HarvardX Data Science Capstone Project

A Machine Learning Model for Classifying Malicious Files

John Hanratty

October 25, 2020

Contents

1 Introduction

This capstone project focused on building a machine learning model for identifying malicious files. Cybersecurity represents a critical issue for all businesses, government agencies, and consumers. These organizations encounter thousands of new files daily, each a security risk. When a new file appears, the anti-virus industry may take weeks or months to discover it and settle on a MALWARE classification. During this period, the new file remains unclassified or often misclassified. Also, attackers often customize malware for a specific organization, so commercial anti-virus products may not recognize these unique attacks.

This project developed a machine learning model to close this detection gap by analyzing a file's attributes and predicting the eventual industry consensus. The resulting model might act as an initial screen for malware analysis labs or detection products by processing voluminous files to identify suspicious candidates for further analysis. With more development, the model could perform automated malware conviction in enterprise environments where there is a very low tolerance for false positives and negatives.

Data sets were created using the ReversinLabs TitanmCloud File Intelligence Service, which collects, analyzes, and catalogs 10 million goodware and malware files daily. The information includes both internal attributes extracted from the files and anti-virus industry classification information. Since applications for the model will vary widely in file mix, the project created multiple data sets for training and testing; each had a different size, file mix, and percentage of malicious files. The model versions were assessed with all test sets to understand the implications of these variations. As expected, test sets that matched the training set's file mix (e.g., partitioned from the same data set) performed percentage points better.

The model used multiple off-the-shelf modeling tools (e.g., RPART, Random Forest, XGB) for prediction. An ensemble voting algorithm improved the accuracy of results but, more importantly, enabled a trade-off between accuracy, false positives, and false negatives. This trade-off is important because applications of the model have different tolerances for incorrectly classifying a file as good or bad. For example, a gateway that blocks good files could interrupt business. In contrast, a malware analysis team may want to identify every suspicious file for analysis, so can tolerate false positives.

The model pre-processed the most predictive file attributes for presentation to the modeling tools. In most cases, these were binary values indicating the presence of an attribute or a count indicated the occurrences of an attribute (e.g., number of strings). Factor values/levels (e.g., file type) were pruned to the most important values.

Table 1: Final Model Performance Averaged Across Test Sets

model	Accuracy	FalsePos	FalseNeg
ens1x	95.4%	4.6%	0.0%
ens2x	94.8%	1.8%	3.4%
ens3x	94.3%	0.9%	4.8%

A theme for this project was studying the effects of different test sets on performance results. The table above shows realistic results by using the *average performance across all test sets*. An average better represents target environments because accuracy for matched test/training sets was higher than for unmatched sets (i.e., different file mixes) . In real-life, file mixes will differ greatly. The three ensembles provide a choice of the best model depending on your tolerance for false positives or negatives. You have a choice!

This Capstone project confirmed some assumptions and taught some valuable lessons about designing a machine learning model. Below are some observations that may merit further investigation in a future project.

Observations

Train Set Size - As expected, increasing the number of objects in the training data set improved the model's performance. The trade-off is **much** longer training time. Performance increased roughly 3-4 percentage points by doubling the number of objects in the training set. Improvements will taper off with further increases, but I didn't have the computing horsepower to train larger data sets. :)

Ensemble Results - Ensemble results exceeded the performance of individual off-the-shelf analysis tools and reduced wide variations (outliers) across datasets with different file mixes. The voting algorithm based on # of convictions by the analysis tools provided a better trade-off between accuracy and false positives/negatives.

Data Composition - Training and test sets created by partitioning (i.e., caret createDataPartition()) a data set from one download session performed better than using sets from different downloads sessions. The effects of file type mix were tested but didn't fully explain the performance difference. Evaluating the model performance using various data sets helps to better predict performance in real-life scenarios that have random file flows. How different data set characteristics affect modeling tool performance requires further research.

Variables Importance - An initial base variable set was chosen by eye-balling varImp() results for training sessions. The reduction in variables increased performance and reduced training time. In a final round, file_subtype, a factor with over 100 levels, was trimmed to 10 levels. VarImp showed that only a handful of file_subtypes were valuable for predictions. The level "Other" contained the rest of the files. Model performance increased slightly compared with the full set of subtypes with a significant reduction in training time.

Sub-model Architecture - An architecture that used separate sub-models for different file types was created as an experiment. The initial idea was to address the long training times and crashing modeling tools by breaking the model into smaller chunks. You could also imagine that each sub-model would provide better performance through specialization. The drawback was the management of training and testing six sub-models (6x the work). The results provided valuable insights into the model's performance by file type. This architecture had comparable accuracy (~0.1% less) than a monolithic model but included no sub-model optimization for file type other than training tuning parameters. In the end, it was better because of time constraints to let random forest and rpart figure out how to divide the variables.

Many researchers have attacked malware detection with novel approaches and achieved varied results. This project certainly does not obviate their work. Several data transformation and modeling approaches wer implemented with promising results. Positive trends were uncovered that identify some fruitful areas for further development and improvement.

2 Data Wrangling

2.1 ReversingLabs TiCloud Database

The ReversingLabs (www.reversinglabs.com) TiCloud file intelligence service processes 10 million files per day and has a catalog of over 10 billion goodware and malware files. Two services provided detailed file information.

File Intelligence Service

- *File internals* - The service decomposes each file to extract thousands of attributes, including internal format, certificates, structure, strings, libraries, and other components.
- *File relationships* - Many files contain or are contained by other files (e.g., zip files, installation package, document illustrations, or packed malware). TiCloud catalogs these parent/child relationships and analyzes the constituent files.

File Reputation Service

- *Multi-vendor anti-virus scan results* - Scanning results for 40+ popular anti-virus products are available.
- *Summary classification data* - Information is summarized by ReversingLabs to provide a threat level, malware family, platform, and anti-virus vendor consensus.

The File Intelligence Service was used to create training/test sets, and the File Reputation Service was used to build the classification reference set. The training/test sets used only attributes extracted directly from the files and no classification information from anti-virus vendors or ReversingLabs. The goal was to use only the attributes of a file to predict it's classification.

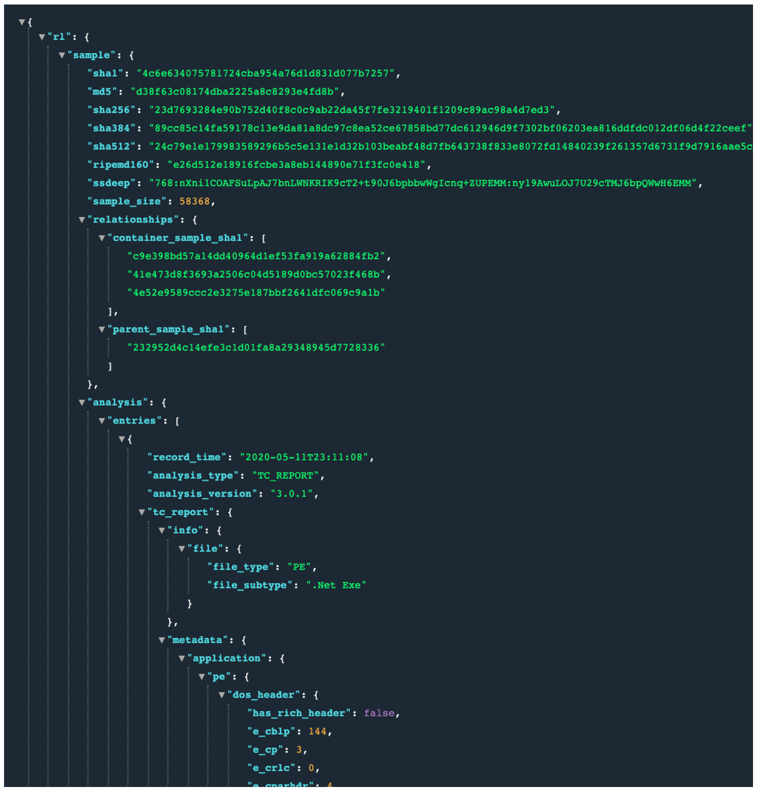
2.2 Downloading the Data Sets

The TiCloud services use a REST services POST command that passes a list of file hashes to query the service. The service returns data for files in a JSON string, which was processed and stored in a dataframe. The download process converted the voluminous and heavily nested JSON file data into a raw data set in a data frame format for model research and development (example JSON tree below). The raw data set was then processed and converted to create the modeling data set. This report submission includes the scripts used for this process.

Since ReversingLabs granted the project access to a few hundred-thousand files, the automated download code had to detect and avoid duplicate or malformed queries. The R scripts implemented error checking and status logging so that processing errors or a stoppage could be debugged and fixed quickly.

2.3 Reference Data Sets

Reference data sets to match the modeling data sets were created to obtain a rating for each file as MALWARE or UNRATED (i.e., innocent until explicitly classified as MALWARE). The TiCloud File Reputation Service classifies files based on historical data and anti-virus scans by 40+ products. Anti-virus scanners provide pretty good detection a month or two after an attack's debut because time is needed to detect new malware and create signatures. ReversingLabs assigns MALWARE classification when a significant number of reliable anti-virus products show a detection. The ReversingLabs classification was deemed sufficient as the reference set for this project.



```
{
  "rl": {
    "sample": {
      "sha1": "4c6e634075781724cba954a76d1d831d077b7257",
      "md5": "d38f63c08174dba2225a8c8293e4fd8b",
      "sha256": "23d7693284e90b752d40f8c0c9ab22da45f7fe3219401f1209c89ac98a4d7ed3",
      "sha384": "89cc85c14fa59178c13e9da81a8dc97c8ea52ce67858bd77dc612946d9f7302bf06203ea816ddfdc012df06d4f22ceef"
      "sha512": "24c79e1e179983589296b5c5e131e1d32b103beabf48d7fb643738f833e8072fd14840239f261357d6731f9d7916aae5c
      "ripemd160": "e26d512e18916fcbe3a8eb144890e71f3fc0e418",
      "ssdeep": "768:nXni1COAFSuLpAJ7bnLWNKRIK9cT2+t90J6bpbbwWgIcnq+ZUPEMM:ny19AwuLOJ7U29cTMJ6bpQWwH6EMM",
      "sample_size": 58368,
      "relationships": {
        "container_sample_sha1": [
          "c9e398bd57a14dd40964d1ef53fa919a62884fb2",
          "41e473d8f3693a2506c04d5189d0bc57023f468b",
          "4e52e9589ccc2e3275e187bbf2641dfc069c9a1b"
        ],
        "parent_sample_sha1": [
          "232952d4c14efe3c1d01fa8a29348945d7728336"
        ]
      },
      "analysis": {
        "entries": [
          {
            "record_time": "2020-05-11T23:11:08",
            "analysis_type": "TC_REPORT",
            "analysis_version": "3.0.1",
            "tc_report": {
              "info": {
                "file": {
                  "file_type": "PE",
                  "file_subtype": ".Net Exe"
                }
              },
              "metadata": {
                "application": {
                  "pe": {
                    "dos_header": {
                      "has_rich_header": false,
                      "e_cblp": 144,
                      "e_cp": 3,
                      "e_crlc": 0,
                      "e_cparhdr": 4
```

Figure 1: JSON for RLDATE

3 Model Data Sets

Model data sets were developed on two dimensions: file mix and attribute selection. File mix impacts real-world performance since production environments have distinct distributions of files. Attribute selection investigated the right combination of file characteristics and relationships to predict malware.

The TiCloud service provides file structure, certificate information, external references, internal strings, and parent/child relationships Many attributes only applied to specific file types or exhibited low correlation to malware. Relevant attributes were transformed and added to the model data set. The trick was to invest time on attributes that have the highest impact on overall model performance. In some cases, carefully chosen investment in less prevalent file types might have a significant impact.

3.1 Data Set File Mix

The TiCloud service identifies over 25 different file formats, each having a unique structure and attributes. For this project, the file formats were consolidated by prevalence into six categories, with “Other” covering around 20 of the least common types. Data sets created from different downloads using random file lists varied in size, file type mix, and percentage of malicious files. The table below shows the data sets make up. These data sets mimicked the variation in production environments.

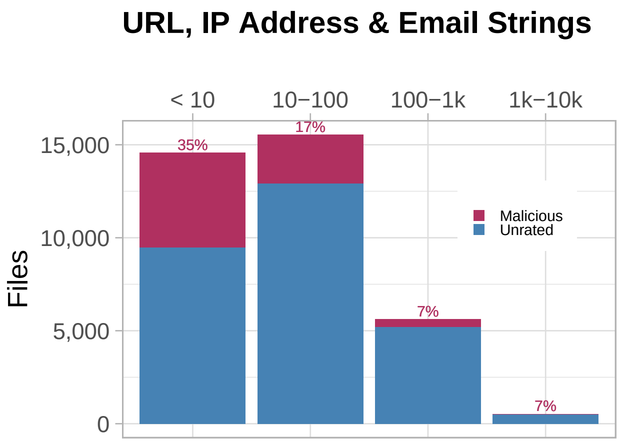
Table 2: Dataset File Mix

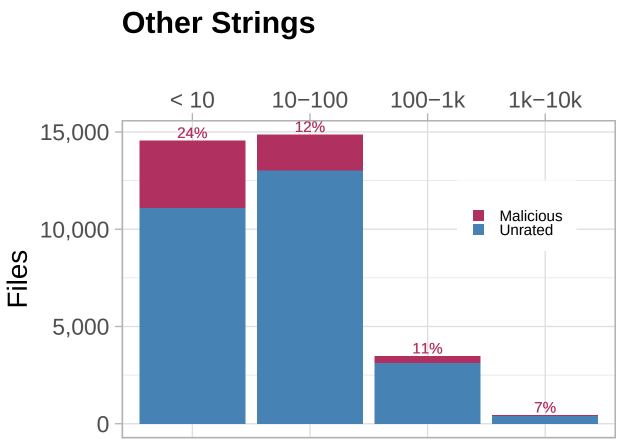
	MST	MEQ	MSU	MSW	TXU
Files	423,870	224,714	134,978	558,848	28,610
Malicious	40%	33%	25%	37%	26%
Download	A	A*	C	A+C	B
File_Types					
PE	55%	40%	41%	NA%	40%
Binary	21%	23%	20%	NA%	23%
Text	11%	21%	21%	NA%	21%
Document	6%	4%	4%	NA%	4%
PEplus	3%	5%	9%	NA%	5%
Other	4%	7%	5%	NA%	6%

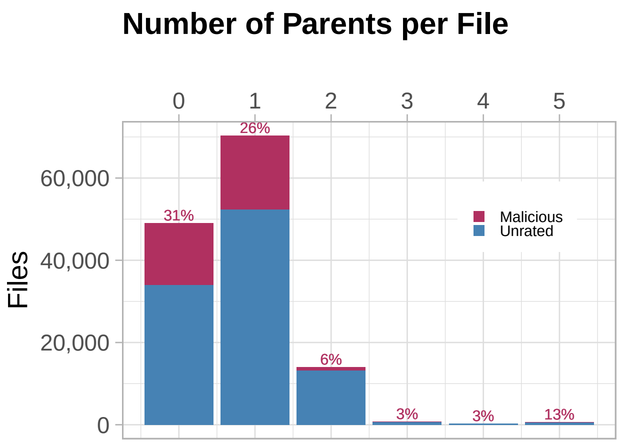
* MEQ was created from MST by deleting files to more closely match the file type mix in MSU

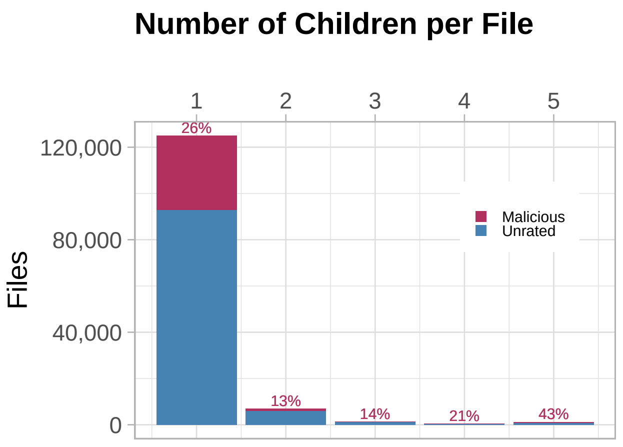
3.2 Strings

Files contain text strings containing various information, including IP addresses, URLs, email addresses, and passwords. The TiCore service provides a list of strings from each file. A count for a few common string types (IP Addresses, email, URL) was extracted and used as input variables for the model. For example, a list of IP addresses might indicate that malicious code that wants to "phone home."







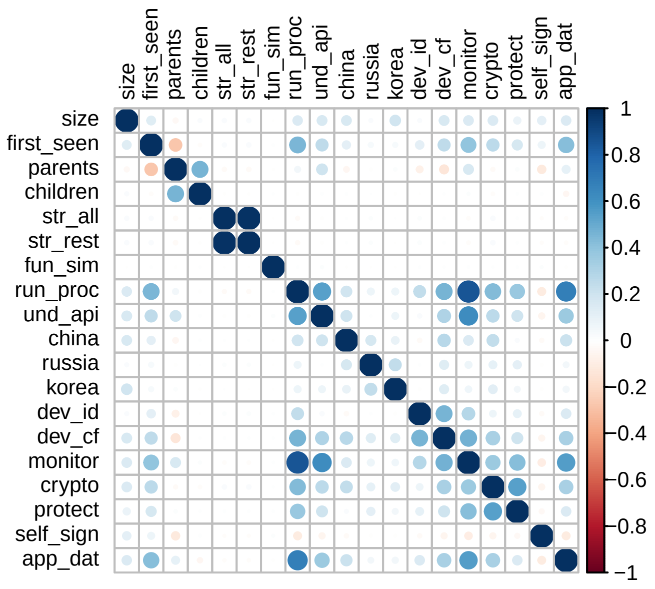


3.3 Parent-Child File Relationships

Files often arrive as 'packages' that contain other files (e.g., zip files, installers, malware packers). The TiCloud file data contains hashes of these parent and child file relationships. For example, an installation package file could contain hundreds of files, each containing more files. This characteristic creates a cluster of relationships useful for analysis since many packages may contain the same file. The presence of a file in a malicious package might increase suspicion when it appears in other packages. The frequency and distribution of particular files across multiple packages can also indicate maliciousness. This model used only the count of child and parent relationships. Further development might investigate the clustering concept further.

3.4 File Attributes

TThe TiCloud service provides thousands of file attributes with a text summary paragraph of relevant information extracted from the file. The attributes required conversion and cleanup to support modeling. Key strings were also extracted from the summary paragraph to augment the attribute list. The correlation diagram below shows the final list of attributes except for file_type and file_subtype, which were factors.



4 Model Description - Malware Identification

4.1 Ensemble Algorithm

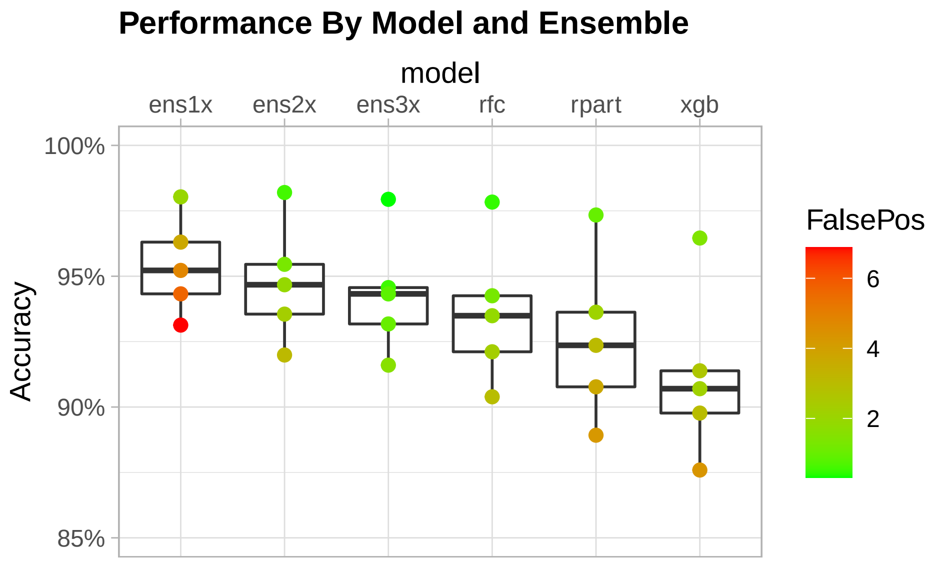
Ensemble results were more consistent and outperformed individual model algorithms and provided options for matching an applications' False Positive/Negative requirements.

The final model uses an ensemble of off-the-shelf analysis tools with a voting system that provides the final prediction. The tools were:, Regression Tree (rpart), Random Forest (rf), and Boosted Gradient (xgb). Other tools were tested but omitted because of redundant results, poor memory management, or slow performance. The ensemble voting algorithm used 1 (ens1x), 2 (ens2x), and 3 (ens3x) MALICIOUS predictions from the off-the-shelf tools to predict an outcome.

All results were tested against multiple data sets with varying file mixtures to understand the effects on Accuracy, False Positives, and False Negatives. The model's eventual target application will define which performance measures are most important. In some situations, false positives could block legitimate business, so minimizing these are critical. In other cases, the model might provide one of several screens to detect all suspicious files, so minimal false negatives are desirable.

The table below shows the performance results. Ensx1 (one or more convictions) showed high accuracy but higher false positives, as shown by the diagram's red dots. Ens2x (2 or more convictions) and ens1x (3 convictions) had lower accuracy but much lower false positives. The choice between ens2x and ens3x depends on your aversion to false positives.

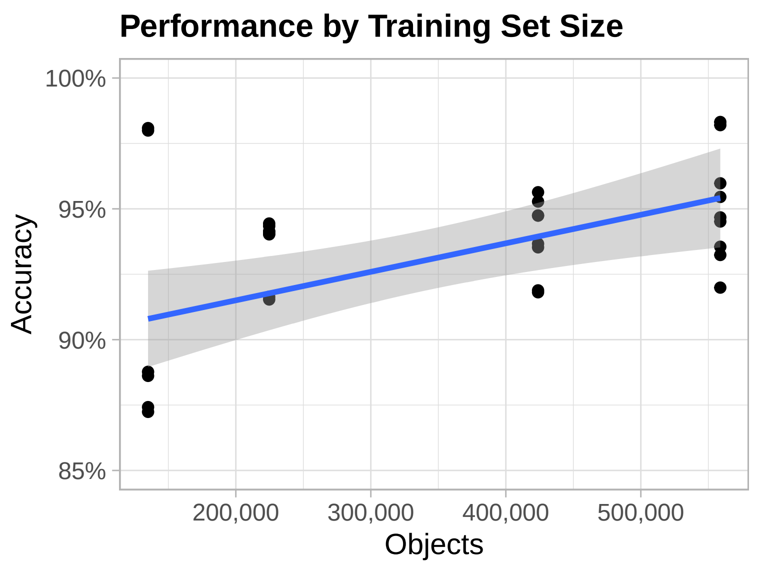
Also, XBG shows the lowest performance of the off-the-shelf analysis tools. A follow on project could investigate a replacement or an additional tool to increase model performance.



4.2 Data Set Size

Larger data sets improved training performance. A bigger computer is needed.

As expected, the size of the train data increased the model performance. Size also increase training time significantly. Amazon and Azure were tested but expensive. More compute power is needed to investigate this further.

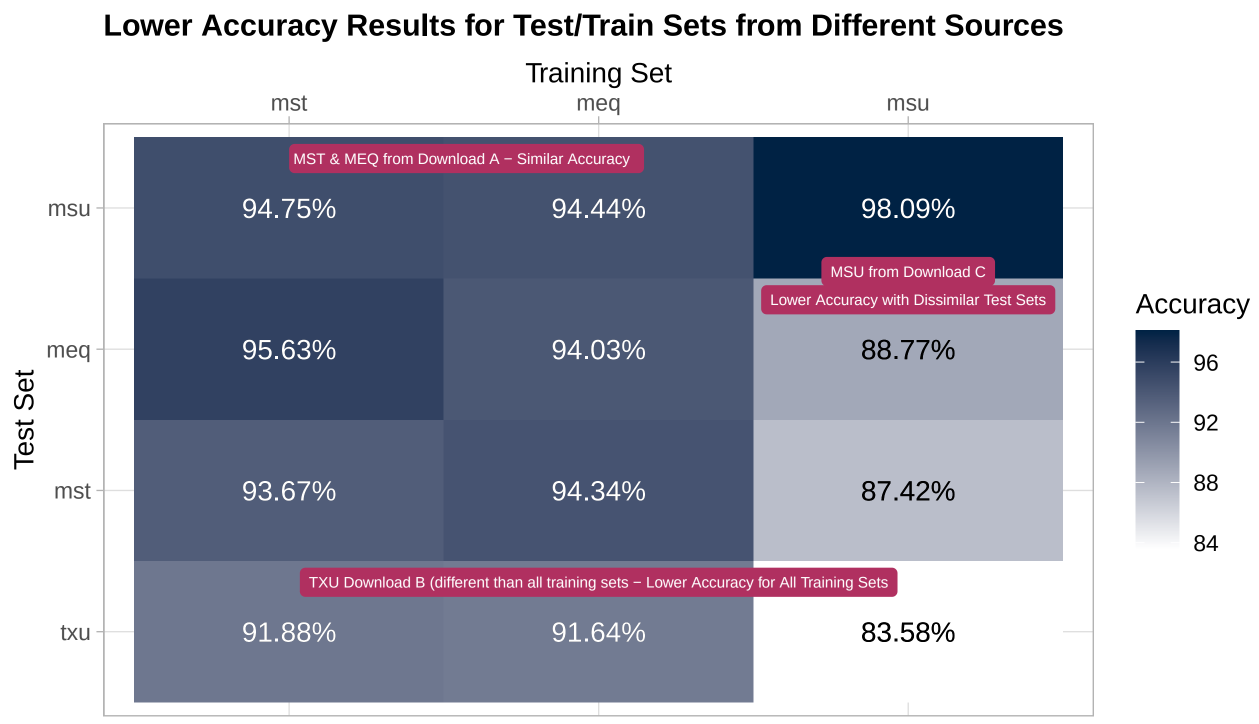


4.3 Data Set File Mix

The model performance decreased for test sets from different sources that the training data set. This project tracked performance usig the average results across several diverse test data sets to mimick real-life file mixes.

Performance differed depending on whether the train/test data sets came from the same download and different downloads. This fact became evident when test data sets under-performed when not partitioned from the same data set as the training set. Further investigation showed that these test sets differed in the file type mix and the percentage of malicious files. The MST, MSU and TXU datasets came from three differenct downloads and contained significantly different file mixes. TXU was a test set only (no corresponing training set) and always showed lower Accuracy results. To test the effects of file type mix, a derivative data set, MEQ, was created from MST that 'equlized' the file mix to better match MSU. As shown in the diagram below, MEQ & MST performed similarly but still differently than MSU. File type mix alone does not account for the file performance variation. Further research is required to understand the causes.

The implication is that the model performance is sensitive to the random file mixes that are found in malware production environments. Using multiple test data sets with from separate downloads provides a more realistic performance characterization (see the diagram). The project measured the average performance across the data sets for assessment.

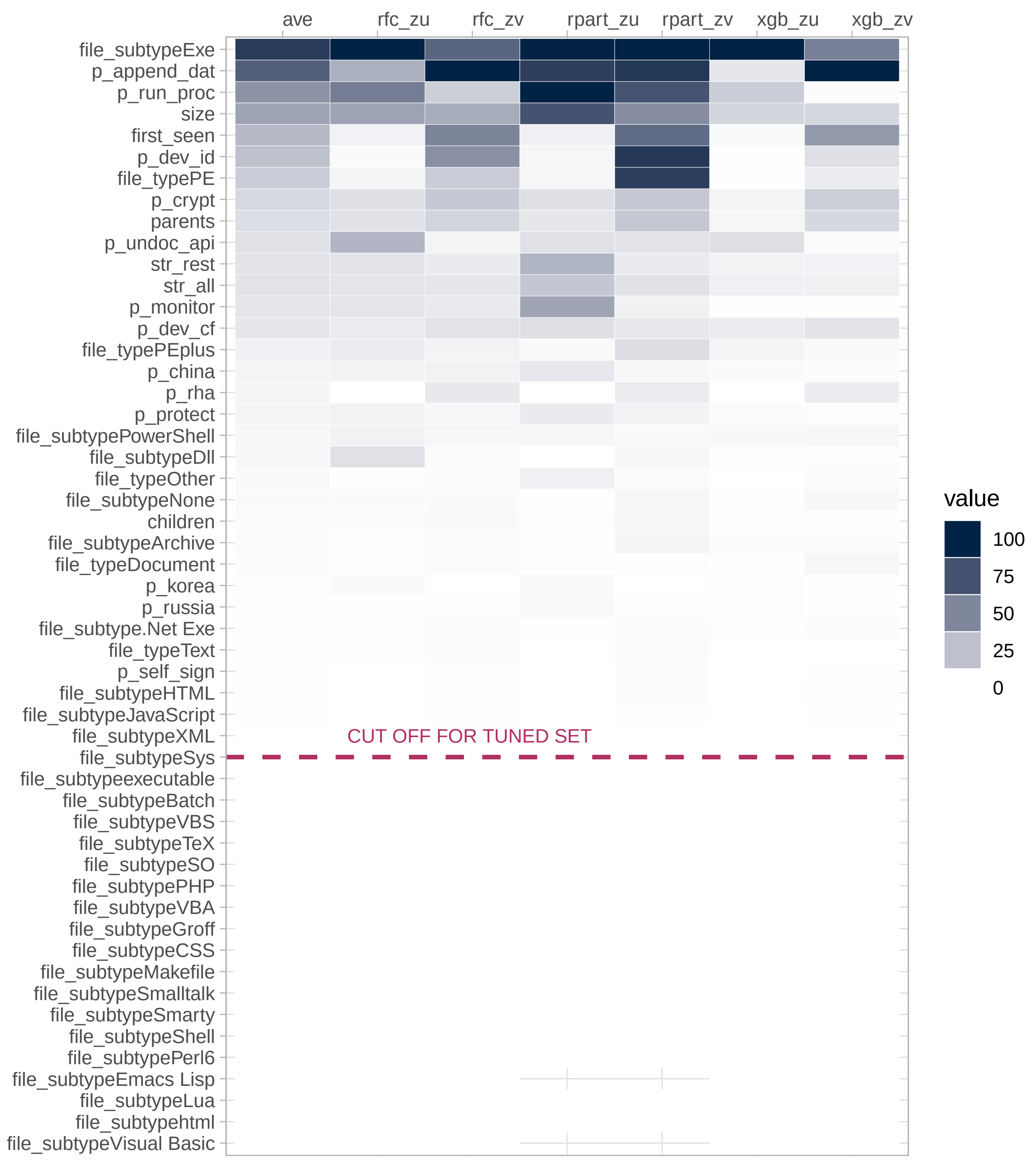


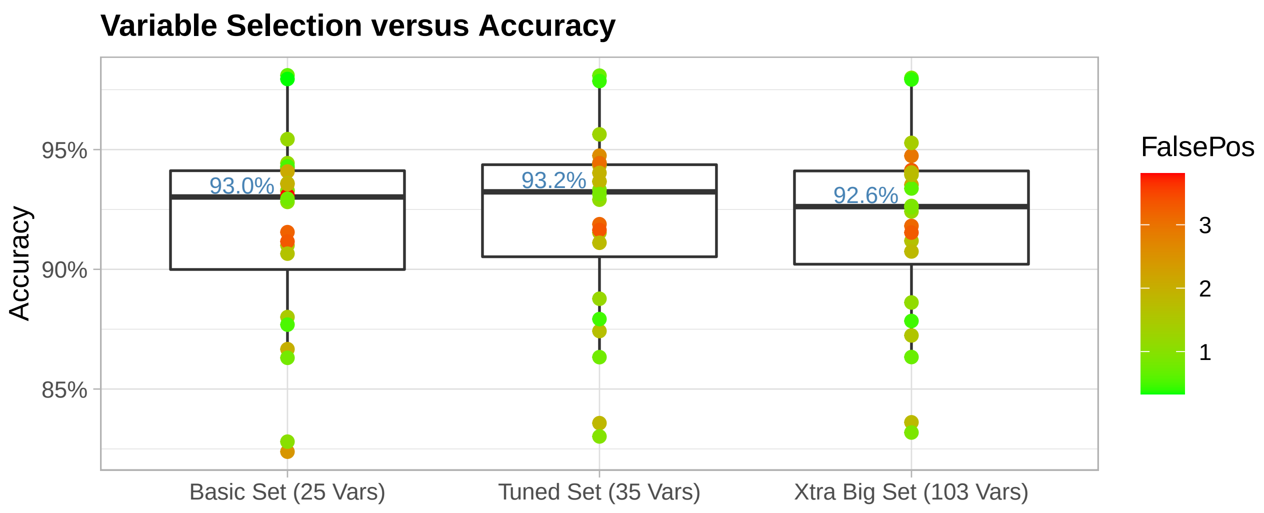
4.4 Data Set Variable Optimizations

More variables significantly increases training time and can decrease model performance. Non-predictive file attributes and factors were removed from the data set, which increased performance with fewer variables.

The systematic assessment added and removed variables and factors based on their importance for prediction. An optimized variable set significantly reduced the training time while increasing performance. Superfluous variables decreased model performance. The table and graph below show a variable optimization pass involving the factor *file_subtype* with over 100 levels. VarImp ratings helped consolidate these into ten levels. The result was quicker training time and sightly better model performance.

Variable Importance (varImp) by Model





4.5 A Segmented Sub-model Architecture

Dividing the model data and processing by file type provided valuable insights but requires more work to attain improved predictive results.

Long training times and crashes were causing headaches. To address this, the project experimented with a segmented architecture that routes data to specialized sub-models based on file type (diagram below), enabling multiple shorter training runs and specific tuning. This approach makes sense because different file formats have radically different characteristics and attributes. There were also potential file type specialization benefits. The downside was the management of many (6x) training sessions, objects, and results.

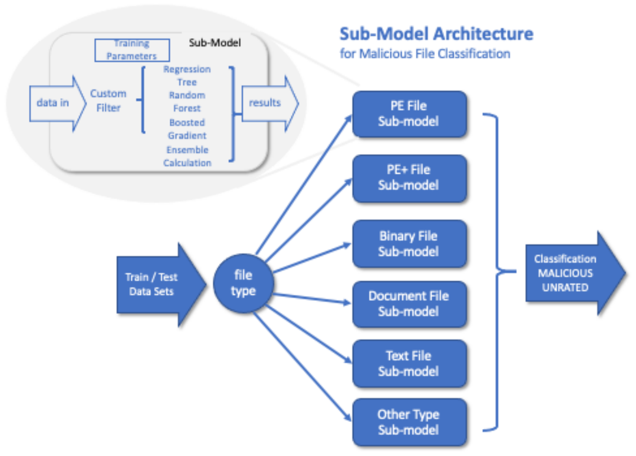
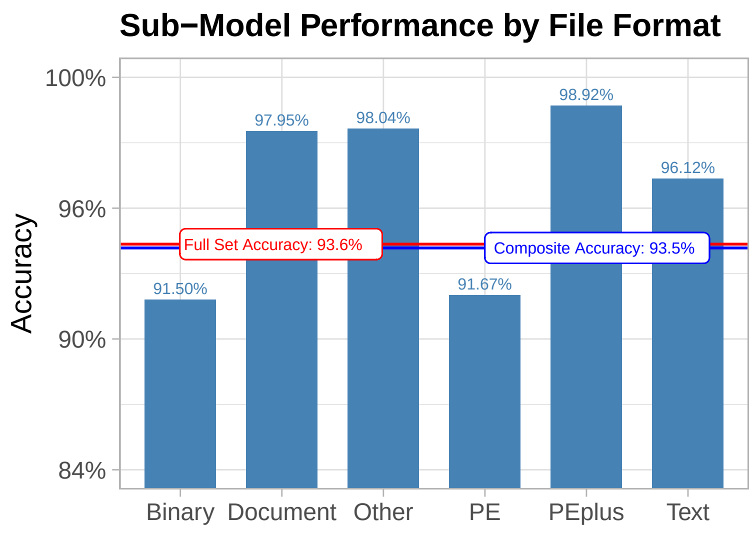


Figure 2: Sub-model Model

The model was partitioned based on file type prevalence. The PE format (Microsoft Windows) represented the largest percentage and the longest sub-model training time. Other sub-models had fewer files and achieved quicker training times. The training and test scripts divided data and processing by file type. The training script divided the data to create separate training sets for six file types: PE, PEplus, Binary, Document, Text, and Other. The output was training object for each analysis tool (i.e. rf, rpart, xgb) and file format. The testing script divided test sets by file type and routed these to the appropriate specialized sub-models to create predictions. The sub-modeltesting consolidated outputs into an overall consolidated performance.

While there was much enthusiasm for this approach, there was not time to further tune the sub-modules. The consolidated accuracy results from the sub-models were comparable (~0.1% less) than with a single model architecture. Individual sub-models were not optimized for their file_type, so performance might increase with some further effort. For this project, letting a single model create the decision tree for all file types was the right policy.



5 Conclusion

This capstone project scratches the surface for applying machine learning to malware classification. The model's performance depends on the creative extraction and transformation of file attributes. This project showed that a test set's file mix impacts performance.

Applying the model to a "real life" stream of files is a worthwhile follow-on project. This project's data sets differ in mix and coverage from enterprise file flows. Real-life, enterprise file mixes should provide some new challenges and dictate model improvements. ReversingLabs sells the tool used by TiCloud to extract the attributes from files. This tool could provide input to a future project with new file sources.

One area explored but not exploited in this project was file clusters base on parent/child relationships. This approach would convict files based on "the company they keep." Files come to computing devices in bundles (e.g., an installation package or a document with graphs). A file/object previously bundled with known malware might indicate maliciousness of a new file bundle. These related files were downloaded, but time did not permit exploitation.

The project provided many challenges through all phases of data wrangling, pre-processing, and analysis. Many valuable lessons will apply to future data science projects.

6 Apendix A - Project Submission Contents

6.1 Suggested Tour

This Capstone submission includes sample code and scripts based on a subset of data because of size and runtime constraints. Contact me if you want a larger set.

1. Look at the included README.TXT file for the latest information
2. Read the Capstone-Report_V3 PDF
3. Run the Capstone-Report_V3.Rmd.
4. Look at the rCapFUNCTIONS - These are sourced and called by the scripts.
5. Look at the rJSON_SCRIPTS_V2.R - You can't run this without ReversingLabs credentials).
6. Run rMODEL_SCRIPT_v2 - This creates the train and test data sets.(runtime 5+ minutes)
7. Run rTRAIN_MODEL_v2 - Trains and tests an RPART model with the data above. (runtime 5+ minutes). You can set the flags in the script to train an RF and XGB (runtime 1-2 hours).
8. Run the rMODEL_PERFORMANCE_V2.R - This tests the models and saves a report file.
9. Look at the CapstoneRptData (readRDS) - This has the test results created by rMODEL_PERFORMANCE_V2.R for all the training and test sets used in the project.

6.2 Components

The files that accompany this are as follows:

- Capstone Report (this document)
  - PDF Format .pdf
  - Markdown Format (Rmd)
    - Images (.png) files for these reports
    - CapstoneRptData performance data
    - Data set data (_msu_test)
- Data Files for the scripts (small subset for demo purposes)
  - msu_model - a model data set for final processing and partitioning
  - msu_test & _msu_train - test and training data sets derived from _msu_model by the rMODEL_SCRIPTS.R script.
  - zumtrain_rpart_ALL - an RPART training object created by the rTRAIN_MODEL_V1.x.R script and used for testing by the rMODEL_PERFORMANCE_V2.R

6.3 Scripts

6.3.1 Functions That Do the Hard Work (rCapFUNTIONS.R)

This module is sourced by most of the scripts included with the submission. It doesn't execute any code, but loads functions used in scripts.

This module loads using source() for use by scripting files. It contains functions that download files, create a modeling data sets, test models and create a performance results database for analysis. The Functions are grouped follows:

- Get Training Results
- Create Model Data Set
- JSON / Download File Information

6.3.2 Data Download Script (rJSON_SCRIPTS_V2.R)

This script will not run without credentials from ReversingLabs and is included for informational purposes.

This module contains scripts to download file information from the ReversingLabs TiCloud database. These scripts call functions defined in the module 'rJSON Functions.R'. The scripts in this module were hand crafted for each download project so they are not completely generic.

6.3.3 Create Training/Test Data Set (rMODEL_SCRIPTS_V2.R)

This script is preconfigured to use msu_model as the input for creating the msu_train training set and msu_test test set. msu_model is included with the submission or by download).

This module contains scripts that create train and test sets from the JSON downloads. Use rJSON_SCRIPTS and rJASON_FUNCTIONS to download and capture file information from ReversingLabs. The script(s) in this module preprocesses and formats the data to create a 'Model' Data Set that is partitioned into train and test sets

6.3.4 Model Training Script (rTRAIN_MODEL_V2.R)

The script is preconfigured to train and test using includes msu_train and msu_test (which are included with the submission or downloadable).

This SCRIPT runs model training batches that are usually executed with Rscript and can run for many hours. The script performs the following:

1. Loads appropriate _test and _train data sets for the batch (BATCH).
2. Selects the appropriate variables (columns) from the training / testing data sets
3. Runs a training session for each specified file_type in the list FORMATS. The file_formats are ALL, PE, PEplus, Binary, Document, Text, and Other.
    a) Filters objects in the input data set for training / testing by file_types specified in FORMATS.
    b) Trains and tests sets for each file_type by using the tools specified by the variables RPART, RFC and XGB.
    c) Stores the training object in the file with a file name: {batch}train_{model}_{file_type} e.g B1train_rpart_Document

6.3.5 Model Testing Script - (rMODEL_PERFORMANCE_V2.R)

This script is preconfigured to load zumtrain_rpart_ALL and generate a report and a results file like the one used by the Capstone_Report-Report_V3.RMD.

This module is a collection of scripts that processes the training objects created by the rTRAIN_MODE:

1. Looks for files of training objects in the and upload them one by one
2. Tests them against 3 or 4 test datasets 3, Grabs details about the training stats, tune parameters, train set, and variable importance.
3. Stores a record for each test result by batch, model, file_type, testset. This is saved and available for creating reports and analysis.