Homework 4

My approach to creating a URL classifier was largely empirical. Given the training set of labelled URLs available to me, I decided to do some statistical analysis on their features to try and identify common patterns within the malicious and safe categories. Using the provided readcorpus.py as a starting point, I modified to the script to read in the training data and, given an arbitrary field name, print a summary of values and frequencies of that feature among both safe and malicious URLs.

For numeric values this approach was relatively straightforward: add up all the values for both malicious and safe URLs and then analyze average values for each category, as well as some other measures such as the standard deviation. For categorical labels within the training data, I created functions that returned simple numerical scores (e.g, 1 and 0) based on some commonsense guesses at how the categorical features should align within the two groups. For instance, I assumed a top-level domain of .com is less suspicious than .cn. These guesses were refined based on how I saw scoring reflected in the statistical output. Essentially, I was mimicking a machine learning algorithm in a primitive and manual way.

Surprisingly, after examining the training data systematically, there were many features that I assumed would be useful that ended up being rather unhelpful. For instance, average URL length is almost identical between malicious and safe URLs; the same is true of host length, path length, and nearly true of the number of domain tokens. Similarly, I found that the presence of query strings was not especially indicative of a malicious URL as I had expected. Neither was the scheme used.

However, there were a small handful of features that were highly indicative of malicious URLs. For instance, domain age is vastly different among the categories: everything newer than one year was malicious in the training data. A malicious URL’s top-level domain is only half as likely as a safe URL to be one of .com, .mil, .edu, .org, or .gov. A similar pattern emerges with default ports, file extensions, and Alexa scores. After this analysis, I was able to vastly narrow down the features that my classifier measured. In the end, my classifier only incorporates six different features of each URL:

* Presence of a fragment
* Domain age
* Top-level domain
* IP association
* Default port
* File extension

Using this classifier, I obtained an accuracy of 99.95% on the training data. This makes me suspect overfitting; however, within the constraints of the exercise and given the size of our training set I was satisfied with how it performed. Generalizing to the classify.json dataset, the classifier flagged 48.86% of URLs as malicious and the other 51.14% as safe, which roughly reflects what we were instructed to expect (i.e., a 50/50 mix).

Script

|  |
| --- |
| #!/usr/bin/python3  # CS 373 - Homework 4  # Caleb Schmidt  # schmical@oregonstate.edu  # 8/9/2019  import sys  import json  import math  import getopt  def usage():  '''  Prints a usage message to stdout.  '''  print("Usage: %s --file=[filename]" % sys.argv[0])  sys.exit()  def main():  '''  Parses command line args and delegates  classification of the specified file.  '''  file = ''  myopts, args = getopt.getopt(sys.argv[1:], "", ["file="])  # Get passed filename  for opt, arg in myopts:  if opt in ('-f', '--file'):  file = arg  else:  usage()  if not len(file):  usage()  # Read in the specified file  with open(file, encoding='latin1') as f:  urldata = json.load(f)  # Delegate to classification function  classify\_urls(urldata)  def classify\_urls(urldata, threshold=3):  '''  Takes URL data and a score threshold and  then aggregates weighted scores from various  scoring functions to create an overall score  and determine if a URL is malicious or not.  Writes gross statistics to stdout and individual  classifications to url\_classifications.txt.  '''  guess = list()  for record in urldata:  # Get each individual score and associate with a weight  weighted\_scores = [  (calc\_fragment\_score(record), 3),  (calc\_domain\_age\_score(record), 3),  (calc\_tld\_score(record), 1),  (calc\_ip\_score(record), 3),  (calc\_port\_score(record), 1),  (calc\_file\_extension\_score(record), 1)  ]  # Calculate the weighted sum of scores  total\_score = sum(score \* weight for score, weight in weighted\_scores)  # Flag as malicious if over the threshold  is\_malicious = 1 if total\_score >= threshold else 0  # Save our guess  guess.append((is\_malicious, record['url']))  # Display overall stats for the classified URLs  print(f'Malicious: {sum(i for i, url in guess) / len(urldata):.2%}')  print(f'Safe: {sum(i == 0 for i, url in guess) / len(urldata):.2%}')  # Save individual classifications to file  with open('url\_classifications.txt', 'wt') as f:  lines = [','.join([str(url), str(flag)]) for flag, url in guess]  f.write('\n'.join(lines))  def calc\_file\_extension\_score(record):  '''  Essentially, only flag as malicious if  the file extension is php.  '''  if record.get('file\_extension', '') == 'php':  return 1  return 0  def calc\_alexa\_score(record):  '''  If a URL has an Alexa rank, return the log 10  divided by 6, roughly scaling to 0-1 for all  possible rankings, with lower scores resulting in  a lower possibility of maliciousness.  '''  rank = record.get('alexa\_rank')  if rank:  return math.log10(int(rank)) / 6  return 1  def calc\_port\_score(record):  '''  Flag a URL as potentially malicious if it  is not using ports 80 or 443.  '''  port = record.get('default\_port', -1)  if port not in [80, 443]:  return 1  return 0  def calc\_ip\_score(record):  '''  Flag URLs with no associated IPs as  potentially malicious.  '''  ips = record.get('ips', [])  if not ips:  return 1  return 0  def calc\_tld\_score(record):  '''  If a TLD is not one of a select handful,  flag it as potentially malicious.  '''  tld = record.get('tld', '')  if tld in 'com org mil edu gov'.split():  return 0  return 1  def calc\_domain\_age\_score(record):  '''  If the domain is less than one year old,  flag it as potentially malicious.  '''  days = int(record.get('domain\_age\_days', '0'))  if days < 365:  return 1  return 0  def calc\_fragment\_score(record):  '''  If a URL has a fragment, flag it as  potentially malicious.  '''  frag = record.get('fragment')  if frag:  return 1  return 0  if \_\_name\_\_ == "\_\_main\_\_":  main() |

Output

|  |
| --- |
|  |