Malware Classification Project Report

Andrea Mambretti

Northeastern University
College of Computer and Information Science
Boston, 20115, USA
mbr@ccs.neu.edu

Problem definition

In recent years, the malware growth and distribution reached unthinkable levels. A 2015 report [1] from TrendMicro reveals that almost 1 million samples are discovered on daily base. Given these numbers, it is easy to realize that a manual approach to handle this immense flow of new threats is absolutely unbearable. Therefore, since beginning of 2000 researcher have been trying to apply automatic techniques to help to catch up with this huge growth.

Several approaches has been proposed, covering the huge spectrum that goes from static to dynamic analysis. In malware analysis, static approaches tend to suffer because packing and encryption. Malware authors employ these techniques to prevent fast and reliable reverse engineering and or automatic analysis on the malware binary. On the other hand, dynamic analysis is instead able to deal with these techniques but suffer of code coverage issues. Most of the time malware relies on specific inputs or time conditions to trigger the malice behaviours and therefore the malicious paths may never be triggered.

Given the complexity of deployment of most of the dynamic or static analysis techniques, researchers have also tried to apply machine learning techniques to the process to easier the analysis and classification of the samples. In this case, researchers exploited the key idea that most of the samples discovered has commonalities with previous discovered samples. Machine learning can help to pre-filter malware based on certain features and classify them in the respective family leaving only a smaller subset of unknown malware to be analyzed to the analysts.

In general, it is fair to assume that the number of malware families is largely smaller compared the number of samples discovered. Most of the new samples represents updated version of what was already known in the past. This offer good ground for machine learning approaches. Previous work such as [4] applied supervised learning (classification) on malware. Others, such as [3] applied unsupervised learning methodology to cluster similar malware without having previous ground truth. All these works concentrate mostly on dynamic features due to the encryption and packing.

In this project, I tackle the problem of malware classification using a complete set of static features. This will allow a fast and inexpensive analysis to pre-filter malware in the respective families. For this work, I use the publicly available Microsoft Kaggle Database which offers a large collection (10868 samples) of malware divided in 9 families. For each sample, the database contains the bytes and the IDApro disassembled view of the malware.

Dataset Description

In this work, I used the publicly available malware dataset offered by Microsoft for the malware classification challenge competition held between February and April 2015. It contains 9 different families of malware showed in Table 1. The dataset contains a total of 10868 samples. The dataset contains also other 17GB of data which goal was to test the various algorithms during the competition. I do not use that part of the dataset in this project since unlabeled.

The dataset offers for each sample two distinct files. It contains a *bytes* file which provides the hexdump of the sample (without the PE header) and a IDApro disassembled file which contains the assembly code in Intel syntax and other useful information (e.g. section names, sub routine start and end etc.). Every sample in the database is not encrypted and not packed. I explicitly exploit this characteristic of the dataset to extract static features that otherwise I would not be able to use. I do not extract any dynamic features (e.g. time of execution or operation triggered by certain inputs) because every sample misses the PE header preventing me its execution.

Based on the Microsoft Malware Classification Challenge website, the database contains malware discovered by Microsoft's real-time detection anti-malware products on over 160M computers worldwide inspecting 700M computers monthly. This dataset represents a very unique point of start for machine learning experiments due to its nature and how the data were collected. All previous works in this field had common issues in their evaluation due to the scarcity of well labeled enough big datasets.

Num	**Family**	**# Samples**
1	Ramnit	1541
2	Lollipop	2478
3	Kelihos_ver3	2942
4	Vundo	475
5	Simda	42
6	Tracur	751
7	Kelihos_ver1	398
8	Obfuscator.ACY	1228
9	Gatak	1013

Table 1: This table shows the number of samples per families in the training Microsoft Malware Classification challenge dataset

Proposed Solution

Feature Selection

Given the dataset and its characteristics described in the previous paragraph, I decided to approach this classification problem trying to define the best features set. The first common element I found looking at the malware of the same family is the size (in bytes) of the malware. The size generally changes of few KB at the maximum in case of samples from the same family meanwhile among families this value changes up to an order of magnitude. A second feature that I discovered to be very specific of the malware family is the strings (sequence of printable characters within the file).

The other features I took into account are related to information contained in the IDApro file provided. More specifically, I counted the number of occurrences of certain common assembly operations such as movs, jmps, calls. Furthermore, I used as feature the sections used within the binary (e.g. .text, .idata, .rdata etc.). Some of these section are actually common but others are very compiler and program specific therefore they tend to characterize file compiled with the same compiler and that shares certain information. Finally, I counted the numer of sub routines for each sample. Also this number is characteristic of the functions implemented within each sample and I would expect this value to change only slightly among samples of the same family.

Num	Feature	Type
1	Strings	Sparse boolean vector
2	Sections	Sparse boolean vector
3	File Size	Integer
4	Subroutine Count	Integer
5	Calls op. Count	Integer
6	Mov op. Count	Integer
7	Jmps op. Count	Integer
8	Pop op. Count	Integer
9	Push op. Count	Integer
10	Xor op. Count	Integer
11	Sub op. Count	Integer
12	Add op. Count	Integer

Table 2: Features collected for each sample

After the collection of all the strings and sections and their conversion to sparser boolean vectors, the features vector for each samples counts a total of 410318 features. Furthermore, I normalized all the integer values in the features vector.

Machine Learning Algorithms

The problem that I'm trying to solve with this project requires multi-class algorithm classifier. Based on the dataset, I have 9 different labels which translates on 9 different classes of malware that I aim to classify. Among the algorithms that support multi-class classification, I choose the following three:

- Support Vector Machines (SVM)
- Random Forest
- Neural Network

One of the goals of the project is comparing the performance and accuracy of the above listed classification methodology. To do so, I trained multiple times, with various settings, each of the classifiers trying to find the best configuration for the features selected. The research of the best setting was made easier thanks to the Grid Research offered by scikit-learn python package which given a dictionary of possible options perform the training for all the combinations among the specified parameters. In the result section, I will show the best settings and the results comparison for the algorithms. For the research of the best settings, I divided the training set in two, I used 33.3% of the dataset for testing and the remaining 66.6% for the training.

Challenges

During the collection of the features, I encountered one major challenge mostly related to the size of the dataset. I will try to describe it in the following paragraph.

The issue I came across is extracting in a proper way the string within the hexdump. The main operations involved for the extraction are string replacement and 2bytes x 2bytes conversion to ascii characters and verification if they belongs to the printable characters. I had very poor performance with both pypy and python such that the its application to all the dataset would have required more than a day. A deeper analysis of the algorithm showed that the first operation was faster in python (about 10x faster than pypy) meanwhile the second operation was a way faster in pypy (8x faster than python). Therefore, based on this observation, I decided to split the script in two pieces and the overall execution time came down to 30 minutes on the whole dataset.

It worth noting that since the nature of the operations (not data dependent) to extract the various features it was possible to largely exploit parallelism. To do so, I mainly relied on the multi-processing python library which allowed me to spawn multiple worker at the time each one on different samples. Then, I collected the results through a callback method which was serialized and allowed me to update the data structure that became my feature matrix.

Results

In this section, I present the results of the project. For each classifier, I show the various configuration tried and which one is providing me the best performance. Lastly, I present a comparison among

the various algorithms used. During the research for the best configuration the dataset is divided in 33% test set and 66% training set. Meanwhile, the test for comparison among the best configuration for each classifier uses 10-fold for cross-validating the results.

Support Vector Machine

During the research of best configuration, I *brute-forced* some of the parameters. Table 3 shows exactly which parameters were tried for SVM.

Parameter	**Values**
kernel	rbf, linear
gamma	1e-3, 3e-4
C	1, 10, 100, 1000

Table 3: Settings tried for SVM best setting research

As result of this search, I found that the best configuration for SVM based on my features is:

```
'kernel' : 'rbf',
'C' : 100,
'gamma' : 0.0001
```

In the appendix A, I show the score given to also the others configuration tried. Hereafter, I report the results on the test set for this configuration.

Class	**Precision**	**Recall**	**F1-score**	**Support**
1	0.98	0.99	0.98	518
2	1.00	1.00	1.00	844
3	1.00	1.00	1.00	956
4	0.90	0.98	0.94	145
5	1.00	0.87	0.93	15
6	0.98	0.99	0.98	248
7	1.00	0.97	0.99	139
8	0.98	0.96	0.97	400
9	0.99	1.00	1.00	322
Tot	0.99	0.99	0.99	3587

Table 4: Results of the SVM best settings on the test dataset

Random Forest

In the Random Forest case, I *brute-foced* the parameters in Table 5.

Parameter	Values
max_depth	3, None
min_samples_split	1,3,10
min_samples_leaf	2,3,10
bootstrap	True, False
criterion	gini, entroy

Table 5: Settings tried for Random Forest best settings research

As result of this search, I found that the best configuration for Random Forest given my features is:

```
'bootstrap': False,
'min_samples_leaf':1,
'min_samples_split':2,
'criterion':'gini',
'max_features': 3,
'max_depth': None
```

In Table 6, I report the results on the test set for this configuration.

Class	Precision	Recall	F1-score	Support
1	0.93	0.98	0.96	518
2	0.99	0.99	0.99	844
3	1.00	0.99	1.00	956
4	0.90	0.98	0.94	145
5	0.90	0.60	0.72	15
6	1.00	0.99	0.97	248
7	0.98	0.96	0.97	139
8	0.98	0.92	0.95	400
9	0.99	1.00	0.99	322
Tot	0.98	0.98	0.98	3587

Table 6: Results of the Random Forest best settings on the test dataset

Neural Network

For Neural Networks, Table 7 shows which combinations of parameters I tried.

Parameters	Values
'Solver'	lbfgs, sgd, adam
'Alpha'	0.0001, 0.001, 0.01, 0.1, 0.9
'Activation'	Identity, Logistic, Tanh, Relu
'Learning_rate'	Constant, Invscaling, Adaptive

Table 7: Settings tried for Neural Network best settings research

As result of this search, I found that the best configuration is

```
'alpha': 0.01,
'activation':'identity',
'learning_rate':'adaptive',
'solver'= 'adam'
```

In Table 8, I report the results on the test set for this configuration.

Class	Precision	Recall	F1-score	Support
1	0.97	0.95	0.96	518
2	0.99	0.99	0.99	844
3	1.00	0.99	0.99	956
4	0.79	0.98	0.87	145
5	0.00	0.00	0.00	15
6	0.97	0.98	0.98	248
7	0.96	0.95	0.95	139
8	0.99	0.92	0.95	400
9	0.93	0.98	0.95	322
Tot	0.97	0.97	0.97	3587

Table 8: Results of the Neural Network best settings on the test dataset

Cross-validation and methods comparison

After the research of the best configuration for each classifier used, I performed 10k-fold validation on each of the classifier in its best settings. Hereafter, I will show the accuracy and F1 score results from the 10-fold per classifier.

Classifier	Accuracy	F1-score
SVM	0.9887 (+/- 0.0077)	0.9888 (+/- 0.0077)
Neural Network	0.9762 (+/- 0.00618)	0.9763 (+/- 0.0090)
Random Forest	0.9850 (+/- 0.0064)	0.9851 (+/- 0.0068)

Table 9: Cross-validation results, the values represents the mean +/- std*2 among the 10 fold

Given the results in Table 9, I can fairly assume that the set of features used for classification well represents the similarity among samples of malware of the same family. Furthermore, comparing the various classifiers, it is pretty clear that svm gives the best results even though this may be a sign of over-fitting. Random forest give slightly worse results but they are somehow comparable with SVM. Meanwhile, neural network at its best settings give the worst results. Also, in comparison on time needed for time neural network needs a considerable longer to be trained on this dataset.

Related Work

The main differences between this work and others is related to the features selected. Generally, during malware classification most of the previous work such as [4] uses dynamic analyzed features of the malware because elements such as strings and instructions are not available due to encryption and packing. The accuracy given using only static features is generally higher because the features extraction can be done in a more reliable manner.

For instance, dynamic features based on the system calls that the program calls during its execution really depends on which path that specific sample is going to take. This may strongly vary between each run. Therefore, this makes the features extracted less reliable and the whole analysis less accurate. Of course, as said, the Microsoft Malware Classification Challenge dataset is a very particular dataset which is very hard to generate and collect.

Some other works exist on the same dataset. Most of them were done during the competition. The competition winners' approach [2] was different from the one I chose. They considered as features opcode count using 2,3,4-grams, the segments count, and the asm pixel intensity as they gold features. In particular the last feature is very characteristic of their approach that takes the bytes of each sample and transform them into an image and then they evaluate the pixel intensity.

Conclusion

In this project, I tried to apply machine learning to a malware classification problem. To achieve my goal, I relied on only static features that were easily extractable from the dataset. I trained three different classifiers and tried to compare their results in their best settings. The results show a good accuracy for all the classifiers even though SVM generally showed to be the best fit for the problem. The results were computed applying 10-fold cross-validation.

Appendix A

Support Vector Machine

Grid scores on development set:

```
0.847 (+/-0.016) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
0.818 (+/-0.015) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
0.975 (+/-0.033) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
0.853 (+/-0.015) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
0.971 (+/-0.024) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
0.976 (+/-0.031) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
0.971 (+/-0.024) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
0.971 (+/-0.024) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
0.970 (+/-0.025) for {'kernel': 'linear', 'C': 1}
0.970 (+/-0.025) for {'kernel': 'linear', 'C': 10}
0.970 (+/-0.025) for {'kernel': 'linear', 'C': 100}
0.970 (+/-0.025) for {'kernel': 'linear', 'C': 1000}
```

Random Forest

Grid scores on development set:

```
0.294 (+/-0.006) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.294 (+/-0.006) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.294 (+/-0.006) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.283 (+/-0.005) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.293 (+/-0.008) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.293 (+/-0.008) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
```

```
0.281 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.281 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.301 (+/-0.005) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.301 (+/-0.005) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.301 (+/-0.005) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.287 (+/-0.012) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.287 (+/-0.012) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.287 (+/-0.012) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.274 (+/-0.002) for {'bootstrap': True, 'min_samples_leaf': 10, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.285 (+/-0.010) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 10, '
    max_depth': 3}
```

```
0.285 (+/-0.010) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 10, '
    max_depth': 3}
0.287 (+/-0.011) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 10,
    'max_depth': 3}
0.288 (+/-0.007) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 10, '
    max_depth': 3}
0.288 (+/-0.007) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 10, '
    max_depth': 3}
0.288 (+/-0.007) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 10,
    'max_depth': 3}
0.274 (+/-0.003) for {'bootstrap': True, 'min_samples_leaf': 10, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 10, '
    max_depth': 3}
0.274 (+/-0.003) for {'bootstrap': True, 'min_samples_leaf': 10, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 10, '
    max_depth': 3}
0.274 (+/-0.003) for {'bootstrap': True, 'min_samples_leaf': 10, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 10,
    'max_depth': 3}
0.975 (+/-0.004) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 1, '
    max_depth': None}
0.975 (+/-0.007) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 1, '
    max_depth': None}
0.972 (+/-0.004) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 1, '
    max_depth': None}
0.287 (+/-0.004) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 1, '
    max_depth': None}
0.284 (+/-0.007) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 1, '
    max_depth': None}
```

```
0.281 (+/-0.001) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 1, '
   max_depth': None}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 1, '
   max_depth': None}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 1, '
   max_depth': None}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 1, '
   max_depth': None}
0.977 (+/-0.005) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
0.972 (+/-0.008) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
0.968 (+/-0.005) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
0.297 (+/-0.012) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
0.306 (+/-0.022) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
0.298 (+/-0.004) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
```

```
0.977 (+/-0.008) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 10, '
   max_depth': None}
0.975 (+/-0.004) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 10, '
   max_depth': None}
0.970 (+/-0.009) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 10,
   'max_depth': None}
0.483 (+/-0.050) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 10, '
   max_depth': None}
0.482 (+/-0.060) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 10, '
   max_depth': None}
0.491 (+/-0.052) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 10,
   'max_depth': None}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 10, '
   max_depth': None}
0.286 (+/-0.038) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 10, '
   max_depth': None}
0.279 (+/-0.011) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 10,
   'max_depth': None}
0.297 (+/-0.010) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': 3}
0.297 (+/-0.008) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': 3}
0.296 (+/-0.013) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   1, 'max_depth': 3}
0.284 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': 3}
```

```
0.284 (+/-0.009) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': 3}
0.286 (+/-0.006) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   1, 'max_depth': 3}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': 3}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': 3}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   1, 'max_depth': 3}
0.300 (+/-0.002) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 3,
    'max_depth': 3}
0.293 (+/-0.010) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 3,
    'max_depth': 3}
0.299 (+/-0.005) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   3, 'max_depth': 3}
0.300 (+/-0.031) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 3,
    'max_depth': 3}
0.303 (+/-0.041) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 3,
    'max_depth': 3}
0.291 (+/-0.009) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   3, 'max_depth': 3}
0.286 (+/-0.037) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 3,
    'max_depth': 3}
0.286 (+/-0.039) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 3,
    'max_depth': 3}
```

```
0.283 (+/-0.030) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   3, 'max_depth': 3}
0.293 (+/-0.008) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': 3}
0.311 (+/-0.047) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': 3}
0.316 (+/-0.058) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': 3}
0.313 (+/-0.063) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': 3}
0.307 (+/-0.058) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': 3}
0.313 (+/-0.063) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': 3}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': 3}
0.278 (+/-0.014) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': 3}
0.276 (+/-0.010) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': 3}
0.975 (+/-0.008) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': None}
0.975 (+/-0.007) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': None}
0.969 (+/-0.005) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   1, 'max_depth': None}
```

```
0.286 (+/-0.016) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features': 1,
     'max_depth': None}
0.280 (+/-0.008) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features': 1,
     'max_depth': None}
0.287 (+/-0.009) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 10, 'criterion': 'entropy', 'max_features':
    1, 'max_depth': None}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features': 1,
     'max_depth': None}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features': 1,
     'max_depth': None}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
    min_samples_split': 10, 'criterion': 'entropy', 'max_features':
    1, 'max_depth': None}
0.975 (+/-0.004) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features': 3,
     'max_depth': None}
0.976 (+/-0.007) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features': 3,
     'max_depth': None}
0.971 (+/-0.004) for {'bootstrap': True, 'min_samples_leaf': 1, '
    min_samples_split': 10, 'criterion': 'entropy', 'max_features':
    3, 'max_depth': None}
0.294 (+/-0.008) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features': 3,
     'max_depth': None}
0.297 (+/-0.010) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features': 3,
     'max_depth': None}
0.307 (+/-0.043) for {'bootstrap': True, 'min_samples_leaf': 3, '
    min_samples_split': 10, 'criterion': 'entropy', 'max_features':
    3, 'max_depth': None}
0.273 (+/-0.000) for {'bootstrap': True, 'min_samples_leaf': 10, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features': 3,
     'max_depth': None}
```

```
0.273 (+/-0.001) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 3,
    'max_depth': None}
0.273 (+/-0.002) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   3, 'max_depth': None}
0.976 (+/-0.006) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': None}
0.974 (+/-0.004) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': None}
0.972 (+/-0.001) for {'bootstrap': True, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': None}
0.517 (+/-0.070) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': None}
0.506 (+/-0.016) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': None}
0.479 (+/-0.121) for {'bootstrap': True, 'min_samples_leaf': 3, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': None}
0.275 (+/-0.007) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': None}
0.288 (+/-0.036) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': None}
0.276 (+/-0.009) for {'bootstrap': True, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': None}
0.292 (+/-0.003) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 1, '
   max_depth': 3}
0.294 (+/-0.005) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 1, '
   max_depth': 3}
```

```
0.290 (+/-0.012) for {'bootstrap': False, 'min_samples_leaf': 1, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.286 (+/-0.010) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.290 (+/-0.003) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.290 (+/-0.006) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.273 (+/-0.001) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.273 (+/-0.000) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.273 (+/-0.000) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 1, '
    max_depth': 3}
0.300 (+/-0.012) for {'bootstrap': False, 'min_samples_leaf': 1, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.299 (+/-0.012) for {'bootstrap': False, 'min_samples_leaf': 1, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.300 (+/-0.010) for {'bootstrap': False, 'min_samples_leaf': 1, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.294 (+/-0.008) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.301 (+/-0.006) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
0.296 (+/-0.008) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 3, '
    max_depth': 3}
```

```
0.274 (+/-0.001) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 3, '
   max_depth': 3}
0.273 (+/-0.000) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 3, '
   max_depth': 3}
0.274 (+/-0.002) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 3, '
   max_depth': 3}
0.327 (+/-0.010) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 10, '
   max_depth': 3}
0.298 (+/-0.011) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 10, '
   max_depth': 3}
0.297 (+/-0.003) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 10,
   'max_depth': 3}
0.304 (+/-0.027) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 10, '
   max_depth': 3}
0.305 (+/-0.028) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 10, '
   max_depth': 3}
0.314 (+/-0.030) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 10,
   'max_depth': 3}
0.277 (+/-0.007) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 10, '
   max_depth': 3}
0.276 (+/-0.008) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 10, '
   max_depth': 3}
0.288 (+/-0.038) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 10,
   'max_depth': 3}
0.979 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 1, '
   max_depth': None}
```

```
0.977 (+/-0.008) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 1, '
   max_depth': None}
0.975 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 1, '
   max_depth': None}
0.291 (+/-0.007) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 1, '
   max_depth': None}
0.288 (+/-0.019) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 1, '
   max_depth': None}
0.292 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 1, '
   max_depth': None}
0.273 (+/-0.000) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 1, '
   max_depth': None}
0.273 (+/-0.000) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 1, '
   max_depth': None}
0.273 (+/-0.000) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 1, '
   max_depth': None}
0.979 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
0.978 (+/-0.003) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
0.975 (+/-0.005) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
0.325 (+/-0.038) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 2, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
0.327 (+/-0.021) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 3, 'criterion': 'gini', 'max_features': 3, '
   max_depth': None}
```

```
0.311 (+/-0.011) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 3, '
    max_depth': None}
0.276 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 3, '
    max_depth': None}
0.275 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 3, '
    max_depth': None}
0.287 (+/-0.039) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 3, '
    max_depth': None}
0.977 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 1, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 10, '
    max_depth': None}
0.978 (+/-0.006) for {'bootstrap': False, 'min_samples_leaf': 1, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 10, '
    max_depth': None}
0.974 (+/-0.007) for {'bootstrap': False, 'min_samples_leaf': 1, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 10,
    'max_depth': None}
0.620 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 10, '
    max_depth': None}
0.612 (+/-0.070) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 10, '
    max_depth': None}
0.616 (+/-0.055) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 10,
    'max_depth': None}
0.277 (+/-0.003) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 2, 'criterion': 'gini', 'max_features': 10, '
    max_depth': None}
0.304 (+/-0.038) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 3, 'criterion': 'gini', 'max_features': 10, '
    max_depth': None}
0.277 (+/-0.005) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 10, 'criterion': 'gini', 'max_features': 10,
    'max_depth': None}
```

```
0.292 (+/-0.007) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': 3}
0.297 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': 3}
0.296 (+/-0.008) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   1, 'max_depth': 3}
0.284 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': 3}
0.287 (+/-0.011) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': 3}
0.287 (+/-0.003) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   1, 'max_depth': 3}
0.274 (+/-0.003) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': 3}
0.273 (+/-0.001) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': 3}
0.273 (+/-0.000) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   1, 'max_depth': 3}
0.297 (+/-0.006) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 3,
    'max_depth': 3}
0.297 (+/-0.002) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 3,
    'max_depth': 3}
0.318 (+/-0.050) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   3, 'max_depth': 3}
0.297 (+/-0.001) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 3,
    'max_depth': 3}
```

```
0.343 (+/-0.131) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features': 3,
     'max_depth': 3}
0.298 (+/-0.006) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 10, 'criterion': 'entropy', 'max_features':
    3, 'max_depth': 3}
0.274 (+/-0.002) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features': 3,
     'max_depth': 3}
0.273 (+/-0.001) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features': 3,
     'max_depth': 3}
0.273 (+/-0.001) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 10, 'criterion': 'entropy', 'max_features':
    3, 'max_depth': 3}
0.308 (+/-0.014) for {'bootstrap': False, 'min_samples_leaf': 1, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': 3}
0.353 (+/-0.112) for {'bootstrap': False, 'min_samples_leaf': 1, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': 3}
0.294 (+/-0.003) for {'bootstrap': False, 'min_samples_leaf': 1, '
    min_samples_split': 10, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': 3}
0.306 (+/-0.025) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': 3}
0.298 (+/-0.009) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': 3}
0.297 (+/-0.007) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 10, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': 3}
0.286 (+/-0.035) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': 3}
0.278 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': 3}
```

```
0.292 (+/-0.033) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   10, 'max_depth': 3}
0.976 (+/-0.001) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': None}
0.977 (+/-0.006) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': None}
0.974 (+/-0.007) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   1, 'max_depth': None}
0.289 (+/-0.005) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': None}
0.297 (+/-0.005) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': None}
0.293 (+/-0.008) for {'bootstrap': False, 'min_samples_leaf': 3, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   1, 'max_depth': None}
0.274 (+/-0.003) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': None}
0.273 (+/-0.001) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 1,
    'max_depth': None}
0.273 (+/-0.001) for {'bootstrap': False, 'min_samples_leaf': 10, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   1, 'max_depth': None}
0.978 (+/-0.007) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 2, 'criterion': 'entropy', 'max_features': 3,
    'max_depth': None}
0.978 (+/-0.003) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 3, 'criterion': 'entropy', 'max_features': 3,
    'max_depth': None}
0.975 (+/-0.006) for {'bootstrap': False, 'min_samples_leaf': 1, '
   min_samples_split': 10, 'criterion': 'entropy', 'max_features':
   3, 'max_depth': None}
```

```
0.358 (+/-0.143) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features': 3,
     'max_depth': None}
0.331 (+/-0.022) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features': 3,
     'max_depth': None}
0.317 (+/-0.006) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 10, 'criterion': 'entropy', 'max_features':
    3, 'max_depth': None}
0.276 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features': 3,
     'max_depth': None}
0.274 (+/-0.002) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features': 3,
     'max_depth': None}
0.274 (+/-0.002) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 10, 'criterion': 'entropy', 'max_features':
    3, 'max_depth': None}
0.979 (+/-0.007) for {'bootstrap': False, 'min_samples_leaf': 1, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': None}
0.978 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 1, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': None}
0.975 (+/-0.006) for {'bootstrap': False, 'min_samples_leaf': 1, '
    min_samples_split': 10, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': None}
0.595 (+/-0.056) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': None}
0.596 (+/-0.052) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': None}
0.616 (+/-0.027) for {'bootstrap': False, 'min_samples_leaf': 3, '
    min_samples_split': 10, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': None}
0.291 (+/-0.031) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 2, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': None}
```

```
0.279 (+/-0.004) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 3, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': None}
0.292 (+/-0.046) for {'bootstrap': False, 'min_samples_leaf': 10, '
    min_samples_split': 10, 'criterion': 'entropy', 'max_features':
    10, 'max_depth': None}
```

Neural Network

Grid scores on development set:

```
0.964 (+/-0.000) for {'alpha': 0.0001, 'activation': 'identity', '
    learning_rate': 'constant', 'solver': 'lbfgs'}
0.964 (+/-0.000) for {'alpha': 0.0001, 'activation': 'identity', '
    learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.964 (+/-0.000) for {'alpha': 0.0001, 'activation': 'identity', '
    learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.965 (+/-0.001) for {'alpha': 0.001, 'activation': 'identity', '
    learning_rate': 'constant', 'solver': 'lbfgs'}
0.965 (+/-0.001) for {'alpha': 0.001, 'activation': 'identity', '
    learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.965 (+/-0.001) for {'alpha': 0.001, 'activation': 'identity', '
    learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.966 (+/-0.002) for {'alpha': 0.01, 'activation': 'identity', '
    learning_rate': 'constant', 'solver': 'lbfgs'}
0.966 (+/-0.002) for {'alpha': 0.01, 'activation': 'identity', '
    learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.966 (+/-0.002) for {'alpha': 0.01, 'activation': 'identity', '
    learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.966 (+/-0.002) for {'alpha': 0.1, 'activation': 'identity', '
    learning_rate': 'constant', 'solver': 'lbfgs'}
0.966 (+/-0.002) for {'alpha': 0.1, 'activation': 'identity', '
    learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.966 (+/-0.002) for {'alpha': 0.1, 'activation': 'identity', '
    learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.965 (+/-0.005) for {'alpha': 0.9, 'activation': 'identity', '
    learning_rate': 'constant', 'solver': 'lbfgs'}
0.965 (+/-0.005) for {'alpha': 0.9, 'activation': 'identity', '
    learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.965 (+/-0.005) for {'alpha': 0.9, 'activation': 'identity', '
    learning_rate': 'adaptive', 'solver': 'lbfgs'}
```

```
0.919 (+/-0.022) for {'alpha': 0.0001, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.919 (+/-0.022) for {'alpha': 0.0001, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.919 (+/-0.022) for {'alpha': 0.0001, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.921 (+/-0.026) for {'alpha': 0.001, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.921 (+/-0.026) for {'alpha': 0.001, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.921 (+/-0.026) for {'alpha': 0.001, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.920 (+/-0.021) for {'alpha': 0.01, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.920 (+/-0.021) for {'alpha': 0.01, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.920 (+/-0.021) for {'alpha': 0.01, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.927 (+/-0.008) for {'alpha': 0.1, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.927 (+/-0.008) for {'alpha': 0.1, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.927 (+/-0.008) for {'alpha': 0.1, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.960 (+/-0.027) for {'alpha': 0.9, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.960 (+/-0.027) for {'alpha': 0.9, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.960 (+/-0.027) for {'alpha': 0.9, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.888 (+/-0.005) for {'alpha': 0.0001, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.888 (+/-0.005) for {'alpha': 0.0001, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.888 (+/-0.005) for {'alpha': 0.0001, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.863 (+/-0.039) for {'alpha': 0.001, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
```

```
0.863 (+/-0.039) for {'alpha': 0.001, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.863 (+/-0.039) for {'alpha': 0.001, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.886 (+/-0.027) for {'alpha': 0.01, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.886 (+/-0.027) for {'alpha': 0.01, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.886 (+/-0.027) for {'alpha': 0.01, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.912 (+/-0.013) for {'alpha': 0.1, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.912 (+/-0.013) for {'alpha': 0.1, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.912 (+/-0.013) for {'alpha': 0.1, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.953 (+/-0.044) for {'alpha': 0.9, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.953 (+/-0.044) for {'alpha': 0.9, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.953 (+/-0.044) for {'alpha': 0.9, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.0001, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.0001, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.0001, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.001, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.001, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.001, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.01, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.01, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
```

```
0.273 (+/-0.000) for {'alpha': 0.01, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.1, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.1, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.1, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.9, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.9, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'lbfgs'}
0.273 (+/-0.000) for {'alpha': 0.9, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.963 (+/-0.009) for {'alpha': 0.0001, 'activation': 'identity', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.151 (+/-0.005) for {'alpha': 0.0001, 'activation': 'identity', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.963 (+/-0.009) for {'alpha': 0.0001, 'activation': 'identity', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.963 (+/-0.009) for {'alpha': 0.001, 'activation': 'identity', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.151 (+/-0.005) for {'alpha': 0.001, 'activation': 'identity', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.963 (+/-0.009) for {'alpha': 0.001, 'activation': 'identity', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.963 (+/-0.009) for {'alpha': 0.01, 'activation': 'identity', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.151 (+/-0.005) for {'alpha': 0.01, 'activation': 'identity', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.963 (+/-0.009) for {'alpha': 0.01, 'activation': 'identity', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.963 (+/-0.009) for {'alpha': 0.1, 'activation': 'identity', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.151 (+/-0.005) for {'alpha': 0.1, 'activation': 'identity', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.963 (+/-0.009) for {'alpha': 0.1, 'activation': 'identity', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
```

```
0.962 (+/-0.008) for {'alpha': 0.9, 'activation': 'identity', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.151 (+/-0.005) for {'alpha': 0.9, 'activation': 'identity', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.962 (+/-0.008) for {'alpha': 0.9, 'activation': 'identity', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.0001, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.095 (+/-0.000) for {'alpha': 0.0001, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.0001, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.001, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.095 (+/-0.000) for {'alpha': 0.001, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.001, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.01, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.095 (+/-0.000) for {'alpha': 0.01, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.01, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.1, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.095 (+/-0.000) for {'alpha': 0.1, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.1, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.9, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.095 (+/-0.000) for {'alpha': 0.9, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.9, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.822 (+/-0.005) for {'alpha': 0.0001, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'sgd'}
```

```
0.121 (+/-0.001) for {'alpha': 0.0001, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.822 (+/-0.005) for {'alpha': 0.0001, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.822 (+/-0.005) for {'alpha': 0.001, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.121 (+/-0.001) for {'alpha': 0.001, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.822 (+/-0.005) for {'alpha': 0.001, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.822 (+/-0.005) for {'alpha': 0.01, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.121 (+/-0.001) for {'alpha': 0.01, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.822 (+/-0.005) for {'alpha': 0.01, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.821 (+/-0.006) for {'alpha': 0.1, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.121 (+/-0.001) for {'alpha': 0.1, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.821 (+/-0.006) for {'alpha': 0.1, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.814 (+/-0.010) for {'alpha': 0.9, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.121 (+/-0.001) for {'alpha': 0.9, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.814 (+/-0.010) for {'alpha': 0.9, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.0001, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.095 (+/-0.000) for {'alpha': 0.0001, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.0001, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.001, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.095 (+/-0.000) for {'alpha': 0.001, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
```

```
0.273 (+/-0.000) for {'alpha': 0.001, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.01, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.095 (+/-0.000) for {'alpha': 0.01, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.01, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.1, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.095 (+/-0.000) for {'alpha': 0.1, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.1, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.9, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'sgd'}
0.095 (+/-0.000) for {'alpha': 0.9, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'sgd'}
0.273 (+/-0.000) for {'alpha': 0.9, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'sgd'}
0.965 (+/-0.007) for {'alpha': 0.0001, 'activation': 'identity', '
   learning_rate': 'constant', 'solver': 'adam'}
0.965 (+/-0.007) for {'alpha': 0.0001, 'activation': 'identity', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.965 (+/-0.007) for {'alpha': 0.0001, 'activation': 'identity', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.965 (+/-0.007) for {'alpha': 0.001, 'activation': 'identity', '
   learning_rate': 'constant', 'solver': 'adam'}
0.965 (+/-0.007) for {'alpha': 0.001, 'activation': 'identity', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.965 (+/-0.007) for {'alpha': 0.001, 'activation': 'identity', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.967 (+/-0.008) for {'alpha': 0.01, 'activation': 'identity', '
   learning_rate': 'constant', 'solver': 'adam'}
0.967 (+/-0.008) for {'alpha': 0.01, 'activation': 'identity', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.967 (+/-0.008) for {'alpha': 0.01, 'activation': 'identity', '
   learning_rate': 'adaptive', 'solver': 'adam'}
```

```
0.966 (+/-0.004) for {'alpha': 0.1, 'activation': 'identity', '
   learning_rate': 'constant', 'solver': 'adam'}
0.966 (+/-0.004) for {'alpha': 0.1, 'activation': 'identity', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.966 (+/-0.004) for {'alpha': 0.1, 'activation': 'identity', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.958 (+/-0.008) for {'alpha': 0.9, 'activation': 'identity', '
   learning_rate': 'constant', 'solver': 'adam'}
0.958 (+/-0.008) for {'alpha': 0.9, 'activation': 'identity', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.958 (+/-0.008) for {'alpha': 0.9, 'activation': 'identity', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.894 (+/-0.008) for {'alpha': 0.0001, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'adam'}
0.894 (+/-0.008) for {'alpha': 0.0001, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.894 (+/-0.008) for {'alpha': 0.0001, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.834 (+/-0.002) for {'alpha': 0.001, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'adam'}
0.834 (+/-0.002) for {'alpha': 0.001, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.834 (+/-0.002) for {'alpha': 0.001, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.834 (+/-0.002) for {'alpha': 0.01, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'adam'}
0.834 (+/-0.002) for {'alpha': 0.01, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.834 (+/-0.002) for {'alpha': 0.01, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.761 (+/-0.089) for {'alpha': 0.1, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'adam'}
0.761 (+/-0.089) for {'alpha': 0.1, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.761 (+/-0.089) for {'alpha': 0.1, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.725 (+/-0.006) for {'alpha': 0.9, 'activation': 'logistic', '
   learning_rate': 'constant', 'solver': 'adam'}
```

```
0.725 (+/-0.006) for {'alpha': 0.9, 'activation': 'logistic', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.725 (+/-0.006) for {'alpha': 0.9, 'activation': 'logistic', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.807 (+/-0.265) for {'alpha': 0.0001, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'adam'}
0.807 (+/-0.265) for {'alpha': 0.0001, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.807 (+/-0.265) for {'alpha': 0.0001, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.899 (+/-0.016) for {'alpha': 0.001, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'adam'}
0.899 (+/-0.016) for {'alpha': 0.001, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.899 (+/-0.016) for {'alpha': 0.001, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.905 (+/-0.004) for {'alpha': 0.01, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'adam'}
0.905 (+/-0.004) for {'alpha': 0.01, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.905 (+/-0.004) for {'alpha': 0.01, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.869 (+/-0.014) for {'alpha': 0.1, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'adam'}
0.869 (+/-0.014) for {'alpha': 0.1, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.869 (+/-0.014) for {'alpha': 0.1, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.615 (+/-0.003) for {'alpha': 0.9, 'activation': 'tanh', '
   learning_rate': 'constant', 'solver': 'adam'}
0.615 (+/-0.003) for {'alpha': 0.9, 'activation': 'tanh', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.615 (+/-0.003) for {'alpha': 0.9, 'activation': 'tanh', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.0001, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.0001, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'adam'}
```

```
0.273 (+/-0.000) for {'alpha': 0.0001, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.001, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.001, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.001, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.01, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.01, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.01, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.1, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.1, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.1, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.9, 'activation': 'relu', '
   learning_rate': 'constant', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.9, 'activation': 'relu', '
   learning_rate': 'invscaling', 'solver': 'adam'}
0.273 (+/-0.000) for {'alpha': 0.9, 'activation': 'relu', '
   learning_rate': 'adaptive', 'solver': 'adam'}
```

References

1. http://blog.trendmicro.com/malware-1-million-new-threats-emerging-daily/, 2015. [Online; accessed 6 December 2016].
2. blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/, 2015. [Online; accessed 6 December 2016].
3. Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, behavior-based malware clustering. Citeseer.
4. Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125. Springer, 2008.