this with gen sub, ignoring the issue of 26! perms.
To crack gen sub shift went through all 26! perm σ:

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T)} \cdot f_E$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T)} \cdot f_E$ is small then σ is incorrect shift. Small ~ 0.035.
3. **Important** Will always be large or small. So we have **a gap**.

Lets try this with gen sub, ignoring the issue of 26! perms.
To crack gen sub shift went through all 26! perm σ:

1. If $f_{\sigma(T)} \cdot f_E$ is large then σ is correct perm. Large ~ 0.065.

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T)} \cdot f_E$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T)} \cdot f_E$ is small then σ is incorrect shift. Small ~ 0.035.
3. **Important** Will always be large or small. So we have **a gap**.

Lets try this with gen sub, ignoring the issue of 26! perms.
To crack gen sub shift went through all 26! perm σ:

1. If $f_{\sigma(T)} \cdot f_E$ is large then σ is correct perm. Large ~ 0.065.
2. If $f_{\sigma(T)} \cdot f_E$ is small then σ is incorrect perm. Small.

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T)} \cdot f_E$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T)} \cdot f_E$ is small then σ is incorrect shift. Small ~ 0.035.
3. **Important** Will always be large or small. So we have **a gap**.

Lets try this with gen sub, ignoring the issue of 26! perms.
To crack gen sub shift went through all 26! perm σ:

1. If $f_{\sigma(T)} \cdot f_E$ is large then σ is correct perm. Large ~ 0.065.
2. If $f_{\sigma(T)} \cdot f_E$ is small then σ is incorrect perm. Small. Hmmm?

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T)} \cdot f_E$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T)} \cdot f_E$ is small then σ is incorrect shift. Small ~ 0.035.
3. **Important** Will always be large or small. So we have **a gap**.

Lets try this with gen sub, ignoring the issue of 26! perms.
To crack gen sub shift went through all 26! perm σ:

1. If $f_{\sigma(T)} \cdot f_E$ is large then σ is correct perm. Large ~ 0.065.
2. If $f_{\sigma(T)} \cdot f_E$ is small then σ is incorrect perm. Small. Hmmm?
3. We have a problem. If σ only changed a few letters around, then likely $f_E \cdot f_{\sigma(T)}$ will be large. We **do not** have a gap!

What to do?

What to do if there is no Gap?

What to do if there is no Gap?

1. Use n-grams instead of 1-grams. This does not close the Gap but will help anyway.

What to do if there is no Gap?

1. Use n-grams instead of 1-grams. This does not close the Gap but will help anyway.
2. Rather than view the **Is-English** program as a YES-NO, view it as comparative:

   T_1 looks **more like English** than T_2.

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
$\sigma_{\rm init}$ is perm that maps most freq to e, etc. Uses 1-gram freq.

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
$\sigma_{\rm init}$ is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to R (R is small, about 5)

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
σ_{init} is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\text{init}}$

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
$\sigma_{\rm init}$ is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\rm init}$
For $i = 1$ to I (I is large, about 2000)

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
$\sigma_{\rm init}$ is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to R (R is small, about 5)

> $\sigma_r \leftarrow \sigma_{\rm init}$
> For $i = 1$ to I (I is large, about 2000)
>
> > Pick $j, k \in \{0, \ldots, 25\}$ at Random.

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
$\sigma_{\rm init}$ is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to $\rm R$ ($\rm R$ is small, about 5)

> $\sigma_r \leftarrow \sigma_{\rm init}$
> For $i = 1$ to $\rm I$ ($\rm I$ is large, about 2000)
>
> > Pick $j, k \in \{0, \ldots, 25\}$ at Random.
> > Let σ' be σ_r with j, k swapped

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
$\sigma_{\rm init}$ is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\rm init}$
For $i = 1$ to I (I is large, about 2000)
Pick $j, k \in \{0, \ldots, 25\}$ at Random.
Let σ' be σ_r with j, k swapped
If $f_{\sigma'(T)} \cdot f_E > f_{\sigma_r(T)} \cdot f_E$ then $\sigma_r \leftarrow \sigma'$

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
σ_{init} is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\mathrm{init}}$
For $i = 1$ to I (I is large, about 2000)
Pick $j, k \in \{0, \ldots, 25\}$ at Random.
Let σ' be σ_r with j, k swapped
If $f_{\sigma'(T)} \cdot f_E > f_{\sigma_r(T)} \cdot f_E$ then $\sigma_r \leftarrow \sigma'$

Candidates for σ are $\sigma_1, \ldots, \sigma_{\mathrm{R}}$

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
$\sigma_{\rm init}$ is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\rm init}$
For $i = 1$ to I (I is large, about 2000)
Pick $j, k \in \{0, \ldots, 25\}$ at Random.
Let σ' be σ_r with j, k swapped
If $f_{\sigma'(T)} \cdot f_E > f_{\sigma_r(T)} \cdot f_E$ then $\sigma_r \leftarrow \sigma'$

Candidates for σ are $\sigma_1, \ldots, \sigma_{\mathrm{R}}$
Pick the σ_r with min good_r or have human look at all $\sigma_r(T)$

Finding Parameters: A Chicken-and-Egg Problem

An old question:
What came first, the chicken or the egg?

Finding Parameters: A Chicken-and-Egg Problem

An old question:
What came first, the chicken or the egg?

Our Problem We need parameters I and R so the answer looks like English. But we then need a notion of **Is English** that does not use a gap. Need a program to tell us that it looks like English.

Finding Parameters: A Chicken-and-Egg Problem

An old question:
What came first, the chicken or the egg?

Our Problem We need parameters I and R so the answer looks like English. But we then need a notion of **Is English** that does not use a gap. Need a program to tell us that it looks like English.

We Trebekked It!

Finding Parameters: A Chicken-and-Egg Problem

An old question:
What came first, the chicken or the egg?

Our Problem We need parameters I and R so the answer looks like English. But we then need a notion of **Is English** that does not use a gap. Need a program to tell us that it looks like English.

We Trebekked It!
On the TV show JEOPARDY Alex Trebek gives you the **answer** and you have to figure out the **question**.

Finding Parameters: A Chicken-and-Egg Problem

An old question:
What came first, the chicken or the egg?

Our Problem We need parameters I and R so the answer looks like English. But we then need a notion of **Is English** that does not use a gap. Need a program to tell us that it looks like English.

We Trebekked It!
On the TV show JEOPARDY Alex Trebek gives you the **answer** and you have to figure out the **question**.

Same here.

Finding Parameters: A Chicken-and-Egg Problem

An old question:
What came first, the chicken or the egg?

Our Problem We need parameters I and R so the answer looks like English. But we then need a notion of **Is English** that does not use a gap. Need a program to tell us that it looks like English.

We Trebekked It!
On the TV show JEOPARDY Alex Trebek gives you the **answer** and you have to figure out the **question**.

Same here.

We find the parameters for texts where we know the answers.

Finding the Parameters

Do the following a large number of times:

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10,000$ characters.

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10,000$ characters.
2. Take a random perm σ.

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10,000$ characters.
2. Take a random perm σ.
3. Compute $\sigma(T)$. (Note- We know σ and T)

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10,000$ characters.
2. Take a random perm σ.
3. Compute $\sigma(T)$. (Note- We know σ and T)
4. Run the n-gram algorithm but with no bound on the number of iterations. Stop when either

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10,000$ characters.
2. Take a random perm σ.
3. Compute $\sigma(T)$. (Note- We know σ and T)
4. Run the n-gram algorithm but with no bound on the number of iterations. Stop when either
   4.1 Get original text T, or

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10{,}000$ characters.
2. Take a random perm σ.
3. Compute $\sigma(T)$. (Note- We know σ and T)
4. Run the *n*-gram algorithm but with no bound on the number of iterations. Stop when either
   - 4.1 Get original text T, or
   - 4.2 Swaps do not improve how close to English (could be in local min). In this case try again.

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10,000$ characters.
2. Take a random perm σ.
3. Compute $\sigma(T)$. (Note- We know σ and T)
4. Run the n-gram algorithm but with no bound on the number of iterations. Stop when either
    4.1 Get original text T, or
    4.2 Swaps do not improve how close to English (could be in local min). In this case try again.
5. Keep track of how how many iterations suffice and how many redos suffice.

David Zhen Found the Parameters

UMCP ugrad CS major David Zhen worked with me on this over the summer.

David Zhen Found the Parameters

UMCP ugrad CS major David Zhen worked with me on this over the summer.

The next three slides show the parameters he found.

David Zhen Found the Parameters

UMCP ugrad CS major David Zhen worked with me on this over the summer.

The next three slides show the parameters he found.

He used a Mac-Book Pro with 2.2 Ghz 6-core Intel Core i7 processor and 16 GB of RAM.

David Zhen Found the Parameters

UMCP ugrad CS major David Zhen worked with me on this over the summer.

The next three slides show the parameters he found.

He used a Mac-Book Pro with 2.2 Ghz 6-core Intel Core i7 processor and 16 GB of RAM.

In English: a normal computer that an ugrad can buy and use.

David Zhen Found the Parameters

UMCP ugrad CS major David Zhen worked with me on this over the summer.

The next three slides show the parameters he found.

He used a Mac-Book Pro with 2.2 Ghz 6-core Intel Core i7 processor and 16 GB of RAM.

In English: a normal computer that an ugrad can buy and use.

He ran the program to find parameters on 150 texts of size approx 10,000 characters:

David Zhen Found the Parameters

UMCP ugrad CS major David Zhen worked with me on this over the summer.

The next three slides show the parameters he found.

He used a Mac-Book Pro with 2.2 Ghz 6-core Intel Core i7 processor and 16 GB of RAM.

In English: a normal computer that an ugrad can buy and use.

He ran the program to find parameters on 150 texts of size approx 10,000 characters:

For each text he generated 1 random perm (will rerun with more later).

Parameters for 1-Grams

Parameters for 1-Grams

Nothing worked.

Parameters for 2-Grams

Nothing worked.

Parameters for 3-Grams

Parameters for 3-Grams

1. The average time to get within 1-2 swaps was 1 minute.

Parameters for 3-Grams

1. The average time to get within 1-2 swaps was 1 minute.
2. The min time was 50 seconds, the max time was 3.5 minutes.

Parameters for 3-Grams

1. The average time to get within 1-2 swaps was 1 minute.
2. The min time was 50 seconds, the max time was 3.5 minutes.
3. Seems hard to get those 1-2 swaps right.

Parameters for 3-Grams

1. The average time to get within 1-2 swaps was 1 minute.
2. The min time was 50 seconds, the max time was 3.5 minutes.
3. Seems hard to get those 1-2 swaps right.
4. The average number of iterations was 900. The MAX number of iterations was 1902. TAKE $I = 2000$.

Parameters for 3-Grams

1. The average time to get within 1-2 swaps was 1 minute.
2. The min time was 50 seconds, the max time was 3.5 minutes.
3. Seems hard to get those 1-2 swaps right.
4. The average number of iterations was 900. The MAX number of iterations was 1902. TAKE $\mathrm{I} = 2000$.
5. The average number of redos the program needed to get within 2 swaps was 1.14. The max number of times was 3. TAKE $\mathrm{R} = 4$.

4-Grams

4-Grams

1. The average time to get it perfect was 6 minutes.

4-Grams

1. The average time to get it perfect was 6 minutes.
2. The min time was 4 minutes, the max time was 30 minutes.

4-Grams

1. The average time to get it perfect was 6 minutes.
2. The min time was 4 minutes, the max time was 30 minutes.
3. The average number of iterations was 1000. The MAX number of iterations was 1966. TAKE $\mathrm{I} = 2000$.

4-Grams

1. The average time to get it perfect was 6 minutes.
2. The min time was 4 minutes, the max time was 30 minutes.
3. The average number of iterations was 1000. The MAX number of iterations was 1966. TAKE $\mathrm{I} = 2000$.
4. The average number of REDOS to get it perfects was 1.3. The max number of times was 7. TAKE $\mathrm{R} = 8$

History of this Approach

History of this Approach

1. There was a paper that claimed to be able to use ML to crack Gen Sub cipher.

History of this Approach

1. There was a paper that claimed to be able to use ML to crack Gen Sub cipher.
2. In Summer 2020 I had ugrad David Zhen look at the paper and code it up. (There were students on the project.)

History of this Approach

1. There was a paper that claimed to be able to use ML to crack Gen Sub cipher.
2. In Summer 2020 I had ugrad David Zhen look at the paper and code it up. (There were students on the project.)
3. When he finished and explained it to me I said

   you can do all of that without ML

   and sketched an algorithm.

History of this Approach

1. There was a paper that claimed to be able to use ML to crack Gen Sub cipher.
2. In Summer 2020 I had ugrad David Zhen look at the paper and code it up. (There were students on the project.)
3. When he finished and explained it to me I said

   you can do all of that without ML

   and sketched an algorithm.
4. David coded up my algorithm and it did not work. My mistake: I thought we needed parameters for gaps.

History of this Approach

1. There was a paper that claimed to be able to use ML to crack Gen Sub cipher.
2. In Summer 2020 I had ugrad David Zhen look at the paper and code it up. (There were students on the project.)
3. When he finished and explained it to me I said

   you can do all of that without ML

   and sketched an algorithm.
4. David coded up my algorithm and it did not work. My mistake: I thought we needed parameters for gaps.
5. David came up with the current algorithm.

History of this Approach

1. There was a paper that claimed to be able to use ML to crack Gen Sub cipher.
2. In Summer 2020 I had ugrad David Zhen look at the paper and code it up. (There were students on the project.)
3. When he finished and explained it to me I said

   you can do all of that without ML

   and sketched an algorithm.
4. David coded up my algorithm and it did not work. My mistake: I thought we needed parameters for gaps.
5. David came up with the current algorithm.
6. Zan found that our algorithm was already known, which did not surprise me. We discuss this on the next slide.

History of this Approach

1. There was a paper that claimed to be able to use ML to crack Gen Sub cipher.
2. In Summer 2020 I had ugrad David Zhen look at the paper and code it up. (There were students on the project.)
3. When he finished and explained it to me I said

   you can do all of that without ML

   and sketched an algorithm.
4. David coded up my algorithm and it did not work. My mistake: I thought we needed parameters for gaps.
5. David came up with the current algorithm.
6. Zan found that our algorithm was already known, which did not surprise me. We discuss this on the next slide.
7. Does ML really help crypto? Not sure.

Our Algorithm Already Known

A Fast Method for Cryptanalysis of Substitution Ciphers by Jakobsen, (1995)
has our approach with the following caveats:

Our Algorithm Already Known

A Fast Method for Cryptanalysis of Substitution Ciphers by Jakobsen, (1995)

has our approach with the following caveats:

1. They use a different IS-ENGLISH function. A better one as we will see.

Our Algorithm Already Known

A Fast Method for Cryptanalysis of Substitution Ciphers by Jakobsen, (1995)
has our approach with the following caveats:

1. They use a different IS-ENGLISH function. A better one as we will see.
2. They use bigrams rather than trigrams.

Our Algorithm Already Known

A Fast Method for Cryptanalysis of Substitution Ciphers by Jakobsen, (1995)
has our approach with the following caveats:

1. They use a different IS-ENGLISH function. A better one as we will see.
2. They use bigrams rather than trigrams.
3. Since they can use bigrams rather than trigrams (I assume) their algorithm is faster.

Our Algorithm Already Known

A Fast Method for Cryptanalysis of Substitution Ciphers by Jakobsen, (1995)

has our approach with the following caveats:

1. They use a different IS-ENGLISH function. A better one as we will see.
2. They use bigrams rather than trigrams.
3. Since they can use bigrams rather than trigrams (I assume) their algorithm is faster.
4. So why did I present ours? (1) Educationally mine and theirs are the same, and (2) I knew all of the parameters of our algorithm and how we got them.

Mostly There But . . .

Since the text was in blocks of five and we want to totally mechanize, need a method to find word breaks.

Mostly There But . . .

Since the text was in blocks of five and we want to totally mechanize, need a method to find word breaks.

We leave this topic for now.

BILL
STOP RECORDING
THIS LECTURE

September 16, 2020