Daniel Shackelford

CSPB 3022

Final Project: Microsoft malware Prediction

GITHUB Repo: https://github.com/dash6877/CSPB3022_FINAL_PROJECT

Kaggle UserId: User ID 5543986

Description of problem:

In this problem, we are asked to run analysis on data from Microsoft concerning Malware statistics in order to predict the probability of a machine getting infected by various families of malware based on the different properties of the machine. The data set contains two files of data, one of which is given as training data, and the other that is given as test data. In the data, there are a total of 83 unique columns that cover a variety of data. The 'HasDetections' column of the data is the column that will be used as our target for comparing our data. The other columns of data contain a variety of values that correspond with both integer/float values and categorical values. These columns and their associated data types can be seen in the associated python notebook of my solution.

Exploratory Data Analysis and cleaning:

The hardest part of this entire project was the sheer size of the dataset. The training data has a total size of 4.08GB and the test data set has a size of 3.53GB, more importantly 83 categories of data means that it would be incredibly computationally intensive to do a regression across all of the values. Loading the train data set alone required 31GB of memory on my machine alone. Upon testing this conclusion by doing a preliminary KNN fit to the uncleaned data and found that it computed for more than 24 hours before halting the kernel. It was then concluded that a large amount of data cleaning would be required.

To begin, the training data needed to be imported and assigned dimension types. I followed the reasoning found here: https://www.kaggle.com/theoviel/load-the-totality-of-the-data. This form is used by many of the data scientists in the competition. In general, it attempts to import the data with dimension types that lessen the amount of data as much as possible and allow the data to be computed on. More specifically, it loads objects in the data as categories, binary without nan terms as int8, binary values with nan as float 16, and attempts to convert float64 values to float 32 and float 16 when possible. This will greatly help with computation later.

Upon importing the data with the correct dimension types, I began to move into data cleaning and analysis. I found the Kaggle thread here: https://www.kaggle.com/jiegeng94/everyone-do-this-at-the-beginning to be very helpful. To start, I created an empty array to hold a list of features that can be reasonably dropped from the data set. I then checked the data features to find all features that contained 99% or more missing data in their columns. Because they contain more than 99% missing data it is reasonably acceptable to drop them from our analysis. These categories were 'PuaMode' and 'Census_ProcessorClass'. I then checked to see if there were any categories whose data was too skewed. In performing an analysis on the data, data that is too skewed is virtually worthless to our computations as it will not effect our algorithm enough. After checking this, it could be seen that there were 12 categories that contained data that was skewed more than 99%. These categories were: 'PuaMode', 'Census_ProcessorClass', 'Census_IsWIMBootEnabled', 'IsBeta', 'Census_IsFlightsDisabled', 'Census_IsFlightingInternal', 'AutoSampleOptIn', 'Census_ThresholdOptIn', 'SMode', 'Census_IsPortableOperatingSystem', 'Census_DeviceFamily', 'UacLuaenable', and 'Census_IsVirtualDevice'. It is important to note that any duplicates in these results and previous results were then accounted for in the array.

Once these categories are dropped, we can then look into features that have more than 10% missing data. This data will need to filled in so that we can use them in our analysis. The columns found to have more than 10% missing data were 'DefaultBrowsersidentifier', 'OrganizationIdentifier', 'SmartScreen', and 'Census_InternalBatteryType'. The missing values in 'DefaultBrowsersidentifier' are all then replaced with 0's. The 'SmartScreen' is a categorical value, we must instead augment this value with a dictionary that linearizes the values. The exact dictionary can be found in the notebook. The 'OrganizationIdentifier' has all of its nan values with 0's, and the 'Census_InternalbateryType' value has all non-normal values replaced with unknown. We can then re-label our category columns to ensure they are correctly dimensioned in our new dataset

After augmenting these columns, we can drop all rows in the data set that contain a nan value. This brings our total shape from (8921483, 83) to (7667789,70). We can then run the code found here: https://www.kaggle.com/timon88/load-whole-data-without-any-dtypes. This code iterates through each of the features and attempts to modify the associated data types to reduce the data memory usage. In essence, it the column is an integer type, it cycles through each of the possible integer types (int8-> float64) and attempts to fit to the lowest possible type. Otherwise it keeps the column as a category. This helped reduce our memory usage from 1703.83 MB to 965.26 MB, a decrease in memory usage of 43.4%.

Now that the memory has been checked for correct dimension type, for missing values, for skewed data, and have filled in data with more than 10% data missing, the correlation of the data can then be checked and the data can be further refined.

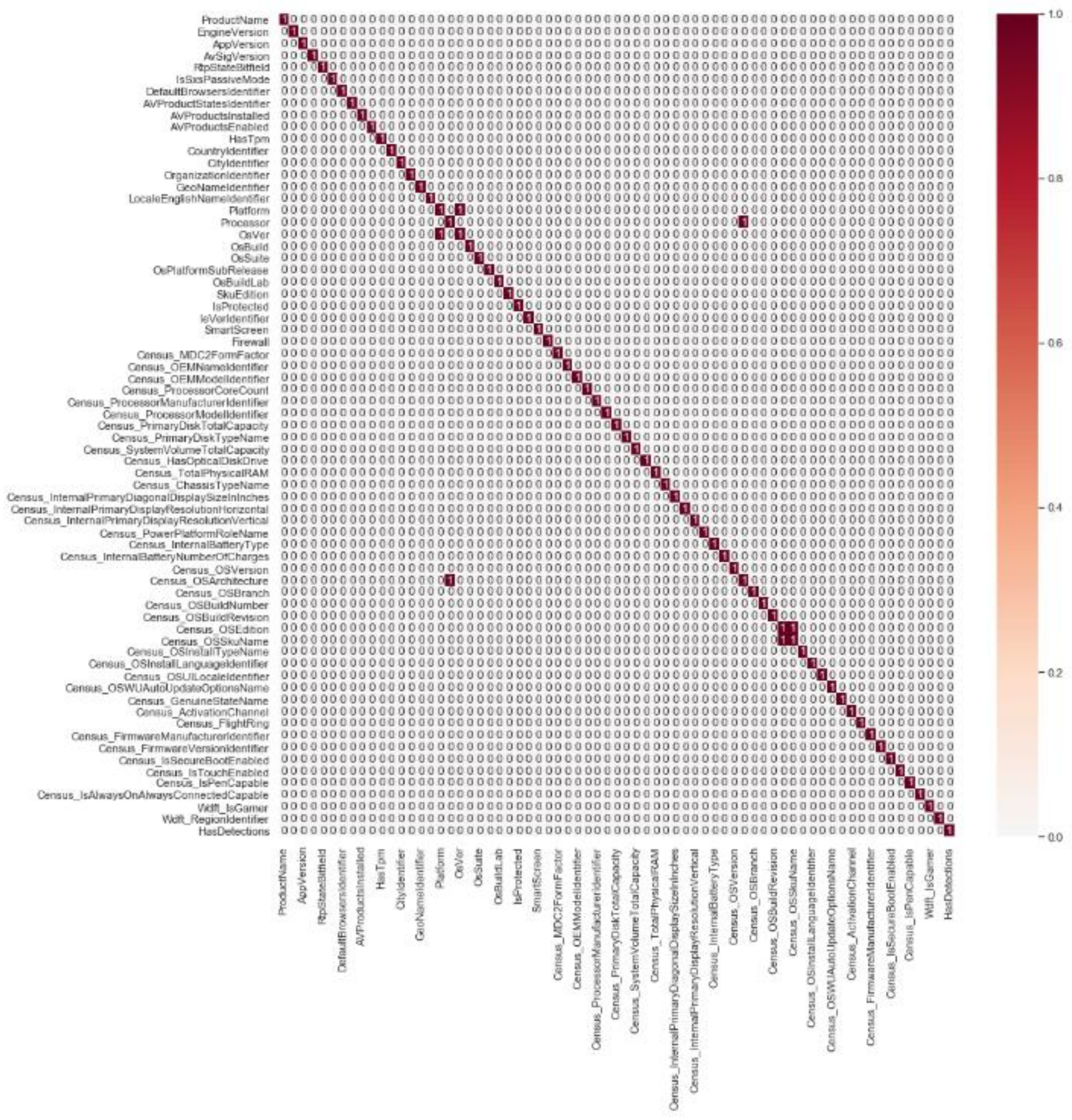

Figure 1: Data correlation across all of the remaining 70 features in the train data set.

To begin, a correlation test was run across the entirety of the train data set, the results of which can be seen in figure 1(above). This test was run to find instances of features with a correlation value of .99 or higher. There are a total of 3 values in the data set, but it is helpful to break down the correlation values into sets of 10 to get a closer look at these values.

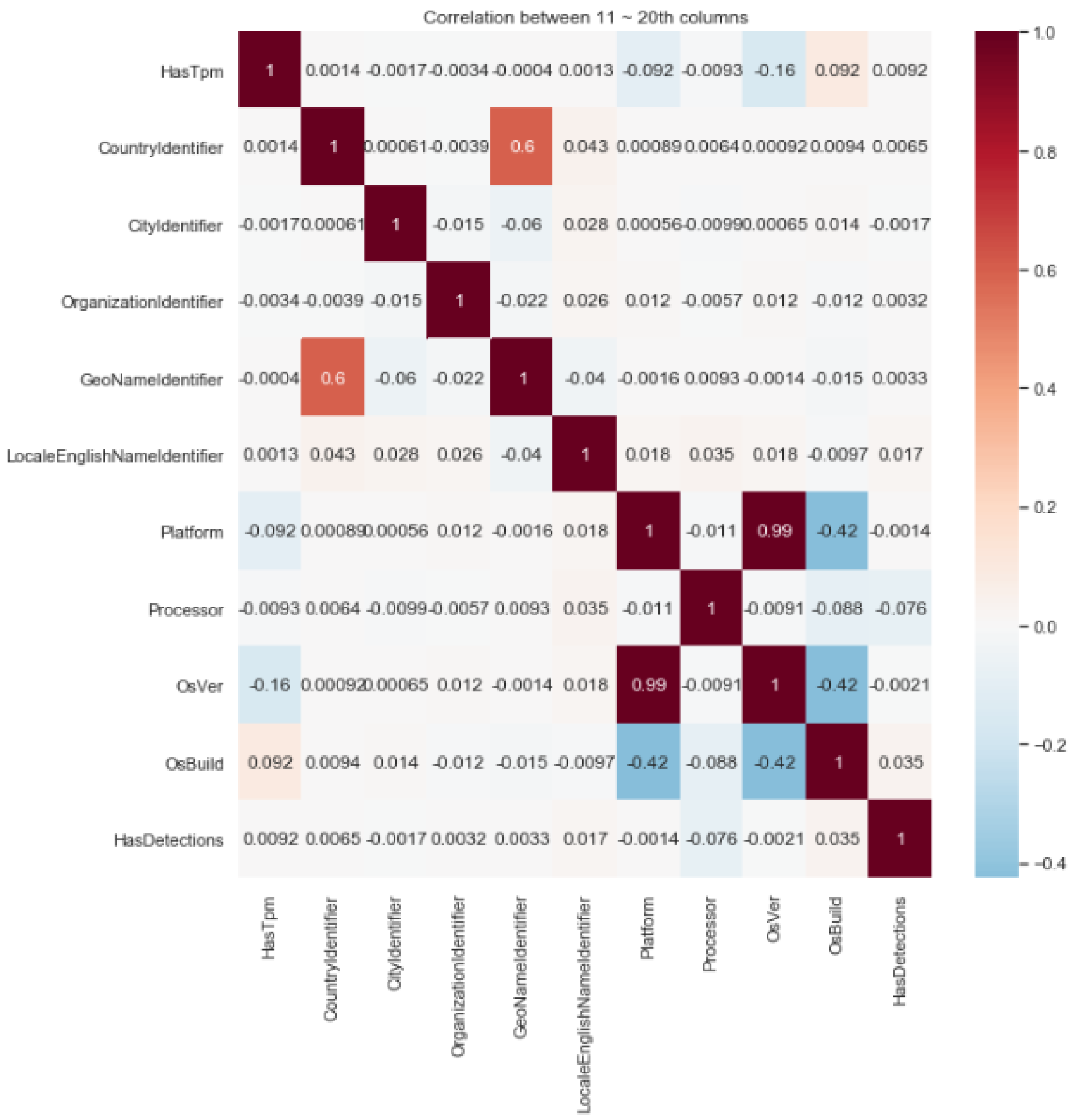

Figure 2: Correlation values for the 11th-20th features in the data set.

The data can be more closely analyzed by dividing the remaining features into sets of 10 and calculating the correlation between all of them. In performing these correlation plots we are looking for data that contains a correlation of greater than 99%. Figure 2 shows an example correlation plot of the data, in this example it can be seen that the 'OsVer' and 'Platform' features are heavily correlated and this can be dropped. These plots were generated for all sets of data up to 70, and correlation values that were greater or equal to 0.99 were dropped from the data set.

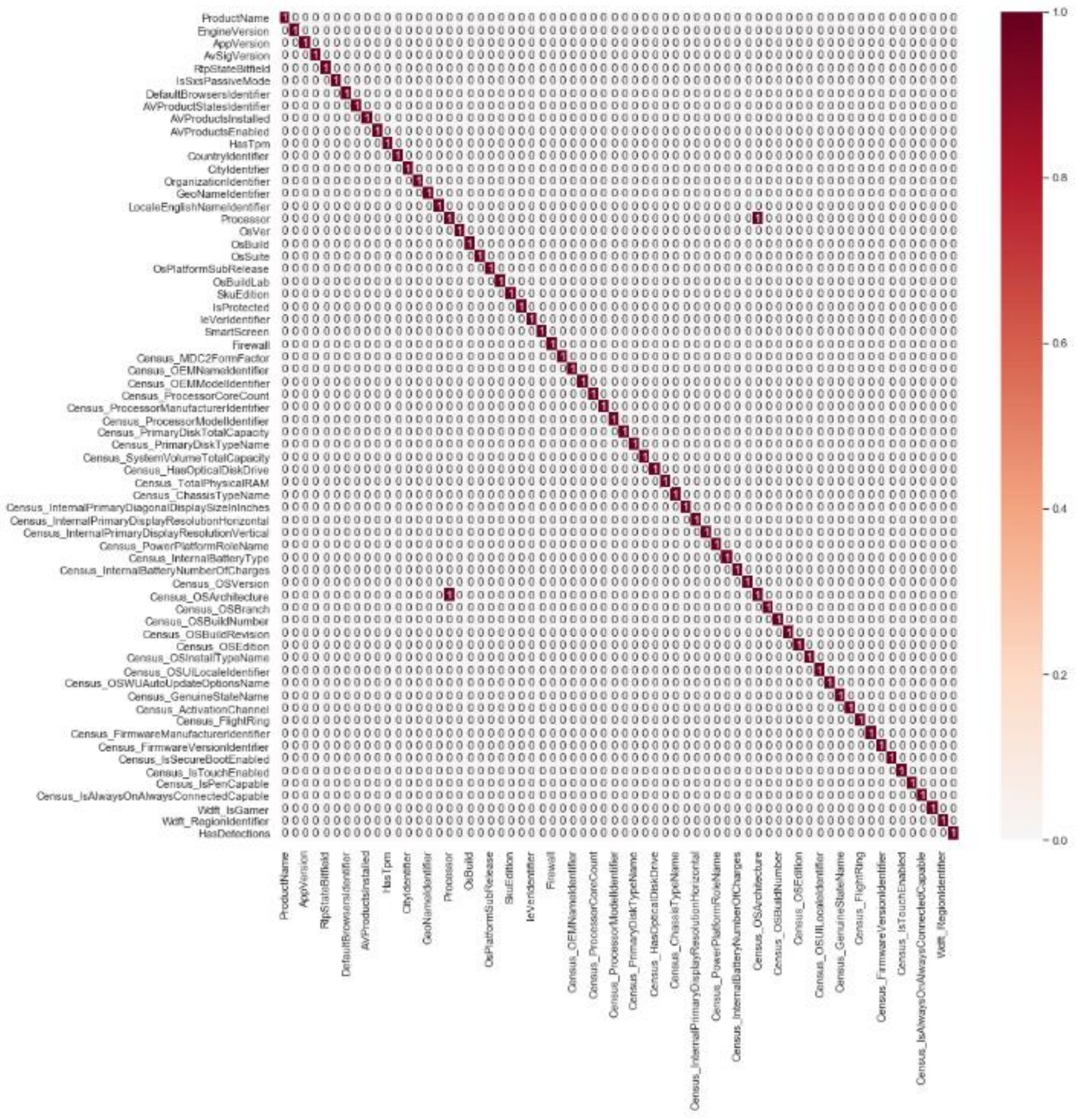

Figure 3: Correlation values for the entire data set after dropping correlation values from the sectioned tests.

The correlation test was then ran one more time after dropping the correlation values from the data set to ensure that there was no correlation values that were overlooked because of the sectioning. Figure 3 displays this information. After running this test, it was found that there was one value that was overlooked, and the associated information was dropped as well.

We are able to drop values with large data correlation because two data sets having a large correlation means that both of the features effect our analysis in the same way. Having both of the features in our analysis would in essence double the amount of data we need to fit to, without providing any benefit in terms of accuracy.

After completing the drops of data from the data analysis the data set was relatively close to being ready for analysis. In the following analysis, the data was fit to multiple regression models. Ultimately, it was found that the data would need slightly more cleaning, but that will be covered later in this report.

The final action of the data analysis and cleaning was to take train data set and split it in half to form a new training and test data set. This allows for us to test our regressions as we go along the analysis so that we can test our data for accuracy, and more importantly, it significantly decreases the time to compute our fits.

Model Architecture and Results:

In determining which model architecture would most effectively encapsulate this data, I decided to go with a three-pronged approach. To begin, because the data set is large and is relatively diverse with a large number of categories, I decided to attempt to run a KNN fit to the data with k=1. Upon running this fit, I quickly found that the data was still too large to run the calculation with the full data set. Instead of moving onto a different analysis technique, I decided to try to optimize the solution.

To begin, the training data set was obviously too large and was taking exorbitant amounts of time to compute. In an attempt to immediately gain some speed, I allocated all processor cores to the calculation, and hard coded the algorithm used to be a kd-tree algorithm. In essence a kd-tree algorithm is better suited to handle larger data structures in a k-dimensional space. I also set the classifier to use the Manhattan distance rather than the Euclidean distance, as it also works better with larger data sets. Perhaps the biggest jump in performance though came from me training the data to every 10th point in the training data set. This is not an ideal approach, but with my computing limitations was a requirement to perform the analysis in a timely manner. This allowed for the fit to be calculated in 3 minutes instead of the days it would have taken to fit the knn before cleaning. Once the data was fit, I proceeded to predict the data values according of the test set. The predictions for the entirety of the test set took a combined 8 minutes to complete and resulted in a 55 percent success rate.

I then performed a logistic fit on the data. The solver used in this process is the familiar 'lbgfs' solver from homework 8. This test was rather straight forward and ultimately resulted in an accuracy of 51 percent. Upon plotting our data on an sns plot, we can see that the prediction probability is centered around .5 for this analysis.

The next test I ran was a basic linear discriminant analysis. This fit only took 37 seconds and ultimately resulted in an accuracy of 60 percent of the data. An accuracy reading of around 60 percent is beginning to approach what we would call an acceptable range for our data set, however there is still room for improvement.

As the final check for our data set, a quadratic discriminant analysis was conducted. This approach ultimately yielded an accuracy metric of 52 percent. This analysis was run at the base level, with the original intent of taking advantage of the multidimensional functionality of the set.

Although the linear discriminant analysis produced the best initial results in the test, I still wanted to see if there was a better way to clean the data so that a more intensive testing procedure could be used. To do this I decided to implement a recursive feature elimination scheme. Recursive feature elimination (RFE) is a package found within the sklearn suite and it allows us to find the features that are 'most relevant' in our data set. We decide to do a logistic regression first to test out the recursive feature elimination tool with 20 features. The RFE computes the most important values and we drop them from our data. We can then run our logistic regression again with the smaller data set, and see that we receive a value of 51 percent for the accuracy, this means that we have computed the same accuracy using a much smaller set of data. Although this is a great way to reduce computation time of our analysis, it does not specifically improve accuracy of the data.

Unfortunately, KNN does not support RFE, so the only real way to improve our KNN metric more is to increase the number of training sets.

Now that we have found that the LDA performs the best against this data set, we can then turn our focus towards analysis of the test data type. After loading the test data type I quickly realized that the categorical values, at least in my test data type were completely different than the categorical values in my training set. I am unsure if I am working with corrupted files or not, but regardless, the test file will need some additional cleaning. To start the categorical values were all checked to see if they could be converted to another form. After completing this I unfortunately was forced to throwing out all values that could not be worked with easily. Given additional time I could have cleaned the data further, but there is still a large amount of data without it to get a relatively viable solution. After throwing out all unwanted data, we could then fit an LDA to our remaining features and run our test set through the fit. After running the test set through the fit, and outputting it to a csv, the csv was submitted and we received a score of .57621

Conclusion:

Overall, the data analysis presented many intense challenges to test our newfound knowledge of machine learning techniques. The most intensive part of the entire project was the EDA and data cleaning by far. Assuming all of the data cleaning was done perfectly the first time, I believe a solution to this problem would be relatively straight forward. Two of the biggest challenges I faced in this project was general lack of ram, and misjudging my initial analysis of the test data. At first glance I wrongfully made the assumption that the test data I had was the same as the training data given to us. After completing most of the project and getting ready to run against the test set, I discovered that the data set was not the same and it was very hard to convert any of the categorical values to something that could be used. I am still unsure if this is on purpose, or if my file was corrupted, as the characteristics experienced did not wholly reflect anything scene in the discussion boards. Unfortunately, due to time constraint, I had to abandon my additional cleaning effort and simply throw out all of the categorical data that could not be converted. Given more time, I believe the solution would be much more precise if I could find a way to include that data. I also did a large amount of exploratory analysis into the possibility of using random forest generation to solve this problem. After reading many message boards about the topic, I believe that the random forest method is the best way to solve this problem currently, and could provide us with a much better result. Unfortunately, due to my lack of knowledge in this field, I was unable to perform this analysis, but I hope to come back and test it in the future. Regardless, a score of 0.57621 is not terrible for a first machine learning project.