Note: Page numbers referring to figures and tables are followed by an italicized f or t respectively.

A

activation functions
common, 178t–180t
defined, 178
add_edge function, 41
add_node function, 49–50
add_question function, 112
add arithmetic instruction, 15
ADS (Alternate Data Streams), 29
Advanced Persistent Threat 1 attacker group. See APT1 attacker group
advanced persistent threats (APTs), 60
Allaple.A malware family, 157, 157f
Alternate Data Streams (ADS), 29
anti-disassembly techniques, 22
API calls, 32–33, 33f
apply_hashing_trick function, 138
APT1 (Advanced Persistent Threat 1) attacker group, 37–39, 38f, 45–47, 45f–47f, 61, 61f, 76, 76f, 86, 222–223
APT1s (advanced persistent threats), 60
ArchSMS family of Trojans, 55
area under the curve (AUC), 209–210, 210f, 213
arithmetic instructions, 15, 15t
.asarray method, 142
assembly language, defined, 12. See also
x86 assembly language
AT&T, 43
AT&T syntax, 13
attributes, 37
adding to nodes and edges, 42
and edges, 48–51
AUC (area under the curve), 209–210, 210f, 213

autoencoder neural networks, 194–195, 195f
automatic feature generation, 188

B

backpropagation, 190–192, 190f–191f
bag of features model, 62–64, 63f
features, defined, 62
Jaccard index and, 65
N-grams, 63–64, 64f
order information and, 63–64
overview, 62–63
bar charts (histograms), 168–170, 168f–169f
base virtual memory address, 6
basic blocks, 19–20
bias parameter, 104
bias term, 178, 181
bipartite networks, 37–39, 38f
bitcoin mining, 158, 160–161, 168f, 172f–173f, 173

callbacks
built-in (Keras package), 212
creating shared callback relationship network, 51–54
custom, 213–214, 214f
call instruction, 17–18
capstone module, 20
Carerra, Ero, 5
chain rule, 191–192
cmp instruction, 18
CNNs (convolutional neural networks), 193–194, 194f
coarsenings, 46
color attribute, 49
comment_sample function, 82–84
COMMENT mode, 229
compile method, 202
compressed_data_weight parameter, 103
compressed_data parameter, 103–104
conditional branches, defined, 15
control flow, 17
graphs, 19–20, 19f
instructions, 17–18
registers, 14–15
convolutional neural networks (CNNs),
193–194, 194f
CPU registers, 13–15, 14f
general-purpose registers, 13–14
stack and control flow registers,
14–15
cross_validation module, 151
cross-validation, 150–153, 151f, 153f
CuckooBox software platform, 27,
33–34, 59
“curse of dimensionality,” 92
cv_evaluate function, 151

dapato malware family, 62, 67f–68f,
70f–72f
DataFrame objects, 158–161
data movement instructions, 15–20, 16t
basic blocks, 19–20, 19f
control flow graphs, 19–20, 19f
control flow instructions, 17–18
stack instructions, 16–17
data science, iii, iv
applying to malware, v
importance of, iv–v
.data section (in PE file format), 4
dateutil package, 164
dec arithmetic instruction, 15
decision boundaries, 93–98, 95f–98f
identifying with k-nearest neighbors, 97–98, 97f–98f
identifying with logistic regression, 96–97, 96f–97f
overfit machine-learning model, 100, 101f
underfit machine-learning model,
99, 99f
well-fit machine-learning model,
100, 100f
decision thresholds, 149
DecisionTreeClassifier class, 130
detection accuracy evaluation, 119–126,
146–153
base rates and precision, 124–126
effect of base rate on precision, 124–125
estimating precision in
deployment environment, 125–126
with cross-validation, 150–153,
151f, 153f
neural networks, 209–211, 210f–211f
possible detection outcomes,
120, 120f
with ROC curves, 123–124, 123f,
147–150, 150f
true and false positive rates,
120–124
relationship between, 121–122,
121f–122f
ROC curves, 123–124, 123f

decision tree–based detectors, 129
importing modules, 129
initializing sample training
data, 130
instantiating classes, 130
sample code, 133–134
training, 130–131
visualizing, 131–133, 132f
follow-up questions, 111
limiting depth or number of
questions, 111–112
pseudocode for, 112–113
root node, 110–111
when to use, 114–115
deep learning, 175–197, 216. See also
neural networks
automatic feature generation, 188
building neural networks, 182–188
neurons, 176
anatomy of, 177–180
networks of, 180–181
overview, 176–177
training neural networks, 189–193
types of neural networks, 193–197
universal approximation theorem,
181–182
deep neural networks. See neural
networks
Dense function, 200–201
describe method, 159
detection accuracy evaluation, 119–126,
146–153
base rates and precision, 124–126
effect of base rate on precision, 124–125
estimating precision in
deployment environment, 125–126
with cross-validation, 150–153,
151f, 153f
neural networks, 209–211, 210f–211f
possible detection outcomes,
120, 120f
with ROC curves, 123–124, 123f,
147–150, 150f
true and false positive rates,
120–124
relationship between, 121–122,
121f–122f
ROC curves, 123–124, 123f

decision trees, 109–115, 109f–110f,
113f–114f
decision tree–based detectors, 129
importing modules, 129
initializing sample training
data, 130
instantiating classes, 130
sample code, 133–134
training, 130–131
visualizing, 131–133, 132f
follow-up questions, 111
limiting depth or number of
questions, 111–112
pseudocode for, 112–113
root node, 110–111
when to use, 114–115
DictVectorizer class, 128–130
directed graphs, 180
distance functions, 107
DLLs (dynamic-link libraries), 13
DOS header (in PE file format), 3
.dot format, 42
dynamically downloaded data, 22–23
dynamic analysis, 25–34
  bag of features model, 63
dataset for, 222
for disassembly, 26
limitations of, 33–34
for malware data science, 26
typical malware behaviors, 27
using malwr.com, 26–33
analyzing results, 28–33
limitations, 33
loading files, 27–28
dynamic API call–based similarity, 72, 72
dynamic-link libraries (DLLs), 13

E
EAX register, 14
EBP register, 14
EBX register, 14
ECX register, 14
edges, 37
  adding attributes, 42
  adding to shared relationship networks, 41
  adding visual attributes to, 48–51
    color, 49, 49f
    text labels, 50–51
    width, 48–49, 48f
EDX register, 14
EFLAGS register, 15
EIP register, 14–15
ELU activation function, 179t
entry point, 3, 19
epochs parameter, 206
ESP register, 14
euclidean_distance function, 107
Euclidean distance, 107
evaluate function, 148
evaluate mode, 231–232
evaluating malware detection systems.
  See detection accuracy evaluation
export_graphviz function, 132
extract_features function, 204–205
ExtractImages helper class, 56–57

F
fakepdfmalware.exe, 7
false negatives, defined, 120, 120f
false positives, 120, 120f
  base rates and precision, 124–126
  false positive rate, 121
  relationship between true and false positive rates, 121–122,
    121f–122f
  ROC curves, 123–124, 123f
fdp tool, 43–45, 45f, 76
feature_extraction module, 129
feature extraction, 134–138
  Import Address Table features, 136
  machine learning–based malware detectors, 90–92, 141–142
  N-grams, 136–137
  Portable Executable header features, 135–136
  shared code analysis, 73, 75
  string features, 135
  training neural networks with Keras package, 203–204
  why all possible features can’t be used at once, 137–138
FeatureHasher class, 140–141
feature hashing. See hashing trick
feature spaces, 93–98, 94f–98f
feed-forward neural networks, 181, 181f, 193
fit_generator function, 204–206, 208, 212, 214
fit method, 130–131, 142
flags, defined, 15
format strings, 70
forward propagation, 189–190

G
Gaussian activation function, 179t
generative adversarial networks (GANs), 195–196
generator parameter, 206
get_database function, 80–82
get_string_features function, 141–142, 144
get_strings function, 82
get_training_data function, 143
get_training_paths function, 143
GETMAIL utility, 223
getstrings function, 73–74
-G flag, 44
gini index, 132, 132f
gradient descent, 105, 190
Graph constructor, 41, 52–53
graphical image analysis, 7–8
  converting extracted .ico files to .png graphics, 8
  creating directory to hold extracted images, 7–8
  extracting image resources using wrestool, 8
GraphViz, 76
decision tree–based detectors, 131–133, 132f
malware network analysis, 43–51
  adding visual attributes to nodes and edges, 48–51
  fdp tool, 44–45, 45f
  neato tool, 47–48, 47f
  parameters, 44
  sfdp tool, 46–47, 46f
  similarity graphs, 76
ground_truth variable, 130

H
hashing trick (feature hashing), 138–141
  complete code for, 139–140
  FeatureHasher class, 140–141
  implementing, 138–139
hidden layer, 181
histograms (bar charts), 168–170, 168f–169f
hostname_projection argument, 225
hyperplanes, 96, 97f

I
IAT. See Import Address Table
icoutils toolkit, 5
IDA Pro, 12
.idata section (imports) (in PE file format), 4
Identity activation function, 178f
Import Address Table (IAT), 4
  dumping using pefile, 6–7
  extracting features, 136
  similarity analysis based on, 71, 71f
import analysis, 6–7
inc arithmetic instruction, 15
information gain, 113
Input function, 200–201

instruction sequence–based similarity, 68f
  limitations of, 68–70
  overview, 67–68
Intel syntax, 13
Internet Relay Chat (IRC), 2
int function, 148
inverted indexing, 82
ircbot.exe bot, 2
  disassembling, 20–21
  dissecting, 5–7
  dumping IAT, 6–7
  strings analysis, 9–10

J
jaccard_index_threshold argument, 227–228
jaccard function, 73
jaccard index, 61, 65, 65f
  building similarity graphs, 73–75
  dynamic API call–based similarity, 72
  instruction sequence–based similarity, 68
  minhash method, 77–79
  scaling similarity comparisons, 77
  strings-based similarity, 70
jge instruction, 18
jmp instructions, 18
jointplot function, 171–172

K
Kaspersky, 62
Keras package, building neural networks with, 199–214
  compiling model, 202–203, 202f
  defining architecture of model, 200–202
  evaluating model, 209–211, 210f–211f
  layers, 200
  saving and loading model, 209
  syntaxes, 200
  training model, 203–209, 211–214
    built-in callbacks, 212
    custom callbacks, 213–214, 214f
    data generators, 204–207, 207f
    feature extraction, 203–204
    validation data, 207–209, 208f
  keyloggers, 158, 168f, 172f–173f, 173
**Index**

**KFold class**, 151–152
**K-fold cross-validation**, 151
**k-nearest neighbors**, 105–109, 106f, 108f
  identifying decision boundaries with, 97–98, 97f–98f
  logistic regression vs., 108–109
  math behind, 107
  pseudocode for, 107
  when to use, 109

**L**
**label attribute**, 50–51
**layers submodule**, 200–201
**lea instruction**, 16
**Leaky ReLU activation function**, 179f
**learned_parameters parameter**, 103
**linear disassembly**, 12
  limitation of, 12
  shared code analysis, 67–68
**LOAD mode**, 229
**logistic_function function**, 103–104, 104f
**logistic_regression function**, 103
**logistic regression**, 102–105, 103f–104f, 154
  gradient descent, 105
  identifying decision boundaries with, 96–97, 96f–97f
  k-nearest neighbors vs., 108–109
  limitation of, 102
  math behind, 103–104
  plot of logistic function, 104f
  pseudocode for, 103
  when to use, 105
**long short-term memory (LSTM) networks**, 196
**Los Alamos National Laboratory**, 41
**loss parameter**, 201–202

**M**
  building basic detectors, 129
  sample code, 133–134
  training, 130–131
  visualizing, 131–133, 132f
  building overview, 90–93
  collecting training examples, 90–91
  designing good features, 92
  extracting features, 90–92
  reasons for, 89–90
  testing system, 90, 93
  training system, 90, 92–93
  building real-world detectors, 141–146
  complete code for, 144–146
  feature extraction, 141–142
  running detector on new binaries, 144
  training, 142–143
  dataset for, 224
  decision boundaries, 93–98, 95f–98f
  evaluating detector performance, 146
  cross-validation, 150–153, 151f, 153f
  ROC curves, 147–150, 150f
  splitting data into training and test sets, 148–149
  feature extraction, 134–138
  Import Address Table features, 136
  N-grams, 136–137
  Portable Executable header features, 135–136
  string features, 135
  why all possible features can’t be used at once, 137–138
  feature spaces, 93–98, 94f–98f
  hashing trick, 138–141
  complete code for, 139–140
  FeatureHasher class, 140–141
  implementing, 138–139
  overfitting and underfitting, 98–99, 99f–101f
  supervised vs. unsupervised algorithms, 93
  terminology and concepts, 128–129
  tool for, 230–232, 231f
  traditional algorithms vs., 90
  types of algorithms, 101, 102f
    decision trees, 109–115, 109f–110f, 113f–114f
    k-nearest neighbors, 97–98, 97f–98f, 105–109, 106f, 108f
    logistic regression, 96–97, 96f–97f, 102–105, 103f–104f
    random forest, 115–116, 116f
  malware_projection argument, 52, 225–227
malware detection evaluation.  
See detection accuracy evaluation
malware network analysis, 35–58, 36f
attributes, defined, 37
bipartite networks, 37–39, 38f
creating shared callback
relationship network, 51–54,
225–226, 226f
code for, 52–54
importing modules, 51–52
parsing command line
arguments, 52
saving networks to disk, 54
creating shared image relationship
networks, 54–58, 55f,
226–227
extracting graphical assets, 57
parsing initial argument and
file-loading code, 55–57
saving networks to disk, 58
dataset for, 222–223
edges, defined, 37
GraphViz, creating visualizations
with, 43–51
fdp tool, 44–45, 45f
neato tool, 47–48, 47f
parameters, 44
sfdp tool, 46–47, 46f
visual attributes, 48–51
NetworkX library, creating
networks with, 40–43
adding attributes, 42
adding nodes and edges, 41
saving networks to disk, 42–43
nodes, defined, 37
projections, 38
shared code analysis and, 60–61
visualization challenges, 39–40
distortion problem, 39–40, 40f
force-directed algorithms, 40
network layout, 39–40
malware samples, 61–62, 222–224
malwr.com, 26–33, 28f
analyzing results on, 28–33
API calls, 32–33, 33f
modified system objects, 30–32
Screenshots panel, 30, 30f
Signatures panel, 29–30, 29f
Summary panel, 30–32, 31f–32f
limitations of, 33
loading files on, 27–28
Mandiant, 61, 76, 223
MAPIGET utility, 223
Mastercard, iii
matplotlib library, 148–150,
162–167, 162/
plotting ransomware and
worm detection rates,
165–167, 166f
plotting ransomware detection
rates, 164–165, 165f
plotting relationship between
malware size and detection,
162–163
max function, 160
mean function, 160–161
memory cells, 196
metrics module, 147–148
metrics parameter, 201–202
min function, 81, 160
minhash approach
combined with sketching, 79
math behind, 78–79, 78f
overview, 77–78
minhash function, 82
ModelCheckpoint callback, 212
Model class, 201
models submodule, 201–202
mov instruction, 15–16
murmur module, 80, 82
mutexes, defined, 32
my_generator function, 205, 207–208
MyCallback class, 213–214

N
neato tool, 47–48, 47f
Nemucod.FG malware family, 157, 157f
NetworkX library, 40–43
creating shared relationship
networks, 41–42
overview, 41
saving networks to disk, 42–43
neural networks, 176, 177–188
automatic feature generation, 188
building
with four neurons, 186–188,
186f–187f, 187t
with three neurons, 184–186,
185f–186f, 185t
with two neurons, 182–184,
182f–184f, 183t–184t
building with Keras package, 199–214
  compiling model, 202–203, 202f
  defining architecture of model, 200–202
  evaluating model, 209–211, 210f–211f
  saving and loading model, 209
  training model, 203–209, 211–214
dataset for, 224
neurons, 176
  anatomy of, 177–180, 177f, 178t–180t
  networks of, 180–181, 181f
training, 189–193
  using backpropagation, 190–192, 190f–191f
  using forward propagation, 189–190
  vanishing gradient problem, 192–193
types of, 193–197
  autoencoder, 194–195, 195f
  convolutional, 193–194, 194f
  feed-forward, 193
  generative adversarial, 195–196
  recurrent, 196
  ResNet, 196–197
universal approximation theorem, 181–182, 182f
neurons, 176
  anatomy of, 177–180, 177f, 178t–180t
  networks of, 180–181, 181f
next method, 205, 208
N-grams, 63–64, 64f
  dynamic API call–based similarity, 72
  extracting features, 136–137
  instruction sequence–based similarity, 67–68
nodes, 37
  adding attributes, 42
  adding to shared relationship networks, 41
  adding visual attributes to, 48–51
    color, 49, 49f
    shape, 49–50, 50f
    text labels, 50–51
    width, 48–49
  in decision trees, 110–111
NUM_MINHASHES constant, 80–81

O
objective function, 189
optimizer parameter, 201–202
optional header (in PE file format), 3–4
output_dot_file argument, 227–228
output_file argument, 52, 225, 227
overfit machine-learning models, 98–99, 101f
overlap parameter, 44

P
packing, 21
  difficulty of disassembling packed malware, 26
  legitimate uses of, 22
pandas package, 158–161
  filtering data using conditions, 161
  loading data, 158–159
  manipulating DataFrame, 159–161
Parkour, Mila, 61
pasta malware family, 62, 67f–68f, 70f–72f
PE. See Portable Executable file format
PE (Portable Executable) header, 3, 135–136
pecheck function, 73–74
pefile module, 5–7
  disassembly using, 20
  dumping IAT, 6–7
  installing, 5, 20
  opening and parsing files, 5–6
  pulling information from PE fields, 6
pefile PE parsing module, 51–52
penwidth attribute, 48–49
persistent malware similarity search systems, 79–87
building
  allowing users to search for and comment on samples, 82–84
  implementing database functionality, 80–81
  importing packages, 80
  indexing samples into system’s database, 82
  loading samples, 85
  obtaining minhashes and sketches, 81–82
  parsing user command line arguments, 84–85
Index

persistent malware similarity search systems, continued
commenting on samples, 86
sample output, 86–87
searching for similar samples, 86
wiping database, 86
pick_best_question function, 112–113
pickle module, 143–144
plot function, 162–163, 167
.png format, 43
pooling layer, 194
pop instruction, 16–17
Portable Executable (PE) file format, 2–5
dissecting files using pefile, 5–7
entry point, 3
file structure, 2–5, 3f
DOS header, 3
optional header, 3–4
PE header, 3
section headers, 4–5
sections, defined, 4
Portable Executable (PE) header, 3, 135–136
position independence, 5
precision, 124–126
effect of base rate on, 124–125
estimating in deployment environment, 125–126
predict_proba method, 144, 149
PReLU activation function, 179f
program stack, defined, 14
projected_graph function, 54
projections, 38
push instruction, 16–17
pyplot module, 148–149, 163

R
random forest
overview, 115–116, 116f
random forest–based detectors, 141–146
complete code for, 144–146
running detector on new binaries, 144
training, 142–143
RandomForestClassifier class, 143, 152
ransomware, 30–31, 31f, 155–158, 156f,
158, 164–168, 165f–166f,
168f, 172–173, 172f–173f/
.rdata section (in PE file format), 4
Receiver Operating Characteristic curves. See ROC curves
rectified linear unit (ReLU) activation function, 177f, 178t, 180,
182f, 183–185, 201
recurrent neural networks (RNNs), 196
registry keys, 32
.reloc section (in PE file format), 5
ReLU (rectified linear unit) activation function, 177f, 178t, 180,
182f, 183–185, 201
ResNets (residual networks), 196–197
resource_projection argument, 52, 227
resource obfuscation, 22
ret instruction, 17–18
reverse engineering, 12
anti-disassembly techniques, 22
dynamic analysis for, 26
methods for, 12
shared code analysis, 60
using pefile and capstone, 20–21
RNNs (recurrent neural networks), 196
ROC (Receiver Operating Characteristic) curves,
123–124, 123f, 126, 147–150,
230–231, 231f
computing, 147–150
cross-validation, 151–152, 153f
neural networks, 209–210,
210f–211f
visualizing, 149, 150f
roc_curve function, 149, 210
.rsrc section (resources) (in PE file format), 4–5

S
sandbox, 26
Sanders, Hillary, 216
savefig function, 165
scan_file function, 144
scan mode, 230–231
scikit-learn (sklearn) machine learning package, 127–128
building basic decision tree–based detectors, 129–134
building random forest–based detectors, 141–146
evaluating detector performance, 146–153
feature extraction, 134–135
hashing trick, 140–141
terminology and concepts, 128–129
classifiers, 129
fit, 129
label vectors, 128–129
prediction, 129
vectors, 128
seaborn package, 168–174, 168/
creating violin plots, 172–174, 172f–173f
plotting distribution of antivirus detections, 169–172, 169f, 171f
search_sample function, 82–84
SEARCH mode, 229
section headers (in PE file format), 4–5
.data section, 4
.idata section (imports), 4
.rdata section, 4
.reloc section, 5
.rssrc section (resources), 4–5
.text section, 4
security data scientists, 215–220
expanding knowledge of methods, 219–220
paths to becoming, 216
traits of effective, 218–219
curiosity, 218–219
obsession with results, 219
open-mindedness, 218
skepticism of results, 219
willingness to learn, 216
workflow of, 216–218, 217f
data feed identification, 218
dealing with stakeholders, 217
deployment, 218
problem identification, 217–218
solution building and evaluation, 218
self-modifying code, 12
set_axis_labels function, 172
sfdp tool, 46–47, 46f
shape attribute, 49–50
shared attribute analysis. See malware network analysis
shared code analysis (similarity analysis), 59–87, 60, 61f
bag of features model, 62–64, 63f
features, defined, 62
N-grams, 63–64, 64f
order information and, 63–64
overview, 62–63
dataset for, 223
Jaccard index, 64–65, 65f
persistent malware similarity search systems, 79–87
allowing users to search for and comment on samples, 82–84
commenting on samples, 86
implementing database functionality, 80–81
importing packages, 80
indexing samples into system database, 82
loading samples, 85
obtaining minhashes and sketches, 81–82
parsing user command line arguments, 84–85
sample output, 86–87
searching for similar samples, 86
wiping database, 86
scaling similarity comparisons, 77–79
difficulties with, 77
minhash method, 77–79, 78f
similarity graphs, 73–76, 76f
declaring utility functions, 73–74
extracting features, 73, 75
importing libraries, 73
iterating through pairs, 75
Jaccard index threshold, 73
parsing user’s command line arguments, 74
visualizing graphs, 76
similarity matrices, 66–72, 66f–67f
concept of, 66
dynamic API call–based similarity, 72, 72f
Import Address Table–based similarity, 71, 71f
instruction sequence–based similarity, 67–70, 68f
strings-based similarity, 70–71, 70f
tools for, 227–230, 228f
shared image relationship networks, 54–58, 55f, 226–227
extracting graphical assets, 57
parsing initial argument and file-loading code, 55–57
saving networks to disk, 58
shelve module, 80
show function, 152, 163, 165, 168
Sigmoid activation function, 180t, 201
sim_graph module, 80, 82
similarity analysis. See shared code analysis
similarity functions, 64–65
similarity graphs, 73–76, 76f
declaring utility functions, 73–74
extracting features, 73, 75
importing libraries, 73
iterating through pairs, 75
Jaccard index threshold, 73
parsing user’s command line arguments, 74
visualizing graphs, 76
similarity matrices, 66–72, 66f–67f
dynamic API call–based similarity, 72, 72f
Import Address Table–based similarity, 71, 71f
instruction sequence–based similarity, 67–70, 68f
strings-based similarity, 70–71, 70f
SKETCH_RATIO constant, 80, 82
sklearn. See scikit-learn machine learning package
skor malware family, 62, 67f–68f, 70f–72f
Softmax activation function, 180t
Sophos, 216
splines parameter, 44
split_regex expression, 203–204
stack, defined, 16
stack instructions, 16–17
stack management registers, 14–15
static malware analysis, 1–23
dataset for, 222
disassembly and reverse engineering, 12
methods for, 12
using pefile and capstone, 20–21
graphical image analysis, 7–8
imports analysis, 6–7
limitations of, 21–23
anti-disassembly techniques, 22
dynamically downloaded data, 22–23
packing, 21–22
resource obfuscation, 22
pefile module, 5–7
Portable Executable file format, 2–5
strings analysis, 8–10
std function, 160
Step activation function, 179t
steps_per_epoch parameter, 206
string_hash function, 81–82
strings
defined, 8
feature extraction, 135, 141–142
strings analysis, 8–10
analyzing printable strings, 8–10
information revealed through, 8
printing all strings in a file to terminal, 8–9
strings-based similarity, 70–71, 70f
strings tool, 8–10
sub arithmetic instruction, 15
summary function, 202–203, 202f
supernodes, 46
suspicious_calls parameter, 103–104
suspiciousness scores, 121–122,
121f–122f

T
Target, iii
target_directory argument, 227–228
target_path argument, 52, 225, 227
TensorFlow, 200, 207
.text section (in PE file format), 4
threat scores, 147
.todense method, 142
train_detector function, 143
training_examples variable, 130
transform method, 131, 140
tree module, 129
Trojans, 54–55, 55f, 158–161, 168f,
172f–173f, 173
true negatives, defined, 120, 120f
true positives, 120, 120f
base rates and precision, 124–126
relationship between true and false positive rates, 121–122,
121f–122f
ROC curves, 123–124, 123f
true positive rate, 121

U
underfit machine-learning models, 98–99, 99f
universal approximation theorem, 181–182, 182f
UPX packer, 29
validation_labels object, 210–211
validation_scores object, 210
vanishing gradient problem, 192–193
vbna malware family, 62, 67f–68f, 70f–72f
vectors, 128
violin plots, 172–174, 172f–173f
VirtualBox, vii–viii, 222
virtual size, 6
VirusTotal.com, 29, 59
visualization, 155–174
  basic machine learning–based
malware detectors, 131–133, 132f
dataset for, 224
importance of, 156–158, 157f
malware network analysis
  challenges to, 40f
  creating with GraphViz, 45f–47f
network analysis
  challenges to, 39–40
  creating with GraphViz, 43–51
ROC curves, 149, 150f, 152–153, 153f
shared code analysis, 76
using matplotlib, 162–167, 162f
  plotting ransomware and
worm detection rates, 165–167, 166f
plotting ransomware detection
  rates, 164–165, 165f
plotting relationship between
malware size and detection, 162–163
using pandas, 158–161
  filtering data using
conditions, 161
  loading data, 158–159
  manipulating DataFrame, 159–161
using seaborn, 168–174, 168f
  creating violin plots, 172–174, 172f–173f
plotting distribution of
antivirus detections, 169–172, 169f, 171f
webprefix malware family, 62, 67f–68f, 70f–72f
weight attribute, 37
weight parameter, 178, 181
Wells Fargo, iii
Wikipedia, 220
wipe_database function, 80–81
wipe mode, 229
work method, 57
worms, 158–161, 165–167, 166f, 168f, 172, 172f–173f
wrestool tool, 55
downloading, 8
  extracting image resources, 7–8
write_dot function, 42–43
x86 assembly language, 12–20
  arithmetic instructions, 15, 15f
  CPU registers, 13–15, 14f
  general-purpose registers, 13–14
  stack and control flow registers, 14–15
  data movement instructions, 15–20, 16f
  basic blocks and control flow
  graphs, 19–20, 19f
  control flow instructions, 17–18
  stack instructions, 16–17
dialects of, 13
shared code analysis, 67
xtoober malware family, 62, 67f–68f, 70f–72f
yield statement, 205
zango malware family, 62, 67f–68f, 70f–72f