Week 9 - Wednesday

COMP 1800

Last time

- What did we talk about last time?
- Cryptanalysis
- Cracking the rail fence cipher

Questions?

Assignment 7

Cryptanalysis of Substitution Ciphers

Simple monoalphabetic substitution cipher

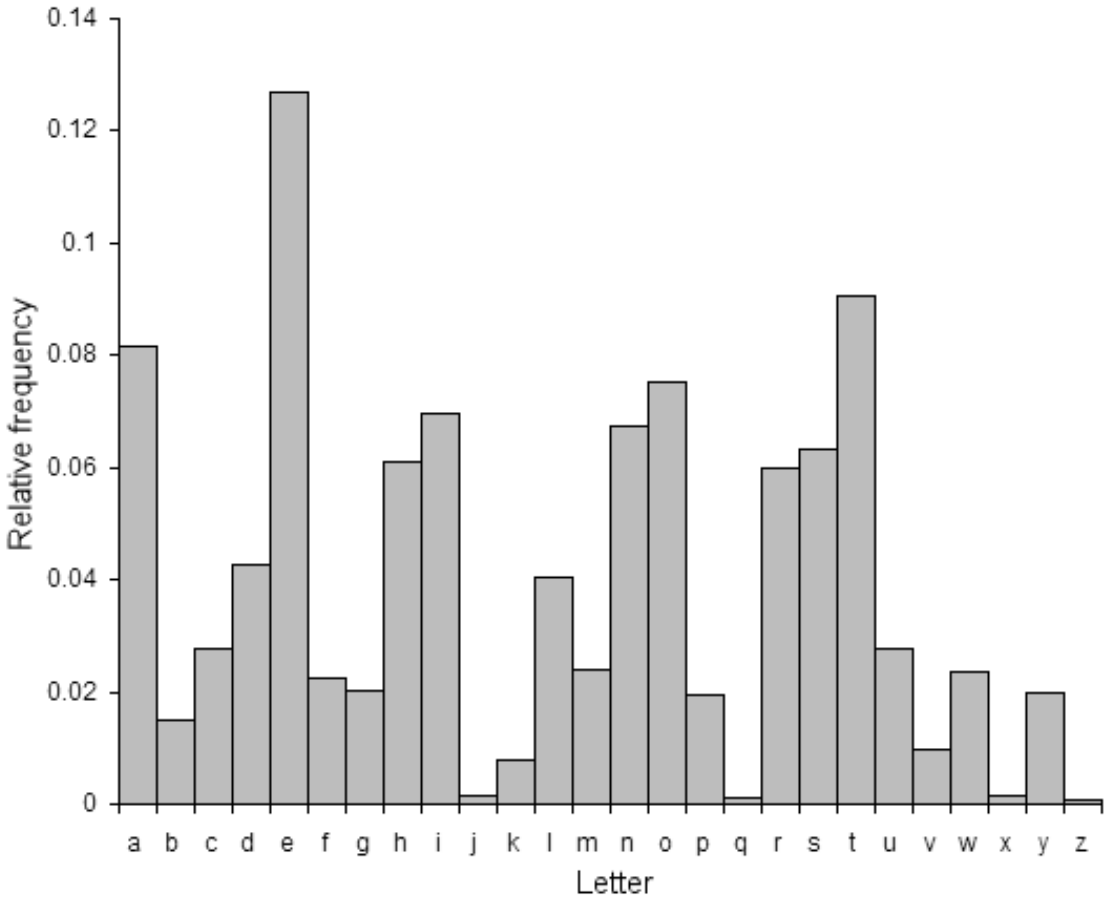
- We can map to a random permutation of letters
- For example:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
I	N	O	V	Z	H	A	P	T	R	G	E	U	F	D	W	S	B	Q	Y	L	K	M	J	C	X

- E("MATH IS GREAT") = "UIYP TQ ABZIY"
- 26! possible permutations
- Hard to check every one

Frequency attack

- English language defeats us
- Some letters are used more frequently than others: ETAOINSHRDLU
- Longer texts will behave more consistently
- Make a histogram, break the cipher



Cipher text

SPHB JLSP K ECGPCQFT GYBKYD, VFCMB C LSPGBYBG, VBKX KPG VBKYD,
SOBY EKPD K RJKCPT KPG HJYCSJU OSMJEB SZ ZSYQSTTBP MSYB -
VFCMB C PSGGBG, PBKYMD PKLLCPQ, UJGGBPMD TFBYB HKEB K TKLLCPQ,
KU SZ USEB SPB QBPTMD YKLLCPQ, YKLLCPQ KT ED HFKEIBY GSSY.
"'TCU USEB OCUCTSY," C EJTTBYBG, "TKLLCPQ KT ED HFKEIBY GSSY -
SPMD TFCU KPG PSTFCPQ ESYB."

KF, GCUTCPHTMD CYBEBEIBY CT VKU CP TFB IMBKX GBHBEIBY;
KPG BKHF UBLKYKTB GDCPQ BEIBY VYSJQFT CTU QFSUT JLSP TFB ZMSSY.
BKQBYMD C VCUFBG TFB ESYYSV; - OKCPMD C FKG USJQFT TS ISYYSV
ZYSE ED ISSXU UJYHBKUB SZ USYYSV - USYYSV ZSY TFB MSUT MBPSYB -
ZSY TFB YKYB KPG YKGCKPT EKCGBP VFSE TFB KPQBMU PKEB MBPSYB -
PKEBMBUU FBYB ZSY BOBYESYB.

KPG TFB UCMXBP, UKG, JPHBYTKCP YJUTMCPQ SZ BKHF LJYLMB HJYTKCP
TFYCMMBG EB - ZCMMBG EB VCTF ZKPTKUTCH TBYYSYU PBOBY ZBMT IBZSYB;
US TFKT PSV, TS UTCMM TFB IBKTCPQ SZ ED FBKYT, C UTSSG YBLBKTCPQ
"'TCU USEB OCUCTBY BPTYBKTCPQ BPTYKPHB KT ED HFKEIBY GSSY -
USEB MKTB OCUCTBY BPTYBKTCPQ BPTYKPHB KT ED HFKEIBY GSSY; -
TFCU CT CU KPG PSTFCPQ ESYB."

Moving toward plain text

SNPE YMSN A LIDNIUHO DTEATF, WHICE I MSNDETED, WEAK AND WEATF,
SVET LANF A XYAINO AND PYTISYR VSCYLE SG GSTUSOOEN CSTE -
WHICE I NSDDED, NEATCF NAMMINU, RYDDENCF OHETE PALE A OAMMINU,
AR SG RSLE SNE UENOCF TAMMINU, TAMMINU AO LF PHALBET DSST.
"'OIR RSLE VIRIOST," I LYOOETED, "OAMMINU AO LF PHALBET DSST -
SNCF OHIR AND NSOHINU LSTE."

AH, DIROINPOCF I TELELBET IO WAR IN OHE BCEAK DEPELBET;
AND EAPH REMATAOE DFINU ELBET WTSYUHO IOR UHSRO YMSN OHE GCSST.
EAUETCF I WIRHED OHE LSTTSW; - VAINCF I HAD RSYUHO OS BSTTSW
GTSL LF BSSKR RYTPEARE SG RSTTSW - RSTTSW GST OHE CSRO CENSTE -
GST OHE TATE AND TADIANO LAIDEN WHSL OHE ANUECR NALE CENSTE -
NALECERR HETE GST EVETLSTE.

AND OHE RICKEN, RAD, YNPETOAIN TYROCINU SG EAPH MYTMCE PYTOAIN
OHTICCED LE - GICCED LE WIOH GANOAROIP OETTSTR NEVET GECO BEGSTE;
RS OHAO NSW, OS ROICC OHE BEAOINU SG LF HEATO, I ROSSD TEMEAOINU
"'OIR RSLE VIRIOET ENOTEAOINU ENOTANPE AO LF PHALBET DSST -
RSLE CAOE VIRIOET ENOTEAOINU ENOTANPE AO LF PHALBET DSST; -
OHIR IO IR AND NSOHINU LSTE."

Real plain text

ONCE UPON A MIDNIGHT DREARY, WHILE I PONDERED, WEAK AND WEARY,
OVER MANY A QUAINT AND CURIOUS VOLUME OF FORGOTTEN LORE -
WHILE I NODDED, NEARLY NAPPING, SUDDENLY THERE CAME A TAPPING,
AS OF SOME ONE GENTLY RAPPING, RAPPING AT MY CHAMBER DOOR.
"'TIS SOME VISITOR," I MUTTERED, "TAPPING AT MY CHAMBER DOOR -
ONLY THIS AND NOTHING MORE."

AH, DISTINCTLY I REMEMBER IT WAS IN THE BLEAK DECEMBER;
AND EACH SEPARATE DYING EMBER WROUGHT ITS GHOST UPON THE FLOOR.
EAGERLY I WISHED THE MORROW; - VAINLY I HAD SOUGHT TO BORROW
FROM MY BOOKS SURCEASE OF SORROW - SORROW FOR THE LOST LENORE -
FOR THE RARE AND RADIANT MAIDEN WHOM THE ANGELS NAME LENORE -
NAMELESS HERE FOR EVERMORE.

AND THE SILKEN, SAD, UNCERTAIN RUSTLING OF EACH PURPLE CURTAIN
THRILLED ME - FILLED ME WITH FANTASTIC TERRORS NEVER FELT BEFORE;
SO THAT NOW, TO STILL THE BEATING OF MY HEART, I STOOD REPEATING
"'TIS SOME VISITER ENTREATING ENTRANCE AT MY CHAMBER DOOR -
SOME LATE VISITER ENTREATING ENTRANCE AT MY CHAMBER DOOR; -
THIS IT IS AND NOTHING MORE."

Digram analysis

- These kinds of attacks can be further refined by analyzing digrams and trigrams (two letter and three letter sequences)
- Digram analysis is also an approach that can be used against transposition ciphers, since you can gain clues about which letters should be next to which others

Digrams	Trigrams
EN	ENT
RE	ION
ER	AND
NT	ING
TH	IVE
ON	TIO
IN	FOR
TF	OUR
AN	THI
OR	ONE

Implementing a substitution cipher

- Before we can try to crack a substitution cipher, we need a way to encrypt using a substitution cipher
- We also need a key
- A few weeks ago, we used these two functions to make a key:

```
def removeChar(string, index):
    return string[:index] + string[index + 1:]
```

```
def makeKey():
    alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    key = ''
    for i in range(len(alphabet)):
        index = random.randint(0, len(alphabet) - 1)
        key = key + alphabet[index]
        alphabet = removeChar(alphabet, index)
    return key
```

Encrypting with a substitution cipher

- Then, we used this function to encrypt a message with the given key:

```
def substitutionEncrypt(plaintext, key):
    ciphertext = ''
    for letter in plaintext:
        value = ord(letter)
        value = value - ord('A')
        value = key[value]
        ciphertext = ciphertext + value
    return ciphertext
```

- Note that it only works when all characters in the message are uppercase letters

Frequencies

- The first step in cracking a message is getting the frequencies of all the letters
- We're going to use a dictionary to record the number of times each letter appears in a file
- If we count how many total letters there are, we can go back and divide these counts to figure out the frequencies

Getting the frequencies

```
def getFrequencies(filename):
```

- Open the file
- Read everything in the file into a single string and convert it to uppercase
- Make a string containing all the letters of the alphabet
- Create a dictionary, and loop over all the letters of the alphabet, setting the value in the dictionary for each one to zero
- Loop over all the letters in the text
  - If it's in the alphabet,
    - Add one to the count for that letter in the dictionary
    - Add one to the total number of letters
- Loop over all the letters in the alphabet
  - Divide the count for that letter in the dictionary by the total
- Return the dictionary

Tuples

- **Tuples** in Python are like lists, except that you can't change them
- You can still access the items in them with square brackets and an index number
- Instead of using square brackets `[]` to say what's in a tuple, you use parentheses `()`

```
things = (4, 'wombat', 2.9)
print(things[0]) # prints 4
print(things[1]) # prints wombat
print(things[2]) # prints 2.9
```

Why are we talking about tuples?

- If you want to return multiple things from a function, you can always return a tuple
- Also, when you iterate over the items in a dictionary, each one is a tuple

```
sounds = {1 : 'Do', 2 : 'Re', 3 : 'Mi'}
for pair in sounds.items():
    print(pair)
# prints:
# (1, 'Do')
# (2, 'Re')
# (3, 'Mi')
```

The `list()` function

- If you want to turn an iterable thing into a list, you can use the **`list()`** function

```
numbers = list(range(5)) # contains [0, 1, 2, 3, 4]
```

- You can also use the **`list()`** function to convert the items in a dictionary into a list of tuples

```
sounds = {1 : 'Do', 2 : 'Re', 3 : 'Mi'}
pairs = list(sounds.items())
# contains [(1, 'Do'), (2, 'Re') (3, 'Mi')]
```

Using `list()` with letter frequencies

- Why did we talk all about tuples and **`list()`**?
- We can use **`list()`** to turn the dictionary we made into a list of letters and their frequencies

```
dictionary = getFrequencies('gettysburg.txt')
frequencies = list(dictionary.items())
```

- If we could only sort this list, we'd have a list from the most common letters down to the least

Sorting a list in an arbitrary way

- If you have a list (called, say, **`things`**), you can sort it with the sort function:

```
things.sort()
```

- But that only works if the items in things are items that Python knows how to sort, like strings or numbers
- If you want to sort arbitrary items, you have to pass in a function that says how you want them sorted, using a special named argument called **`key`**

```
things.sort(key=howToSort)
```

Sorting tuples

- In our case, we have a list of tuples that look like this:

  `('A', 0.08162203832186278)`
- We want to sort by the second thing, the frequency
- We can write a simple function that gives the second thing (which has index 1) in a tuple

```
def second(pair):
    return pair[1]
```

Putting it all together

- With custom sorting, we can get the frequencies from a file and sort them as follows:

```
dictionary = getFrequencies('gettysburg.txt')
frequencies = list(dictionary.items())
frequencies.sort(key=second)
```

- Unfortunately, doing so sorts the frequencies from least frequent to most frequent
- To fix it, we have to pass True to another special named argument called reverse:

```
dictionary = getFrequencies('gettysburg.txt')
frequencies = list(dictionary.items())
frequencies.sort(key=second, reverse=True)
```

Final substitution cracking function

```
def crackSubstitution(cipherFile, sampleFile):
```

- Sort the letters by frequency from the ciphertext file
- Sort the letters by frequency from the sample file
- Make a dictionary
- Loop through all the letters sorted by frequency from the ciphertext
  - Store the corresponding letter (sorted in the same order) from the sample file into the dictionary
- Open the ciphertext file again
- Loop over all the letters in the ciphertext
  - If it's an uppercase letter
    - Look up its corresponding letter from the dictionary and put it into the output string
- Return the output string

Some observations

- In most cases, this approach won't work right away
- Some of the letters will probably be right
- Many will probably be off by a few places because they were more or less common than the sample file
- As we mentioned earlier, tricks using multiple letter sequences are needed to improve the output

Upcoming

Next time…

- Work time for Assignment 7

Reminders

- Work on Assignment 7