Practical 3: Classifying Malicious Software

Writeup due 23:59 on Friday 14 March 2014
Kaggle submission closes at 11:59am on Friday 14 March 2014

CS 181 Students: You will do this assignment in groups of three. You can seek partners via Piazza. Course staff can also help you find partners. Submit one writeup per team by the due date via the iSites dropbox.

CSCI E-181 Students: You will do this assignment on your own. Submit your writeup by the due date via the Extension School iSites dropbox.

Competing on Kaggle: You are expected to submit at least one set of predictions to the Kaggle competition online at

http://inclass.kaggle.com/c/cs181-practical-3-classifying-malicious-executables

CS 181 students should submit these as a team. You should be a part of exactly one team. Do not make submissions separately from your team, and please use your real name for your user identity so that we can identify your results. There is a limit of four submissions per day, where "day" is determined by UTC. (Your deadlines are still in eastern time.) **Note that the Kaggle submission site closes 12 hours before the iSites dropbox. This is to ensure that you are able to write up any last-minute submissions.** You should be able to join the competition by registering with your `harvard.edu` or `mit.edu` email address. If you have trouble joining the contest, please email the staff list.

Warm-Up

In the warm-up, you'll be classifying three different kinds of fruit, based on their heights and widths. Figure 1 is a plot of the data. Iain Murray collected these data and you can read more about this on his website at http://homepages.inf.ed.ac.uk/imurray2/teaching/oranges_and_lemons/. I have made a slightly simplified (collapsing the subcategories together) version of this available as `fruit.csv`, which you can download from the course website. The file has three columns: type (1=apple, 2=orange, 3=lemon), width, and height.

Implement the three-class generalization of logistic regression for these data. Make a plot that shows the decision boundaries. After this, implement a simple generative classifier with Gaussian class-conditional densities. Fit this model with maximum likelihood and plot the resulting decision boundaries. Describe in words the differences that you see in the boundaries for the two models. Which one looks like it can fit these simple data better? Why do you think this is true?

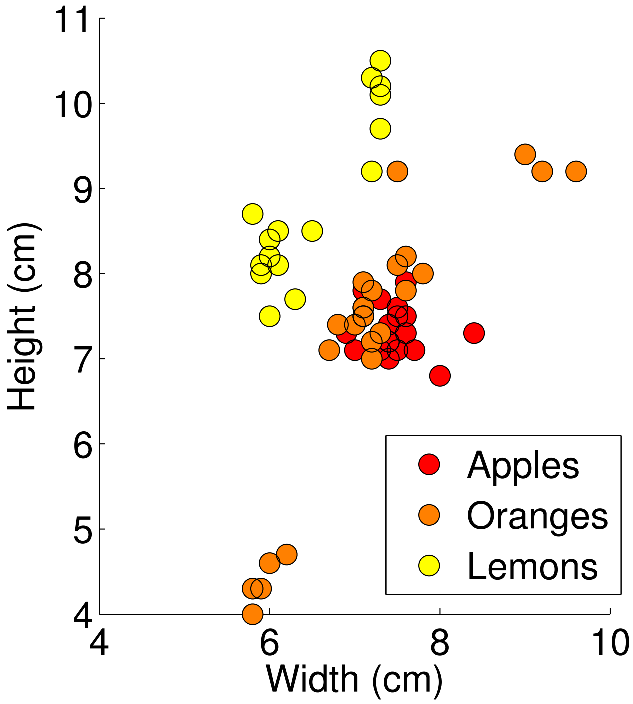

Figure 1: Heights and widths of apples, oranges, and lemons. These fruit were purchased and measured by Iain Murray: http://homepages.inf.ed.ac.uk/imurray2/teaching/oranges_and_lemons/.

Classifying Malicious Software Executables

In this practical, you will classify executable files collected from people's computers (over a period of several days) into any of 14 known malware classes, or determine that the executables are not malware. You will train on 3086 executables of known provenance that were collected on a single day. Your predictions on 1900 out of 3724 executables collected on a subsequent day will appear on the public leaderboard, and your predictions on the remaining 1824 executables will appear in the private leaderboard. In making your predictions, you will primarily have at your disposal logs of the system calls (and arguments) made by the processes invoked by the executables when run.

Identifying malware can be tricky because there are often many variants of any particular class of malware that exhibit slightly different behavior. Malware can also behave differently depending on the environment in which it finds itself, or act with some randomness. Being able to classify malware into broader classes is useful, however, because it can both suggest ways of disinfecting infected systems, as well as allow us to easily identify new variants of particular classes that are being introduced. The malware classes under consideration in this practical are: Agent, AutoRun, FraudLoad, FraudPack, Hupigon, Krap, Lipler, Magania, Poison, Swizzor, Tdss, VB, Virut, and Zbot.

Data Files

There are three files of interest, which can be downloaded from Kaggle:

- train.tar.gz and test.tar.gz – These file contains information about the 3086 executables in the training set, and the 3724 executables in the test set, respectively. They are gzipped tarballs of directories, which contain an XML file for each datum. The training files have the form

  <hex_string>.<malware_class>.xml

  where hex_string is a unique identifier, and malware_class is either one of the 14 malware classes under consideration or is None. For example:

  fc9b35928deb723b0e0105263d1661e38ad033337.FraudLoad.xml

  You will use the malware_class label in the filename when training. After unzipping test.tar.gz, you should also have a directory containing XML files named according to the above convention, except the malware_class in the filename has been replaced in each instance with an X.

  When submitting your predictions, you will use the hex_string in the filename as a unique identifier. For example, when predicting on the test file

  ffc47163a530c51ef2e6572d786aefbaed99890f2.X.xml

  you will use ffc47163a530c51ef2e6572d786aefbaed99890f2 as the Id in your submission file. Your prediction for each unique hex_string will be an integer between 0 and 14 (inclusive). See the "Evaluation" section for more details.

  Each train and test file is a valid XML document containing a log of the executable's execution history as well as some metadata. The XML adheres to the following format:

```
<?xml version="1.0"?>
<processes>
 <process ...>
  <thread>
   <all_section>
     <system_call1 ...>
     </system_call1>
     ... more system calls
   </all_section>
  </thread>
  ... more threads
 </process>
```

```
  ... more processes
</processes>
```

That is, the root element is called `processes`, and it contains a list of `process` elements, each corresponding to one of the processes invoked by the executable. Each `process` element may contain some metadata as attributes, and its children are `thread` elements. The execution history of a particular thread is contained in an `all_section` element, which is likely the most important part of the document. The `all_section` element lists the system calls made by the thread (in order) together with various arguments. Note that this will not literally have `system_call1` elements, but the element names will correspond to system calls such as `load_dll` and `create_thread`.

The following is an example `system_call` element from an `all_section`:

```
<load_image \
  filename="c:\342c547b28e9517f6fcf6c703933c0d9.EX" \
  successful="1" address="&#x24;400000" \
  end_address="&#x24;414000" size="81920" \
  filename_hash="hash_error"/>
```

The name of the system call, in this case `load_image`, is given by the tag of the element, and its arguments appear as attributes. The above system call element does not have children, though some system call elements do.

- `sample_predictions.csv` – A sample submission file. You will produce a similar file. The format is comma-delimited, with the first column being the `hex_string` and the second column being your class prediction, an integer between 0 and 14 (inclusive).

```
Id,Prediction
0aefbb082e0461675d05e3147473045acdf2894cb,2
7070018d4360b1b45a6dcb001acc4e463369d2e9f,2
4fcb33dd28a6f88533562958c22f26d5bfbb683b1,2
a2be4cf8927a6f2dbff67a02f9487982511768e21,13
a7abe16d5197f7d2257b81724a241c4b0b3f35bed,13
e44f52dfce3fdef015ee4f77f4564d1e18bc908a9,13
a3b09caec6edcfeb2f3ef321b62a5100a6c2f23f9,2
...
```

The class numbers correspond to the predicted classes, in alphabetical order; see table below. Note that `None` is a special class indicating that the executable is not malware.

Class Distribution

The distribution of malware classes in the training data is approximately as follows. It may be worthwhile to keep in mind that some classes are very infrequent.

0	`Agent`	3.69%
1	`AutoRun`	1.62%
2	`FraudLoad`	1.20%
3	`FraudPack`	1.03%
4	`Hupigon`	1.33%
5	`Krap`	1.26%
6	`Lipler`	1.72%
7	`Magania`	1.33%
8	`None`	52.14%
9	`Poison`	0.68%
10	`Swizzor`	17.56%
11	`Tdss`	1.04%
12	`VB`	12.18%
13	`Virut`	1.91%
14	`Zbot`	1.30%

Evaluation

The evaluation metric for this practical is categorization accuracy. That is, you will be scored on the percentage of the test executables that are correctly classified. In math:

$$\text{Categorization Accuracy} = \frac{\text{Number Correctly Classified Examples}}{\text{Total Number of Examples}}.$$

Sample Code

Two Python files are available from the course website. The file `classification_starter.py` and `util.py` are meant to help you get going. You definitely don't have to use them, but they provide some potentially-useful tools in which you can fill in the gaps. Specifically, it helps you write some functions that can generate features from the data. The file has lots of comments, so hopefully you can figure out how it works. Thanks to Sam Wiseman for putting this together!

Solution Ideas

As in the previous practicals, you have a lot of flexibility in what you might do. You could focus on feature engineering, i.e., coming up with fancy inputs for your method, or you could focus on fancy classification techniques that use the features. Here are some ideas to get you started:

- **Logistic regression on basic features:** A good place to start is to turn the data into a vectorial feature representation, and use a logistic regression technique. You could use quantitative features such as the number of times each system call was made.

- **Use a generative classifier:** You could build a model for the class-conditional distribution associated with each type of malware and compute the posterior probability for prediction.

- **Use a neural network:** If you think there isn't enough flexibility, you could implement a multi-layer perceptron and train it with backpropagation.

- **Use a support vector machine:** If you prefer your objectives convex, you could jump the gun and learn about support vector machines.

- **Go totally Bayesian:** Worried that you're not accounting for uncertainty? You could take a fully Bayesian approach to classification and marginalize out your uncertainty in a generative or discriminative model.

- **Use a decision tree:** If you think a linear classifier is too simple but don't want to train a neural network, you could try a decision tree.

- **Use KNN:** Have a great way to think about similarities between the executables? You could try K nearest neighbors and see how that works.

Questions and Answers

What should I turn in via the dropbox? The main deliverable of this practical is a three-to-four page typewritten document in PDF format that describes the work you did. The warmup asks you to implement a basic algorithm and turn in code, but the bulk of your writeup should be about how you tackled the practical prediction task. This may include figures, tables, math, references, or whatever else is necessary for you to communicate to us how you worked through the problem. Concretely, you should turn in via the dropbox:

- A 3-4 page PDF writeup that shows results from your warm-up and explains your approach to malware classification. Make sure to include the name of the team and the names of all partners.

- A zipped or gzipped folder containing your warm-up code and a README file explaining how to run it.

How will my work be assessed? This practical is intended to be a realistic representation of what it is like to tackle a problem in the real world with machine learning. As such, there is no single correct answer and you will be expected to think critically about how to solve it, execute and iterate your approach, and describe your solution. The upshot of this open-endedness is that you will have a lot of flexibility in how you tackle the problem. You can focus on methods that we discuss in class, or you can use this as an opportunity to learn about approaches for which we do not have time or scope. Except for the warm-up, you are welcome to use whatever tools and implementations help you get the job done. Note, however, that you will be expected to *understand* everything you do, even if you do not implement the low-level code yourself. It is your responsibility to make it clear in your writeup that you did not simply download and run code that you found somewhere online.

You will be assessed on a scale of 25 points, divided evenly into five categories:

1. **Warm-Up:** In the warmup, you'll be expected to implement a simple algorithm from scratch, run it on some simple data, and turn in your code. You'll be graded on correctness of the implementation.

2. **Effort:** Did you thoughtfully tackle the problem? Did you iterate through methods and ideas to find a solution? Did you explore several methods, perhaps going beyond those we discussed in class? Did you think hard about your approach, or just try random things?

3. **Technical Approach:** Did you make tuning and configuration decisions using quantitative assessment? Did you compare your approach to reasonable baselines? Did you dive deeply into the methods or just try off-the-shelf tools with default settings?

4. **Explanation:** Do you explain not just what you did, but your thought process for your approach? Do you present evidence for your conclusions in the form of figures and tables? Do you provide references to resources your used? Do you clearly explain and label the figures in your report?

5. **Execution:** Did you create and submit a set of predictions? Did your methods give reasonable performance? Don't worry, you will not be graded in proportion to your ranking; we'll be using the ranking to help calibrate how difficult the task was and to award bonus points to those who go above and beyond.

Bonus Points for CS 181 Students: The top three teams among CS 181 (Harvard College and local cross-registrations) students will be eligible for extra credit. The first place team will receive an extra five points on the practical, conditioned on them giving a five-minute presentation to the class at the next lecture, in which they describe their approach. The second and third place teams will each receive three extra points, conditioned on them posting an explanation of their approach on Piazza.

Bonus Points for CSCI E-181 Students: The top three individuals among CSCI E-181 (Extension School) students will be eligible for extra credit. The first place team will receive an extra five points on the practical and the second and third place teams will each receive three extra points, all conditioned on posting an explanation of their approach on Piazza. Extension school students who choose to form teams will be pooled with the Harvard College teams for purposes of awarding bonus points.

What language should I code in? You can code in whatever language you find most productive. We will provide some limited sample code in Python and can also provide some support for Matlab. You should not view the provided Python code as a required framework, but as hopefully-helpful examples.

Can I use {scikit-learn | pylearn | torch | shogun | other ML library}? You can use these tools, but not blindly. You are expected to show a deep understanding of the methods we study in the course, and your writeup will be where you demonstrate this. You should not use these tools for the warm-ups.

These practicals do not have conceptual questions. How will I get practice for the midterms? We will provide practice problems and solutions in section. You should work through these to help learn the material and prepare for the exams. They will not be a part of your grade.

Can I have an extension? There are no extensions to the Kaggle submission and your successful submission of predictions forms part of your grade. Your writeup can be turned in up to a week late for a 50% penalty. There are no exceptions, so plan ahead. Find your team early so that there are no misunderstandings in case someone drops the class.

Changelog

This format for assignments is somewhat experimental and so we may need to tweak things slightly over time. In order to be transparent about this, a changelog is provided below.

- **v1.0** – 28 February 2014 at 23:59pm