Microsoft Malware Prediction

Amir Babaei, Christian Ortiz, Christian Cortes
University of California, Santa Cruz
Santa Cruz, California

ABSTRACT

The machine learning project we chose was the Microsoft Malware Prediction competition from Kaggle. The purpose of this competition was to predict malware on a computer. Microsoft provided test data which comprised of 82 features and 8.9 million rows. The dataset leaned heavily toward categorical data over numerical data. After cleaning and feature engineering the data we compared different versions of neural nets and random forest trees. After testing the model we converged on a 64% test accuracy.

ACM Reference Format:
Amir Babaei, Christian Ortiz, Christian Cortes. 2019. Microsoft Malware Prediction. In *Microsoft Malware Prediction, March 14, 2019, Santa Cruz, CA.* ACM, New York, NY, USA, 5 pages.

1 MOTIVATION

The malware industry continues to be a well-organized, well-funded market dedicated to evading traditional security measures. Once a computer is infected by malware, criminals can hurt consumers and enterprises in many ways.

With more than one billion enterprise and consumer customers, Microsoft takes this problem very seriously and is deeply invested in improving security. Microsoft believes the ability to predict malware before it happens can help protect billions of machines.

2 OBJECTIVE

The goal of this competition is to predict a Windows machine's probability of getting infected by various families of malware, based on different properties of that machine. The telemetry data containing these properties and the machine infections was generated by combining heartbeat and threat reports collected by Microsoft's endpoint protection solution, Windows Defender.

Each row in this dataset corresponds to a machine, uniquely identified by a Machine Identifier. Has Detection is the ground truth and indicates that Malware was detected on the machine. Using the information and labels in the dataset provided by Microsoft known as the train.csv. The purpose of our project is to predict the value of the HasDetection feature as the result from the training data set of each machine

Malware detection is inherently a time-series problem, but it is made complicated by the introduction of new machines, machines that come online and offline, machines that receive patches, machines that receive new operating systems, etc.

While the dataset provided in the dataset has been roughly split by time, the complications and sampling requirements mentioned above may mean we may see imperfect agreement between our cross validation

3 DATASET

The training dataset contained 82 features and 8921482 million rows. The features were distributed based on the figure below:

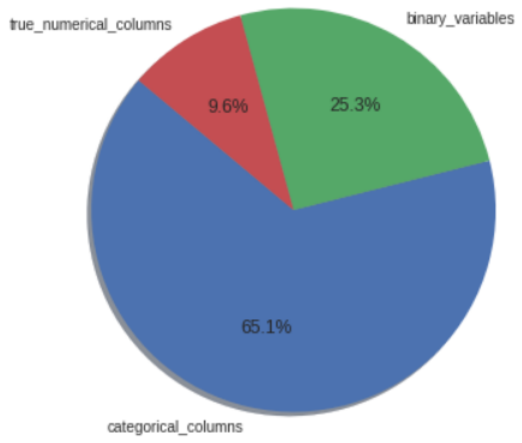
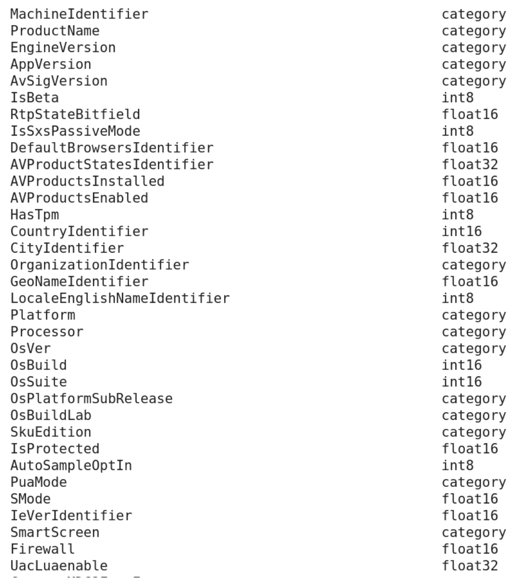


Figure 1: Distribution of Features

From the distribution of features figure we can see that there not many true numerical features with the least out of the three. A little under 2/3 of the features were categorical. The categorical data needed to be one hot encoded in order to be usable in the models.

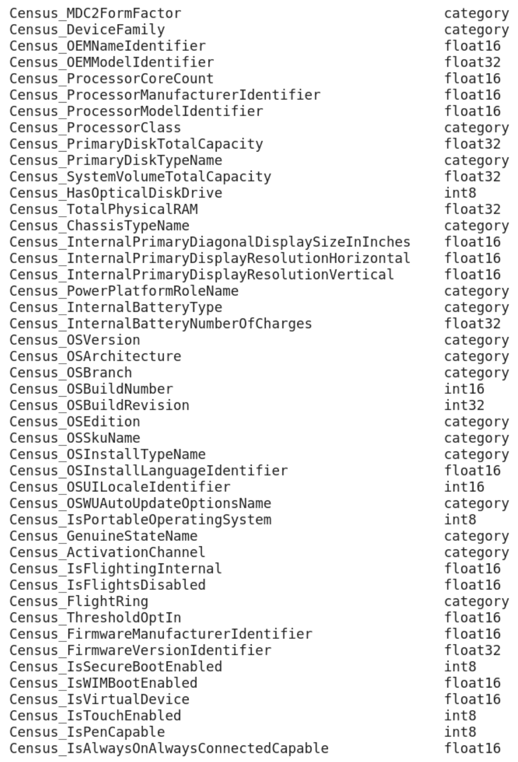
Each of the features provided by Microsoft can be broken down into three parts. The first part includes the MachineIdentifier, ProductName, Versions of the Computer, and Identifiers. It also includes information about the operating system and other various features. We attempted to set the types of all the features by looking at them.

The second part of the data is from the computer census information. This makes up most of the features in the data set. The information from the census varies from processor information, ram information, OS information, to storage. It also contains a few boolean values such as touch enable, pen enable, virtual device, and always on capability.



```
MachineIdentifier                                    category
ProductName                                          category
EngineVersion                                        category
AppVersion                                           category
AvSigVersion                                         category
IsBeta                                               int8
RtpStateBitfield                                     float16
IsSxsPassiveMode                                     int8
DefaultBrowsersIdentifier                            float16
AVProductStatesIdentifier                            float32
AVProductsInstalled                                  float16
AVProductsEnabled                                    float16
HasTpm                                               int8
CountryIdentifier                                    int16
CityIdentifier                                       float32
OrganizationIdentifier                               category
GeoNameIdentifier                                    float16
LocaleEnglishNameIdentifier                          int8
Platform                                             category
Processor                                            category
OsVer                                                category
OsBuild                                              int16
OsSuite                                              int16
OsPlatformSubRelease                                 category
OsBuildLab                                           category
SkuEdition                                           category
IsProtected                                          float16
AutoSampleOptIn                                      int8
PuaMode                                              category
SMode                                                float16
IeVerIdentifier                                      float16
SmartScreen                                          category
Firewall                                             float16
UacLuaenable                                         float32
```

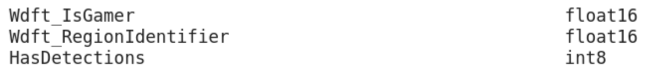
Figure 2: Features Part 1



```
Census_MDC2FormFactor                                category
Census_DeviceFamily                                  category
Census_OEMNameIdentifier                             float16
Census_OEMModelIdentifier                            float32
Census_ProcessorCoreCount                            float16
Census_ProcessorManufacturerIdentifier               float16
Census_ProcessorModelIdentifier                      float16
Census_ProcessorClass                                category
Census_PrimaryDiskTotalCapacity                      float32
Census_PrimaryDiskTypeName                           category
Census_SystemVolumeTotalCapacity                     float32
Census_HasOpticalDiskDrive                           int8
Census_TotalPhysicalRAM                              float32
Census_ChassisTypeName                               category
Census_InternalPrimaryDiagonalDisplaySizeInInches    float16
Census_InternalPrimaryDisplayResolutionHorizontal    float16
Census_InternalPrimaryDisplayResolutionVertical      float16
Census_PowerPlatformRoleName                         category
Census_InternalBatteryType                           category
Census_InternalBatteryNumberOfCharges                float32
Census_OSVersion                                     category
Census_OSArchitecture                                category
Census_OSBranch                                      category
Census_OSBuildNumber                                 int16
Census_OSBuildRevision                               int32
Census_OSEdition                                     category
Census_OSSkuName                                     category
Census_OSInstallTypeName                             category
Census_OSInstallLanguageIdentifier                   float16
Census_OSUILocaleIdentifier                          int16
Census_OSWUAutoUpdateOptionsName                     category
Census_IsPortableOperatingSystem                     int8
Census_GenuineStateName                              category
Census_ActivationChannel                             category
Census_IsFlightingInternal                           float16
Census_IsFlightsDisabled                             float16
Census_FlightRing                                    category
Census_ThresholdOptIn                                float16
Census_FirmwareManufacturerIdentifier                float16
Census_FirmwareVersionIdentifier                     float32
Census_IsSecureBootEnabled                           int8
Census_IsWIMBootEnabled                              float16
Census_IsVirtualDevice                               float16
Census_IsTouchEnabled                                int8
Census_IsPenCapable                                  int8
Census_IsAlwaysOnAlwaysConnectedCapable              float16
```

Figure 3: Features Part 2

The last part includes the feature HasDetections which is the most important. It is used to test against all other features. The training set was evenly split with half of the information having a detection and the other half not having a detection.



```
Wdft_IsGamer                                         float16
Wdft_RegionIdentifier                                float16
HasDetections                                        int8
```

Figure 4: Features Part 3

Many of the features in the dataset was categorical and needed to be one hot encoding. An issue with one hot encoding this dataset was keeping the dimensions of the data small. A few of the features had many unique values with some having over 5000 unique values.

Each row in this dataset corresponds to a machine, uniquely identified by a MachineIdentifier. HasDetections is the ground truth and indicates that Malware was detected on the machine. Using the information and labels in train.csv, you must predict the value for HasDetections for each machine in test.csv.

4 MODELS & ALGORITHMS

The models and algorithms we chose for this project were based on the fact that this was a binary classification problem. We attempted to use some of the code from homework assignment 2 which had a neural network template. We began this project with neural nets and attempted to try different amounts of layers, dropouts, and loss functions.

After exhausting all we could we branched out to figure out a new solution. Researching binary classification we found that random forest tree were able to solve this type of problem. It took a lot of understanding to make sense what this type of model was trying to do. After getting model working with good accuracy we attempted many different iterations.

4.1 Neural Nets

The first neural net model we created was a one-layer sequential model. The layer had ten neurons a relu activation function. The kernel initializer was normal. The layer had a drop out rate of 20%. The final layer had one neuron and a sigmoid activation function. The model was compiled using binary cross entropy loss function. The model had an 'adam' optimizer which combines the benefits of adaptive gradient and root mean square propagation.

The second neural net we tested was equivalent to the first with eight layers. Each of the eight layers had 64 neurons compared to the ten in the previous model. All other features of the model were the same, including the drop out rate, loss, optimizer, and activation.

We attempted many other implementation including adding more layers and different loss functions. Many of those failed to reach 50% accuracy. The limiting factor of adding more layers was Google Colab, which could not handle the data size.

4.2 Random Forest Tree

The random forest tree model was tested by splitting the dataset into a train and test split using 25% of the dataset as the test. The random forest tree model we used was sklearn's ensemble RandomForestClassifier model. The parameters used for this model was using five minimum samples for each leaf of the tree. One hundred estimates for each feature. Ten minimum samples to split. The number of max features was based on the square root of the number of input. The maximum depth was chosen to be six. No maximum leaf nodes were chosen.

Only one model was used for the final testing. Many other implementations were used including altering the number of max features and number of estimators. The parameters chosen above was picked based on the time it took to complete as well as accuracy.

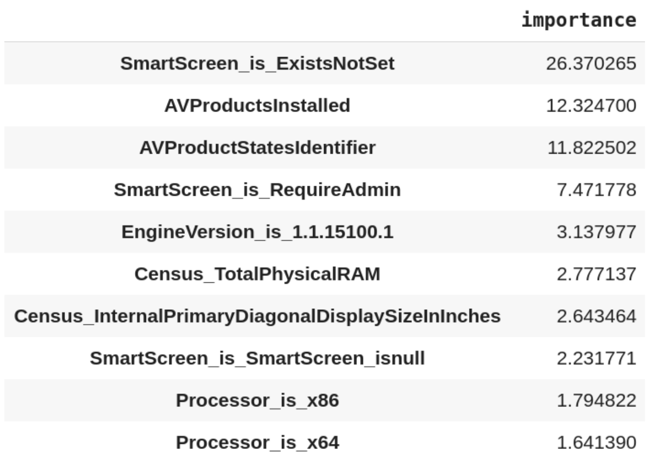
One of the main reasons the Random Forest model was chosen was to view the importance of each of the features. The importance of each of the features was used to show what features mattered when predicting malware.

5 RESULTS

After running the three models we were able to get an accuracy 64% for the one layer neural net, 64% eight layer neural net. The random forest model was able to produce a 64% accuracy which is comparable to the one layer neural net.

Model	Accuracy (%)
One-layer Neural Net	64
Eight-layer Neural Net	64
Random Forest Tree	62

From the RandomForestClassifier we were able to view the importance of each feature.



	importance
SmartScreen_is_ExistsNotSet	26.370265
AVProductsInstalled	12.324700
AVProductStatesIdentifier	11.822502
SmartScreen_is_RequireAdmin	7.471778
EngineVersion_is_1.1.15100.1	3.137977
Census_TotalPhysicalRAM	2.777137
Census_InternalPrimaryDiagonalDisplaySizeInInches	2.643464
SmartScreen_is_SmartScreen_isnull	2.231771
Processor_is_x86	1.794822
Processor_is_x64	1.641390

Figure 5: Importance

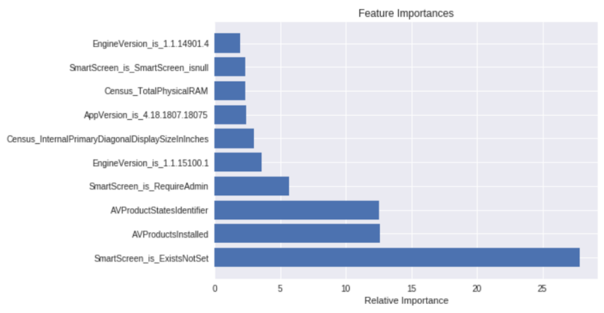


Figure 6: Importance Chart

From this we are able to view each of the feature that correlated with having defections. It can be noted that the features that include *_is_* were features that were one hot encoded from categorical features. This data can be used to infer which features may have played a large role.

The accuracy of one layer neural net was computed using kfold cross validation. The number of folds was chosen to be five. The accuracy of the model over each fold can be viewed in the figure below.

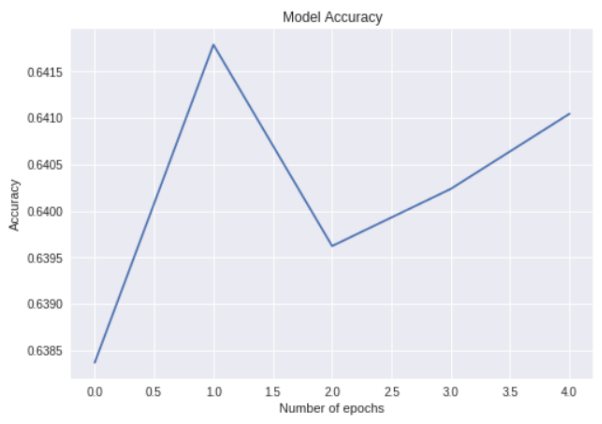


Figure 7: One layer kfold accuracy

The accuracy of eight layer neural net was computed using kfold cross validation. The number of folds was chosen to be five. The accuracy of the model over each fold can be viewed in the figure below.

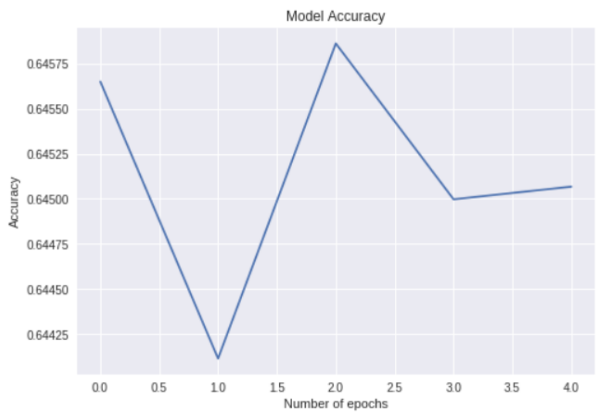


Figure 8: Eight layer kfold accuracy

6 ANALYSIS

After getting the results from different models, we realized that the given data is not the best for predicting the malware we high accuracy. the best score reached in the whole competition was around 67% after the final submission, and that's not the best accuracy for a binary classification task.

Over all we are very happy with results, because our first models wouldn't pass 5% accuracy. Out of all the models mention, results of 2 stood out. The one layer neural net, which gave us a high accuracy and surprisingly the results didn't show any over fitting. RandomForestClassifier was a great model for our task; because although using it we managed to get feature importance (figure 6). As you can see smart screens had the most to do with having malware, which made us question the quality of the provided data and the results expected from training this data. in farther note, one interesting thing we noticed was that K-folding worked great on our data, we just wished we had better hardware to try fitting the entire data to it, maybe we could get better results and intuition. Do to one hot encoding we needed to decrease size of the sample even more, because the hardware could handle it. This project was categorized as research level in the competition, which was one of our worries before choosing this project, and that might explain the low accuracy results throughout the whole competition.

7 CONTRIBUTIONS

The groups contributions were split as evenly as possible. We made many attempts to meet, however, our schedules had many conflicts. We did most of the work separately and discussed our results in short meetings and on our discord server.

7.1 Amir B.

As the project leader, I mostly spent my time researching and giving the team direction. For example, in the beginning, since we had a big dataset with over 80 features for each item, I decided to split the features into 3 equal portions and have each team member go through his portion and understand the features and remove the ones necessary and report on what they did and why they did it. I also tried to find the best tools that would work well for our data and also the whole team has easy access to them and can share their progress with the rest of the team easily. I made all the team meetings and made sure we participate in the meetings at least twice a week. Other than planing the direction of where the project is going; I also participated in every step of coding. In the middle of the quarter, we started testing different models and approaches to see what gives us the best result and why? ML projects are somewhat new to me especially when it comes to working with the team since we all wanted to learn every step taken to finish a ML project, we decided that we all attack every step together so it kind of took away from the agile development aspect of our project. If I want to mention specifically what I worked on in the code. First I looked around and found a way to compress the data, and where I found how to do it is cited in the code. I tried some NN models with different aspects to see how our data trains on them (not so good in the beginning). I tried to provide some graphs and visualizations of the data, so the team and people looking at our code will have a better understanding of the data we are working with. I helped to tune some of the models that were made by teammates.

7.2 Christian C.

Spent time studying the data in order to evaluate the best decision to take in order to answer the question being asked in the Kaggle competition. Spent time looking at all the columns in the dataset and discussing with teammates if whether or not to keep or delete the feature.

After cleaning the dataset, wrote a script to see if there was strong correlation between the features kept and would remove the one that were to close together, almost redundant. Discussed with teammates before final decisions were made. Programming was not the strongest so paired programming was involved when trying algorithms and trying to understand how to make NNs.

Mostly contributed ideas like understanding the dataset, how we break the data by cities and look at the machines attacked in each city, what all the machines had in common that got attacked and how we can expand that to countries and make better generalizations based off that information. Contributed to the idea of using dropout. It did not work well at all but still something worth trying.

Attempted different Neural Networks mostly change the depth of the neural net and the size of each layer. For example, attempted a 13 layer neural net where the first layer was 100 neurons, increased to 500 for layers 2-10 then dropped to 100 to 10 then to 2 as the output and just changed these to see if results would increase. Did not work so just continued to smaller neural nets and team ideas that worked a lot better.

7.3 Christian O.

Researched the meaning behind each of the features and made decisions on what should be kept based on the relevancy to the objective. Realized some of the numerical features in the dataset were not continuous. Developed a one-hot-encoding function that solved two issues:

- Categorical Features with many unique items.
- Memory

As stated in the dataset, one of the main issues with this dataset was keeping the size of the memory as low as possible to run the neural nets. The one hot encoding function took in an extra parameter which was the maximum number of features to one hot encode. The maximum number of features were the features with the highest frequency.

When performing the actual one-hot-encoding the function turned the column size from *int64* to *int8* allowing seven extra features in the same memory space.

8 FUTURE WORKS

Given more time and resources we would have spent more time learning and testing different neural nets. We were constrained by Google Colab's GPU and RAM size. No matter how many different features we removed we were never able to run the entire dataset on Colab with crashing the site.

Another avenue to explore would have been to split the data up by screen sizes to see if that feature was truly important, when running a correlation, we noticed that screen size contributed to determining if a machine was vulnerable to malware or not. But knowing that 13 inch are more common than other sizes we think this is not truly a valid feature to keep. Especially if you want a good general neural net that will not discriminate against other screen sizes. The plan was to check the separated data and see if the ratio of the HasDetections feature was the same across the board.

Another thing we would have liked to check if the different cities the machines were located. And seeing what those machines had in common to see if we can find something that would connect all the machines together and get a better understanding of what may cause a machine to be vulnerable. Knowing the similarities would have helped us pick out strong features or even keep some features we might have thought about as useless.

We would have tried setting up an environment using Amazon services including S3 to hold the train and test set and SageMaker to run the models. This would have allowed us to run the entire dataset without many worries of the size.

9 KAGGLE NOTE

One thing to note was Kaggle's competition as a way of researching solutions and issues we had with it. Since Kaggle is an open competition and competitors are able to view the Public leader board and comment on their own implementation it was easy to feel inferior to some of the competitors. As a group with little machine learning background we felt our goal was to beat out the top competitors by any means possible. However, when the competition ended and the winners were announced big change was noticed in the leader boards.

Many of the top teams in the public leader boards had a accuracy 70% which was a 2/3 of the test set. The private leader boards, the true winners, had an accuracy of 67% on the other 1/3 of the test set.

This information made us realize that many of the solutions in the public leader boards were over fitting their models. These teams worked hard to get their training sets to be as high as possible instead of generalizing their models.

Our models were built using k-fold cross validation which prevents over fitting. Our accuracy, 64%, would have placed in the top 100 of the scores. In our k-fold models our standard deviation was less than 1%. This proves our model was generalized across the entire set.

Instead of feeling bad for failing to reach 70% test accuracy we should have tried much harder to generalize our algorithm as best as possible. Had we submitted our model and let our best work be ran we would have done much better than we anticipated. This was a great learning experience and will be noted in all future works.