Homework 2

Author: Benjamin Heindl

Course: IST 664

Date: November 6, 2023

This program was adapted from the Stanford NLP class SpamLord homework assignment.

The code has been rewritten and the data modified, nevertheless please do not make this code or the data public.

This version has two patterns that were suggested in comments in order to get you started .

In [39]:
```python
import sys
import os
import re
import pprint
from io import open
```

For Part 1 of our assignment, add to these two lists of patterns to match examples of obscured email addresses and phone numbers in the text.

For optional Part 3, you may need to add other lists of patterns.

In [40]:
```python
# patterns1: Creating a list of regex patterns for matching various obfuscated email formats

patterns1 = [
    # Matching email addresses ending with '.EDU'. Capturing username and domain
    # Example: user@domain.EDU
    '([A-Za-z.]+)@([A-Za-z.]+)\\.EDU',

    # Similar to the first pattern, but case-insensitive to '.edu'
    # Example: user@domain.edu.
    '([A-Za-z.]+)@([A-Za-z.]+)\\.edu',

    # Matching email addresses with spaces around the '@' symbol
    # Example: user @ domain.edu
    '([A-Za-z.]+)\\s@\\s([A-Za-z.]+)\\.edu',

    # Matching emails obfuscated with HTML character references for '@'
    # Example: user &#64; domain.edu
    '([A-Za-z0-9._]+)\\s*&#[A-Za-z0-9._]+;\\s*([A-Za-z0-9._]+)\\.edu',

    # Matching emails where 'AT' is used instead of '@', and 'DOT' instead of '.', and can end with
    # different variations of 'edu' or 'com', in both lowercase and uppercase
    # Example: user AT domain DOT edu
    '([A-Za-z0-9._]+)\\s*AT\\s*([A-Za-z0-9._]+)\\s*DOT\\s*[edu|com|COM|EDU]',

    # Matching emails where 'WHERE' is used for '@' and 'DOM' for '.edu'
    # Example: user WHERE domain DOM edu
    '([A-Za-z0-9._]+)\\s*WHERE\\s*([A-Za-z0-9._]+)\\s*DOM\\s*edu',

    # Matching a specific obfuscation function call that contains an email address
    # Example: obfuscate('domain.edu','user')
    "obfuscate\\('([A-Za-z0-9]+)\\.edu','([A-Za-z0-9]+)'\\)"
]
```

In [41]:
```python
# patterns2: Creating a list of regex patterns for matching various obfuscated email formats with more variations
patterns2 = [
    # Matching email with spaces around the '@' symbol, possibly to avoid simple email scrapers
    '([A-Za-z.]+)  @  ([A-Za-z.]+)\\.edu',

    # Matching emails where 'AT' replaces '@' and 'DOT' replaces '.', with spaces around them
    '([A-Za-z.]+) AT ([A-Za-z.]+) DOT edu',

    # Matching emails with the words 'WHERE' and 'DOM' in place of '@' and '.edu'
    '([A-Za-z.]+) WHERE ([A-Za-z.]+) DOM edu',

    # Matching emails with 'at' in square brackets or parentheses, used to avoid scrapers
    '([A-Za-z.]+) [at]+ ([A-Za-z.]+)\\.edu',

    # Matching emails where '@' is replaced with the HTML entity '&#x40;'
    '([A-Za-z.]+)&#x40;([A-Za-z.]+)\\.edu',

    # Matching emails with '<del>' HTML tag before the '@', perhaps to confuse scrapers
    '([A-Za-z.]+)<del>@([A-Za-z.]+)\\.edu',

    # Matching emails where 'at' is placed within brackets, parentheses,
    # less/greater than signs, or curly braces
    '([A-Za-z.]+)[\\[\\(\\<\\{]at[\\]\\)\\>\\}]\\s*([A-Za-z.]+)\\.edu',

    # Matching emails with the words 'at' and 'dot' spelled out and separated by spaces
    '([A-Za-z.]+)\\s+at\\s+([A-Za-z.]+)\\s+dot\\s+edu',

    # Matching emails with a placeholder text '<at symbol>' for the '@' character
    '([A-Za-z.]+)\\s<at symbol>\\s([A-Za-z.]+)\\.edu',

    # Matching obfuscated emails with HTML character references
    '([A-Za-z0-9._]+)\\s*&#[A-Za-z0-9._]+;\\s*([A-Za-z0-9._]+)\\.edu',

    # Matching 'AT' and 'DOT' with variations in '.edu' or '.com', including case variations
    '([A-Za-z0-9._]+)\\s*AT\\s*([A-Za-z0-9._]+)\\s*DOT\\s*[edu|com|COM|EDU]',

    # Similar to one of the patterns in `patterns1`, matching 'WHERE' and 'DOM'
    '([A-Za-z0-9._]+)\\s*WHERE\\s*([A-Za-z0-9._]+)\\s*DOM\\s*edu',

    # Matching obfuscated emails with a character (possibly trying to represent '@') in angle brackets
    '([A-Za-z0-9._]+)\\s*[<][A-Za-z0-9._]+[>]\\s*@\\s*([A-Za-z0-9._]+)\\.edu',

    # Matching emails with a complex combination of characters and HTML entities before the '@'
    '([A-Za-z0-9._]+)\\s*\\([\\s*A-Za-z0-9.&;#_]*["|;}]@([A-Za-z0-9._]+)\\.edu',

    # Matching emails with 'at' and 'dt' as placeholders for '@' and '.', but only 'com' domains
    '([A-Za-z0-9._]+)\\s*at\\s*([A-Za-z0-9._]+)\\s*dt\\s*[com|COM]',

    # Matching emails with 'at' and semi-colons as placeholders, allowing various domain endings
    '([A-Za-z0-9._]+)\\s*at\\s*([A-Za-z0-9._]{2,})\\s*;\\s*([A-Za-z0-9._]{2,})\\s*;\\s*[com|edu|COM]',

    # Matching emails with 'at' and 'dot' placeholders, including two 'dot' sections for domains like '.co.uk'
    '([A-Za-z0-9._]+)\\s*at\\s*([A-Za-z0-9._]{2,})\\s*dot\\s*([A-Za-z0-9._]{2,})\\s*dot\\s*[com|edu|COM]',

    # Matching emails with 'at' as a placeholder and two domain sections, for longer domain names
    '([A-Za-z0-9._]+)\\s*at\\s*([A-Za-z0-9_]{2,})\\s*([A-Za-z0-9_]{5,})\\s*edu',

    # Similar to previous patterns, this one matches 'at' and 'dot' placeholders with a
    # requirement for the username to be at least two characters long
    '([A-Za-z0-9._]{2,})\\s+at\\s+([A-Za-z0-9._]+)\\s*dot\\s*edu',

    # Matching emails with 'at' as a placeholder, a domain, and a subdomain, ending with a less-than sign,
    # possibly part of HTML
    '([A-Za-z0-9_.-]+)\\s*at\\s*([A-Za-z0-9]{2,})\\.*([A-Za-z0-9_]{5,})\\.edu<',

    # Matching a call to an 'obfuscate' function similar to one in `patterns1`, but with escaped single quotes
    "obfuscate\\(\\'([A-Za-z0-9]+)\\.edu\\',\\'([A-Za-z0-9]+)\\'"
]
```

In [42]:
```python
# patterns3: A list of regex patterns for matching various formats of phone numbers
patterns3 = [
    # Matching phone numbers in a 3-digit, 3-digit, 4-digit sequence separated by hyphens.
    # Capturing each sequence of digits separately
    # Example: 123-456-7890
    '(\\d{3})-(\\d{3})-(\\d{4})',

    # Matching phone numbers where the area code is enclosed in parentheses,
    # followed by a space or no space, then another 3-digit sequence, a hyphen, and a 4-digit sequence
    # Capturing each sequence of digits separately
    # Example: (123) 456-7890 or (123)456-7890
    '\\((\\d{3})\\)\\s*(\\d{3})-(\\d{4})',

    # Matching phone numbers that can be separated by a hyphen or a space
    # It captures the sequences of digits, which are assumed to be in groups of three and then four
    # Example: 123-456-7890, 123 456 7890, 123-456 7890, or 123 456-7890
    '([0-9]{3})[- ]([0-9]{3})[- ]([0-9]{4})'
]
```

This function takes in a filename along with the file object and scans its contents against regex patterns. It returns a list of (filename, type, value) tuples where type is either an 'e' or a 'p' for e-mail or phone, and value is the formatted phone number or e-mail.

The canonical formats are: (name, 'p', '###-###-#####') (name, 'e', 'someone@something')

If the numbers you submit are formatted differently they will not match the gold answers

For Part 3, if you have added other lists, you should add additional for loops that match the patterns in those lists and produce correctly formatted results to append to the res list.

In [43]:
```python
import re

def process_file(name, f):
    res = []  # Initialize a list to store results

    # Iterate over each line in the file
    for line in f:

        # Processing patterns1 for email addresses
        for p1 in patterns1:
            matches = re.findall(p1, line)  # Finding all matches of the pattern in the line
            if 'com' in p1 or 'COM' in p1:
                # If the pattern includes 'com' or 'COM', format the match as an email with .com domain
                for m in matches:
                    email = '%s@%s.com' % m
                    res.append((name, 'e', email))  # Append the result as a tuple
            elif '-e-d-u' in p1:
                # If the pattern includes '-e-d-u', remove hyphens from user and domain parts
                for m in matches:
                    user = m[0].replace('-', '')  # Remove hyphens from user
                    domain = m[1].replace('-', '')  # Remove hyphens from domain
                    email = user + '@' + domain + '.edu'
                    res.append((name, 'e', email))  # Append the formatted email to the results
            elif 'obfuscate' in p1:
                # For patterns that indicate an obfuscation function
                for m in matches:
                    user = m[1]  # The second group in the match is the user
                    domain = m[0]  # The first group is the domain
                    email = user + '@' + domain + '.edu'
                    res.append((name, 'e', email))  # Append the reformatted email to results
            else:
                # For all other patterns, format the match as an email with .edu domain
                for m in matches:
                    email = '%s@%s.edu' % m
                    res.append((name, 'e', email))  # Append the email to results

        # Process patterns2 for email addresses but with different obfuscations
        for p2 in patterns2:
            matches = re.findall(p2, line)  # Find all matches of the pattern in the line
            for m in matches:
                if len(m) == 2:
                    # If the match has two groups, it's a simple email format
                    email = '%s@%s.edu' % m
                else:
                    # If the match has more than two groups, it's a more complex format
                    email = '%s@%s.%s.edu' % m
                res.append((name, 'e', email))  # Append the email to results

        # Process patterns3 for phone numbers
        for p3 in patterns3:
            matches = re.findall(p3, line)  # Find all matches of the pattern in the line
            for m in matches:
                phone = '%s-%s-%s' % m  # Format the match as a phone number
                res.append((name, 'p', phone))  # Append the phone number to results

    return res  # Return the list of results
```

You should not edit this function.

In [44]:
```python
def process_dir(data_path):
    # save complete list of candidates
    guess_list = []
    # save list of filenames
    fname_list = []

    for fname in os.listdir(data_path):
        if fname[0] == '.':
            continue
        fname_list.append(fname)
        path = os.path.join(data_path,fname)
        f = open(path,'r', encoding='latin-1')
        # get all the candidates for this file
        f_guesses = process_file(fname, f)
        guess_list.extend(f_guesses)
    return guess_list, fname_list
```

You should not edit this function.

Given a path to a tsv file of gold e-mails and phone numbers this function returns a list of tuples of the canonical form: (filename, type, value)

In [45]:
```python
def get_gold(gold_path):
    # get gold answers
    gold_list = []
    f_gold = open(gold_path,'r', encoding='latin-1')
    for line in f_gold:
        gold_list.append(tuple(line.strip().split('\t')))
    return gold_list
```

You should not edit this function.

Given a list of guessed contacts and gold contacts, this function computes the intersection and set differences, to compute the true positives, false positives and false negatives.

It also takes a dictionary that gives the guesses for each filename, which can be used for information about false positives.

Importantly, it converts all of the values to lower case before comparing.

In [46]:
```python
def score(guess_list, gold_list, fname_list):
    guess_list = [(fname, _type, value.lower()) for (fname, _type, value) in guess_list]
    gold_list = [(fname, _type, value.lower()) for (fname, _type, value) in gold_list]
    guess_set = set(guess_list)
    gold_set = set(gold_list)

    # for each file name, put the golds from that file in a dict
    gold_dict = {}
    for fname in fname_list:
        gold_dict[fname] = [gold for gold in gold_list if fname == gold[0]]

    tp = guess_set.intersection(gold_set)
    fp = guess_set - gold_set
    fn = gold_set - guess_set

    pp = pprint.PrettyPrinter()
    #print 'Guesses (%d): ' % len(guess_set)
    #pp.pprint(guess_set)
    #print 'Gold (%d): ' % len(gold_set)
    #pp.pprint(gold_set)

    print ('True Positives (%d): ' % len(tp))
    # print all true positives
    pp.pprint(tp)
    print ('False Positives (%d): ' % len(fp))
    # for each false positive, print it and the list of gold for debugging
    for item in fp:
        fp_name = item[0]
        pp.pprint(item)
        fp_list = gold_dict[fp_name]
        for gold in fp_list:
            s = pprint.pformat(gold)
            print('   gold: ', s)
    print ('False Negatives (%d): ' % len(fn))
    # print all false negatives
    pp.pprint(fn)
    print ('Summary: tp=%d, fp=%d, fn=%d' % (len(tp),len(fp),len(fn)))
```

You should not edit this function.

It takes in the string path to the data directory and the gold file

In [47]:
```python
def main(data_path, gold_path):
    guess_list, fname_list = process_dir(data_path)
    gold_list =  get_gold(gold_path)
    score(guess_list, gold_list, fname_list)
```

commandline interface assumes that you are in the directory containing "data" folder

It then processes each file within that data folder and extracts any matching e-mails or phone numbers and compares them to the gold file

In [48]:
```python
if __name__ == '__main__':
    print('Assuming ContactFinder.py called in directory with data folder')
    main('./dev', './devGOLD')
```

```
Assuming ContactFinder.py called in directory with data folder
True Positives (112):
{('ashishg', 'e', 'ashishg@stanford.edu'),
 ('ashishg', 'e', 'rozm@stanford.edu'),
 ('ashishg', 'p', '650-723-1614'),
 ('ashishg', 'p', '650-723-4173'),
 ('ashishg', 'p', '650-814-1478'),
 ('balaji', 'e', 'balaji@stanford.edu'),
 ('bgirod', 'p', '650-723-4539'),
 ('bgirod', 'p', '650-724-3648'),
 ('bgirod', 'p', '650-724-6354'),
 ('cheriton', 'e', 'cheriton@cs.stanford.edu'),
 ('cheriton', 'e', 'uma@cs.stanford.edu'),
 ('cheriton', 'p', '650-723-1131'),
 ('cheriton', 'p', '650-725-3726'),
 ('dabo', 'e', 'dabo@cs.stanford.edu'),
 ('dabo', 'p', '650-725-3897'),
 ('dabo', 'p', '650-725-4671'),
 ('engler', 'e', 'engler@lcs.mit.edu'),
 ('engler', 'e', 'engler@stanford.edu'),
 ('eroberts', 'e', 'eroberts@cs.stanford.edu'),
 ('eroberts', 'p', '650-723-3642'),
 ('eroberts', 'p', '650-723-6092'),
 ('fedkiw', 'e', 'fedkiw@cs.stanford.edu'),
 ('hager', 'e', 'hager@cs.jhu.edu'),
 ('hager', 'p', '410-516-5521'),
 ('hager', 'p', '410-516-5553'),
 ('hager', 'p', '410-516-8000'),
 ('hanrahan', 'e', 'hanrahan@cs.stanford.edu'),
 ('hanrahan', 'p', '650-723-0033'),
 ('hanrahan', 'p', '650-723-8530'),
 ('horowitz', 'p', '650-725-3707'),
 ('horowitz', 'p', '650-725-6949'),
 ('jks', 'e', 'jks@robotics.stanford.edu'),
 ('jurafsky', 'e', 'jurafsky@stanford.edu'),
 ('jurafsky', 'p', '650-723-5666'),
 ('kosecka', 'e', 'kosecka@cs.gmu.edu'),
 ('kosecka', 'p', '703-993-1710'),
 ('kosecka', 'p', '703-993-1876'),
 ('kunle', 'e', 'darlene@csl.stanford.edu'),
 ('kunle', 'e', 'kunle@ogun.stanford.edu'),
 ('kunle', 'p', '650-723-1430'),
 ('kunle', 'p', '650-725-3713'),
 ('kunle', 'p', '650-725-6949'),
 ('lam', 'e', 'lam@cs.stanford.edu'),
 ('lam', 'p', '650-725-3714'),
 ('lam', 'p', '650-725-6949'),
 ('latombe', 'e', 'asandra@cs.stanford.edu'),
 ('latombe', 'e', 'latombe@cs.stanford.edu'),
 ('latombe', 'e', 'liliana@cs.stanford.edu'),
 ('latombe', 'p', '650-721-6625'),
 ('latombe', 'p', '650-723-0350'),
 ('latombe', 'p', '650-723-4137'),
 ('latombe', 'p', '650-725-1449'),
 ('levoy', 'e', 'ada@graphics.stanford.edu'),
 ('levoy', 'e', 'melissa@graphics.stanford.edu'),
 ('levoy', 'p', '650-723-0033'),
 ('levoy', 'p', '650-724-6865'),
 ('levoy', 'p', '650-725-3724'),
 ('levoy', 'p', '650-725-4089'),
 ('manning', 'e', 'dbarros@cs.stanford.edu'),
 ('manning', 'e', 'manning@cs.stanford.edu'),
 ('manning', 'p', '650-723-7683'),
 ('manning', 'p', '650-725-1449'),
 ('manning', 'p', '650-725-3358'),
 ('nass', 'e', 'nass@stanford.edu'),
 ('nick', 'e', 'nick.parlante@cs.stanford.edu'),
 ('nick', 'p', '650-725-4727'),
 ('ok', 'p', '650-723-9753'),
 ('ok', 'p', '650-725-1449'),
 ('ouster', 'e', 'ouster@cs.stanford.edu'),
 ('ouster', 'e', 'teresa.lynn@stanford.edu'),
 ('pal', 'e', 'pal@cs.stanford.edu'),
 ('pal', 'p', '650-725-9046'),
 ('psyoung', 'e', 'patrick.young@stanford.edu'),
 ('rajeev', 'p', '650-723-4377'),
 ('rajeev', 'p', '650-723-6045'),
 ('rajeev', 'p', '650-725-4671'),
 ('rinard', 'e', 'rinard@lcs.mit.edu'),
 ('rinard', 'p', '617-253-1221'),
 ('rinard', 'p', '617-258-6922'),
 ('serafim', 'e', 'serafim@cs.stanford.edu'),
 ('serafim', 'p', '650-723-3334'),
 ('serafim', 'p', '650-725-1449'),
 ('shoham', 'e', 'shoham@stanford.edu'),
 ('shoham', 'p', '650-723-3432'),
 ('shoham', 'p', '650-725-1449'),
 ('subh', 'e', 'subh@stanford.edu'),
 ('subh', 'e', 'uma@cs.stanford.edu'),
 ('subh', 'p', '650-724-1915'),
 ('subh', 'p', '650-725-3726'),
 ('subh', 'p', '650-725-6949'),
 ('thm', 'e', 'pkrokel@stanford.edu'),
 ('thm', 'p', '650-725-3383'),
 ('thm', 'p', '650-725-3636'),
 ('thm', 'p', '650-725-3938'),
 ('tim', 'p', '650-724-9147'),
 ('tim', 'p', '650-725-2340'),
 ('tim', 'p', '650-725-4671'),
 ('ullman', 'e', 'ullman@cs.stanford.edu'),
 ('ullman', 'p', '650-494-8016'),
 ('ullman', 'p', '650-725-2588'),
 ('ullman', 'p', '650-725-4802'),
 ('widom', 'e', 'siroker@cs.stanford.edu'),
 ('widom', 'e', 'widom@cs.stanford.edu'),
 ('widom', 'p', '650-723-0872'),
 ('widom', 'p', '650-723-7690'),
 ('widom', 'p', '650-725-2588'),
 ('zelenski', 'e', 'zelenski@cs.stanford.edu'),
 ('zelenski', 'p', '650-723-6092'),
 ('zelenski', 'p', '650-725-8596'),
 ('zm', 'e', 'manna@cs.stanford.edu'),
 ('zm', 'p', '650-723-4364'),
 ('zm', 'p', '650-725-4671')}
False Positives (5):
('ullman', 'e', 'support@gradiance.edu')
   gold:  ('ullman', 'e', 'support@gradiance.com')
   gold:  ('ullman', 'e', 'ullman@cs.stanford.edu')
   gold:  ('ullman', 'p', '650-494-8016')
   gold:  ('ullman', 'p', '650-725-2588')
   gold:  ('ullman', 'p', '650-725-4802')
('jure', 'e', 'server@cs.stanford.edu')
('jurafsky', 'e', 'stanford@jurafsky.edu')
   gold:  ('jurafsky', 'e', 'jurafsky@stanford.edu')
   gold:  ('jurafsky', 'p', '650-723-5666')
('plotkin', 'e', 'server@infolab.stanford.edu')
('subh', 'e', 'subh@stanford.com')
   gold:  ('subh', 'e', 'subh@stanford.edu')
   gold:  ('subh', 'e', 'uma@cs.stanford.edu')
   gold:  ('subh', 'p', '650-724-1915')
   gold:  ('subh', 'p', '650-725-3726')
   gold:  ('subh', 'p', '650-725-6949')
False Negatives (5):
{('dlwh', 'e', 'dlwh@stanford.edu'),
 ('nass', 'p', '650-723-5499'),
 ('nass', 'p', '650-725-2472'),
 ('ullman', 'e', 'support@gradiance.com'),
 ('vladlen', 'e', 'vladlen@stanford.edu')}
Summary: tp=112, fp=5, fn=5
```

ContactFinder Enhancement Summary

Overview

The `ContactFinder.py` script has been refined with additional regular expressions to improve the extraction of obfuscated email addresses and phone numbers. The update utilizes `patterns1`, `patterns2`, and `patterns3` to capture a broader range of obfuscations.

Results

After the introduction of new patterns, the script's performance has improved:

- **True Positives (TP):** Increased to 112, indicating a higher number of correct matches.
- **False Positives (FP):** Remained low at 5, suggesting that the new patterns are precise and not overly broad.
- **False Negatives (FN):** Reduced to 5, demonstrating the effectiveness of the new patterns in capturing previously missed obfuscations.

Conclusion

The enhancements to the `ContactFinder.py` script have led to a significant reduction in false negatives and an increase in true positives, without a corresponding rise in false positives. This indicates a successful refinement in the script's ability to detect and extract obfuscated contact information.

In [ ]: