HW2

Name: Rashmi Shivanna
SUID: 329589544

Initial status: With the pattern '(\d{3})-(\d{3})-(\d{4})' for phone numbers and '([A-Za-z.]+)@([A-Za-z.]+)\.edu' and '([A-Za-z.]+)\s@\s([A-Za-z.]+)\.edu' for emails, provided by the professor, few emails and phone numbers were picked up correctly and below was the summary of the result.
True Positives (41):
False Positives (0):
False Negatives (76):
Summary: tp=41, fp=0, fn=76

In order to handle the other email IDs and phone numbers, I started creating more patterns. I have tried to cover the requirements in option 3 and explained the patterns I came up with to match all of them correctly.

Handling phone numbers:

Step 1: Picked the below false negative phone numbers:
('bgirod', 'p', '650-723-4539'), ('bgirod', 'p', '650-724-3648'), ('bgirod', 'p', '650-724-6354'),

when I looked at the phone number in bgirod's website, they were (650) 724-6354, (650) 723-4539, (650) 724-3648. They had Parenthesis around the 1st 3 digits which is followed by a space and 3 more digits and - and 4 digits. So, I added a new pattern, pattern2 = '\((\d{3})\) (\d{3})-(\d{4})'

And below string formatting was done to remove the parenthesis and extra space b/w the first 3 digits and the 2nd 3digits.

```
temp = str(m[0]).replace('(','')
temp = temp.replace(')','')
match = (temp,m[1],m[2])
phone = '%s-%s-%s' % match
```

After this, the true positive count increased to 80 and false negative decreased to 37 from 76
Summary: **tp=80, fp=0, fn=37**

Step 2: Picked up Nass's numbers who was categorized as false negative and got the actual phone numbers which were [650] 723-5499 and [650] 725-2472. This format is similar to what I saw in the last pattern except that instead of (this format has [around the 1st 3 digits.

So, I updated pattern2 to '\[*\(*(\d{3})\]*\)* (\d{3})-(\d{4})' so that it matches () or [] around the 1st 3 digits. I used * for [and (so that it matches zero or more of () or []

Now, 3 more phone numbers were matched which increased tp count
Summary: **tp=84, fp=0, fn=33**

Step 3: Picked up phone numbers in the file ashishg which were (650)723-1614, (650)814-1478 and (650)723-4173. These were in the same format like before except that there was no space after the parenthesis and the 2nd 3digits. So, I updated the pattern which I created previously to include zero or more spaces.

pattern2 = '\[*\(*(\d{3})\]*\)* *(\d{3})-(\d{4})'. No string formatting was needed as the parenthesis were not picked when the pattern was matched.

Now, tp count increased further along with fp **Summary: tp=92, fp=2, fn=25**

Step 4: There was only one phone number left which was in the format +1 650 725 9046. To handle this I created a new pattern pattern3= '\+\d (\d{3}) (\d{3}) (\d{4})' and this helped is picking up +1 650 725 9046 correctly.

But the phone numbers ('shoham', 'p', '650-723-3432'), ('shoham', 'p', '650-725-1449') were listed as false negative. These numbers were in format +1 650 723-3432. This was similar to pattern 3 but since they had - instead of space after 2n 3digits and last 4digits, they were not matching with the pattern exactly.

So I updated the pattern to pattern3= '\+\d (\d{3}) (\d{3})(?:\-|)(\d{4})' so that it picks phone numbers with a space or - after the 2nd 3digits and last 4digits. This helped with all phone numbers starting with +1

Step 5(False Positive numbers): Below phone numbers were listed as false positive ('nass', 'p', '114-910-7699'), ('nass', 'p', '375-742-1353'). These numbers were present in the below format in nass's website's html code and not in the website's body.

<LI><i><a href="http://www.amazon.com/exec/obidos/ASIN/1575860538/qid=1110995710/sr=2-1/ref=pd_bbs_b_2_1/104-3**375742-1353**522">
<LI><i><a href="http://www.amazon.com/exec/obidos/ASIN/0262140926/qid=1116182714/sr=2-1/ref=pd_bbs_b_2_1/103-3**114910-7699**828">

These numbers are not present in the gold set as they are not actually phone numbers. So it should ideally be not matched. But the pattern \[*\(*\(d{3})\]*\)* *(\d{3})-(\d{4}) was picking these phone numbers since it detects zero or more parenthesis and spaces. So I updated the pattern to include either (or] instead of zero or more (and [
pattern2= '(?:\[|\()(\d{3})(?:\]|\)) *(\d{3})-(\d{4})'

This removed the false positives. Now, the results were **Summary: tp=94, fp=0, fn=23**

All the phone numbers were handled correctly after creating 3 patterns:
pattern1 = '(\d{3})-(\d{3})-(\d{4})'
pattern2 = '((?:\[|\()\d{3}(?:\]|\))) *(\d{3})-(\d{4})'
pattern3 = '(\+\d) (\d{3}) (\d{3})(?:\-|)(\d{4})'

Handling email IDs:

I updated the pattern which was written by professor to ([A-Za-z.]+)\s*@\s*([A-Za-z.]+)\.edu to match dabo @ cs.stanford.edu since it was similar to normal email IDs but with space around the first word. \s* is taking care of this. It will pick email IDs if they have zero or more spaces around the first word.

Step 1: I picked the below emails which were listed as false negative.

hager at cs dot jhu dot edu
jks at robotics;stanford;edu
lam at cs.stanford.edu

There were few more emails of above pattern which had a word followed by symbol @ or the word at followed by a word again and then a . or dot or ; followed by word and . or dot or ;. So to match these emails, I created the pattern '(\w+)(\s(?:at|@)\s)(\w+)((?: dot |.))(\w+)((?: dot |.))(edu)'

This pattern returns 7 words out of which we needed only 3. But I am returning 7 words since the domain name can contain 3 words and those 3 words could be separated by . or word dot. In that case we could use the whole domain part but if they are separated by space then we won't know which word comes before . in the domain part. For e.g. the email ID pal at cs stanford edu has 3 words in the domain part cs, stanford and edu. Now we return just 2 words from this where 2nd word is the domain then we won't know what are the 2 words in the domain name. If they are separated by . like pal at cs.stanford.edu then returning cs.stanford as one word will make sense but it won't work if space is used as separator. So, I am returning 7 words matching this pattern.

Since we need just 3 words from the returned list, one word before @ and 2 words in the domain part I updated the code to format these emails by replacing dot by . and at by @. Please refer the code attached.

This increased the tp count to 100 but it generated 2 fp emails ('jure', 'e', 'server@cs.stanford.edu'), ('plotkin', 'e', 'server@infolab.stanford.edu'). Since both of these emails begin with server and none of the expected emails had server as the first word, I added a filter(if m[0]!='Server':) while selecting the matched emails so that these emails are not detected.

After this, all Fps were removed. The result was **Summary: tp=100, fp=0, fn=17**

Step 2: For the 2 FN emails ouster@cs.stanford.edu and teresa.lynn@stanford.edu, the format in the files given was 'teresa.lynn (followed by "@stanford.edu")' and 'ouster (followed by “@cs.stanford.edu”)'. These emails didn't match with the any of the patterns created so far since thy had the special characters and random letters attached in the end of the email address. So created the following pattern to match these:

'(\w+(?:.|\s)\w+\s)\(followed by (?:"|“)@(\w+.\w+).edu'

But since the words before @ had spaces in the end, I updated the process_file function to remove those spaces.

Step 4: Few emails had the domain name stroke through. They were present in the html in the below format
latombe@cs.stanford.edu: latombe~~@cs.stanford.edu~~
asandra@cs.stanford.edu: asandra~~@cs.stanford.edu~~
liliana@cs.stanford.edu: liliana~~@cs.stanford.edu~~
manning@cs.stanford.edu: manning <at symbol> cs.stanford.edu</at>
dbarros@cs.stanford.edu: dbarros <at symbol=> cs.stanford.edu</at>

To handle these, I created another pattern (\w+)(?:\s<at symbol>\s|<del>@)(\w+.\w+).edu. So it matched the emails IDs embedded b/w the tags <del> or <at symbol>. This changed the result to **tp=109, fp=0, fn=8**

Step 5: The 2 emails IDs "subh AT stanford DOT edu" and "uma at cs.Stanford.EDU", were still listed as FN. But these should have matched the pattern (\w+)(\s(?:at|@)\s)(\w+)((?: dot |.))(\w+)((?: dot |.))(edu) since they start with a word and then space and then word at and space and word etc. But since these email IDs had capital letters they didn't match the pattern exactly. So, make the pattern match these email IDs, I added [aA][tT] instead of at and [eE][dD][uU] but since this picks a character at a time, it gave many false positive emails. If there is a way to match the letters irrespective of case, the same pattern can be modified to match these email IDs. Since I was not able to find one, I created a pattern (\w+)(?: at | AT)(\w+(?: DOT edu|.\w+.EDU)).

Since they had edu attached to the string which was returned after matching with the pattern and the letters were capitalized, I updated process_file function to convert them to lower case, replace at by @ and dot by .

This gave the result **tp=111, fp=0, fn=6**

Email IDs with special characters and patterns embedded in between:

1. vladlen@stanford.edu was present in the file in the format vladlen at <!-- die!--> stanford <!-- spam pigs!--> dot <!-- die!--> edu. This was not matching with any of the patterns defined till now as it had special characters and words embedded b/w the words in the email. These kind of email IDs are hard to match using a general pattern. So, I created the below pattern specific to this email ID and this helped in picking this email as the TP by the program.
   (\w+)\sat <!-- die!--> (\w+) <!-- spam pigs!--> dot <!-- die!--> edu

2. The emails in the file levoy were present in the following format:
   melissa@graphics.stanford.edu
   ada@graphics.stanford.edu

These formats couldn't be matched with any of the patterns created before as they had special characters in place of at/@. We could update the pattern created before t match these by saying match the email with @ or at or @. But that would make the pattern complex as it will need 3 conditions in the OR part. So, I created a separate pattern (\w+)@(\w+.\w+).edu which matched both the emails in levoy.

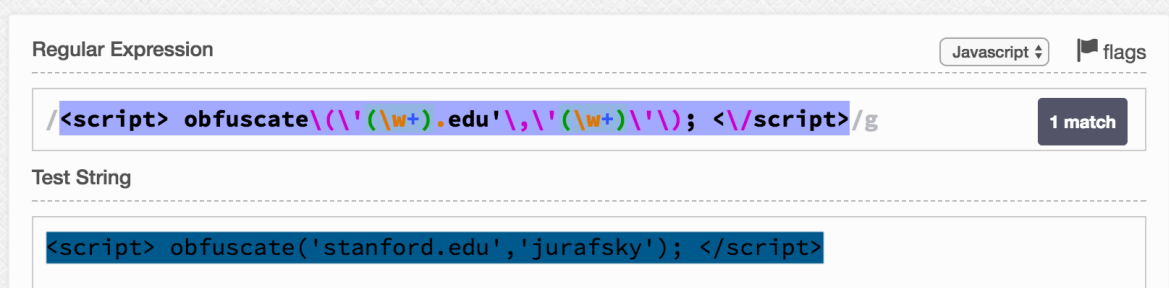
3. There was another email d-l-w-h-@-s-t-a-n-f-o-r-d-.-e-d-u which was not matched with any of the patterns since it had - b/w all the letters in the email. So I created a pattern '([\w-]+)@-([-\w]+).-e-d-u' and added below lines of code to format the email

Finally, result was **tp=116, fp=0, fn=1**.

Un-matched email ID:
Email ID jurafsky@stanford.edu was not matched with any of the patterns written.

jurafsky@stanford.edu was present in the file in the below format
<script> obfuscate('stanford.edu','jurafsky'); </script>.

So I created the pattern <script> obfuscate\(\'(\w+).edu'\,\'(\w+)\'\); <\/script> which matched the email exactly in regexPal.



But it didn't work when I used it in the python program. When I tried to look the strings which were matched with this pattern, I found the below strings which were matched with this pattern in the order mentioned. I noticed that, since the pattern has ' in it and I tried to escape by using \' there must have been an issue matching this email with the pattern by python.

<script> obfuscate('stanford.edu','jurafsky'); </script>
('s', '', 'c', 'r', 'i', 'p', ’t')

<script> obfuscate('stanford.edu','jurafsky'); </script>
('obfu', '', 's', 'c', 'a', 't', ‘e’)

<script> obfuscate('stanford.edu','jurafsky'); </script>
('stanfor', '', 'd', '.', 'e', 'd', ‘u’)

<script> obfuscate('stanford.edu','jurafsky'); </script>
('jur', '', 'a', 'f', 's', 'k', ‘y’)

<script> obfuscate('stanford.edu','jurafsky'); </script>
('s', '', 'c', 'r', 'i', 'p', ’t')

Conclusion:

It was easy to match all the phone numbers since they just had () or [] or spaces or +1 attached. I could match all the phone numbers using only few patterns.

However, email IDs matching was a bit tricky as they had different patterns from each other. Most of the email IDs had general format where they either had spaces or at or AT or dot or DOT, some of them had 3 words in the domain part and some had 2 words in domain. Such email IDs were easy to match using only few patterns which were generalized. But there were few email IDs with special characters and words embedded in between the words in the email ID. I tried creating a generalized patterns to match these, but that created many false positive email IDs. So, patterns had to be created for such email IDs separately. I was able to match all the items using the patterns discussed above except for the email ID jurafsky@stanford.edu. Due to complexity involved in using ‘ inside the pattern in python program, I was not able to match this even though it matched exactly in regexpal.

I updated the process_file function in the python program in order to remove the spaces, replace at \AT with @, dt\dot\DOT by and few emails had 2 and few had 3 words in the domain name, I updated the function to pick the words depending on the length of the domain name. Also, there were email IDs with com in the domain part instead of edu. So for such email IDs, I updated the process_file function to include com instead of the default .edu which was used for other email IDs. There was no update needed to the phone number part in the process_file function. All the patterns and the updated process_file function can be found in the python program SpamLord.py which is uploaded along with this report. Please refer the file for complete code and the output of the program where 115 items were marked as TP and 1 was FN.