

**Ghost Hunting:**  
**Finding a Botnet in NetFlow Traffic**

John Tomaselli, Emma Delehanty,  
Steven Moore, Mukila Rajasekar

DSC522

Data Science Program, University of North Carolina Wilmington

**Abstract**

Searching for malware is like hunting ghosts. Purposefully designed to blend in with normal system traffic, the average network security officer can easily miss malware transmissions traversing his or her network amongst the vast amount of other material (Gunter, 2020). This project is designed to determine the efficacy of machine learning classification and data modification techniques in identifying those malware transmissions. First, we clean the data and make determinations about how to extract the maximum amount of information from the available variables. Next, we directly employ a range of classification techniques to discern which are optimized for this type of dataset. Finally, dimension and dataset reduction techniques are employed to see how efficiently high accuracy results can be achieved. In most cases, our efforts were successful, validating the use of many of the machine learning classification and data manipulation techniques explored for use in classifying botnet traffic.

## **Introduction**

No network is safe. Malicious entities target both enterprises and individuals every day, consistently putting intellectual property, money, and privacy in jeopardy. This trend is growing with a staggering 13% increase in known corporate intrusions in 2019 alone (Fruhlinger, 2020). To counter this trend, network security officials employ a range of network monitoring and intrusion prevention systems that identify and stop network exploitation (Dalton & Turner, 2019). Our goal was to leverage a sample of network traffic and the techniques employed in DSC522 to classify malicious botnet traffic. The success we found in correctly identifying traffic emanating from malware allowed us to appreciate the potential for these techniques as well as to understand how network traffic needs to be processed to accommodate many machine learning methods.

## **Project Overview and Cleaning**

Before reviewing our results, an overview of the dataset is needed. In 2011, the Czech Technical University (CTU) generated network packet capture (PCAP) in order to show how malicious botnets, a form of malware, appear when present in otherwise normal, benign network traffic (Stratosphere Lab, 2020). The aforementioned PCAP, known as CTU-13, was posted publicly online for researchers to use. This project used the NetFlow of a portion of that PCAP, showing high-level summary data characterizing each network. While only a small part of the overall CTU-13, the selected dataset still contained approximately 1.29 million NetFlow observations. Figure 1 shows a sample of the NetFlow data. Figure 2 defines each NetFlow variable.

Date flow start	Durat	Prot	Src IP Addr:Port	Dst IP Addr:Port	Flags	Tos	Packets	Bytes	Flows	Label Labels
2011-08-15 17:13:40.449	4.230	TCP	90.177.154.197:20127 ->	147.32.84.144:22	PA_	0	180	12628	1	Background
2011-08-15 17:13:40.449	4.204	TCP	147.32.84.144:22 ->	90.177.154.197:20127	PA_	0	329	478578	1	Background
2011-08-15 17:13:40.453	4.985	TCP	198.36.38.132:55530 ->	147.32.86.183:443	PA_	0	94	6212	1	Background
2011-08-15 17:13:40.456	4.975	UDP	41.103.64.21:54617 ->	147.32.86.183:51246	INT	0	182	61555	1	Background
2011-08-15 17:13:40.456	4.975	TCP	147.32.86.183:443 ->	198.36.38.132:55530	PA_	0	185	66524	1	Background
2011-08-15 17:13:40.462	4.984	TCP	147.32.80.13:80 ->	147.32.84.162:60840	PA_	0	74	85085	1	LEGITIMATE
2011-08-15 17:13:40.463	4.984	TCP	147.32.84.162:60840 ->	147.32.80.13:80	A_	0	36	2160	1	LEGITIMATE

Figure 1: Example NetFlow Data

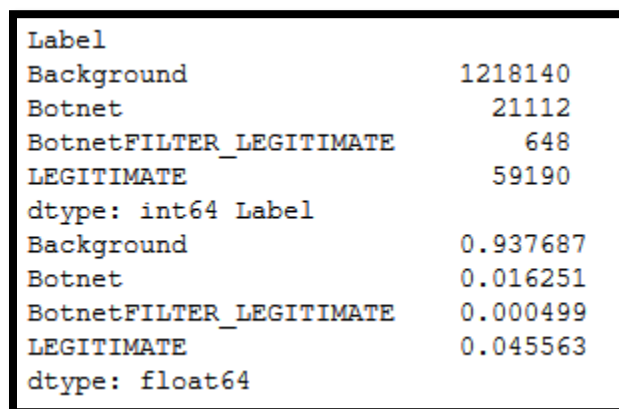
Variable	Definition
Date Flow Start	The time, down to the microsecond, when the traffic flow associated with each connection began.
Duration	The amount of time in which the connection occurs.
Protocol	The type of Internet Protocol (IP) used for relaying datagrams across network boundaries.
Source IP Address / Port	The pairing of the logical location and service from which requests across the internet are sent.
Destination IP Address / Port	The receiving pair of location and service from which an action is requested.
Flags	Indicators of the type of action executed in the traffic flow.
Type of Service (TOS)	In this context, the precedence given the packet.
Packets	The total number of datagram packets in the flow.
Flows	The number of flows required to complete the transmission.
Label(s)	The response variable, indicating whether the flow contains <ul style="list-style-type: none"> <li>1.) Background traffic, or traffic not from an infected host.</li> <li>2.) Legitimate traffic, or benign traffic from an infected host.</li> <li>3.) Botnet traffic, or traffic supporting malicious activities.</li> </ul>

Figure 2: NetFlow Definitions

Cleaning the above dataset proved challenging. Missing values or non-alphanumeric characters, such as the flow arrows seen in Figure 1, were either filled-in or removed, depending on how each value affected the information contained. One-hot encoding the protocol, port, and flags variables enabled these categorical variables to remain and contribute to the final model. All the date values contained the same strings and were removed. While the flow start, source IP, and destination IP variables were

initially retained, these played no direct part in classification of network traffic. Instead, all 3 variables would later be evaluated and further processed in questions 1 and 2.

After initial regression tests, evidence mounted for the need to conduct more modifications to the dataset. All tests returned very high classification values of above 95%, but classification of botnet traffic suffered greatly compared to background and legitimate traffic. Further investigations revealed that a large imbalance existed between the number of botnet observations compared to the number of legitimate and background traffic. As Figure 3 shows, botnet traffic only accounted for about 1.7% of observations with the remaining 98.3% of observations either being background or legitimate traffic. Our results skewed because the produced models heavily favored classification of legitimate and background traffic over botnet traffic.



Label	
Background	1218140
Botnet	21112
BotnetFILTER_LEGITIMATE	648
LEGITIMATE	59190
dtype: int64	Label
Background	0.937687
Botnet	0.016251
BotnetFILTER_LEGITIMATE	0.000499
LEGITIMATE	0.045563
dtype: float64	

**Figure 3: Observation Labels and Label Percentages**

Downsizing the dataset produced the balanced results we desired. Without the ability to inflate the number of botnet observations, we used the `resample()` function to scale down the number of observations associated with non-Botnet traffic. Figure 4 displays the results of downsizing, producing a total of 43,520 observations. The response variable, `Label`, was separated from the dependent variables and further

consolidated into two bins, indicating 'yes' if the observation was associated with botnet traffic and 'no' if otherwise. A total of 21760 observations were included with each bin. Consolidation of all the previous steps produced the variables seen in figure 5, with the letters following the Bytes variable indicating one-hot encoded flags and the numbers at the end indicating one-hot encoded ports.

```
Botnet          21112
Background      20745
LEGITIMATE      1015
BotnetFILTER_LEGITIMATE  648
Name: Label, dtype: int64
```

**Figure 4: Downsized Dataset.**

	Flow_Start	Source_IP	Destination_IP	Duration	Packets	Bytes	A_	CON	ECO	ECR	...	83	832	843	877	88	888	916	98	993	995
4117	17:13:56.723	64.12.175.136	147.32.84.165	0.000	1	263	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6269	17:14:06.078	212.117.171.138	147.32.84.165	0.000	1	66	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8919	17:14:17.100	94.63.149.150	147.32.84.165	2.712	4	256	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10127	17:14:22.915	94.63.149.150	147.32.84.165	0.000	1	64	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

**Figure 5: Consolidated Dataset Before Question 1**

## Question 1

Further modification of the dataset occurred in questions 1 and 2. Question 1 asked whether the variables in the dataset can be effectively expanded using the GeoIP2 Python third-party library. The GeoIP2 library uses IP addresses and the GeoNames database to extract data points about the logical and physical locations in which IP addresses are based (Python Software Foundation, 2020). Using this platform, we first asked if the geographic location of each observation can be ascertained down to the city and country. Next, we asked what further information that

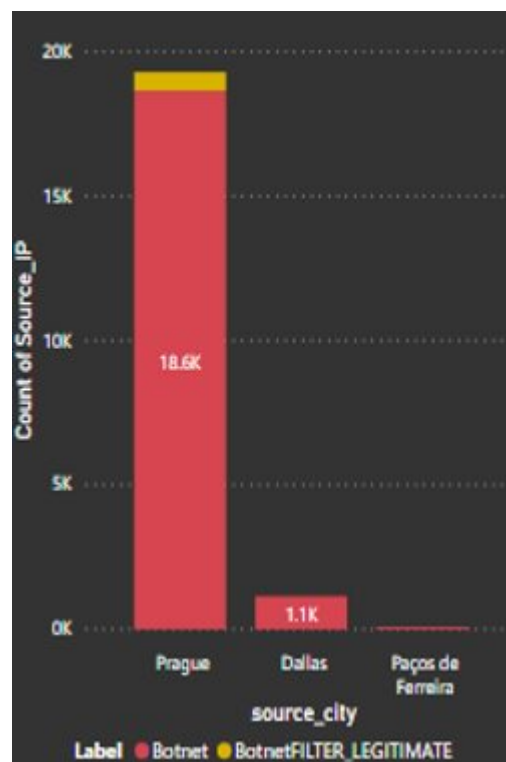
can be effectively derived from the data. To answer the above questions, only the source IP address for each observation was employed with GeoIP2 for two reasons. First, depending on the type of botnet attack, the victim can be disproportionately represented in destination IP addresses as UDP packets are not connection-oriented (CloudFlare, 2020). Second, in attacks involving the TCP protocol or other connection-oriented protocols, we would double count the sending and receiving locations by including both the source and destination IP addresses.

The results were very focused. For most observations, both the source city and country populated. Figure 6 shows that while Botnet traffic did originate from all around the globe, the tightest concentration occurred in Prague, the capital of the Czech Republic. Figure 7 shows that the amount of botnet traffic originating from Prague vastly outweighed the amount of traffic coming from the next-highest location. While other variables available through GeoIP2 were also explored, such as the postal code and continent from which traffic came, these variables proved to contain redundant information and were discarded. Once chosen, the city and country variables complimented the other variables by being one-hot encoded and added to the processed dataset, as shown in figures 8 and 9.



**Figure 6: Botnet Traffic Patterns**

**(Packet Origination in Red Bubbles with Packet Destination at Line End without Bubble)**



**Figure 7: Botnet Source**



	IPX/SPX	PIM	RTCP	RTP	TCP	UDP	Alameda	Almaty	Amsterdam	Arichuna
4117	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
6269	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
8919	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
10127	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
11546	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

**Figure 8: Demarcation between Protocol and City One-Hot Encoded Variables**

	Ukraine	United Arab Emirates	United Kingdom	United States	Uzbekistan	Venezuela	Zimbabwe	0	1	1000
4117	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
6269	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8919	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10127	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11546	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

**Figure 9: Demarcation between Country and Port One-Hot Encoded Variables**

## Question 2

Exploring more ways to add information to the dataset, we asked in question 2 whether we could effectively expand the variable list by splitting certain variables and meaningfully extracting values from them. Determining this mostly relied on qualitative research concerning the variables we acquired with CTU-13, although some analysis of the data occurred. Starting with the Flags variable, Figure 10 displays the full list of values available. Several of these flags did not appear to have a direct correlation with known TCP flags (Rubicon Communications LLC, 2020). Figure 11 shows why as several unexpected protocols appeared in the dataset. Exemplifying the results of our investigation into the flags of these protocols, flags associated with the Internet Control

Message Protocol (ICMP), displayed in Figure 12, represent transmission states with multiple letters (Su, 2020). This differs from TCP, where different attributes of a transmission are represented by different individual letters, as in Figure 13. Even if the TCP flags were split, we could not find evidence that this would produce more information. Given the above, we decided to keep all flags unmodified.

```
df.Flags.unique()
executed in 70ms, finished 09:20:11 2020-04-26
array(['PA_', 'INT', 'A_', 'SA_', 'FSPA_', '_FSPA_', 'URP', 'S_', 'RA_',
      'FSRPA_', 'SRPA_', 'FA_', 'SPA_', 'FPA_', 'RED', 'ECR', 'URH',
      'R_', 'FSA_', '_FSA_', 'FRA_', '_FSRPA_', 'FRPA_', 'SR_', 'ECO',
      'SRA_', 'CON', 'TXD', 'PAC_', 'RPA_', 'NRA', 'URN', 'FSRA_',
      'URFIL', 'NNS', 'FRPAC_', 'URHPRO', '_FSRA_', 'FSRPAC_', 'URO',
      'F_', 'FSPAC_', 'RE_', 'FS_', 'FPAC_', 'SPAC_', 'URF', 'FAU_',
      'RPA_FRPA_', '_FSPAC_', 'RC_', 'FR_', 'FSR_', 'SRC', 'SEC_', 'FAC_',
      'FRAEC_'], dtype=object)
```

Figure 10: Unique Flags in CTU-13

```
df.Protocol.unique()
executed in 57ms, finished 13:53:54 2020-04-01
array(['TCP', 'UDP', 'ICMP', 'IGMP', 'ARP', 'PIM', 'IPX/SPX', 'RTCP',
      'IPV6-ICMP', 'IPV6', 'RTP'], dtype=object)
```

Figure 11: Unique Protocols in CTU-13

```
Only_ICMP_Flags = df[df['Protocol']=='ICMP']
Only_ICMP_Flags.Flags.unique()
<
executed in 108ms, finished 09:22:45 2020-04-26
array(['URP', 'RED', 'ECR', 'URH', 'ECO', 'TXD', 'URN', 'URFIL', 'URHPRO',
      'URO', 'URF', 'SRC'], dtype=object)
```

Figure 12: ICMP Flags in CTU-13

```

Only_TCP_Flags = df[df['Protocol']=='TCP']
Only_TCP_Flags.Flags.unique()

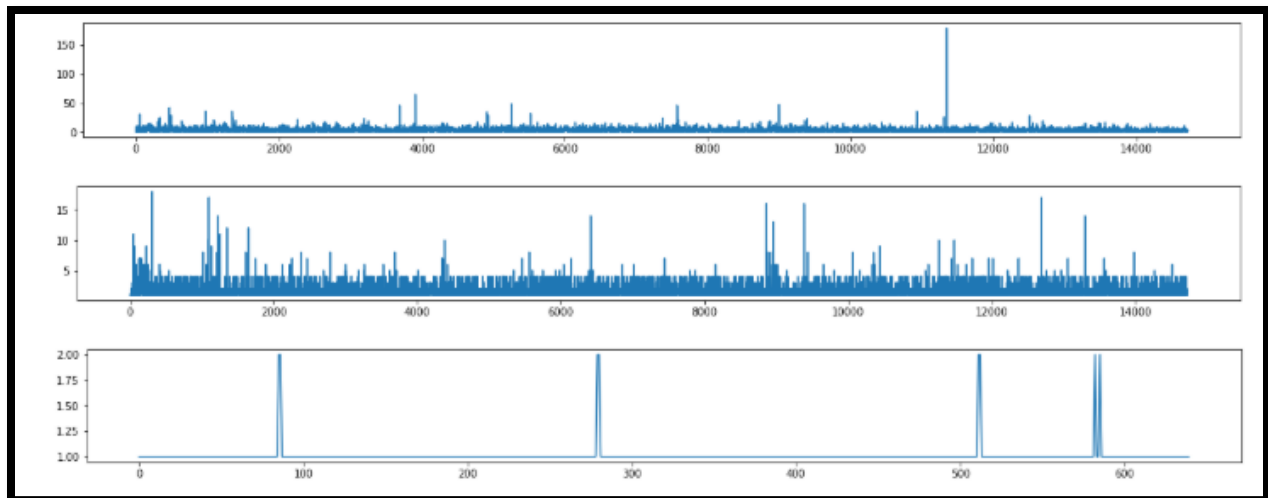
executed in 249ms, finished 09:22:45 2020-04-28

array(['PA_', 'A_', 'SA_', 'FSPA_', '_FSPA', 'S_', 'RA_', 'FSRPA_',
      'SRPA_', 'FA_', 'SPA_', 'FPA_', 'R_', 'FSA_', '_FSA', 'FRA_',
      '_FSRPA', 'FRPA_', 'SR_', 'SRA_', 'PAC_', 'RPA_', 'FSRA_',
      'FRPAC_', '_FSRA', 'FSRPAC_', 'F_', 'FSPAC_', 'RE_', 'FS_',
      'FPAC_', 'SPAC_', 'FAU_', 'RPA_FRPA', '_FSPAC', 'RC_', 'FR_',
      'FSR_', 'SEC_', 'FAC_', 'FRAEC_'], dtype=object)

```

**Figure 13: TCP Flags in CTU-13**

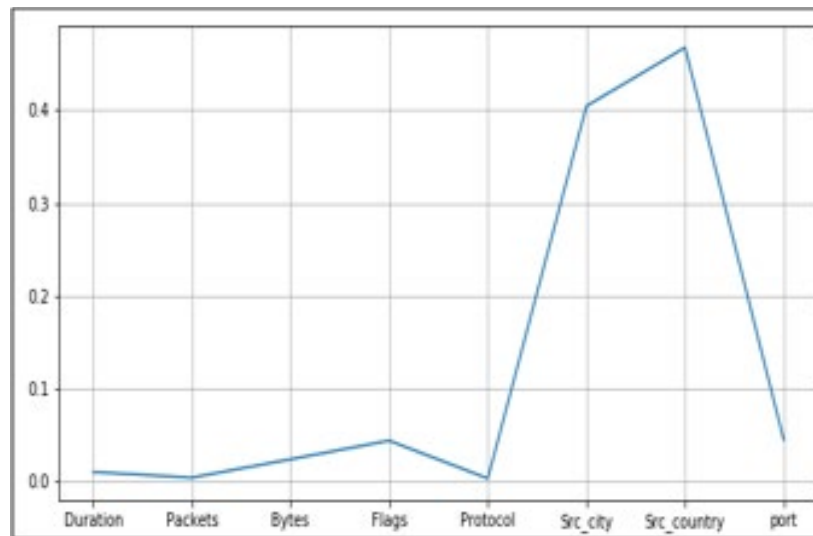
The flow start and IP address variables also proved to be better left without modification. Figure 14 shows the number of packets transmitted at different intervals present in the Flow Start variable. While there are clearly spikes in traffic at certain times, no regular pattern occurs that enables us to better classify botnet traffic over normal, benign traffic. A similarly disappointing conclusion was reached concerning IP addresses. Question 1 already provided location data concerning the city and country from which and to which transmission occurred. Breaking down the individual octaves of each IP address into the network and host components only left us with half of the IP address to consider (Oracle Corporation and/or its affiliates, 2010). Researching how network components are assigned revealed that the amount of geographic information included with these numbers is already reflected in the data generated from GeoIP2 (IANA, 2020). As such, no further information could profitably be extracted.



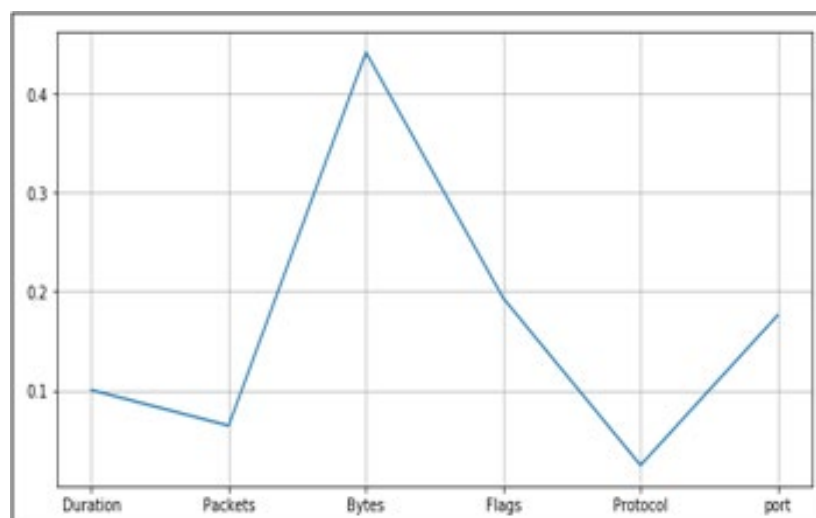
**Figure 14: Flow Start Timeseries Chart**

### Question 3

Question 3 asks which techniques most effectively classifies or aides in classifying botnet traffic. Before detailing our results, one item requires explaining. Each classification variable was applied to both the entire cleaned dataset as well as a reduced version of the dataset without the source city and source country variables. Figures 15 and 16 reveal why this change occurred. Without modification, the dataset heavily skewed towards the source city and source country variables. More about why this occurred will be discussed in question 4, but this skew distracted from the more technical, universal variables that could be used to identify botnet traffic regardless of location. As such, we elected to demonstrate the potential of these classification algorithms both with the location variables and without them.



**Figure 15: Feature Importance of the Original Dataset**



**Figure 16: Feature Importance without Source City and Source Country**

Several different categories of classification techniques presented themselves as suitable candidates to apply to the cleaned CTU-13 dataset. First were instance-based classification algorithms, such as K-Nearest Neighbors (KNN) and Support Vector Machines (SVMs). These types of algorithms form databases of sample data that are contrasted with new data in order to make a prediction (Brownlee, 2019). Ensemble

algorithms, such as Bootstrapped Aggregation (also known as Bagging), Gradient Boosting, and Random Forest, were attempted next. All ensemble algorithms poll an array of independently trained weaker algorithms to determine a final prediction. Also included were clustering algorithms such as K-Means Clustering, Hierarchical Clustering, and Density-Based Spatial Clustering of Applications with Noise (DBSCAN). Clustering algorithms use hierarchical or centroid-based modeling to identify patterns in data. Finally, a Gini Decision Tree and Naïve Bayes rounded out our analysis. Decision trees leverage structures in the data to create dataset-tailored forks for classification and regression. Naïve Bayes assumes strong independence between variables and is often integrated into spam filters.

For each algorithm, the Python library employed needed to be carefully considered. Figure 17 displays the libraries used for each classification algorithm. In most cases, the standard Sci-Kit Learn library was employed with default parameters as alternatives provided no clear advantage. The green boxes below display these standard libraries. The blue boxes display instances where different libraries offered capabilities that justified their inclusion. With SVMs, the NuSVC() library enabled us to use a dataset above 10,000 observations, a limitation when using SVC() (Developers, 2019). Concerning Gradient Boosting, our decision to use XGBoost instead of Sci-Kit Learn's GradientBoostingClassifier() relied primarily on speed. Figure 18 compares the training time and accuracy of XGBoost to other Python libraries. While XGBoost and GradientBoostingClassifier() achieve comparable accuracy scores, XGBoost trains approximately 100 times faster (Morde, 2019).

Classification Method	Algorithm
KNN	<code>sklearn.neighbors.KNeighborsClassifier()</code>
SVM	<code>sklearn.svm.NuSVC()</code>
Bagging	<code>sklearn.cross_validation.Bootstrap()</code>
Gradient Boosting	<code>Xgboost()</code>
Random Forest	<code>sklearn.ensemble.RandomForestClassifier()</code>
K-Means Clustering	<code>sklearn.cluster.kmeans()</code>
Hierarchical Clustering	<code>sklearn.cluster.AgglomerativeClustering()</code>
DBSCAN	<code>sklearn.cluster.DBSCAN()</code>
Gini Decision Tree	<code>sklearn.tree.DecisionTreeClassifier()</code>
Naïve Bayes	<code>sklearn.naive_bayes.GaussianNB()</code>

Figure 17: Employed Python Libraries

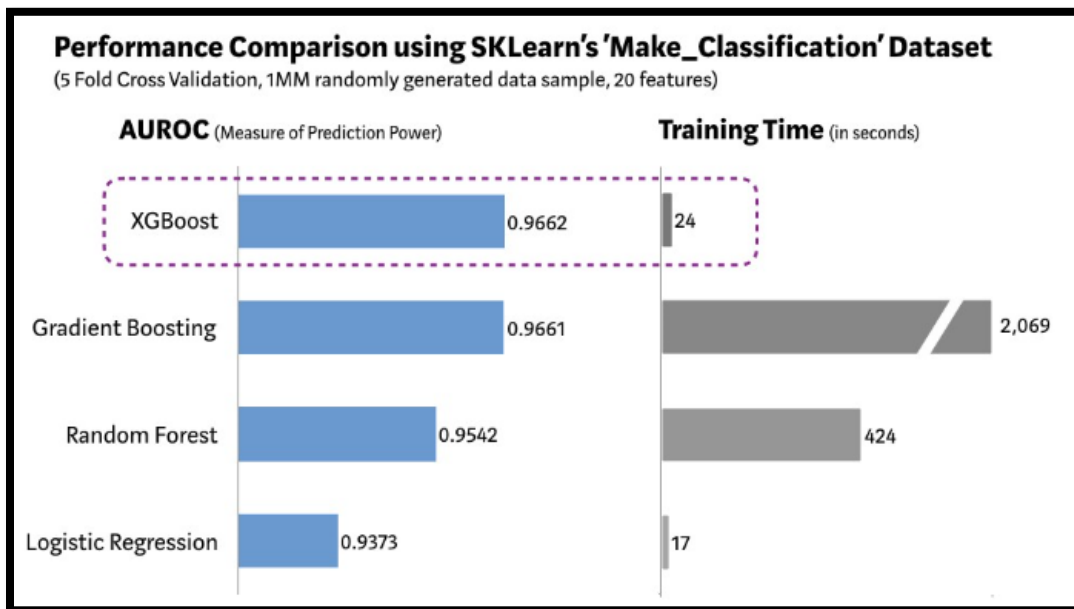


Figure 18: XGBoost Training Time vs Other Algorithms

The results of our investigation reflect in Figure 19. Random Forest proved to be the most effective algorithm at classifying all variables. Two other ensemble algorithms, Gradient Boosting and Bagging, also achieved above 99% accuracy along with the Gini Decision Tree. Complimenting these high marks are the SVM and KNN instance-based algorithms, which both almost achieved a 97% accuracy score. Performing poorly in this category were Naïve Bayes and the clustering algorithms.

A more nuanced picture is painted looking at the accuracy scores when the source city and source country variables were removed. The Gini Decision Tree achieves the highest accuracy score of 92.1% followed closely by Bagging. Gradient Boosting follows with an 86.86% accuracy score trailed by Random Forest, KNN, and SVM. The clustering algorithms and Naïve Bayes again performed poorly on the dataset, although one anomaly with these results occurs when compared to the original results. The algorithms that performed well when all variables were included still performed well when the source city and source county variables were removed. However, these models experience a 7% to 18% drop in their accuracy without the source city and source county variables. Algorithms that performed poorly with both datasets witnessed comparable results.

Algorithm or Method	All Variables		City and Country Removed	
	Accuracy Score	Accuracy Rank	Accuracy Score	Accuracy Rank
Random Forest	100%	1	83.7%	4
Gini Decision Tree	99.9%	2	92.1%	1
Gradient Boosting	99.89%	3	86.86%	3
Bagging	99.2%	4	92.09%	2
K-Nearest Neighbors	96.9%	5	83.2%	5
Support Vector Machine	96.82%	6	78.91%	6
K-Means Clustering	58.35%	7	58.35%	7
DBSCAN	55.79%	8	55.96%	8
Naïve Bayes	50.8%	9	52.1%	9
Hierarchical Clustering	50.00%	10	50.00%	10

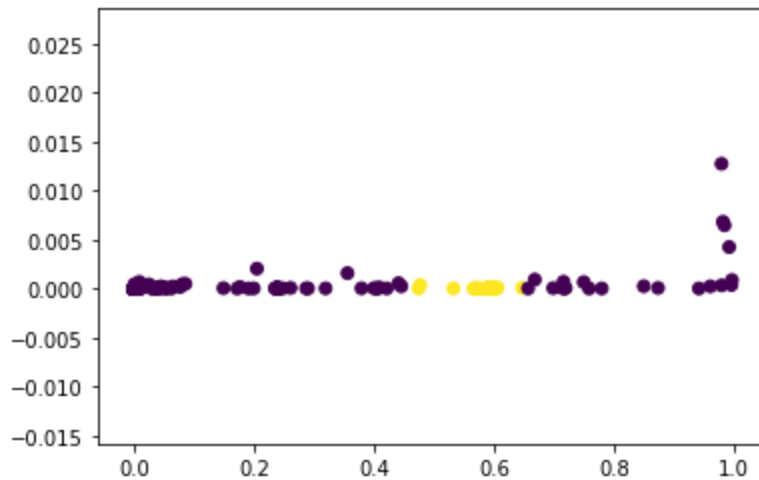
**Figure 19: Accuracy Scores for All Classification Algorithms and Methods**

An explanation for the strength of certain algorithms likely involves how each respective class of algorithm creates predictions. The best performing group on the complete dataset, ensemble algorithms, use a group of low-correlation models voting as a committee to make predictions (Yiu, 2019). This means that any individual error is



discarded by the votes of the larger group. Instance-based algorithms enjoy a similar privilege from a different perspective. Instead of a set of models voting, the group of neighbors informing an individual response value's prediction compensate for any minority of neighbors erroneously informing the group. Later, we will review the results of dominance analysis, showing very strong relative importance concentrated in a small number of variables for both datasets. This scenario favors algorithms that use majority-voting schemes, such as ensemble and instance-based algorithms.

The weakness detected in clustering and Naïve Bayes algorithms probably comes from two distinct considerations with the dataset. In regard to clustering algorithms, Figure 20 shows the distribution of scaled values classified as botnet or not botnet based on our active learning solution in question 5. The yellow botnet observations are in one homogenous band, but the purple non-botnet observations are split into two different parts. Neither condition aides the clustering algorithms in performing accurate predictions (Ullman, Poggio, Harari, Zysman, & Seibert, 2014). Without a focused centroid or a tight group of observations to anchor predictions on, large number of errors occur. Conversely, Naïve Bayes assumes variable independence. Given the premise of this dataset, to identify malware that likely exhibits a set of similar properties, that assumption is effectively null.



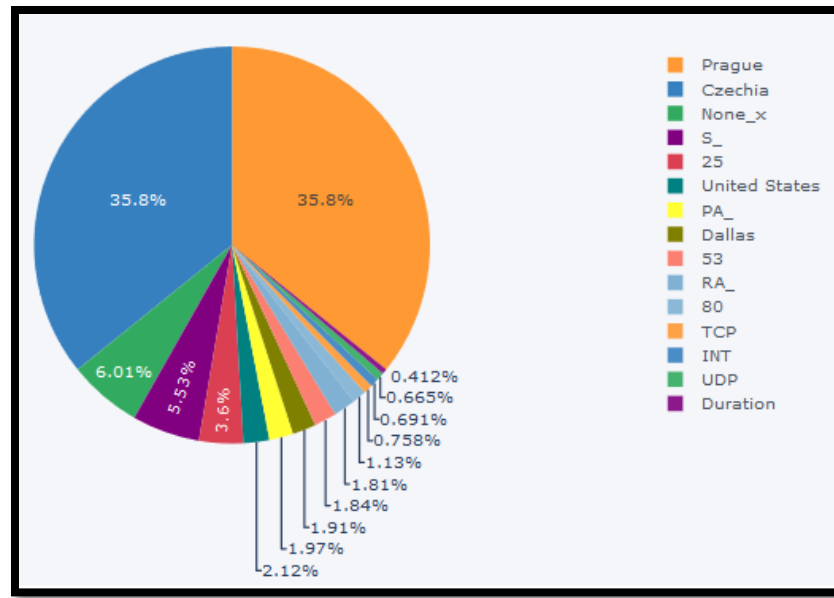
**Figure 20: Classification Based on Duration and Bytes**

On a final note, before moving forward, we wanted to attempt a feature extraction technique to verify our classification results with at least one algorithm. Principal Component Analysis (PCA) performed this function. Through the creation of new principal components from combinations of highly correlated original variables, PCA reduced the dimensionality of both the original as well as the modified dataset. This action helps prevent overfitting, enabling a more balanced view of the dataset (Brems, 2017). Using the Min Max Scaler in order to preserve the distribution of the original dataset, an explained variance of 95% was applied to the PCA() function before being fitted to a KNN classifier. Accuracy scores of 82.8% with the original dataset and 82.2% with the modified dataset were achieved (Hale, 2019). While the original dataset result fell sharply, the modified dataset result remained relatively unchanged. Nevertheless, this result appears to confirm the general effectiveness of our previous results at classifying botnet traffic with a reasonable degree of accuracy.

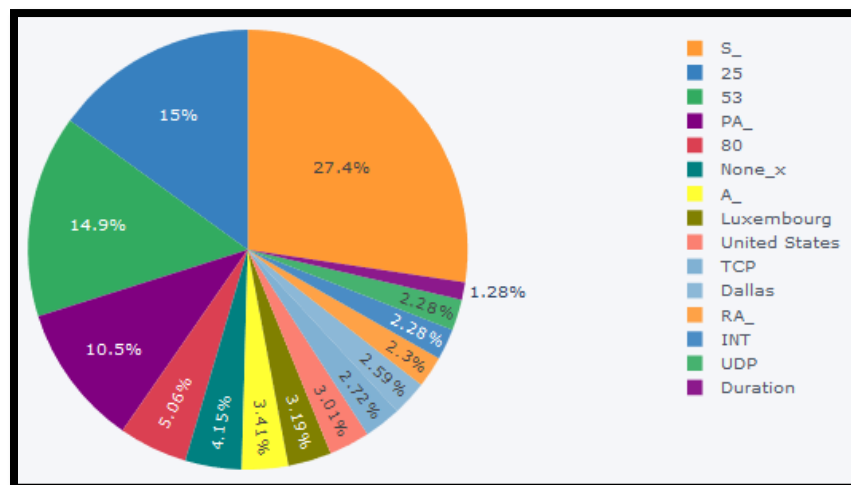
#### Question 4

Further exploring how to reduce the dimensionality of the dataset fell to question 4 and dominance analysis. Question 4 asks which of the variables in the dataset are most important and least important, explicitly directing the use of dominance analysis to find out. Dominance Analysis is well-suited to answer this question as instead of conducting feature extraction, like with PCA, dominance analysis focuses on feature elimination, or the selection of a subset of the original variables for continued use (Brems, 2017). Dominance Analysis accomplishes this goal by identifying the relative importance of each variable through comparing the R-squared contribution each variable provides every subset of the variable set (dominance-analysis, 2020).

Dominance Analysis's results confirm the feature importance conducted at the beginning of question 3. Figure 21 shows how the Prague and Czechia variables account for 71.6% of the relative importance of all variables in the original dataset. This reflects the findings of Figure 15, where the source country and source city variables appeared overweight compared to all other variables. With an R-Squared value of 98.26%, the model in Figure 21 fits the variation seen in the dataset extremely well. Removing just Czechia and Prague results in the pie chart seen in Figure 22. Now, the list of the top 15 most important variables includes a mixed list of ports, flags, locations, and protocols. While still heavily concentrated in the first 3 variables, SYN (S\_), port 25 (SMTP), and port 53 (DNS), these are technical indicators that can be more universally applied to botnet identification anywhere. However, this change comes at the price of a R-Squared value of 41.53%, explaining about 57% less of the variance in the original model than before.

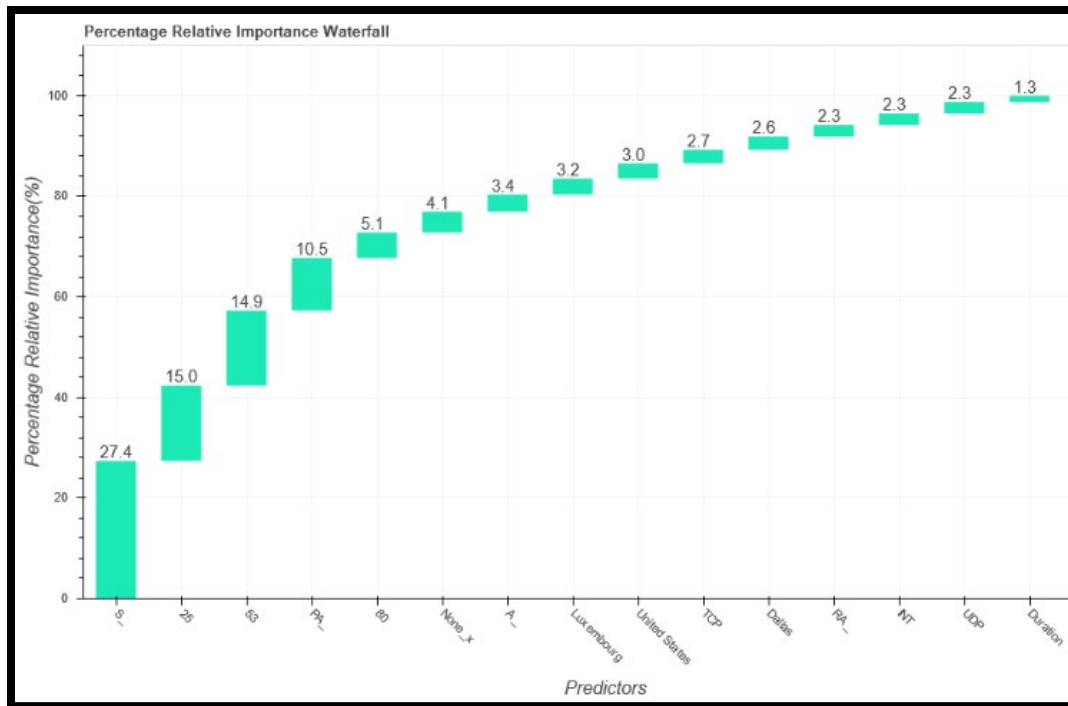


**Figure 21: Dominance Analysis on the Original Dataset**



**Figure 22: Dominance Analysis without the Prague and Czech Republic Variables**

Determining the 5 or 10 least important variables is practically irrelevant. Reviewing Figure 23, one sees that the top 15 variables take approximately 99.99% of the relative importance. Simply excluding all other variables produces results comparable to those seen in question 3.



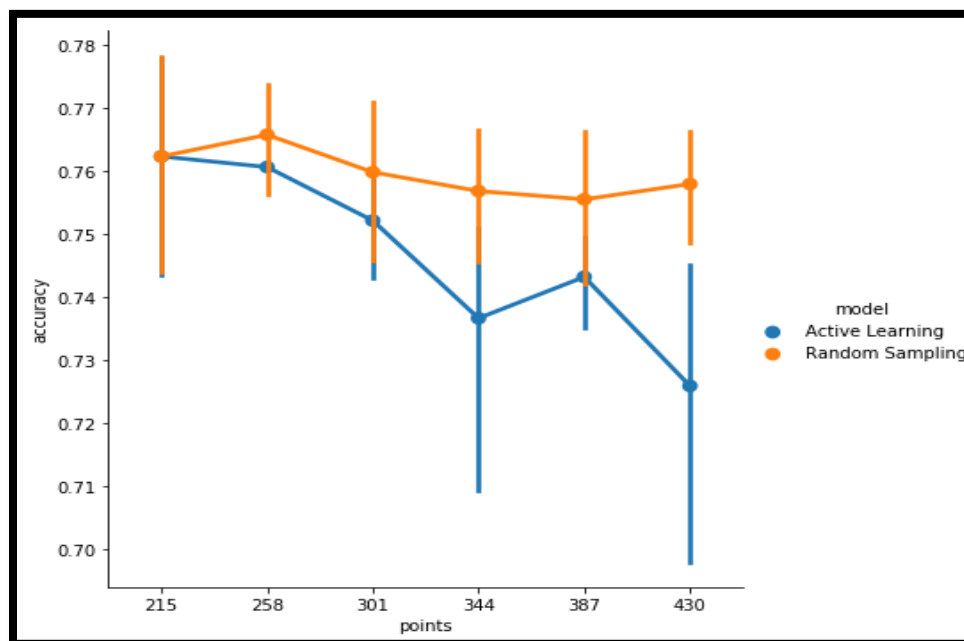
**Figure 23: Percentage Relative Importance Waterfall**

### Question 5

Question 5 tackles the same idea as question 4, but from a different perspective. To reduce computational output and preserve time, question 5 seeks to reduce the size of the dataset through Active Learning. Instead of reducing the dimensionality, as with PCA, Active Learning starts with a low number of observations and iteratively increases the number of observations used for classification. As the number of observations increase, the accuracy should increase, allowing the user to assess what balance of accuracy and observations he or she finds acceptable (Cohen, 2019). Ideally, after a number of iterations, the accuracy achieved with a portion of the observations should be similar to the accuracy achieved with the larger set of observations.

Our experience proved to be an example of this ideal. Using an Active Learning example from Kaggle as a base, we applied this technique to a 2-dimensional portion of

the dataset with the duration of each transmission and the number of bytes as our variables (Mader, 2018). Employing Random Forest, KNN, and SVM as our classifiers, we received Figures 24, 25, and 26 as results. These results started with 0.5% of the down sampled dataset, 215 observations, and increased over 6 iterations to 1% of the down sampled dataset. Accuracy scores were relatively depressed compared to our previous classifications with Random Forest and KNN obtaining accuracy scores between 75% and 80% while SVM achieved substantially better results between 85% and 87%. Figure 27 documents the effect Active Learning had on each algorithm. Active Learning modestly boosted KNN's accuracy by up to 2% with KNN, SVM only very slightly benefited with up to a 0.5% increase, and Random Forest suffered up to a little more than 4% decrease.



**Figure 24: Active Learning Random Forest Results**

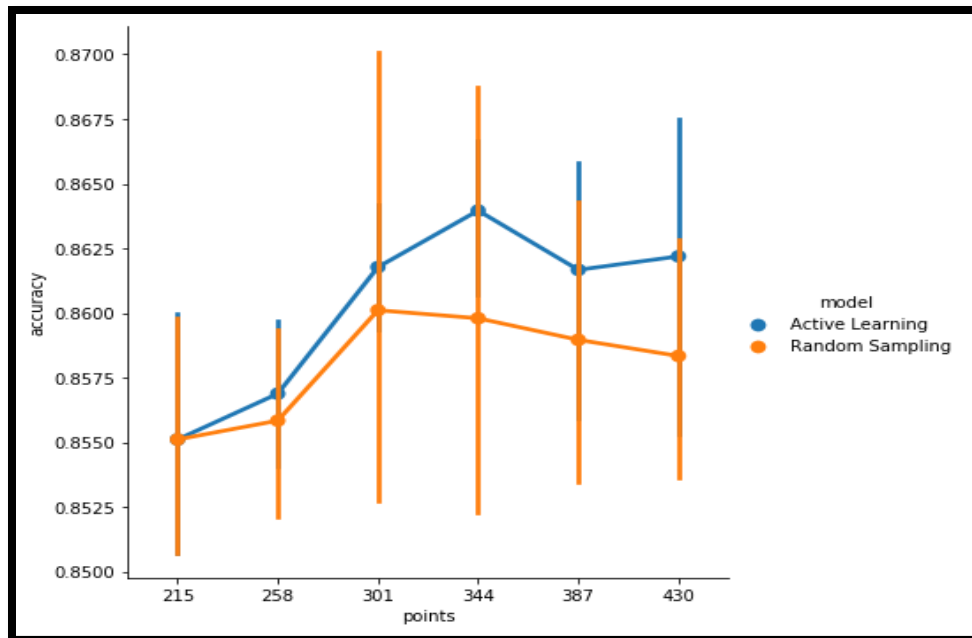


Figure 25: Active Learning SVM Results

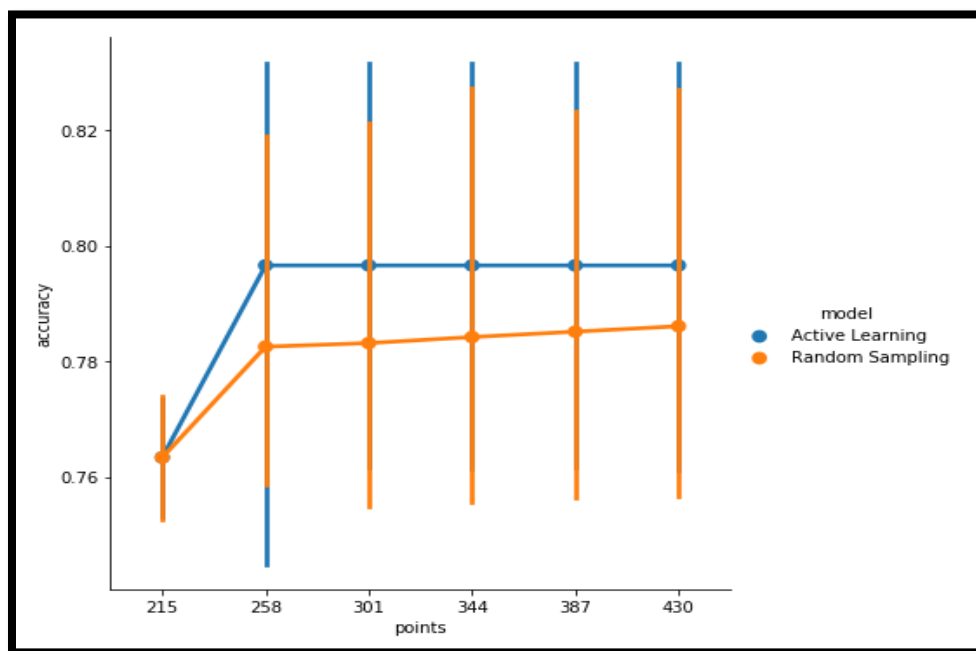
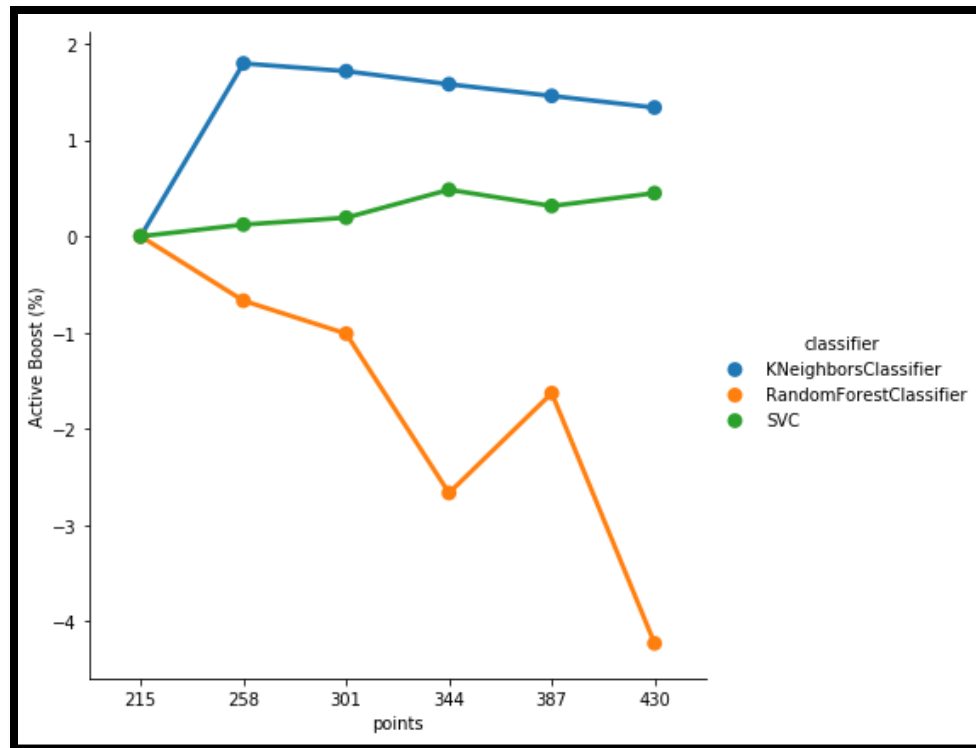


Figure 26: Active Learning KNN Results



**Figure 27: Increase or Decrease in Results Due to Active Learning**

## Bonus

As a bonus, we aimed to apply 3 of our most successful classification algorithms to a distinct NetFlow dataset. In order to keep the same assumptions concerning the variables, we used another portion of the CTU-13 dataset. Instead of being NetFlow based on PCAP from August 15, 2011, we selected NetFlow based on PCAP from August 10, 2011. Figure 28 shows the results, with very similar accuracy scores to those encountered in question 3. We attempted to remove just the source city and leave the source country in place. This returned comparable results to including all variables, confirming our earlier assertion that they contain the same information.



Classification Algorithm	All Variables	With the City Removed
Bagging	96.61%	96.83%
Gini Decision Tree	95.75%	95.68%
Gradient Boosting	92.06%	97.99%

**Figure 28: Bonus Question Accuracy Results**

## Conclusion

This project successfully confirmed that multiple different types of classification, dimension reduction, and dataset size reduction techniques can be applied to NetFlow data. Both with and without location data, individual techniques returned usable accuracy scores. Moreover, the dimension reduction and dataset reduction techniques identified unnecessary components of NetFlow traffic that can be disregarded, saving time and computation resources. While the techniques involved imply batch or mini-batch processing with in-line application, that is still more useful than manual analysis of individual transmissions.

## References

Brems, M. (2017, Apr 17). *A One-Stop Shop for Principal Component Analysis*.

Retrieved from towardsdatascience: <https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c>

Brownlee, J. (2019, December 5). *A Tour of Machine Learning Algorithms*. Retrieved from Machine Learning Mastery: <https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/>

CloudFlare. (2020). *What is a DDoS Attack?* Retrieved from CloudFlare: <https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/>

Cohen, O. (2019, April 19). *Active Learning Tutorial*. Retrieved from towardsdatascience: <https://towardsdatascience.com/active-learning-tutorial-57c3398e34d>

Dalton, W., & Turner, B. (2019, November 14). *Best network monitoring tools of 2020: remote monitoring and management (RMM)*. Retrieved from TechRadar: <https://www.techradar.com/best/best-network-monitoring-tools>

Developers, S.-K. L. (2019). *sklearn.svm.SVC*. Retrieved from Sci-Kit Learn: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html?highlight=impractical>

dominance-analysis. (2020). *Dominance-Analysis : A Python Library for Accurate and Intuitive Relative Importance of Predictors*. Retrieved from Github: <https://dominance-analysis.github.io/dominance-analysis/>

Fruhlinger, J. (2020, March 9). *Top cybersecurity facts, figures and statistics for 2020*.

Retrieved from CSO: <https://www.csoonline.com/article/3153707/top-cybersecurity-facts-figures-and-statistics.html>

Gunter, D. (2020). *A Practical Model for Conducting Cyber Threat Hunting*. Retrieved

from SANS: ♣ <https://www.sans.org/reading-room/whitepapers/threathunting/practical-model-conducting-cyber-threat-hunting-38710>

Hale, J. (2019, March 4). *Scale, Standardize, or Normalize with Scikit-Learn*. Retrieved

from towardsdatascience: <https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02>

IANA. (2020). *Number Resources*. Retrieved from IANA: Internet Assigned Numbers

Authority: <https://www.iana.org/numbers>

Mader, K. S. (2018). *Active Learning: Optimization != Improvement*. Retrieved from

Kaggle: <https://www.kaggle.com/kmader/active-learning-optimization-improvement>

Morde, V. (2019, April 7). *XGBoost Algorithm: Long May She Reign!* Retrieved from

towardsdatascience: ♣ <https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d>

Oracle Corporation and/or its affiliates. (2010). *Parts of the IP Address*. Retrieved from

TCP/IP and Data Communications Administration Guide:  
<https://docs.oracle.com/cd/E19504-01/802-5753/planning3-18471/index.html>

Python Software Foundation. (2020). *geoip2 3.0.0*. Retrieved from Python Software Foundation: <https://pypi.org/project/geoip2/0.1.0/>

Rubicon Communications LLC. (2020). *TCP Flag Definitions*. Retrieved from Netgate Docs: <https://docs.netgate.com/pfsense/en/latest/firewall/tcp-flag-definitions.html>

Stratosphere Lab. (2020, March 14). *The CTU-13 Dataset. A Labeled Dataset with Botnet, Normal and Background traffic*. Retrieved from Stratosphere Lab: <https://www.stratosphereips.org/datasets-ctu13>

Su, Z.-S. (2020, March 25). *Internet Control Message Protocol (ICMP) Parameters*. Retrieved from IANA: <https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml>

Ullman, S., Poggio, T., Harari, D., Zysman, D., & Seibert, D. (2014). *Unsupervised Learning: Clustering*. Retrieved from Center for Brains, Minds, and Machines: <http://www.mit.edu/~9.54/fall14/slides/Class13.pdf>

Yiu, T. (2019, June 12). *Understanding Random Forest*. Retrieved from towardsdatascience: <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>