Jason DiMedio
CS373
August 16, 2019

Homework 4

The general strategy for approaching this objective is to create a series of tests to run for each URL. Each test will have a score and a weight. The score is the actual number awarded to the URL. The weight is the maximum possible score for that test. In each test, I create a multiplier (e.g. 0, .1, .2, .5, … 1) based on a series of evaluations (e.g. if statements) of the given attributes for the URL. At the end of each test, the multiplier is multiplied by the weight to calculate the score.

For example, if we know that a URL having attribute A makes it very likely to be malicious but otherwise the attribute doesn't make much of a different, we would determine if the URL has attribute A. If it does, the multiplier is set to 0, indicating the maximum chance that the URL is malicious. If it does not have attribute A, the multiplier is set to .5, indicating that this test has minimal impact on whether the URL is malicious.

In some examples, the multiplier has many different possibilities, including calculating the multiplier directly using numerical attributes of the URL.

The weight is set for each test by assigning a number (e.g. from 1-20). Tests whose outcomes have a strong likelihood of affecting the results have a higher number assigned, while tests whose outcomes have a low likelihood of affecting the results have a lower number assigned. Thus, the difference between the assigned score and the weight will push the overall score toward or away from being evaluated as malicious. Specifically the midpoint of the total weight is the neutral point, with scores above the midpoint being evaluated as not malicious and scores below the midpoint being evaluated as malicious.

To start, all of the weights were set at 10, with the plan being that they would be adjusted based on the results of evaluating the training set and validating the results against the actual data.

Additionally, in the early stages of developing the script, instead of iterating through every URL, the script would choose one at random. This allowed me "unit test" the tests while at the same time trying them out on different data to see if the tests were working as they were being developed.

The tests break down to a number of different categories. Each feature being used is identified and explained below, with a justification for weights and thresholds being used for each test. For convenience, relevant portions of code are reproduced below. The full script is included with this document in the "readcorpus.py" file.

1. top level domain

```
# TLD
x = "tld"
val = record[x]
weights[x] = 10
scores[x] = 0
print x + ": " + val

if (val == "com") or (val == "org") or (val == "net") or (val == "edu") or (val == "gov"):
    mult = 1
else:
    mult = .5

scores[x] = round(mult * weights[x])
```

Here, the goal is to favor URLs with well known TLDs, including .com, .org, .net, .edu and .gov. Any URLs with one of these top level domains are given a higher likelihood of being good. If the URL does not have one of these top level domains, however, the effect on the outcome is neutral. These particular TLDs were thought of intuitively. It is possible that with statistical data this test could be improved.

2. domain age

```
# AGE
x = "domain_age_days"
val = int(record[x])
weights[x] = 20
scores[x] = 0
print x + ": " + str(val)

if (val < 10):
    mult = 0
elif (val < 20):
    mult = .1
elif (val < 30):
    mult = .2
elif (val < 90):
    mult = .3
elif (val < 180):
    mult = .4
elif (val < 360):
    mult = .5
elif (val < 500):
    mult = .75
else:
    mult = 1

scores[x] = round(mult * weights[x])
```

From studying the training data set, it seemed that URLs for domains that were only a few days old were almost certain to be malicious, and URLs for domains that were greater than about a year were extremely likely to be good. Thus, in this test, it is determined how old the domain is. Any domain that is greater than 500 days old is given a significant increase in likelihood to be good. On the other hand, any domain that is less than 10 days old is very likely to be malicious. In between, the likelihood gradually increases at 20 days, 30 days, 90 days, 180 days, and 360 days.

3. IP addresses

```
# IPS
x = "ips"
val = len(record[x]) if record[x] else 0
weights[x] = 10
scores[x] = 0
print x + ": " + str(val)

if (val == 0):
    mult = 0
else:
    mult = .5

scores[x] = round(mult * weights[x])
```

In order to detect fast-flux domains, the number of IP addresses for each URL is determined. If there are none, the likelihood of being malicious is increased. Otherwise, there is no effect.

4. alexa ranking

```
# ALEXA
x = "alexa_rank"
val = int(record[x]) if record[x] else None
weights[x] = 10
scores[x] = 0
print x + ": " + str(val)

if (val == None):
    mult = .2
else:
    mult = float(float(1000000 - val) / float(1000000))

scores[x] = round(mult * weights[x])
```

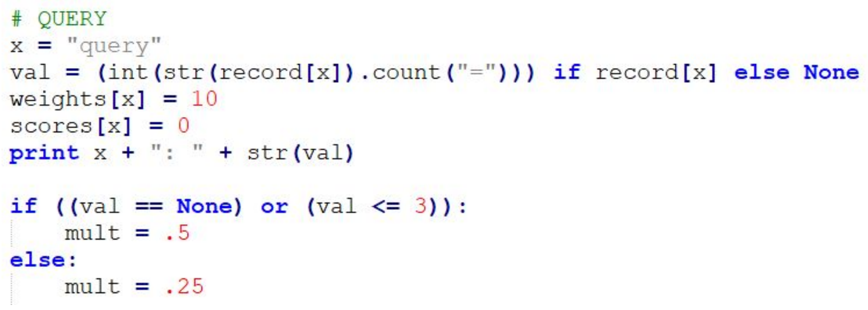
Here, the test is administered from two angles. First, if a URL does not have an alexa ranking, it is given a lower multiplier (.2). If the URL does have an alexa ranking, it is assigned a multiplier that is proportional to its place within the top 1,000,000.

5. file extension

```
# EXTENSION
x = "file_extension"
val = str(record[x]) if record[x] else None
weights[x] = 10
scores[x] = 0
print x + ": " + str(val)

if ((val == None) or (val != "exe")):
    mult = .5
else:
    mult = 0

scores[x] = round(mult * weights[x])
```

In this test, it is simply determined whether the URL has a ".exe" file extension, in which case the URL is given a higher likelihood of being malicious. Any other case (e.g. not having an extension, or having a different extension) has a neutral effect on the result.

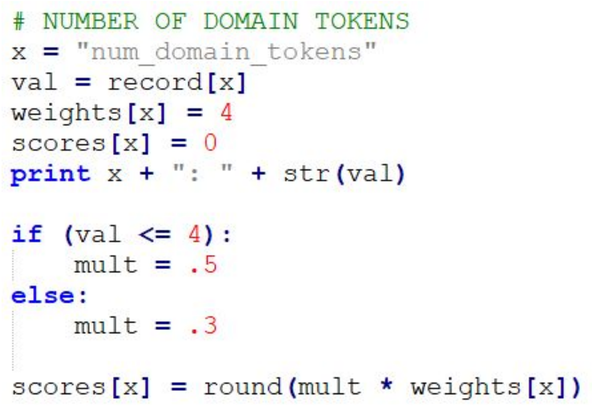
6. query



```
# QUERY
x = "query"
val = (int(str(record[x]).count("="))) if record[x] else None
weights[x] = 10
scores[x] = 0
print x + ": " + str(val)

if ((val == None) or (val <= 3)):
    mult = .5
else:
    mult = .25
```

Several of the URLs in the training file appeared to have long queries with several tokens. This test gives a higher likelihood of being malicious to any query that has more than 3 tokens. Otherwise, there is no affect on the outcome. The threshold of 3 tokens was arbitrary, based on informal observation of the training set. A more rigorous statistical analysis might optimize this test.

7. domain tokens



```
# NUMBER OF DOMAIN TOKENS
x = "num_domain_tokens"
val = record[x]
weights[x] = 4
scores[x] = 0
print x + ": " + str(val)

if (val <= 4):
    mult = .5
else:
    mult = .3

scores[x] = round(mult * weights[x])
```

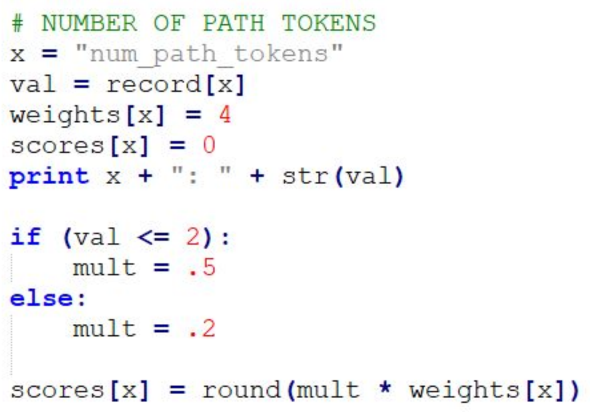
The idea here is to try to suss out URLs with long, complicated domains (including the ones that seemed to try to sneak a well known domain in front of a more obscure domain (e.g. "www.google.com.oiwjeofijwoejf.abc.cn"). The assumption here is that, accounting for "www", the top level domain, and a domain and sub-domain, URLS with 4 or fewer domain tokens would be fairly common, but URLs with more than 4 domain tokens, while not necessarily malicious, are more likely to be. Therefore, any more than 4 domain tokens causes the URL to have a slightly higher chance of being malicious.

8. host characteristics

```
# HOST CHARACTERISTICS
x = "host"
val = record[x]
weights[x] = 10
scores[x] = 0
print x + ": " + str(val)

mult = .5

# if host contains "-"
if val.count("-") >= 1:
    mult -= .1

# if .com is in the middle of the host
if val.count(".com.") >= 1:
    mult -= .1

# if host is exceedingly long
if len(val) > 15:
    mult -= .1
if len (val) > 23:
    mult -= .1
if len(val) > 30:
    mult -= .1

reg = len(record["registered_domain"]) if record["registered_domain"] else 0


# if host is significantly longer than registered domain
if (len(val) - reg > 10):
    mult -= .1
if (len(val) - reg > 20):
    mult -= .1

if mult < 0:
    mult = 0

scores[x] = round(mult * weights[x])
```

This test is actually a series of accumulating affects based on the host, which is the part of the URL before the path and queries. A few traits of malicious URLs stood out from studying the training data set. For example, many malicious URLs had dashes separating words. As mentioned before with respect to the domain tokens test, several malicious URLs buried well known domains within a longer host name. Based on these observations, a series of accumulating if statements are evaluated, each one chipping away at the score if it turns out to be true. They are as follows: if the host has a dash, if the string ".com." is found anywhere in the host, if the host is greater than 15, 23, or 30 characters, and if the host is significantly longer than the registered domain. The character counts were based on an informal survey of the training set data. It is possible that a more rigorous statistical analysis would result in a more optimal threshold.

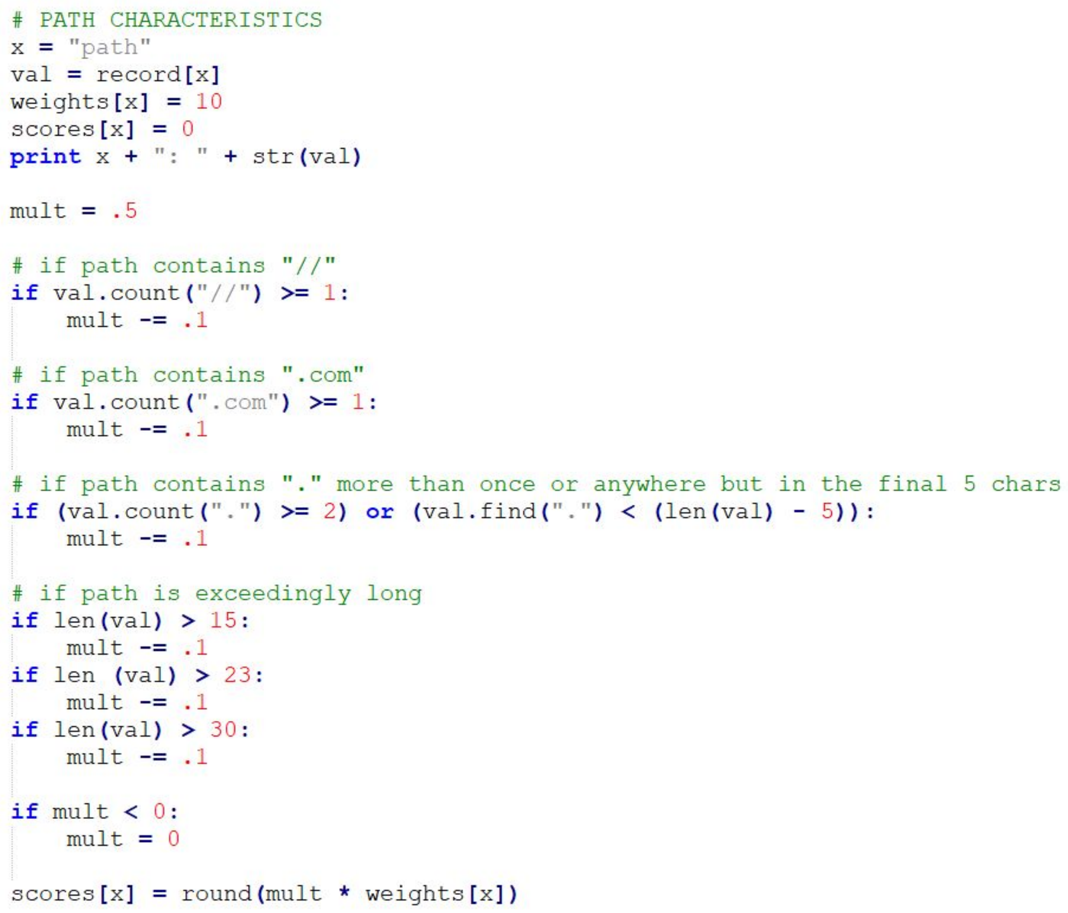
9. path tokens



```
# NUMBER OF PATH TOKENS
x = "num_path_tokens"
val = record[x]
weights[x] = 4
scores[x] = 0
print x + ": " + str(val)

if (val <= 2):
    mult = .5
else:
    mult = .2

scores[x] = round(mult * weights[x])
```

Given that several of the malicious URLs in the training set seemed to have long, complicated path names, this test assigns a greater likelihood that a URL is malicious if is has greater than 2 path tokens. This number is somewhat arbitrary, which is why this test is given a lower weight.

10. path characteristics

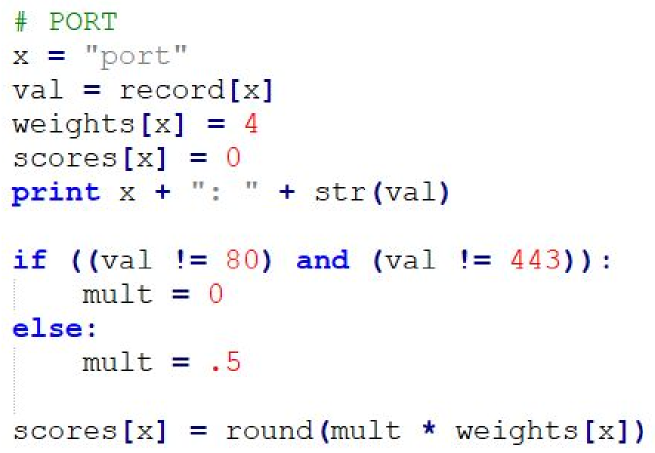


```
# PATH CHARACTERISTICS
x = "path"
val = record[x]
weights[x] = 10
scores[x] = 0
print x + ": " + str(val)

mult = .5

# if path contains "//"
if val.count("//") >= 1:
    mult -= .1

# if path contains ".com"
if val.count(".com") >= 1:
    mult -= .1

# if path contains "." more than once or anywhere but in the final 5 chars
if (val.count(".") >= 2) or (val.find(".") < (len(val) - 5)):
    mult -= .1

# if path is exceedingly long
if len(val) > 15:
    mult -= .1
if len (val) > 23:
    mult -= .1
if len(val) > 30:
    mult -= .1

if mult < 0:
    mult = 0

scores[x] = round(mult * weights[x])
```

Here, a series of tests are run on the path string based on several observations about the paths for malicious URLs from the training set. On an accumulating basis, if the path contains "//",

".com", the character "." more than once or anywhere outside of the final 5 characters, or if the path is longer than 15, 23, or 30 characters, it is given a higher likelihood of being malicious.

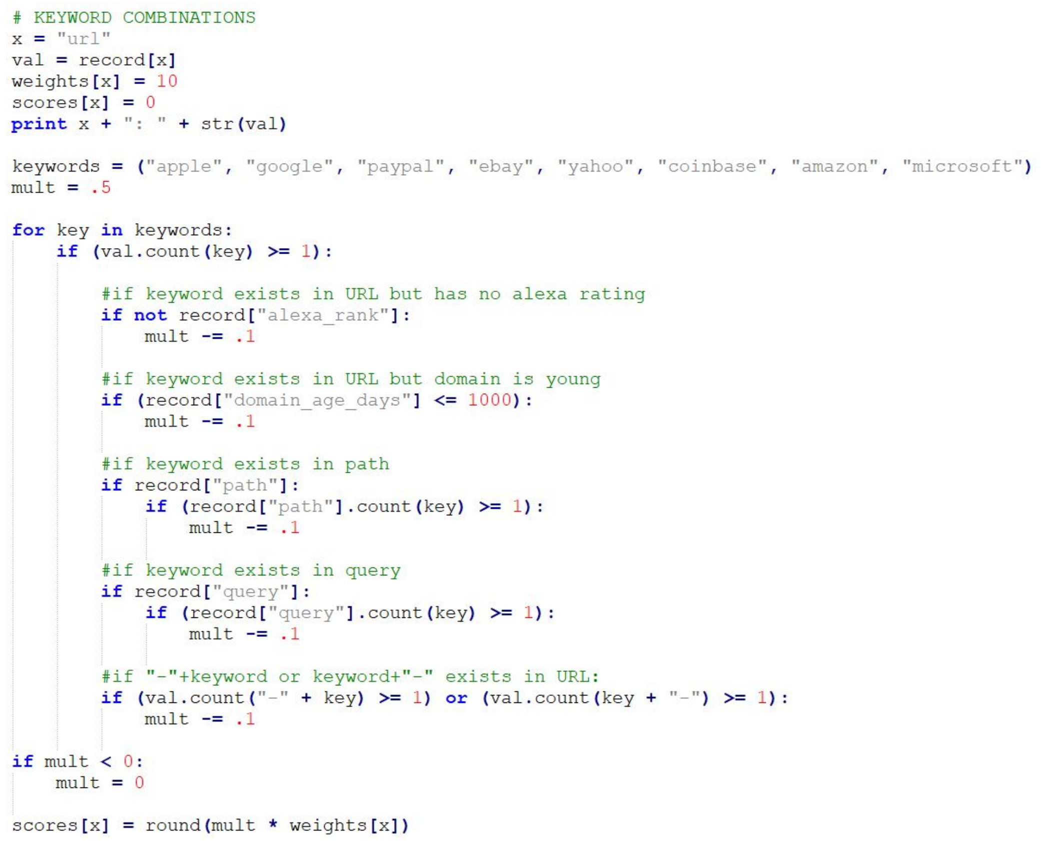
11. port



```
# PORT
x = "port"
val = record[x]
weights[x] = 4
scores[x] = 0
print x + ": " + str(val)

if ((val != 80) and (val != 443)):
    mult = 0
else:
    mult = .5

scores[x] = round(mult * weights[x])
```

Based on the observation that the training set largely uses standard ports as well as several well known URLs that I tested outside of the training set, I implemented a test that assigns a higher likelihood of being malicious if a non-standard port is used. If a standard port is used, there is no effect.

12. keyword combinations

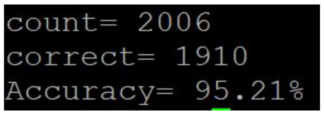


```
# KEYWORD COMBINATIONS
x = "url"
val = record[x]
weights[x] = 10
scores[x] = 0
print x + ": " + str(val)

keywords = ("apple", "google", "paypal", "ebay", "yahoo", "coinbase", "amazon", "microsoft")
mult = .5

for key in keywords:
    if (val.count(key) >= 1):

        #if keyword exists in URL but has no alexa rating
        if not record["alexa_rank"]:
            mult -= .1

        #if keyword exists in URL but domain is young
        if (record["domain_age_days"] <= 1000):
            mult -= .1

        #if keyword exists in path
        if record["path"]:
            if (record["path"].count(key) >= 1):
                mult -= .1

        #if keyword exists in query
        if record["query"]:
            if (record["query"].count(key) >= 1):
                mult -= .1

        #if "-"+keyword or keyword+"-" exists in URL:
        if (val.count("-" + key) >= 1) or (val.count(key + "-") >= 1):
            mult -= .1

if mult < 0:
    mult = 0

scores[x] = round(mult * weights[x])
```

Here, it is determined whether unusual combinations of attributes are present for the URLs. In particular, these combinations depend on whether one of a set of keywords representing very well known domains (e.g. "google", "apple") is present in the URL. If one of the keywords is present but the URL has no alexa rating, the domain is very young, if the keyword exists in the path or query (rather than the host) and if the keyword exists alongside a dash in the front or back (e.g. "apple-" or "-apple"). The idea here is the pick up people trying to sneak these well known names into malicious URLs, which is common, for example, in phishing.

Training Results

The first complete run through the training set resulted in a 95.21% accuracy. This was higher than I was expecting. I attribute this initial high accuracy to the way in which the tests were developed, where I evaluated how the tests were matching up with the actual data during the development process.



```
count= 2006
correct= 1910
Accuracy= 95.21%
```

That said, based on the results, there were definitely areas that could be improved.

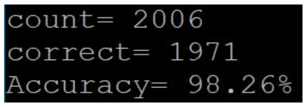
In one example, I caught a logical error in the **port** test. Strangely, upon fixing the logical error, the error rate increased! As a result, I determined that the port test should be de-emphasized. This is also where I discovered that the weights must be even numbers, or the rounding would skew the results in favor of non-malicious URLs (e.g. because 5/2 = 2.5, which is always rounded up to 3). After fixing these issues, the percentage increased.

I also reweighted the **path tokens** test from 5 to 4 and **keyword combinations** from 5 to 10.

The threshold for the **alexa rank** was changed to penalize the URLs without a rank less than before, because there was evidence that it was causing too many false positives.

However, the most significant change was doubling the weight of the **domain age** test. The data seemed to point to the fact that this characteristic was a particularly good indicator of both a malicious URL (for very young domains) and a good URL (for much older domains).

As a result of these adjustments, I was able to increase the accuracy to 98.26%.



```
count= 2006
correct= 1971
Accuracy= 98.26%
```

Classification Results

Included along with this document is a "results" text file providing the results output by the classification script when it was executed on the classify.json data set.