**SE526 Final Project**

**Wills Mckenna**

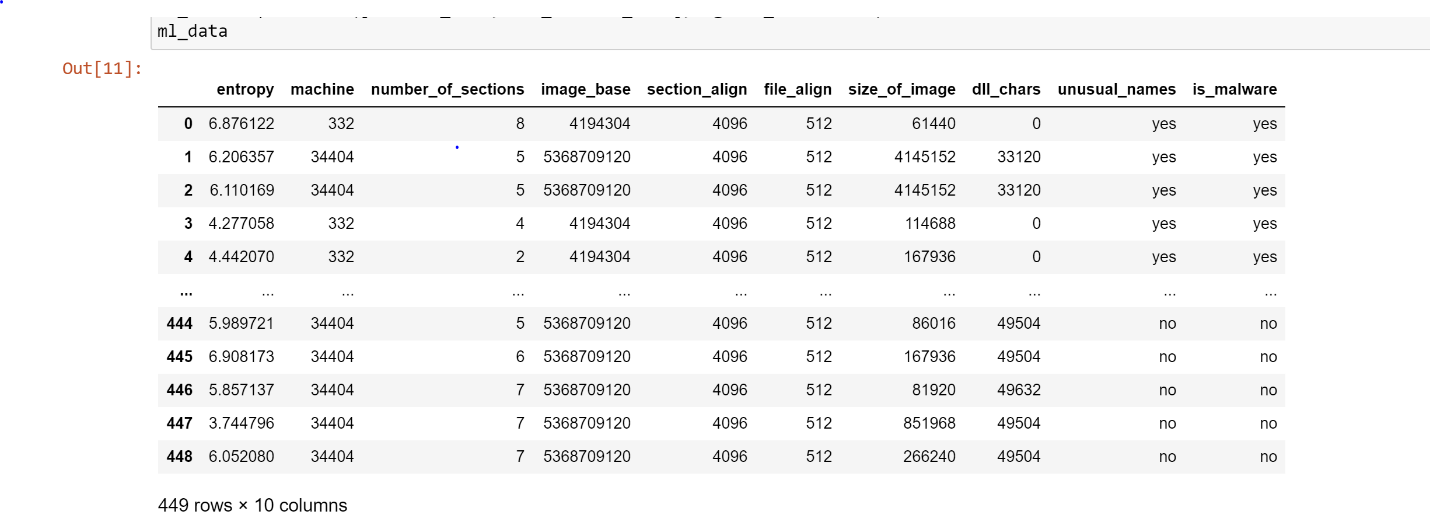
**March 14, 2021**

Malware prediction with Machine Learning models

Due to the steadily increasing numbers of cybersecurity attacks and newly developed malware that the IT industry faces today, new and effective methods of addressing some of these issues must be developed. It has been found that some of the problems can be solved using Artificial Intelligence and Machine Learning. According to Gilbert, Mateu, and Planes (2020)- “There is general belief among cybersecurity experts that AI-powered antimalware tools will help detect modern malware attacks and improve scanning engines.” One such antimalware tool to combat the influx of new malware in today’s world is malware detection using Machine Learning classification methods and models. Classification in Machine Learning is a form of supervised learning where our program learns from the data it collects or that we provide to it, and this data is associated with a corresponding class label (Harrington, 2012). Class labels are a single identifying data feature that describes that particular data instance- so in the case of malware, a class label would simply be “yes” or “no” for whether it is malware or not. The focus of this project was to develop a simple prototype for classifying malware and experiment with optimal models to create the best predictor program possible.

DATA COLLECTION, ANALYSIS, AND PREPROCESSING

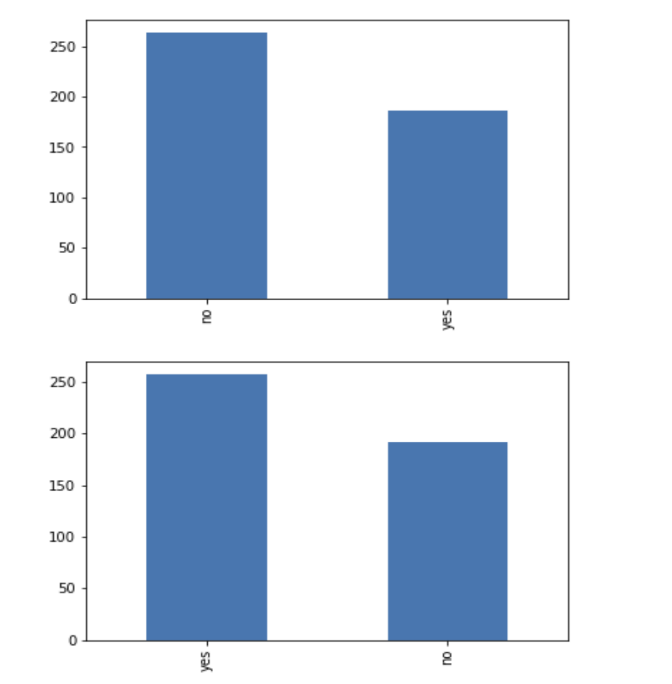
For this project over 1300 samples of Malware were provided for analysis and model development, however due to some parsing difficulties and time constraints 257 samples of malware were included in the dataset and 197 samples of non-malware samples were included. Non-malware samples were collected by searching .exe programs on this writer’s Windows machine. A “file parser” program was implemented using the Python PE File module. The program took a file and recorded 8 characteristics of each- the file’s entropy, machine, number of sections, image base, section align, size of image, dll characteristics, and the names of the PE sections. Entropy is a number that reflects a programs’ overall level of randomness, and is often telling if a file has been obfuscated or not (Morgenstern, 2016). The higher the entropy, the more likely the program is extremely hard to make sense of, and malware developers will often compress or obfuscate their code in order to avoid being detected by AV software. Odd or unusual names for PE sections outside the realm of what is usually in a PE file (i.e .text or .data) is also possibly indicative of malware. For this project, files were grouped into two categories: either they had unusual pe section names, or they did not. Rationale was based on common PE section names found in malware. The final dataset that was arrived at is shown below:

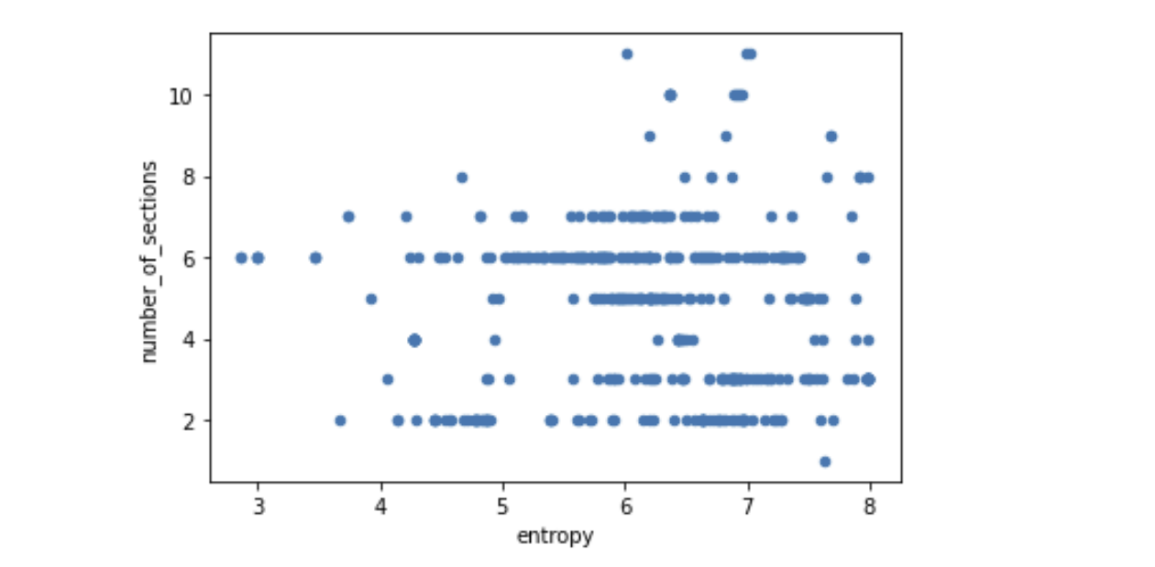


Entropy and number of sections were continuous values, and the rest were categorical. To get the data set ready for the machine learning models, the data’s categorical features were “dummied”, meaning they were turned into binary representations. The continuous values were normalized on a scale of 0 to 1 using Scikit learn’s preprocessing module.

The data was split between non-malware and malware fairly evenly, with a slight lean towards malware. Most malware samples had unusual PE section names, and all non-malware samples did not have any unusual PE section names. Some malware samples, however, also did not have unusual PE section names. Below are bar charts illustrating these numbers:

*(Top graph shows numbers of data points with is\_malware = ‘yes’ vs. is\_malware = ‘no’, bottom shows ‘unusual\_names’ = ‘yes’ vs. ‘unusual\_names’ = ‘no’. )*



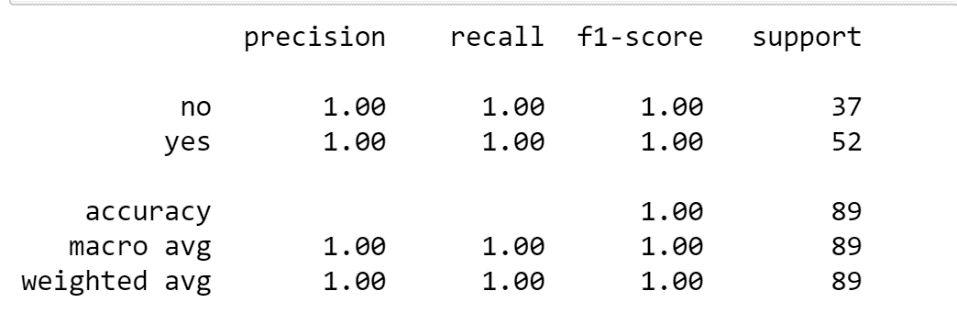


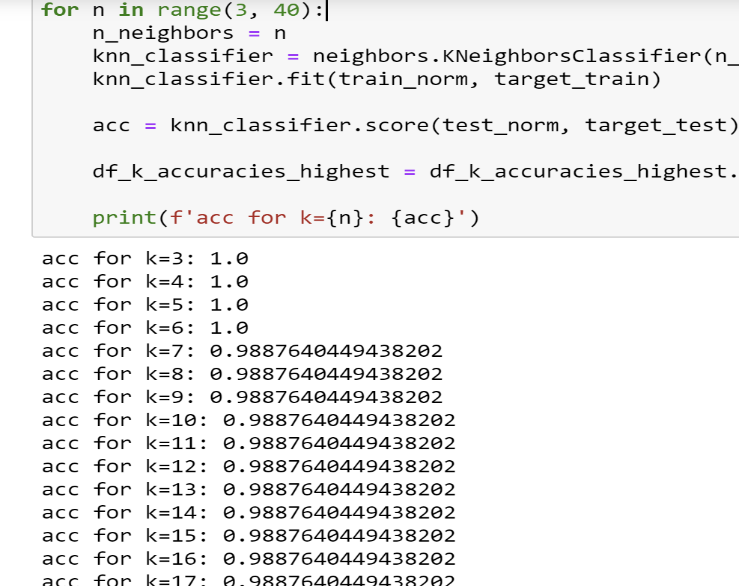
The above graph was a scatter plot of the two continuous features- entropy and number of sections. There appears to be a slight correlation between these two values, in that as the entropy of a file increases, so does its number of sections.

Lastly, before the classification algorithms were run on the data set, some additional data preprocessing was done to bring all the values to the same scale. This was done through using the min-max normalization function in scikit-learn, and all values were brought down to a scale of 0 to 1. This is so certain models, namely K-nearest-neighbors, could be the most effective and accurate.

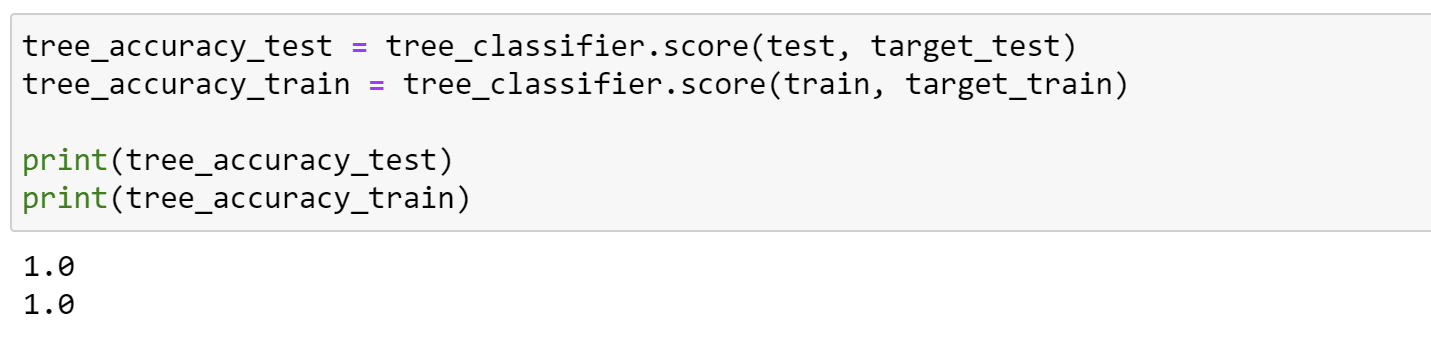
CLASSIFIER DEVELOPMENT

Two classifiers were chosen for this project- the Decision Tree and K-nearest-neighbors. The rationale for this was that these two models are some of the most common, they are easy to understand and implement, and in the case of Decision Tree, does feature selection for you. Both algorithms were implemented using the scikit-learn module. Initially, all features were used for the KNN implementation. Distance metric used throughout was Euclidean. Results showed a perfect accuracy of 1.0 on the training set for K’s between 3 and 6, and then dropping off from there as K increased. For the final KNN model a K of 5 was chosen. Other evaluations on the model were performed as well including examining Precision and Recall, which all were 1.0. The results of the KNN experiment are shown below:





The Decision Tree was implemented using entropy as a criterion parameter and a min-max-split of 3. The tree’s accuracy in both training and testing was also 1.0, similar to some KNN outcomes. Below are the results:



After examining the results of both classifiers, it was decided that some feature reduction should take place in an attempt to shore up the model and create something that could be more accurate on new data. A new KNN model was developed that only included three features: entropy, number of sections, and the unusual pe section names flag. Results of testing accuracies on this model are shown below:



The model in this case was slightly less accurate on the training set, but this could be beneficial due to it being less overfit. The best K of three was chosen. Overall, the decision tree with all the features proved to be the most accurate on training and testing, but the KNN with only the three features proved to be the most effective when handling new input/data. This was ultimately used in the app implementation of the model.

APPLICATION

The final piece of the project was creating an application that used the above found models to classify never before seen files. User input at the command line was used and the application was developed in PyCharm. The application used elements of both the file parser program developed early on in addition to the KNN and Decision Tree classification models. The user was prompted to enter an absolute file path at the command line, and this file was located by the program, parsed, converted to a Pandas Dataframe, and normalized to fit the format of the training and testing data. A simple yes or no answer was provided to the user upon running of the model. An example screen shot of what user input looked like is below, with a non-malware executable found on this writer’s local machine:



STUDY FINDINGS, LIMITATIONS AND CONCLUSION

The above explorations show that the development of malware identification technology lends itself very well to machine learning models, particularly classification algorithms due to the binary nature of the end result (“yes” for malware, “no” for non-malware). Indeed, this small application has shown that machine learning and more broadly artificial intelligence could be a vital tool in cybersecurity. Machine learning could be an important form of defense in fighting the growing number of cyberattacks and malware that plagues the IT industry today.

More specifically, the findings in this study showed that machine learning techniques can be used effectively to correctly classify, with a relatively small amount of information, previously unseen files. However, this application is in the prototype stages and needs more fine tuning and data. Due to time constraints a small subset of data was used for the models, and the application could be more accurate and effective if all 1300 malware samples were processed. As explained above, because there were roughly 250 malware samples and 250 non-malware samples, this caused the models to overfit on the training data, which means that when the models were then tested on previously unseen data, they may have been more prone to predicting incorrectly, supplying either a false positive or negative. Additionally, more algorithms, such as Naïve-Bayes or Support Vector Machine, could have been applied to examine their efficacy. Ensemble methods such as random forest, which uses multiple decision trees, could also have been applied. These additions could be demonstrated in future iterations of the application to provide a more effective and complete model.

WORKS CITED

Gilbert, Mateu, and Planes. “The rise of machine learning for detection and classification of malware: research developments, trends, and challenges.” *Journal of Network and Computer Applications.* Vol. 153,1 March 2020.

Harrington, Peter. *Machine Learning In Action.* Shelter Island, NY, Manning Publications, 2012.

Morgenstern, Tal. “Malware Terms for Non-Techies- Code Entropy”. December 15th, 2016. https://www.cyberbit.com/blog/endpoint-security/malware-terms-code-entropy/.