2023

Applying machine learning to categorize distinct categories of network traffic

Isaac M. Dunham

Applying machine learning to categorize distinct categories of network traffic

Abstract

The recent rapid growth of the field of data science has made available to all fields opportunities to leverage machine learning. Computer network traffic classification has traditionally been performed using static, pre-written rules that are easily made ineffective if changes, legitimate or not, are made to the applications or protocols underlying a particular category of network traffic. This paper explores the problem of network traffic classification and analyzes the viability of having the process performed using a multitude of classical machine learning techniques against significant statistical similarities between classes of network traffic as opposed to traditional static traffic identifiers.

To accomplish this, network data was captured, processed, and evaluated for 10 application labels under the categories of video conferencing, video streaming, video gaming, and web browsing as described later in Table 1. Flow-based statistical features for the dataset were derived from the network captures in accordance with the “Flow Data Feature Creation” section and were analyzed against a nearest centroid, k-nearest neighbors, Gaussian naïve Bayes, support vector machine, decision tree, random forest, and multi-layer perceptron classifier. Tools and techniques broadly available to organizations and enthusiasts were used. Observations were made on working with network data in a machine learning context, strengths and weaknesses of different models on such data, and the overall efficacy of the tested models.

Ultimately, it was found that simple models freely available to anyone can achieve high accuracy, recall, and F1 scores in network traffic classification, with the best-performing model, random forest, having 89% accuracy, a macro average F1 score of .77, and a macro average recall of 76%, with the most common feature of successful classification being related to maximum packet sizes in a network flow.

Degree Type

Open Access Senior Honors Thesis

Department or School

College of Engineering and Technology

First Advisor

Omar Darwish, Ph.D.

Second Advisor

James M. Banfield, Ph.D.

Third Advisor

Sean Che, Ph.D.

Subject Categories

Information Security

For Departmental & Highest Honors:

APPLYING MACHINE LEARNING TO CATEGORIZE DISTINCT CATEGORIES OF NETWORK TRAFFIC

By

Isaac M. Dunham

A Senior Project Submitted to the

Eastern Michigan University

Honors College

In Partial Fulfillment of the Requirements for Graduation

with Departmental Honors in Information Assurance & Cyber Defense

and with Highest Honors

Approved in Ypsilanti, MI on April 20, 2023

Project Advisor:	Omar Darwish, Ph.D.
Departmental Honors Advisor:	James M. Banfield, Ph.D.
School Director:	Sean Che, Ph.D.
Dean of The Honors College:	Ann R. Eisenberg, Ph.D.

Abstract

The recent rapid growth of the field of data science has made available to all fields opportunities to leverage machine learning. Computer network traffic classification has traditionally been performed using static, pre-written rules that are easily made ineffective if changes, legitimate or not, are made to the applications or protocols underlying a particular category of network traffic. This paper explores the problem of network traffic classification and analyzes the viability of having the process performed using a multitude of classical machine learning techniques against significant statistical similarities between classes of network traffic as opposed to traditional static traffic identifiers.

To accomplish this, network data was captured, processed, and evaluated for 10 application labels under the categories of video conferencing, video streaming, video gaming, and web browsing as described later in Table 1. Flow-based statistical features for the dataset were derived from the network captures in accordance with the "Flow Data Feature Creation" section and were analyzed against a nearest centroid, k-nearest neighbors, Gaussian naïve Bayes, support vector machine, decision tree, random forest, and multi-layer perceptron classifier. Tools and techniques broadly available to organizations and enthusiasts were used. Observations were made on working with network data in a machine learning context, strengths and weaknesses of different models on such data, and the overall efficacy of the tested models.

Ultimately, it was found that simple models freely available to anyone can achieve high accuracy, recall, and F1 scores in network traffic classification, with the best-performing model, random forest, having 89% accuracy, a macro average F1 score of .77, and a macro average recall of 76%, with the most common feature of successful classification being related to maximum packet sizes in a network flow.

Acknowledgments

This paper would not have been possible without the help of dedicated and passionate staff at Eastern Michigan University. In alphabetical order, the author thanks Dr. Ann Eisenberg for facilitating the author's Honors College and project success despite his nontraditional background and discipline, Christopher Krieger for providing excellent and rigorous instruction on the subject of network forensics, Dr. James Banfield for critical support and supervision of the project, Dr. Mohammed Alsaleh for providing critical early opportunities in academic and research-oriented projects, and Dr. Omar Darwish for immensely valuable guidance and advising of the project.

Table of Contents

SECTION 1: INTRODUCTION

Background and Motivation

Applications on a computer utilize network resources on a per-port basis and are generally standardized in use by the Internet Assigned Numbers Authority (IANA). Thus, ports are the most typical identifier of an application. Prior to modern next-generation firewalls (NGFWs) that are capable of performing deep packet inspection (DPI) against traffic to identify unique traffic signatures, network traffic classification and filtering were primarily based on network protocol (e.g., TCP, UDP) and corresponding port numbers.

However, port number usage, while standardized, is not mandated and can vary from implementation to implementation of an application. Further compounding issues, it is trivial for network attackers to modify the port numbers used by their tools to bypass port-based filtering schemes. The modern basis for interest in using machine learning and statistical modeling to perform network traffic classification lies in the fact that, like port numbers, there are no perfectly consistent identifiers of application or traffic category in regular communications. Instead, researchers largely seek to use machine learning to identify statistical commonalities in traffic and to perform classification based on these commonalities.

The motivation behind this paper is to explore and evaluate how different machine learning models may be applied in network traffic classification using easily accessible tools and products. Where most existing literature primarily focuses on existing datasets, simple and well-known applications, or niche aspects of the network traffic classification problem, this paper demonstrates the creation of a simple, realistic, recent dataset purpose-built to contain a variety of common modern applications in the most common categories of use. It does not seek to solve the decades-old problem actively being worked on by the best and brightest in industry and

research, but instead to illustrate and evaluate some of the existing capabilities and difficulties with modern, readily available solutions.

Contributions

This paper contributes not by pushing the boundary of what is possible with machine learning, but by evaluating and elaborating on what is already commonly available to network administrators today. In a bulleted list, some of the contributions of this paper are that this paper:

- Provides a replicable workflow to use machine learning on network traffic, including the processes of capturing network traffic, identifying traffic of interest, labeling traffic, extracting useful statistical features from traffic, and training models to correctly classify traffic
- Provides a performance summary of common machine learning classifiers in the context of a network dataset with a variety of common, modern application labels
- Identifies important features in network traffic classification for use in feature selection or further feature creation
- Reveals insight into strengths, weaknesses, and nuances of different classifiers and how they interact with many applications varying in nature and degree of presence

Literature Review

Despite its lack of regular use in enterprise networks, the concept of machine learning is not new to network traffic classification. In 2005, Andrew Moore and Denis Zhuev wrote what is still one of the most cited papers on the subject. Their research used Bayesian analysis techniques to achieve up to a 95% accuracy rate using traffic featuring 248 per-flow discriminators, including TCP port numbers. The traffic was hand-labeled by broad categories as opposed to individual applications (e.g., SMTP, IMAP, and POP traffic were labeled “MAIL”)

(2005). That same year, 2005, researchers were able to achieve an average of 86.5% accuracy in traffic classification using just six features: packet inter-arrival time, packet length, mean, and variance, flow size, and duration, notably excluding port numbers of any sort while also using Bayesian techniques (Zander et al).

More recent research improves on the performance of past research using other traditional, classical learning models and also explores completely novel methods. In 2013, researchers used a "Bag-of-Words" (BoW) model with latent semantic analysis (LSA) to create similar traffic clusters capable of accurately identifying applications previously unknown to a network (Zhang et al). In 2015, Hajjar et al. achieved relatively high recall scores using message size analysis of the first few messages in a given network conversation with a combination of K-means clustering and Dempster's EM algorithm. In 2017, researchers evaluated different models and algorithms including J47, random forest, k-NN, and Bayesian networks against two datasets and achieved the greatest accuracy (90+%) using random forest and k-NN, depending on the dataset and feature sets used (Yamansavascilar et al). In 2019, researchers compared the accuracy of multilayer perceptron, C4.5, SVM, and naïve Bayes against specific encrypted network applications, achieving the best results (88.29% accuracy) using C4.5 (Al-Obaidy et al).

The non-academic community, too, has a number of projects adjacent to or directly involved in using machine learning to categorize network traffic. NFStream, which will be discussed later, is a Python "machine learning oriented" package that seeks to "make machine learning approaches for network traffic management reproducible and deployable" ("NFStream: Flexible Network Data Analysis", n.d.). CANIV TECH is a network analytics-as-a-service organization that seeks to help predict network faults, mitigate DDoS attacks, and classify network traffic using machine learning (CANIV TECH, n.d.). Adjacent to network traffic

classification, computer network equipment manufacturers also commonly tout their use of machine learning for network security purposes, such as Palo Alto Networks WildFire inline ML antivirus offering (Palo Alto Networks, n.d.).

SECTION 2: METHODOLOGY

In data science and machine learning, there is a broad process that must be performed to effectively create a machine learning model that can accurately classify data. That process, as will be laid out below, includes data collection, data preparation, modeling, and model evaluation. A flowchart showing the broad process of this paper from start to finish is included below.

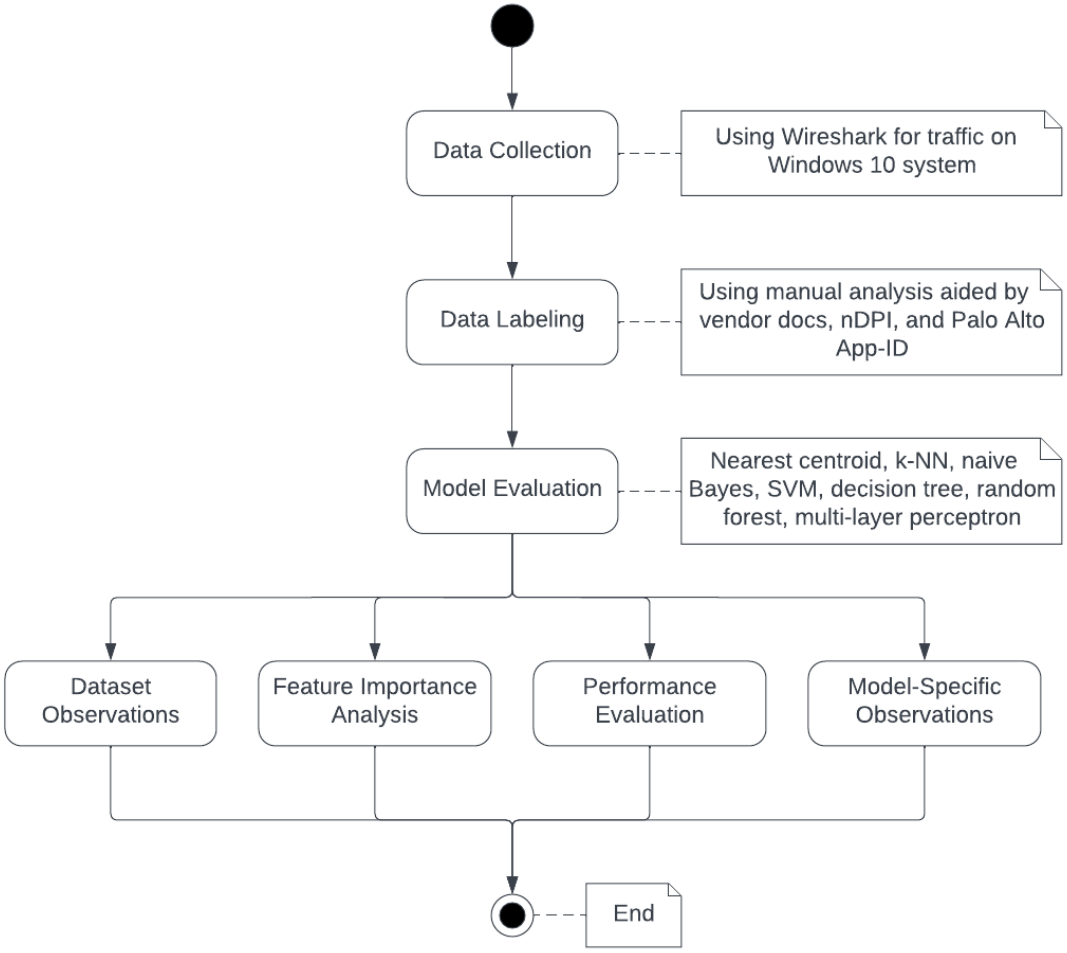

Figure 1: Process overview

Data Collection

Broadly speaking, most computer data that travels across a network is encapsulated into datagrams. At different stages of travel across a network, these datagrams exhibit different

properties and take on different names. Most commonly, people think of network data in terms of “packets,” representing network data that includes application, transport, and network layer properties in the traditional TCP/IP stack. Because of the quantity and variety of applications and protocols that underlie typical network traffic, individual recorded packets only contain a limited number of consistent features. This most notably includes time, length, source and destination IP address (IPv4 or IPv6), source and destination port, protocol, and flags. Additionally, individual packets associated with many similar applications are often difficult to distinguish from each other. As an example, fundamentally, one might expect a packet related to YouTube video and another packet related to Vimeo video streaming to be nearly indiscernible, aside from specific, non-static identifier fields like IP address.

Therefore, much of the existing literature on network traffic classification does not rely on individual packets as the raw units of data, but, instead, *flows*. Flows are aggregations of packets based on a tuple consisting of five properties: source and destination IP address, source and destination port, and protocol. Given a flow, many different properties can be extrapolated, such as total packets sent and received, flow duration, average packet interarrival times, etc. In this paper, data is originally collected as packets in PCAP (packet capture) format, and traffic classification is done on the basis of 5-tuple flows with such features and statistical properties. Discussion on how these properties are derived is included in the Data Preparation section.

All network data originated from a Windows 10 system. Network traffic was captured using Wireshark and saved as .pcap files. In the data collection process, first, Wireshark packet captures would be started. Then, the application in question would be run for 10 minutes before being turned off. Finally, the Wireshark capture would be stopped and saved. Accompanying the packet capture was the collection of NetFlow data. This NetFlow data was forwarded from an

intermediary Palo Alto Networks PA-450 firewall able to identify applications on a per-flow basis to an Ubuntu server hosting Elastic Elasticsearch, Logstash, and Kibana. This NetFlow data was not used as the basis of any traffic classification but is used later as a tool aiding the data labeling process in conjunction with nDPI-identified applications and manual analysis based on IP addresses, protocols, and port numbers.

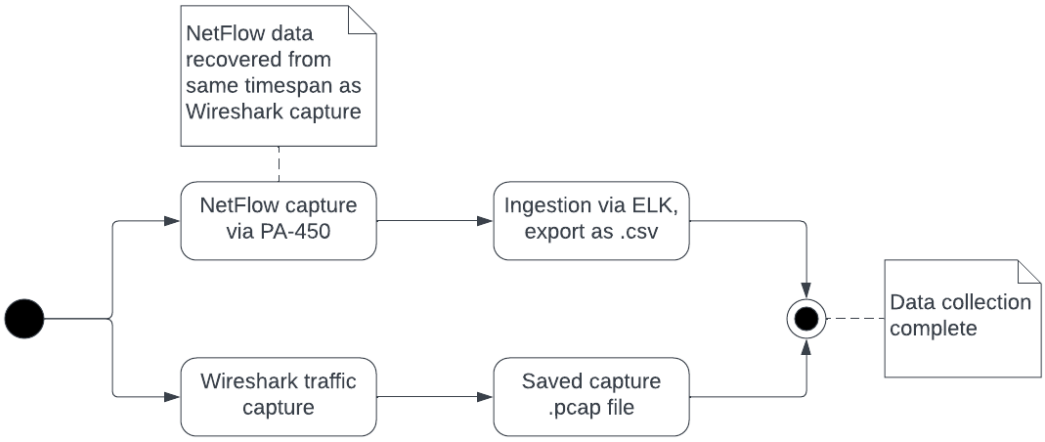

Figure 2: Data collection process

Data Preparation

Traffic Labeling

Unfortunately, labeling network traffic is not as simple as classifying all data from a network capture for a specific application as belonging to that application. Most modern major operating systems, such as Windows, operating on modern networks inherently have a lot of background network noise. Whether sending heartbeats and telemetry information to a service provider or checking for update information, almost any network capture will include traffic unrelated to the primary application being run on the system at the time of capture. Therefore, extraneous traffic must be identified and removed from each capture file so that the rest of the traffic can be accurately labeled per its associated application.

Isolating a particular application's network traffic has varying degrees of difficulty. In using a pre-configured, well-known, simple, and consistent application, like SSH, one could simply remove all traffic not destined for TCP port 22 (or to some other unshared port as configured by the SSH server). In other situations, however, such as in trying to distinguish YouTube traffic from other background traffic hosted on the web via HTTPS, as is done in this paper, identifying undesired traffic is more difficult. To accomplish this, a mixture of methods was used, including nDPI-based application identification, Palo Alto App-ID identification, and manual analysis based on IP addresses, protocols, and port numbers.

nDPI-based application identification was performed using *NFStream*, described later in this paper, to identify application usage on a per-flow basis. nDPI is a popular open source LGPLv3 library for deep packet inspection created by ntop that is a common basis for application identification in products and research (Aouini, Z. & Pekar, A, 2022). Palo Alto App-ID information is found in the NetFlow captures when the "PAN-OS Field Types" option is configured from the firewall. App-ID information served the same function as the information derived using nDPI, but the App-ID library of applications is both very different and much larger. Both Palo Alto App-ID and nDPI application identification are performed by analyzing packet contents for signatures associated with particular applications. For example, at the time of writing, the halflife2_and_mods.c Half-Life 2 identifier in nDPI's GitHub repository checks packets for the text "halflife2 client req detected, waiting for server reply." These application identification tools were used as useful indicators in trying to determine which flows in a capture corresponded with the application in question.

Manual application analysis was done primarily on the bases of IP addresses and port numbers. Where possible, vendor documentation pertaining to the specifics of which IP

addresses and port ranges were used by an application was used as a source of truth. Zoom's documentation, for example, provided a list of over 300 Zoom-specific IP address ranges which served to identify which packets were sent to and from Zoom. In general, for each PCAP capture file, the following process was performed to create a display filter and to save a final representative file to be used in later data processing:

1. Identify port numbers and/or IP addresses as appropriate associated with the application in question.
2. In the capture file, filter out traffic not using these ports and/or IP addresses.
3. Analyze the remaining traffic to determine if it seems to be associated with the application in question not using the ports discovered above. Record any newly discovered indicators of relevance.
4. Repeat the above process until all traffic is generally categorized into desired and undesired traffic.
5. Filter out all undesired traffic and save a new capture file with an appropriate name.
6. Visually confirm that the saved file is appropriately filtered.

Table 1, shown below, describes the applications and labels captured for this paper.

Label	Category	Description
csgo	Video gaming	Video game Counter-Strike: Global Offensive
discord	Video conferencing	Discord is a popular audio/video calling platform
hl2dm	Video gaming	Video game Half-Life 2: Deathmatch
minecraft	Video gaming	Video game Minecraft
netflix	Video streaming	Netflix streaming over website
skype	Video conferencing	Skype is a popular audio/video calling platform
vimeo	Video streaming	Vimeo is a popular video streaming platform
webbrowsing	Web browsing	Miscellaneous web browsing on Amazon.com, Wikipedia, and Gmail
youtube	Video streaming	YouTube is a popular video streaming platform
zoom	Video conferencing	Zoom is a popular audio/video calling platform.

Table 1: Dataset labels and categories

Flow Data Feature Creation

Following the process up to this point, the data exists in a PCAP file format without useful features. To aggregate the data into per-flow records and to extract statistical flow information, NFStream was utilized. At the time of writing, NFStream's GitHub page describes itself as "a multiplatform Python framework providing fast, flexible, and expressive data structures designed to make working with online or offline network data easy and intuitive." NFStream provides a sound, consistent manner in which to derive flow information, including underlying application using the nDPI library, and has been used in a number of scholarly, peer-reviewed papers. To recover useful statistical flow properties, the "statistical_analysis = True" argument was provided to the NFStreamer NFStream function. Resultant flows were then saved into a CSV format.

Initial Feature Selection, Transformation, and Scaling

Different machine learning models benefit in different ways from the presence and absence of different features. However, it is always best practice to remove known misleading and effectively useless features. A table of features initially removed from the dataset is included below.

Feature Removed	Explanation
Source and destination IP address	Long-term, these are unreliable indicators. Source addresses change based on the location and ISP of the user, and destination addresses often change when an application provider moves servers providing services.
Port numbers	Port numbers are a biased and historically notably vulnerable means of identifying applications. Using ports could be used to achieve extremely high accuracy, but would render the model ineffective at identifying applications using deliberately irregular ports.
Layer 2 data (MAC addresses, VLAN ID, etc.)	Layer 2 network information is only relevant at the local network level and is unreliable as a general indicator for app identification.

All features with no variation in value	Features that all had the same value (e.g., Echo of Congestion Encountered (ECE) packets in this dataset) were removed because they provide no information and add to the dimensionality problem.
Flow start / end time	Start/end times are dependent upon when a conversation occurs and are unreliable indicators of an application in the context of this paper.
Source to destination and destination to source SYN packets	These features were redundant because they were always identical in the dataset; whenever TCP was the protocol, they would share a value of 1, else, they would share a value of 0. The bidirectional syn packet feature was kept to preserve this feature.

Table 2: Features removed from NFStream defaults

After useless features are removed, it is important to identify categorical and/or ordinal data types remaining in the dataset. Aside from the class labels, the only categorical variable is "protocol." Its values are integers dependent upon the IANA-defined protocol number. This paper used one-hot encoding to transform its values into separate binary TCP, UDP, and ICMP features.

Next, the data underwent scaling. Scaling is particularly useful for machine learning models whose algorithms are dependent upon determining distances between a data point and data points associated with a particular class, such as nearest centroid or k-nearest neighbors (k-NN). To perform data scaling, standardization was performed on appropriate features in accordance with $x = (x - \text{mean}(x)) / \sigma$.

Dataset Balancing

A common issue in machine learning exercises is an imbalanced dataset. An imbalanced dataset is a dataset that has many samples of a particular class, but not many of another. Models trained on imbalanced datasets may believe that some classes are inherently uncommon (whether or not that is necessarily true) and will therefore often ignore strong indicators of their presence due to the perception that the likelihood that a class will be present at all is very low. If the real-world parent distribution is similar to that of the training dataset, this problem can be difficult to notice in performance. Consider, for example, that to achieve 99.98% accuracy in classifying a

dataset of 4,999 3-leaf clovers and one 4-leaf clover, a model simply needs to classify every single clover as a 3-leaf clover.

Network traffic is almost inherently imbalanced. Different applications make use of a different number of servers, services, and communication flows. The most imbalanced ratio in this paper's dataset is between the CS:GO-labeled traffic (maximum sample count) and the Netflix-labeled traffic (minimum sample count), with the former flow count making up only 4.5% (a ratio of roughly 1:22) of the latter flow count. There exists a great deal of literature covering potential solutions and proposed principles for handling the problem of imbalanced datasets. However, as Dr. Gary Weiss of Fordham University writes, "…ultimately what we care about is how the imbalance impacts learning, and, in particular, the ability to learn the rare classes (2013)." Different methods of undersampling overrepresented classes (e.g., CS:GO & Minecraft) and oversampling underrepresented classes (e.g., Netflix & YouTube) were explored, but neither was found to be of great benefit in improving the models. Therefore, to maintain the benefits of having an accurate, realistic, unbiased dataset and to reduce complexity, the dataset balance was left unchanged. However, to ensure that the training process did not forego certain classes because of their chance non-appearance in training sets, stratification (ensuring similar representation ratios of certain labels) was employed in k-fold cross-validation. A pie chart showing the relative proportions of traffic types in the dataset is shown in Figure 3 below.

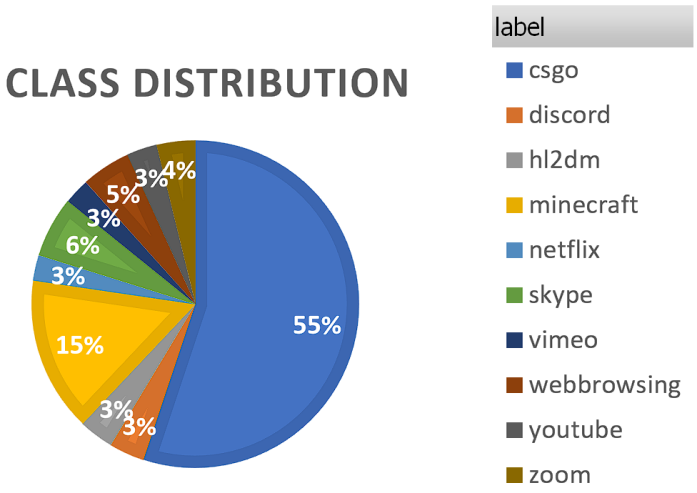

Figure 3: Dataset label distribution

Common Modeling and Evaluation Processes

Machine learning model evaluation was performed using the Scikit-Learn library for Python. For each model, the dataset is shuffled and stratified k-fold validation is used to evaluate the model against different subsets of the data for an aggregate score. As applicable, SHAP values, confusion matrices, permutation importance reports, and correlations are used to describe the performance of the models and the effect of the data on the model in question. Confusion matrix discussions will sometimes reference what this paper calls “confusers,” classes or class categories (e.g., video streaming) that are most misclassified over the true label. Confusion matrices showing results by class category reflect previously shown classification results binned into their respective categories, not results of classification using broad categories as labels. Permutation importance is measured using the average permutation importance value of features during the stratified k-fold validation process using 30 repeats in the sklearn “permutation_importance” function. SHAP values are measured as described in each appearance.

Evaluation of the classifiers includes scoring based on overall accuracy, average recall, precision, and F1-score, and average weighted precision, recall, and F1-score. F1-score is a metric that accounts for both precision and recall to strike a broad balance between the two. To

optimize models to achieve as best scores possible, grid searching using the Python Scikit-Learn library for parameters of interest is performed using accuracy as the performance metric. Any model tests that use details deviating from what has been described thus far will have these exceptions noted in their discussion.

SECTION 3: MODEL EVALUATION

Nearest Centroid

The nearest centroid classifier works by separating data points into clusters representing the classes that they are labeled with. The average point of these clusters is called a "centroid," and given a test piece of data against a trained nearest centroid classifier, the test point would be classified based on what the *nearest centroid* to that point is using Euclidean distance.

The ultimate classification results of the nearest centroid classifier are shown below.

	Precision	Recall	F1-Score	Support
csgo	1.00	0.74	0.85	350
discord	0.52	0.64	0.57	22
hl2dm	0.22	0.18	0.20	22
minecraft	0.71	0.91	0.80	97
netflix	0.37	0.69	0.48	16
skype	0.25	0.16	0.19	38
vimeo	0.27	0.50	0.35	16
webbrowsing	0.39	0.48	0.43	31
youtube	0.12	0.32	0.17	19
zoom	0.38	0.54	0.45	24
Accuracy	0.67			635
Macro Average	0.42	0.52	0.45	635
Weighted Average	0.75	0.67	0.69	635

Table 3: Nearest centroid classification results

Confusion matrices representing the total classifications and recall per label are respectively shown in Table 4 and Table 5.

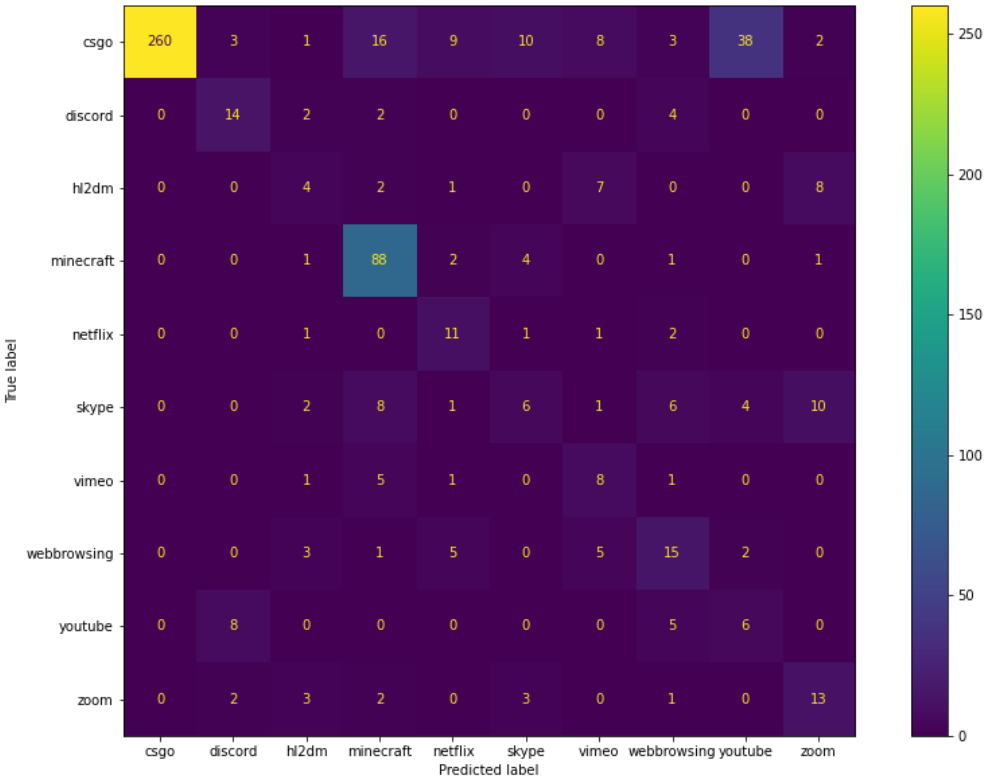

Table 4: Nearest centroid confusion matrix

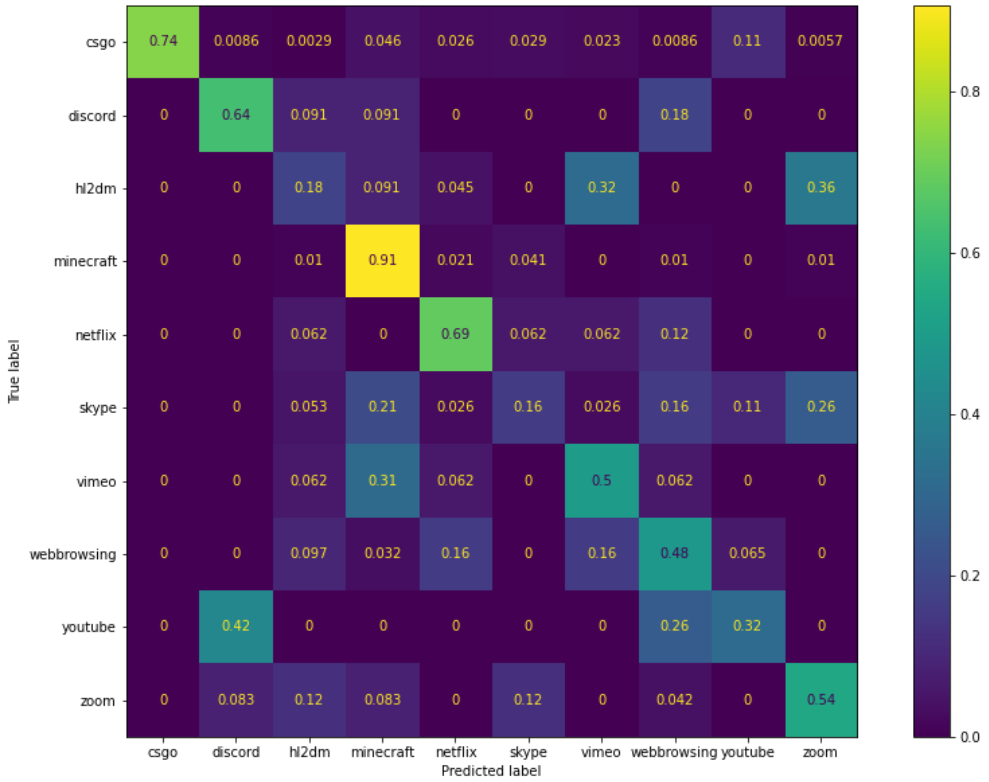

Table 5: Nearest centroid normalized confusion matrix

For misclassification, one would expect that similar applications would be misclassified as each other. However, this doesn't seem to be true in the results for this classifier. Table 6, shown below, lists each label by its strongest confuser by true label proportion. Below that, Table 7 is a confusion matrix representing classification summarized by broad category.

Class	Class Category	Strongest Confuser	Strongest Confuser Category
csgo	Video gaming	youtube	Video streaming
discord	Video conferencing	webbrowsing	Web browsing
hl2dm	Video gaming	zoom	Video conferencing
minecraft	Video gaming	skype	Video conferencing
netflix	Video streaming	webbrowsing	Web browsing
skype	Video conferencing	zoom	Video conferencing
vimeo	Video streaming	minecraft	Video gaming
web browsing	Web browsing	vimeo/netflix	Video streaming
youtube	Video streaming	discord	Video conferencing
zoom	Video conferencing	skype/hl2dm	Video conferencing/gaming

Table 6: Nearest centroid label & confuser comparison

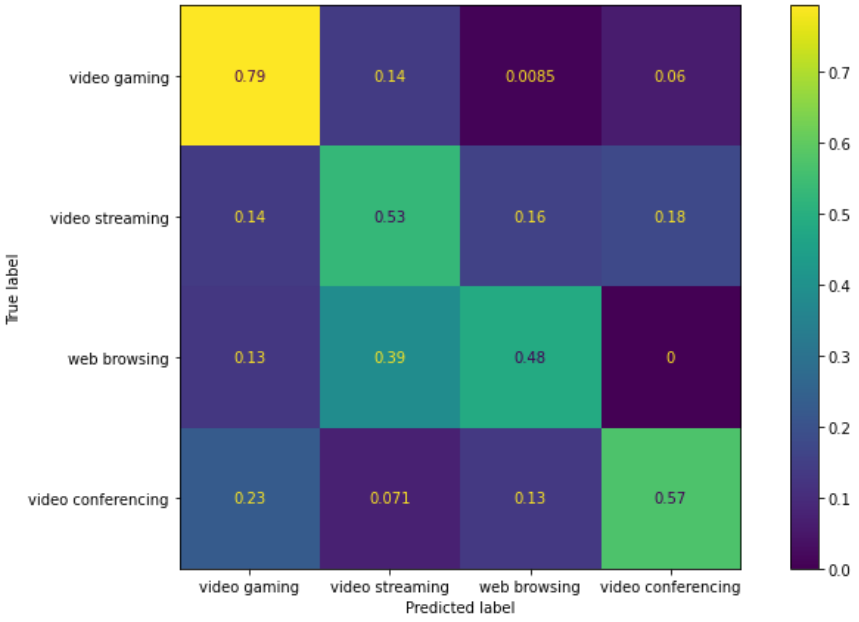

Table 7: Nearest centroid broad category confusion matrix

In total, if applications in the same broad category are to be considered "similar" to each other, applications' most frequent classifications were only in the same category for two of the

ten classes, with one of those two having a tie with another category. There are no strong patterns for confusion between categories. Video conferencing was an overrepresented confuser category, whereas video gaming was an underrepresented confuser category, despite CS:GO traffic having the largest number of samples. To better understand this confusion, it helps to explore feature importance.

The ten most important features to this model as derived from permutation importance are shown in Table 8 below.

Feature	Weight ± Std. Dev
dst2src_rst_packets	0.010394 ± 0.003906
dst2src_stddev_ps	0.007192 ± 0.006178
src2dst_max_ps	0.007139 ± 0.005679
bidirectional_max_ps	0.005617 ± 0.007453
src2dst_max_piat_ms	0.005459 ± 0.005531
bidirectional_rst_packets	0.005459 ± 0.003471
src2dst_duration_ms	0.005249 ± 0.005148
bidirectional_duration_ms	0.004987 ± 0.004877
dst2src_max_piat_ms	0.004934 ± 0.005384
bidirectional_max_piat_ms	0.004934 ± 0.005303

Table 8: Nearest centroid feature importance by permutation importance

The strongest features were related to packet size, packet inter-arrival times, and reset packet counts. Largely, these features were uncorrelated (r-correlation values lower than 0.8), aside from the dst2src_max_piat_ms, src2dst_max_piat_ms, and bidirectional_max_piat_ms group (sharing an r-correlation > .94 between each other).

The strongest features largely make intuitive sense to be correlated on a per-application basis. Packet sizes are often well-correlated with specific applications as they often require different amounts of data. For example, file downloading might have large packet sizes to increase absolute throughput. Video conferencing applications, on the other hand, need to send data as immediately as possible, and, in the case of audio-only conferences, absolute data sizes are often small.

Packet inter-arrival times can be closely correlated with application usage because distinct applications typically have a pattern of sending a number of packets per second. As an example, a high-definition video conferencing UDP-based application, like Zoom, which has the 3rd lowest mean packet inter-arrival time, might be expected to send a lot of packets quickly and thus have an average low packet inter-arrival time.

Reset packet counts make less obvious, intuitive sense to vary on a per-application basis. While some applications may be designed to forcefully terminate connections using reset packets, this is very rare and does not seem to be the case for the applications used in this paper.

Table 9 conveys the average packet size, packet interarrival time, and reset packet counts and is included below, though differences in per-feature variance should be considered. Based on this table alone, some of the model's performance is explainable. CS:GO-classified was most often confused with YouTube traffic. It so happens that their average values for src2dst_max_ps, one of the most important features according to weight, are very similar to each other. However, because the nearest centroid model performed so poorly and there are so many potential causes of this performance, whether they be due to the nature of the classifier or due to statistical similarities across the dozens of different features, such analysis is not necessarily consistent or reliable for this model.

Label	Mean src2dstmax_ps ± Std. Dev	Mean src2dst max_piat_ms ± Std. Dev	Mean dst2src_rst_packets ± Std. Dev
csgo	1228 ± 315	7873 ± 14040	0 ± 0
discord	995 ± 431	7658 ± 15077	0.0455 ± 0.213
hl2dm	235 ± 177	16465 ± 17792	0 ± 0
minecraft	370 ± 187	5271 ± 8874	0.0619 ± 0.242
netflix	947 ± 175	42016 ± 7168	0.0625 ± 0.250
skype	687 ± 471	20731 ± 26197	0.316 ± 0.471
vimeo	1051 ± 235	26698 ± 21507	0 ± 0
webbrowsing	800 ± 335	40638 ± 11674	0 ± 0
youtube	1126 ± 292	23823 ± 15264	0.158 ± 0.375
zoom	439 ± 484	6659 ± 14373	0.333 ± 0.702

Table 9: Feature means and standard deviations

K-Nearest Neighbors

The k-nearest neighbors classifier (k-NN) works similarly to the nearest centroid classifier. Instead of computing per-class centroids, the k-NN model compares a new sample to the class labels assigned to a pre-defined number (k) of the nearest training samples, "neighbors." For example, if using k-NN where k = 3 and a test record were found to be closest to two data points assigned to class A and one assigned to class B, the test point would be classified as belonging to class A. In essence, k-NN features no "training" whatsoever. Instead, new values are simply compared to existing, labeled values from the "training" set.

The ultimate classification results of the optimized k-NN classifier are show in Table 10.

	Precision	Recall	F1-Score	Support
csgo	0.97	0.97	0.97	350
discord	0.65	0.59	0.62	22
hl2dm	0.57	0.55	0.56	22
minecraft	0.95	0.89	0.91	97
netflix	0.62	0.62	0.62	16
skype	0.58	0.68	0.63	38
vimeo	0.67	0.75	0.71	16
webbrowsing	0.77	0.77	0.77	31
youtube	0.68	0.79	0.73	19
zoom	0.72	0.54	0.62	24
Accuracy	0.87			635
Macro Average	0.72	0.72	0.71	635

Weighted Average	0.87	0.87	0.87	635

Table 10: k-NN classification results

To get the best performance from k-NN, the most important hyperparameter to optimize is k. Different data distributions and feature sets will result in different values of k being most effective. Ultimately, the k value providing the highest accuracy results was found to be 1.

The primary cause of the low k value is the distribution of classes in the dataset. Indeed, though shuffled and stratified random train-test splits were employed, ultimately, there were not always more than a few samples for certain classes. An example of this issue in the dataset can be observed in a sample 80% training split created during cross-validation. In this training split, there existed only 13 total Vimeo samples. Thus, increasing the number of nearest neighbors from one to any higher amount easily results in misclassification. A graph comparing the accuracy with the number of neighbors used for the classifier fitted on a 5:1 train-test split is shown in Figure 4.

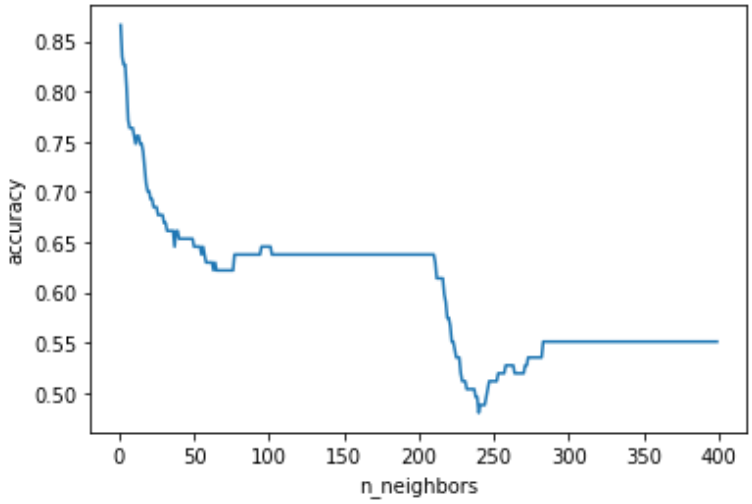

Figure 4: Accuracy rate vs number of neighbors

Despite the shortcomings, the results are much better than those of the nearest centroid model. Confusion matrices representing the total classifications and recall per label are respectively shown in Table 11 and Table 12.

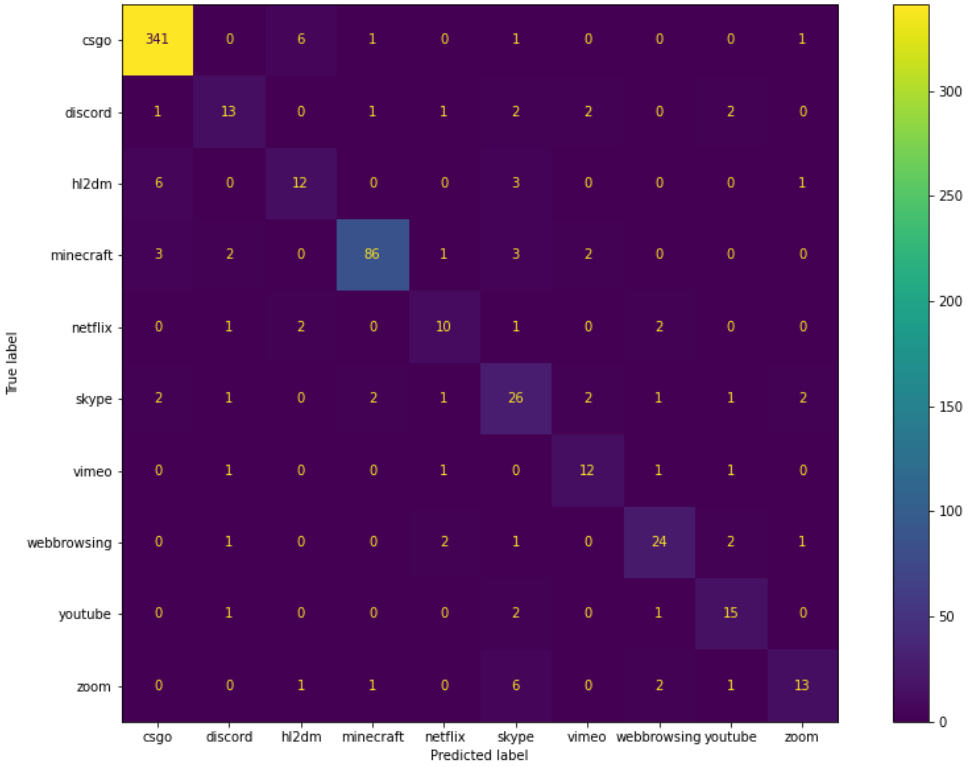

Table 11: k-NN confusion matrix

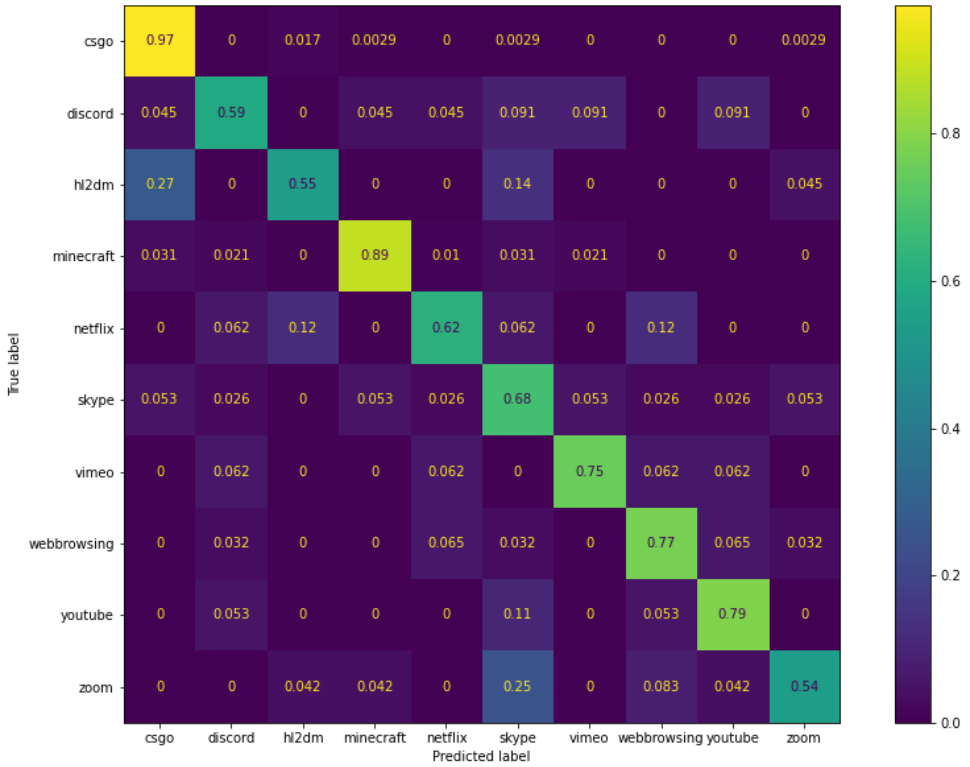

Table 12: k-NN normalized confusion matrix

A table comparing the k-NN classifier's common confusers and a confusion matrix showing recall results by broad category are shown below in Table 13 and Table 14.

Class	Class Category	Strongest Confuser	Strongest Confuser Category
csgo	Video gaming	hl2dm	Video gaming
discord	Video conferencing	skype / vimeo / youtube	Video conferencing / video streaming
hl2dm	Video gaming	csgo	Video gaming
minecraft	Video gaming	csgo / skype	Video gaming / Video conferencing
netflix	Video streaming	hl2dm / webbrowsing	Video gaming / web browsing
skype	Video conferencing	csgo / minecraft / vimeo / zoom	Video gaming, video streaming, video conferencing
vimeo	Video streaming	discord / netflix / webbrowsing / youtube	Video conferencing, video streaming, web browsing
web browsing	Web browsing	netflix / youtube	Video streaming
youtube	Video streaming	skype	Video conferencing
zoom	Video conferencing	skype	Video conferencing

Table 13: k-NN label & confuser comparison

Table 14: k-NN normalized broad category confusion matrix

The ten most important features to this model as derived from permutation importance are shown below in Table 15.

Feature	Weight ± Std. Dev
src2dst_max_ps	0.065722 ± 0.015302
dst2src_rst_packets	0.036115 ± 0.008005
src2dst_rst_packets	0.031129 ± 0.006146
src2dst_stddev_ps	0.023832 ± 0.010014
dst2src_min_piat_ms	0.021312 ± 0.005587
dst2src_mean_ps	0.016063 ± 0.010617
bidirectional_mean_ps	0.014698 ± 0.011116
bidirectional_max_piat_ms	0.014646 ± 0.009183
dst2src_max_piat_ms	0.014593 ± 0.009097
src2dst_max_piat_ms	0.012651 ± 0.009345

Table 15: k-NN feature importance by permutation importance

In nearly every meaningful way, the results of the k-NN classifier are better and make more intuitive sense than those of the nearest centroid. Confusers are significantly more understandable, with three broad categories directly matching between classes and their confusing classes, as opposed to only one, and four broad categories being tied with their correct match and other broad confusing classes. The most important features, similar to those of nearest centroid, are related to packet sizes, packet interarrival times, and packet reset counts. The relatively frequent misclassification of HL2:DM traffic as CS:GO traffic is also expected as both applications are not only video games but are also built on the same underlying game engine.

Naïve Bayes

The naïve Bayes classifier's logic is built on Bayes' theorem in calculating the probability that a piece of data belongs in a particular class. In doing this, it assumes that features are statistically independent (hence being called "naïve"). While this is rarely the case, naïve Bayes often perform well, despite the generally incorrect assumption.

There are different categories of naïve Bayes classifiers. Commonly used ones in popular tools and libraries include Bernoulli, categorical, complement, multinomial, and Gaussian, with

Gaussian being one of the most commonly employed. In optimization, these different methods of implementing a naïve Bayes classifier were tested and Gaussian was found to perform the best.

The ultimate classification results of the optimized Gaussian naïve Bayes classifier are shown below.

	Precision	Recall	F1-Score	Support
csgo	0.97	0.87	0.92	350
discord	0.57	0.55	0.56	22
hl2dm	0.28	0.59	0.38	22
minecraft	0.77	0.91	0.83	97
netflix	0.31	0.69	0.42	16
skype	0.39	0.18	0.25	38
vimeo	0.36	0.25	0.30	16
webbrowsing	0.39	0.55	0.45	31
youtube	0.44	0.37	0.40	19
zoom	0.42	0.21	0.28	24
Accuracy	0.74			635
Macro Average	0.49	0.52	0.48	635
Weighted Average	0.77	0.74	0.74	635

Table 16: Naïve Bayes classification results

To optimize the performance of the classifier, the most important parameter to consider in the sklearn library is the var_smoothing parameter, which represents the percentage of the variance of all features added to maximize calculation stability. Grid searching between 5,000 evenly spaced numbers in a logarithmic space between 10^0 and 10^{-9} found that the best-performing value ≅ .26983 performed best.

Broadly speaking, the naïve Bayes classifier with all features performs better than nearest centroid but worse than k-NN. This is not unexpected. Many of the features in the dataset are known to not be statistically independent of each other, such as features related to the "total" of some trait and the separate "source to destination" and "destination to source" features that contribute to the total feature. Furthermore, Gaussian naïve Bayes classifiers simply perform best when the underlying data is of a Gaussian/normal distribution. Many features of the dataset are

not normally distributed, even on a per-class basis, nor are real-world networks inclined to inherently have normally distributed properties.

The ten most important features to this model as derived from permutation importance are shown below.

Feature	Weight ± Std. Dev
bidirectional_syn_packets	0.024987 ± 0.008751
tcp	0.023937 ± 0.008300
udp	0.022310 ± 0.008289
dst2src_rst_packets	0.018950 ± 0.007448
dst2src_stddev_ps	0.017690 ± 0.008738
dst2src_max_ps	0.017585 ± 0.009955
bidirectional_max_ps	0.017270 ± 0.009816
src2dst_max_ps	0.015801 ± 0.009320
dst2src_mean_piat_ms	0.011811 ± 0.007253
src2dst_stddev_ps	0.011759 ± 0.005768

Table 17: Naïve Bayes feature importance by permutation importance

Interestingly, many of the most important features are in some way related to one another. bidirectional_max_ps, src2dst_max_ps, and dst2src_max_ps are related to each other in that the bidirectional property is the maximum of the other two. The binary presence of TCP was directly related to the presence of bidirectional SYN packets and was nearly a mutually exclusive feature with UDP (r-correlation <-.99, on account of the low presence of ICMP packets). Having such interdependent features rank so highly in importance for the naïve Bayes classifier is a sign that the dataset should be investigated for tuning further, as, ideally, the classifier cares the most about statistically independent features.

To attempt to better fulfill the statistical independence assumption of naïve Bayes, model evaluation was performed against the dataset with some mathematically related features removed. This proved to slightly improve the performance of the classifier. Accuracy and score improvements were not universal for all classes, but were broad improvement and noticeable.

The results of classification using a dataset with some select features removed are included below:

	Precision	Recall	F1-Score	Support
csgo	0.99	0.87	0.93	350
discord	0.57	0.55	0.56	22
hl2dm	0.28	0.59	0.38	22
minecraft	0.73	0.91	0.81	97
netflix	0.33	0.69	0.45	16
skype	0.38	0.13	0.20	38
vimeo	0.57	0.50	0.53	16
webbrowsing	0.49	0.65	0.56	31
youtube	0.45	0.53	0.49	19
zoom	0.38	0.25	0.30	24
Accuracy	0.75			635
Macro Average	0.52	0.57	0.52	635
Weighted Average	0.78	0.75	0.76	635

Table 18: Improved naïve Bayes classification results

To achieve this moderately better result, manual and statistical analysis was performed on the relationships between different features. While the most helpful features to remove were often highly correlated with one feature or another, simply removing one highly correlated feature from each pair of related features did not produce results as effective as intermingling manual analysis with statistical analysis. To achieve the above result, the following features were removed: tcp, bidirectional_psh_packets, bidirectional_min_ps, bidirectional_ack_packets, bidirectional_rst_packets, bidirectional_packets, and bidirectional_bytes. Removing these same features did not notably improve the performance of other classifiers in this paper; these features' removal's positive effect on the performance of the naïve Bayes classifier is likely due to its underlying assumption of statistical independence of features.

The features ranked by permutation importance in accordance with the new, more tuned feature set are included in Table 19.

Feature	Weight ± Std. Dev
bidirectional_syn_packets	0.038688 ± 0.011815
udp	0.034541 ± 0.011819
dst2src_rst_packets	0.033228 ± 0.011074
dst2src_max_ps	0.026457 ± 0.012369
src2dst_max_ps	0.019580 ± 0.009276
bidirectional_max_ps	0.019108 ± 0.009909
src2dst_mean_ps	0.017848 ± 0.010379
src2dst_fin_packets	0.017743 ± 0.008775
bidirectional_mean_ps	0.014331 ± 0.006812
src2dst_stddev_ps	0.014278 ± 0.007613

Table 19: Improved naïve Bayes feature importance by permutation importance

Many of the most valuable features by permutation are still greatly correlated with each other, but further feature removal only reduced classifier performance. For instance, the UDP feature had an r-correlation with a magnitude greater than .93 with the bidirectional_syn_packets feature, but the removal of either feature harmed performance. A heatmap showing features by their absolute r-correlation values with each other is shown in Table 20.

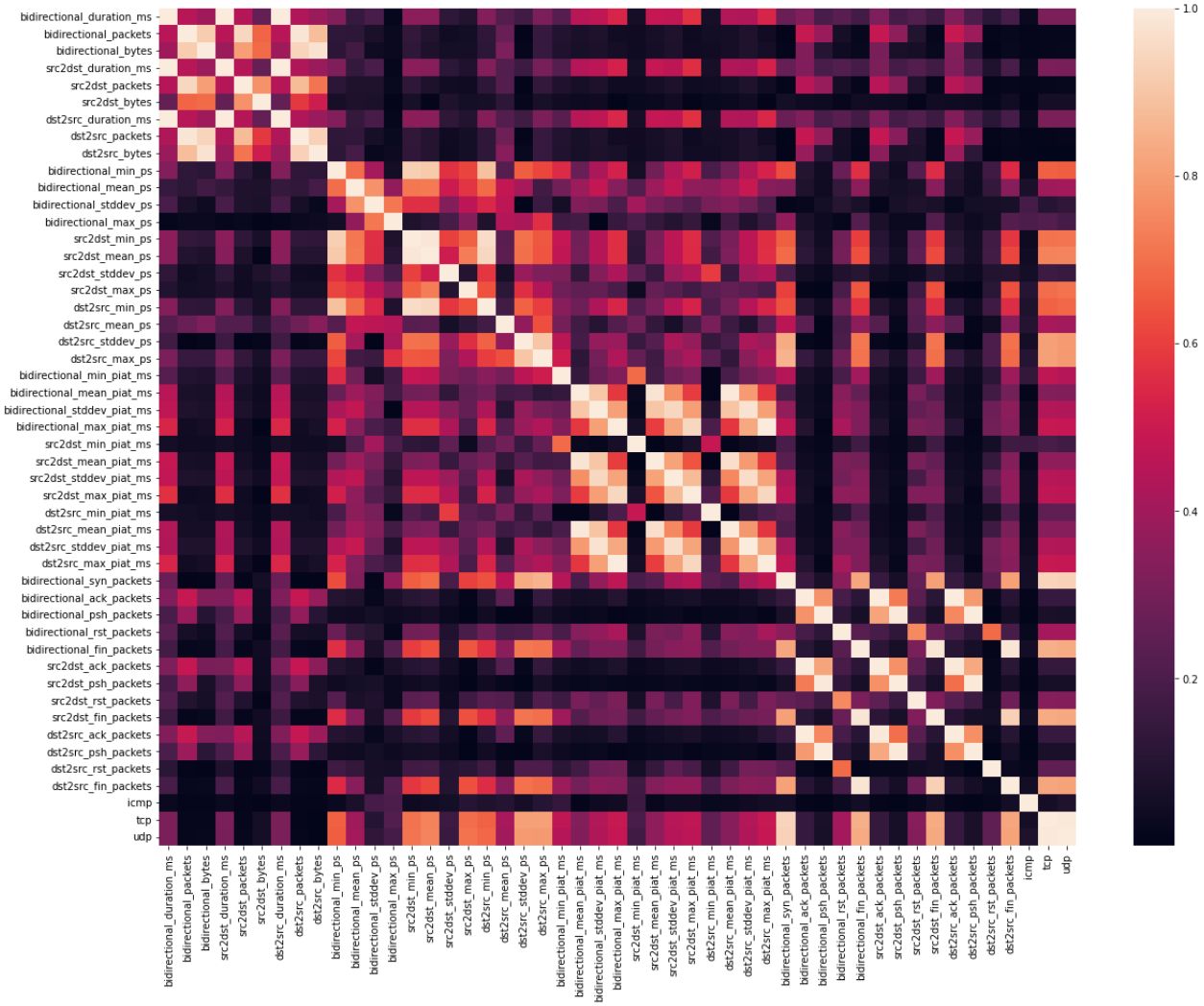

Table 20: Feature and absolute r-correlation heatmap

Still notable, however, is that of these important features, none constitute an effective standard distribution. Most of the features' distributions peak, valley, and then peak again in a manner much the opposite of a normal distribution, even on a per-class basis. Density plots for the four most important features of the feature-tuned Gaussian classifier with accompanying reference normal distributions are shown below. Where this paper was able to improve the performance of the Gaussian classifier by removing statistically dependent features, it was unable to use a similar process with features following a non-Gaussian distribution as few

features follow a normal distribution and data normalization and standardization methods had no effect on distribution density.

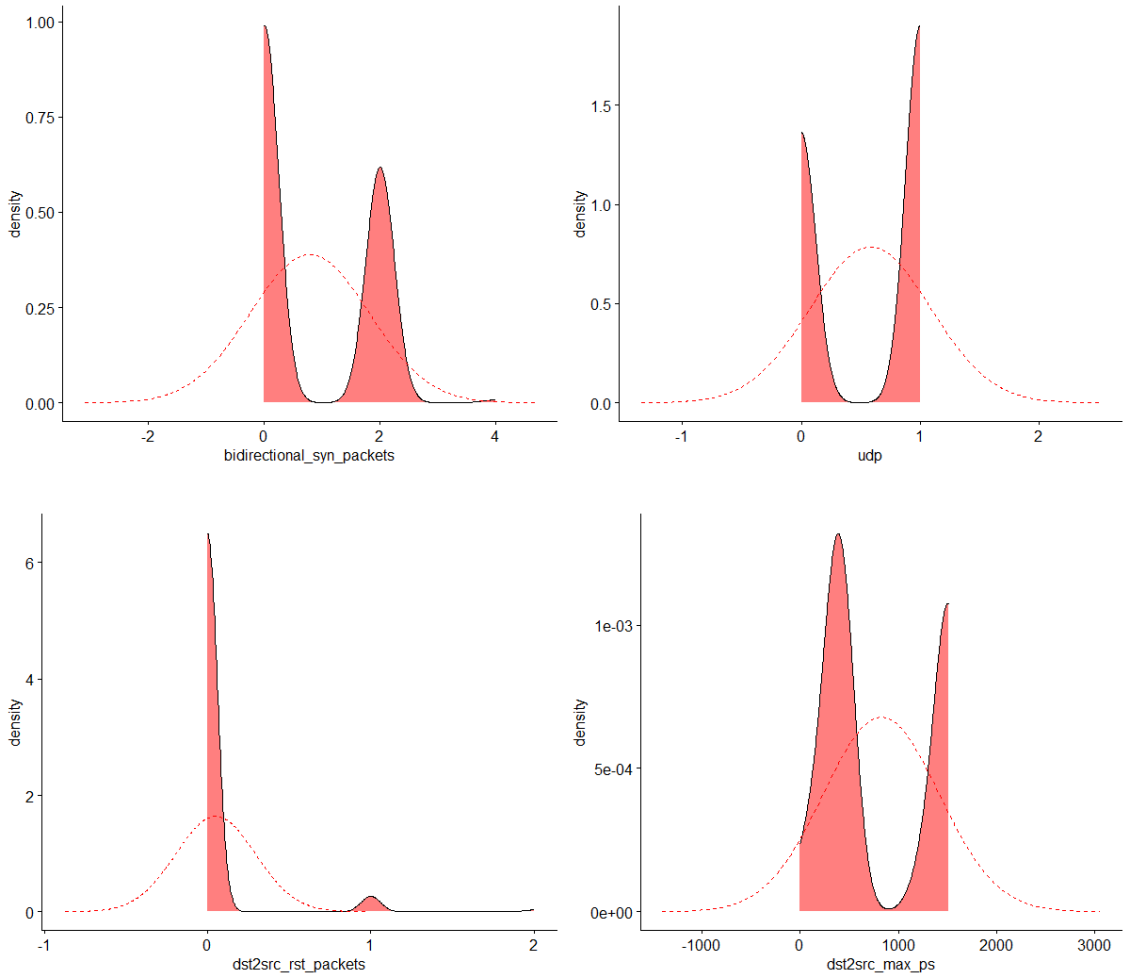

Figure 5: Feature value distribution vs. normal distribution

Finally, confusion matrices representing the total classifications, recall per label, and results by broad category are respectively shown below for the feature-tuned naïve Bayes classifier.

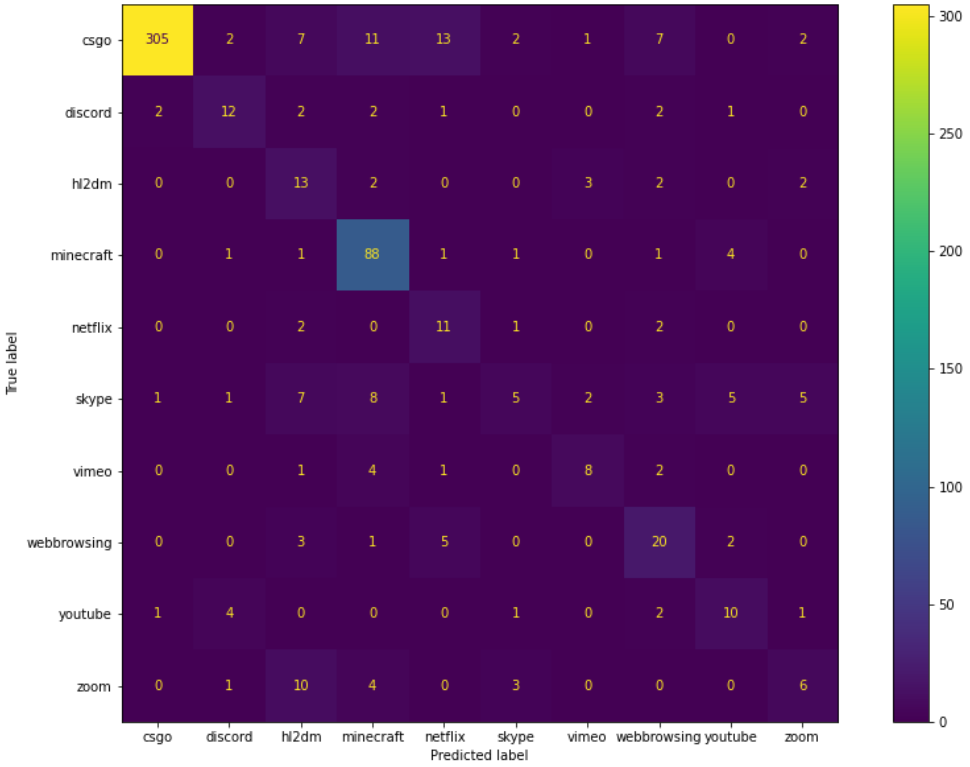

Table 21: Improved naive Bayes confusion matrix

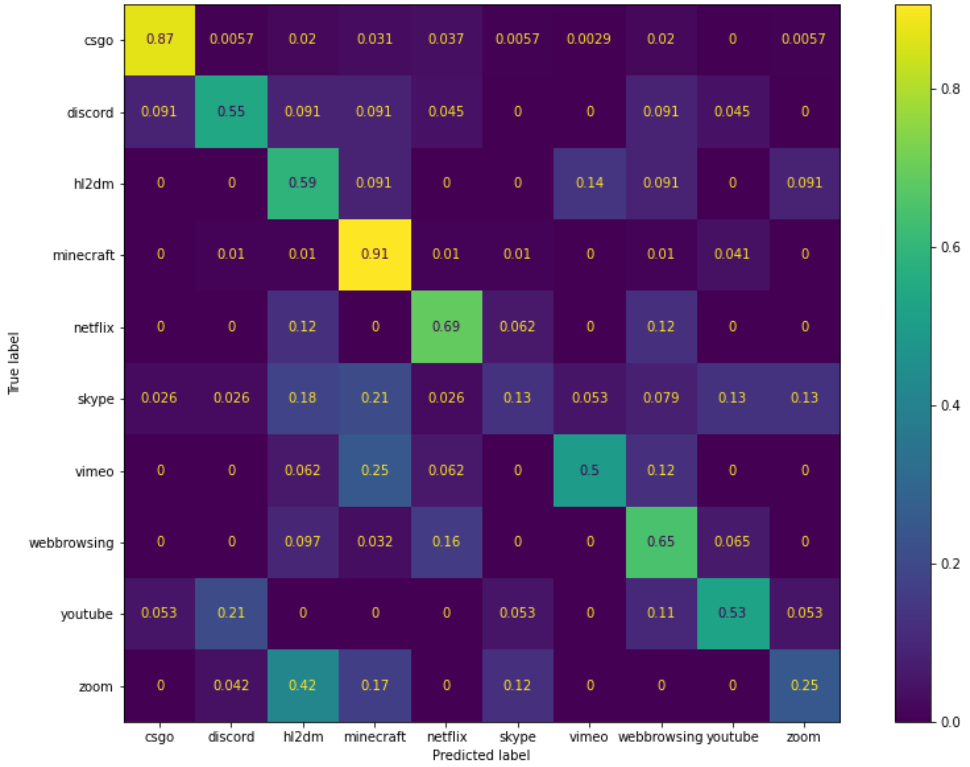

Table 22: Improved naive Bayes normalized confusion matrix

Table 23: Improved NB normalized broad category confusion matrix

Support Vector Machines

Support vector machines (SVM) operate on the basis that there exists a mathematical hyperplane that can separate data points into different classes. This hyperplane is separated by an optimally large margin as distant as possible from different training samples defining the boundaries of different classes. These training samples, called support vectors, define the margin. A kernel function is used to optimize the location of the hyperplane that produces the maximum possible margin between the hyperplane and its support vectors.

The classification results of SVM using the Python sklearn library's C-Support Vector Classification (SVC) using an optimized RBF function are included in Table 24.

	Precision	Recall	F1-Score	Support
csgo	0.97	0.98	0.97	350
discord	0.58	0.64	0.61	22
hl2dm	0.64	0.64	0.64	22
minecraft	0.90	0.92	0.91	97
netflix	0.77	0.62	0.69	16
skype	0.62	0.66	0.64	38
vimeo	0.73	0.69	0.71	16
webbrowsing	0.82	0.74	0.78	31
youtube	0.76	0.84	0.80	19
zoom	0.71	0.62	0.67	24
Accuracy	0.88			635
Macro Average	0.75	0.73	0.74	635
Weighted Average	0.88	0.88	0.88	635

Table 24: SVM classification results

Optimization of the SVM classifier found that the best underlying kernel was a radial basis function. Two other parameters, γ and C, were found to be optimal at values ~0.020236 and 39.07, respectively.

The SVM classifier performed well**.** Support vector machines are most well-known for producing good results without the need for extensive computational resources when compared to similar traditional models. There is little to comment on regarding the dataset or the classifier based on its performance. It largely does well at what most of the other classifiers shown thus far do and poorly where other classifiers also tend to fail. The rest of this section will include summary information for reference in later discussion.

The permutation importance-derived most important features are included in Table 25.

Feature	Weight Std. Dev
src2dst_max_ps	0.083465 ± 0.017599
dst2src_rst_packets	0.033333 ± 0.008511
dst2src_min_piat_ms	0.024462 ± 0.006562
bidirectional_max_ps	0.018320 ± 0.007033
src2dst_stddev_ps	0.018320 ± 0.008716
src2dst_max_piat_ms	0.016535 ± 0.007807
bidirectional_max_piat_ms	0.014961 ± 0.007388
dst2src_max_piat_ms	0.014593 ± 0.007233
dst2src_stddev_ps	0.014541 ± 0.010175
src2dst_fin_packets	0.013228 ± 0.006173

Table 25: SVM feature importance by permutation importance

Confusion matrices representing the total classifications, recall per label, and results by broad category are respectively shown below.

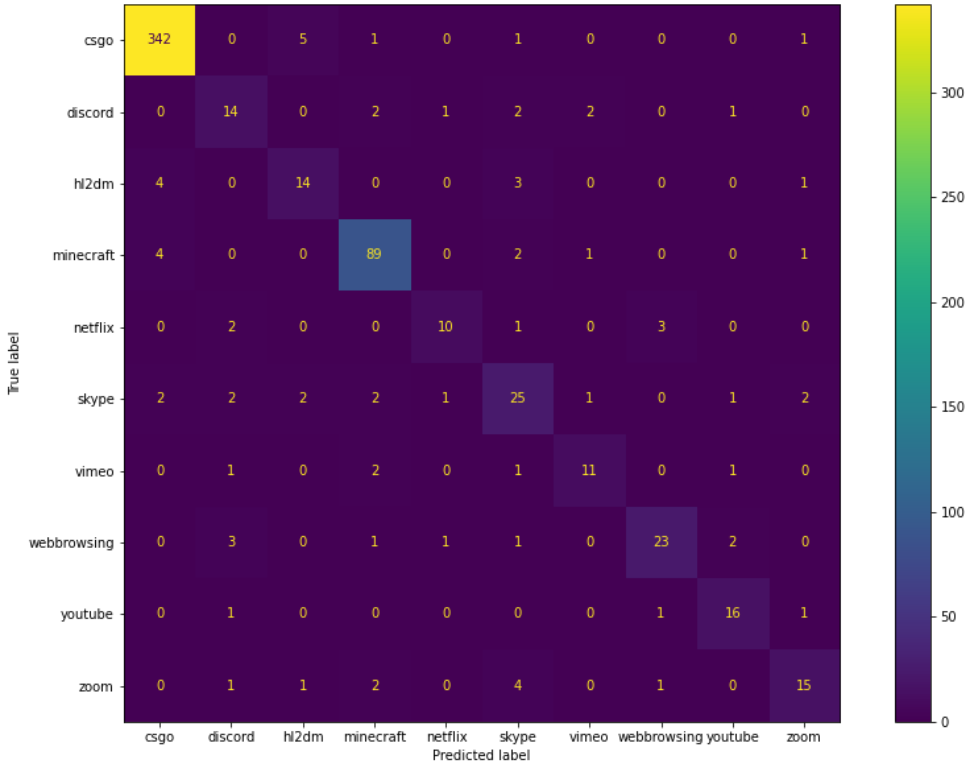

Table 26: SVM confusion matrix

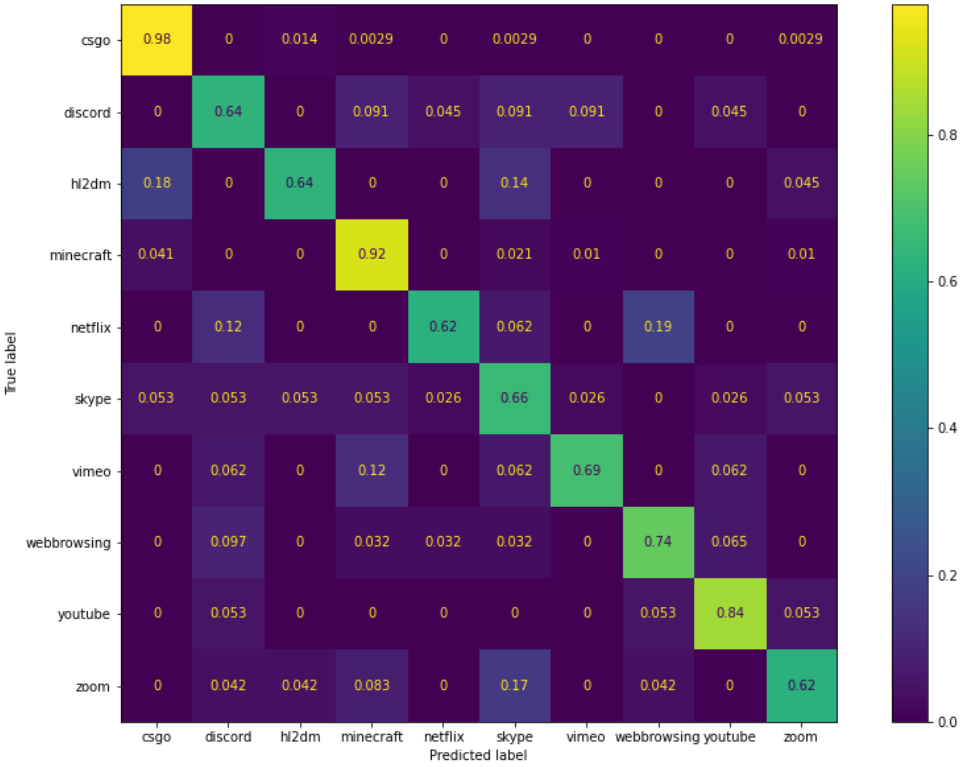

Table 27: SVM normalized confusion matrix

Table 28: SVM normalized broad category confusion matrix

Decision Trees

Functionally, decision trees are one of the most easily understood classification models. Decision trees consist of nodes that define a condition and branch based on whether that condition is fulfilled. The conditions set at each node are based on brute force determinations of which features and criteria most purely split the dataset samples by class in accordance with a purity function, such as Gini impurity. They can be thought of as elaborate flow diagrams and are frequently visually represented using tree diagrams.

Unlike some of the other models used in this paper, tree-based algorithms like decision trees are not sensitive to the magnitude of feature variance. Therefore, standardization was not necessary on the dataset and, in fact, for k-folds cross-validation, the decision tree classifier's performance between a standardized dataset and a non-standardized dataset was completely identical.

Decision trees are very sensitive to the features upon which decisions can be made. With misleading features, model performance can significantly decrease. Throughout the process of testing the decision tree classifier, two separate feature sets were tested: the full feature set, and a feature set consisting of only the ten most important items for the decision tree classifier as determined using permutation importance. Classification results for an optimized decision trees classifier using a standardized dataset with all features are included in Table 29.

	Precision	Recall	F1-Score	Support
csgo	0.95	0.97	0.96	350
discord	0.62	0.68	0.65	22
hl2dm	0.63	0.55	0.59	22
minecraft	0.95	0.90	0.92	97
netflix	0.83	0.62	0.71	16
skype	0.67	0.63	0.65	38
vimeo	0.67	0.62	0.65	16
webbrowsing	0.61	0.74	0.67	31
youtube	0.85	0.89	0.87	19
zoom	0.57	0.50	0.53	24
Accuracy	0.87			635
Macro Average	0.73	0.71	0.72	635
Weighted Average	0.87	0.87	0.86	635

Table 29: Decision tree classification results

Hyperparameter optimization of the sklearn library's implementation of the decision tree classifier was employed against multiple parameters. When using all features, the most effective quality function was Gini impurity (as opposed to entropy or logistic loss), and the most effective class weight function was "balanced," in which weights are inversely proportional to class frequencies.

Classification results for a decision tree evaluated using a dataset with only the ten most important features to the previous decision tree are included in Table 30. These ten most important features were bidirectional_min_ps, src2dst_max_ps, src2dst_min_ps, dst2src_mean_ps, src2dst_stddev_ps, bidirectional_stddev_ps, dst2src_stddev_ps, dst2src_ack_packets, src2dst_duration_ms, and dst2src_max_piat_ms.

	Precision	Recall	F1-Score	Support
csgo	0.96	0.98	0.97	350
discord	0.60	0.68	0.64	22
hl2dm	0.65	0.50	0.56	22
minecraft	0.90	0.88	0.89	97
netflix	0.92	0.75	0.83	16
skype	0.57	0.61	0.59	38
vimeo	0.62	0.62	0.62	16
webbrowsing	0.73	0.77	0.75	31
youtube	0.80	0.84	0.82	19
zoom	0.74	0.58	0.65	24
Accuracy	0.87			635
Macro Average	0.75	0.72	0.73	635
Weighted Average	0.87	0.87	0.87	635

Table 30: Decision tree feature importance improved classification results

When using only these features, the optimized parameters were the same as for the decision tree using all features. The reason this second set of classification results is included is to show how nearly identical and even numerically better results can be achieved using higher quality, but significantly fewer features.

Simple decision tree classifiers have some noteworthy problems. As will be discussed further below, they are prone to overfitting on meaningless statistical nuances of a particular dataset. For a particular training set, some features may hold some meaningless information that happens to form a relative pattern between classes that is not typically reflected in reality. Decision trees can often make significant use of every feature they are given, even if they are effectively noise, leading to overfitting and meaningless features being interpreted as valuable. The issues of overfitting and attributing importance to unimportant features can be better resolved by using a random forest classifier, which this paper explores later.

The decision tree diagram drawn from a decision tree classifier ran against the dataset using all features is too large to display, with over 40 individual nodes. However, examining just the top of a decision tree can be highly informative. If there are indeed very strong features and

commonalities that can accurately separate different classes, one can develop a strong understanding of some statistical commonalities in a dataset and obtain an inherent explainer of how the decision tree classifier works. The root of a tree using all features and its immediate two sub nodes are shown in Figure 6.

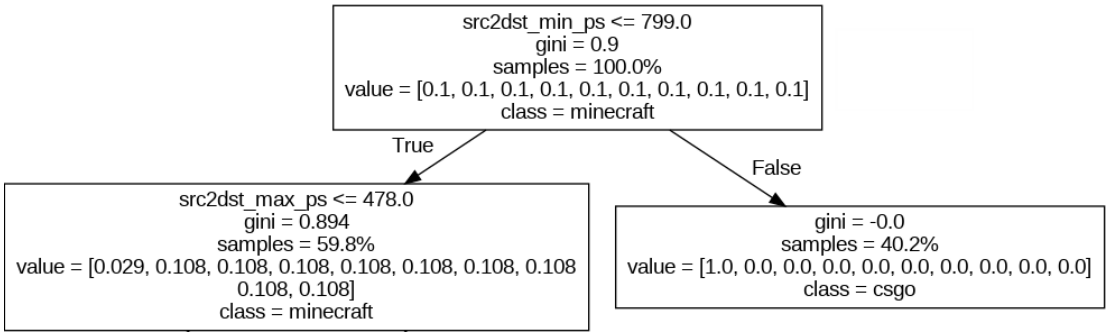

Figure 6: Root of all features decision tree

The root most important distinguishing feature in this decision tree classifier was src2dst_min_ps. A reasonably tight 59.8:40.2 split was made based on its condition. Significantly, with just one decision at the root of the tree, 40% of the samples were already classified.

The better-performing decision tree, using features derived from the permutation importance of the “all features” decision tree classifier, has a different decision tree diagram. Its root can be examined below.

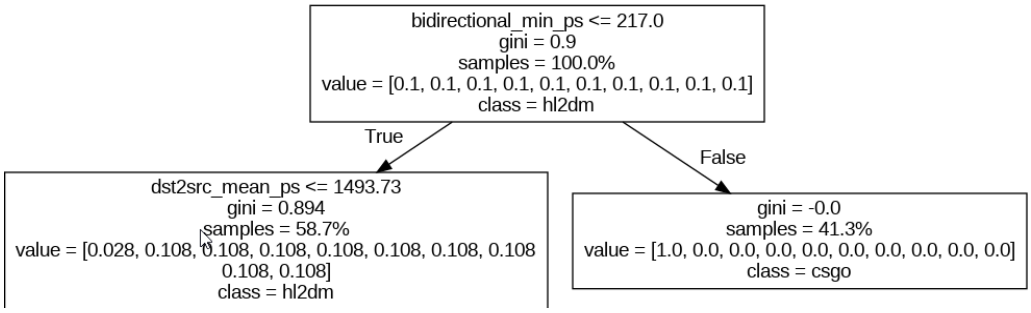

Figure 7: Root of better-performing decision tree

Both the better-performing narrow feature set decision tree and the “all features” decision tree start with some form of minimum packet size as the root of their tree. Like the last one, the root of this decision tree was able to classify about 40% of the flow with one single decision. Given its presence in both decision tree roots, minimum packet size seems to be a very high-quality feature. Note that the field “value,” representing the relative proportion of samples present, is shown to be apparently equal at the root of the above tree because the classes are adjusted to be balanced in weight, as opposed to only being based on sample count.

Not all nodes make equally valid splits, even using only 10 features from the better-performing decision tree, many of the end leaf nodes are fairly overfitted and invalid in their assumptions. Consider the bottom nodes shown below from a decision tree using the better-performing feature set.

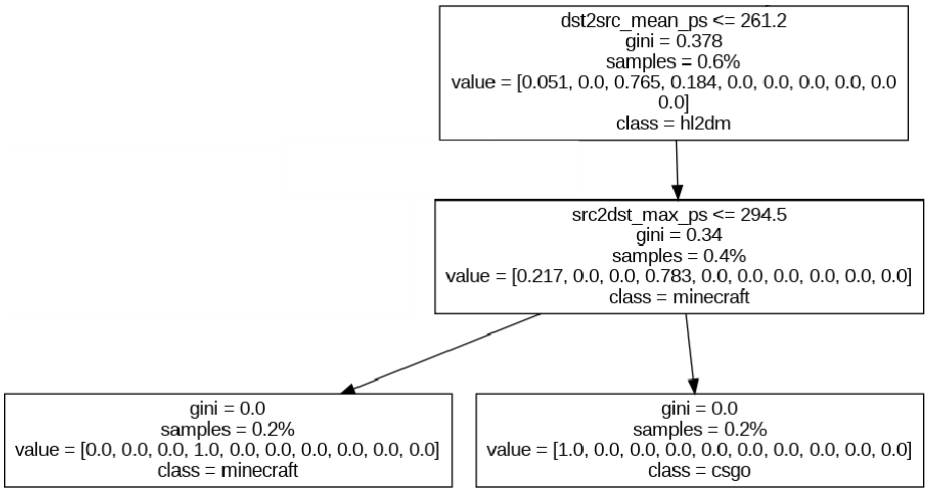

Figure 8: Example best-performing decision tree leaf nodes

Exact statistical relationships, such as minute range differences between this dataset’s dst2src_mean_ps and src2dst_max_ps features, are rarely exactly generalizable to new data in the context of network traffic. While the final shown decision’s conditionality was a determinant in only 0.4% of the data samples, it can also be thought of as only having been true for 0.4% of a

subset of a very specific dataset representing just a single experiment in time. It is almost certainly not a broadly, legitimately meaningful determinant of a class of network traffic.

Tree classifiers are a category of classifiers easily and broadly supported by a popular modern feature importance explanation technique, Shapley (SHAP) values. SHAP values have their origin in game theory and show the effect a particular data point's feature values had in the determination of its label. Where permutation importance can help explain model performance, SHAP values are best suited for understanding how a model makes classifications. A bar plot showing the average SHAP value across folds in the cross-validation process for the classifier using all features for the entire dataset is shown below.

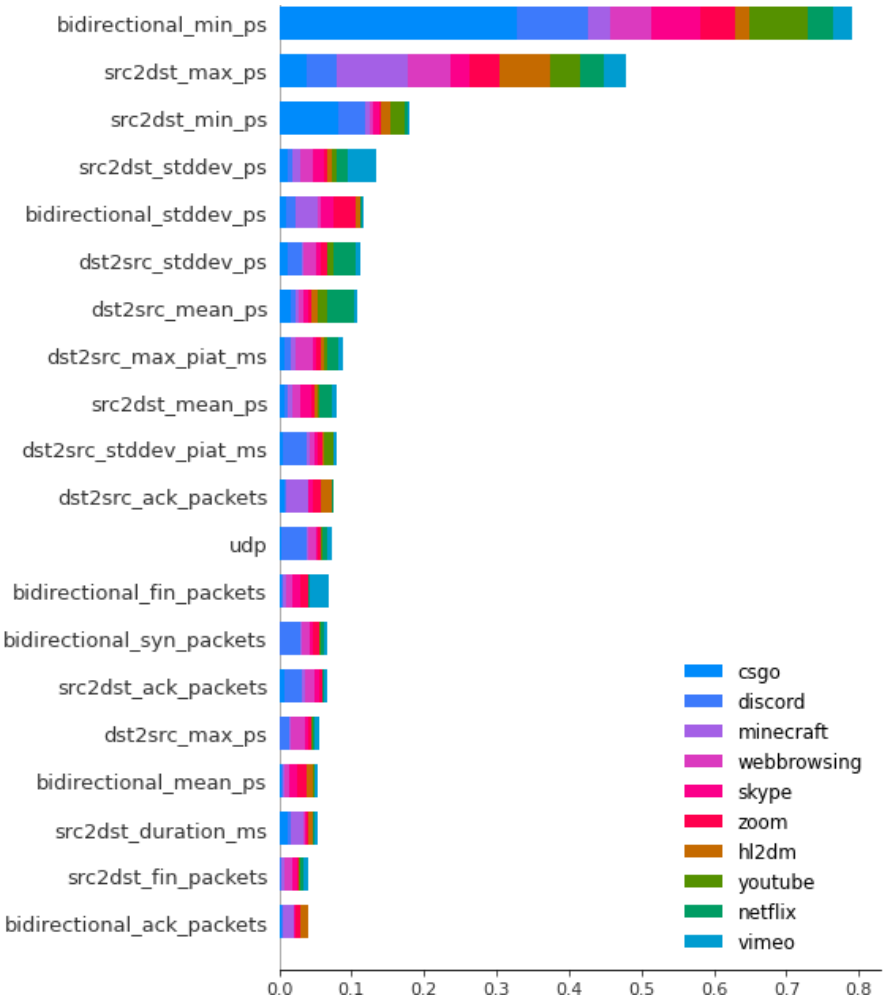

Figure 9: Top features by absolute SHAP value importance

Just as might be inferred from the classifier's tree diagram, some of the most important features in determining a class are located near the top of the tree: src2dst_max_ps and src2dst_min_ps. The most important feature in the figure, bidirectional_min_ps, was the root of the fewer-featured decision tree. The SHAP values in this bar plot also help in understanding what features most strongly affect classification on a per-label basis. For example, CS:GO-tagged traffic was strongly impacted by the bidirectional minimum packet size, but was not strongly affected by the source-to-destination maximum packet size. HL2:DM-tagged traffic, on the other hand, exhibited the opposite relationship.

In addition to the magnitude of the effect of a feature, SHAP values can explain the directionality of how a feature affects the classification of a sample. A beeswarm chart for Skype traffic classification in order of most impactful features is shown below.

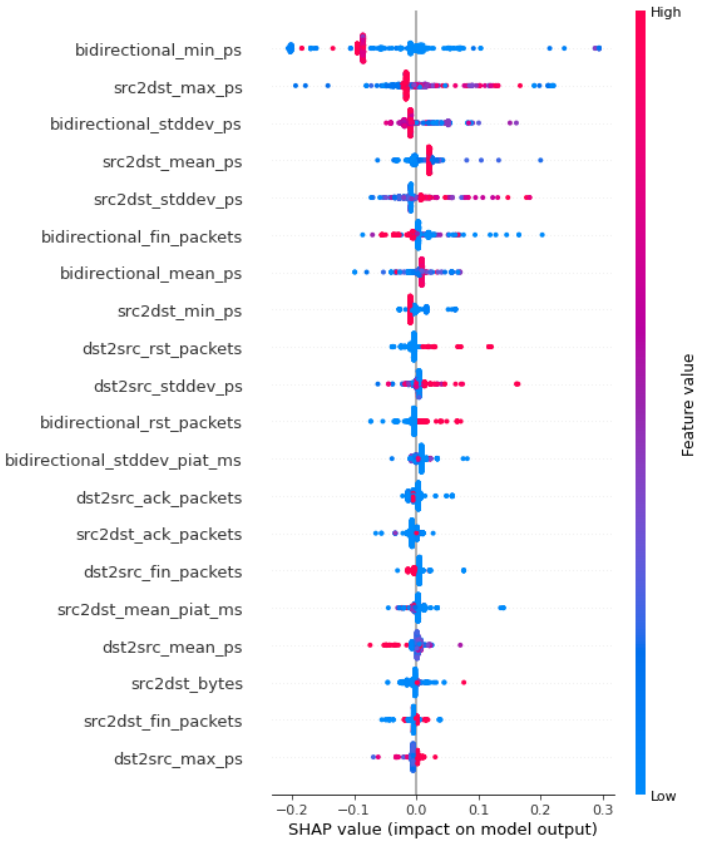

Figure 10: Beeswarm chart for Skype traffic classification

From this beeswarm chart, it can be observed that a high value for the bidirectional_stddev_ps feature was a light indicator that a traffic flow did not belong to Skype, whereas low values were a generally positive indicator that a flow belonged to Skype.

Feature importances as measured by permutation importance are shown in Table 31 for the decision tree classifier using all features. When the dataset was isolated to only these features, it produced the second-shown and better-performing decision tree classifier in this section.

Feature	Weight ± Std. Dev
bidirectional_min_ps	0.286404 ± 0.029184
src2dst_max_ps	0.192913 ± 0.024339
src2dst_min_ps	0.085564 ± 0.010584
dst2src_mean_ps	0.040997 ± 0.009049
src2dst_stddev_ps	0.040630 ± 0.012795
bidirectional_stddev_ps	0.035066 ± 0.011234
dst2src_stddev_ps	0.028084 ± 0.008489
dst2src_ack_packets	0.027454 ± 0.005766
src2dst_duration_ms	0.026089 ± 0.007620
dst2src_max_piat_ms	0.025039 ± 0.005454

Table 31: All feature decision tree feature importance by permutation importance

Notably, many of the ten most permutation importance-determined important features are shared with the SHAP-determined important features, though not always sharing the same position and weight. A comparison of the most important features derived using these two methods is shown below.

Permutation Importance Position	SHAP Value Importance Position	Shared Important Feature
1	1	birectional_min_ps
2	2	src2dst_max_ps
3	3	src2dst_min_ps
5	4	src2dst_stddev_ps
6	5	bidirectional_stddev_ps
4	7	dst2src_mean_ps
10	8	dst2src_max_piat_ms

Table 32: Feature importance from permutation importance and SHAP value comparison

Confusion matrices showing the recall results for the decision tree classifier evaluated against the entire feature set and against the optimized feature set are shown below. Additionally, a broad category confusion matrix is shown for the optimized feature set classifier.

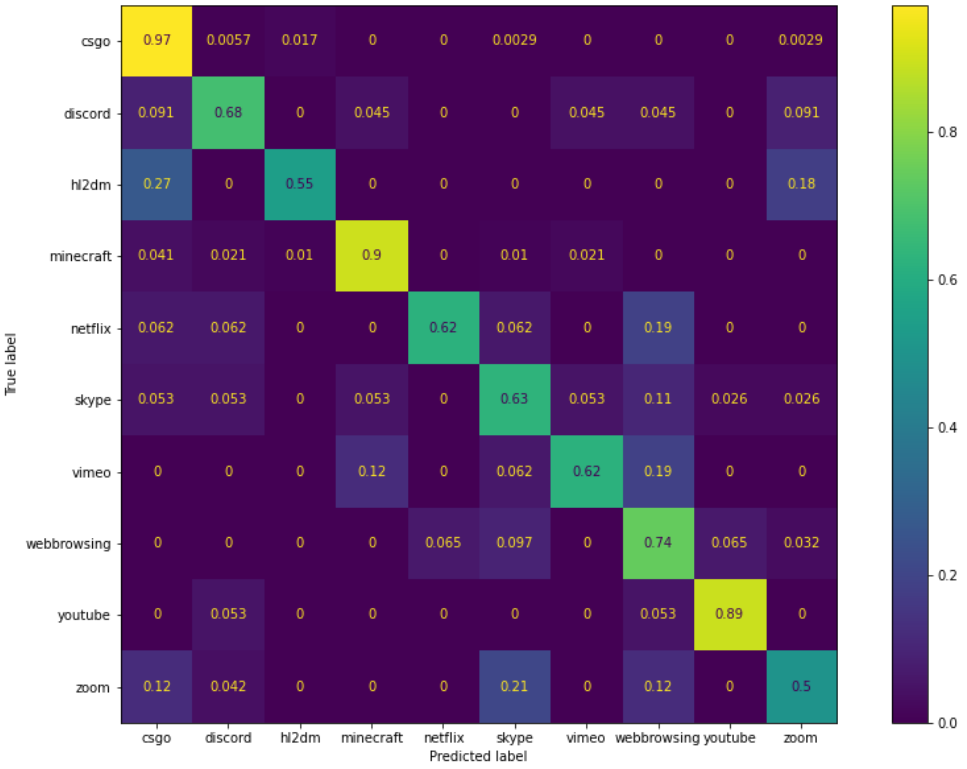

Table 33: All features decision tree normalized confusion matrix

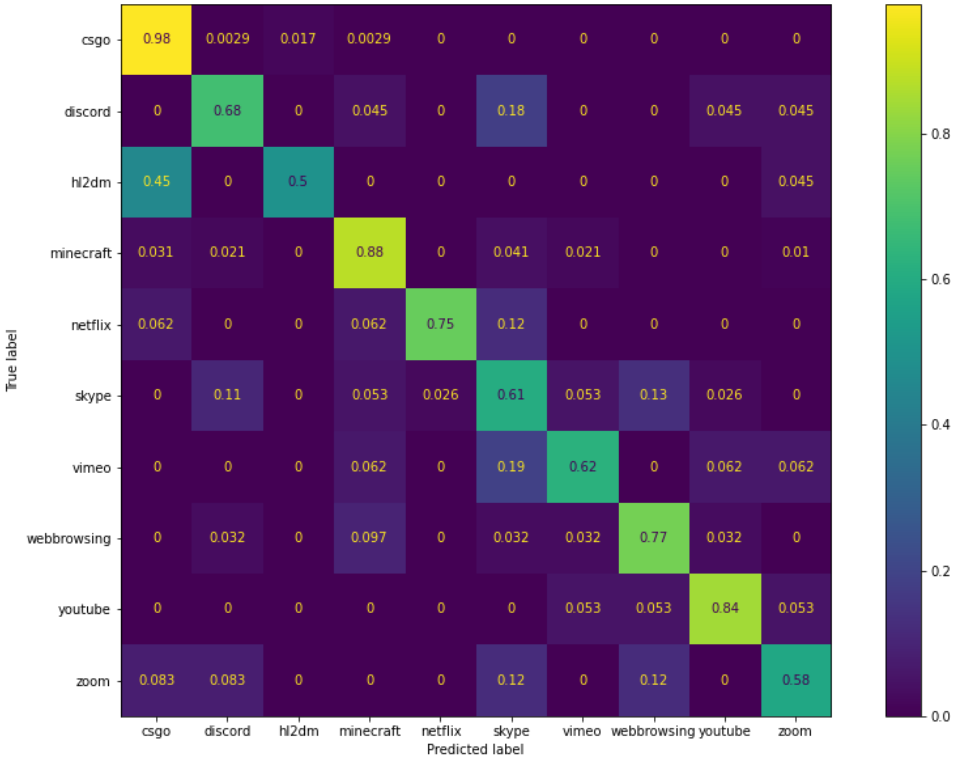

Table 34: Better-performing decision tree normalized confusion matrix

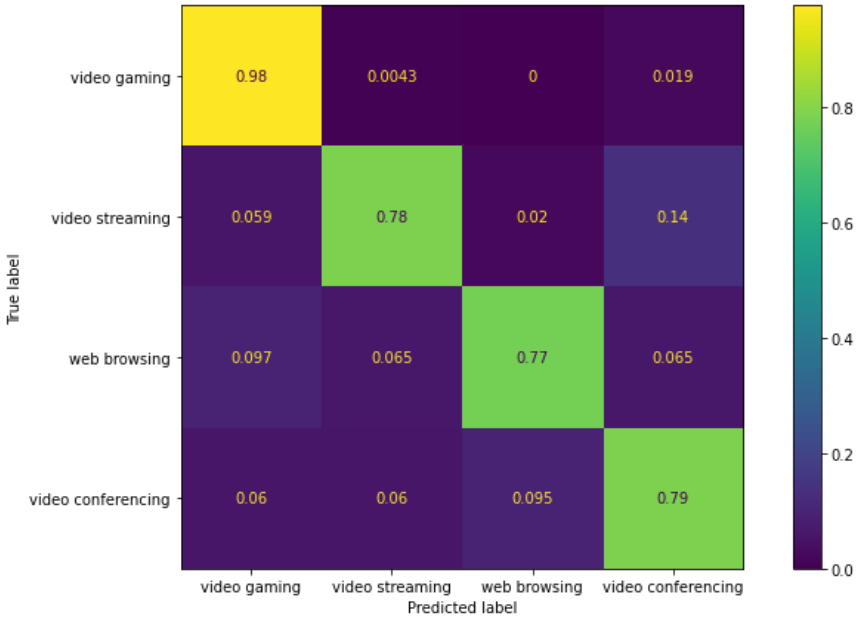

Table 35: Better-performing DT normalized broad category confusion matrix

Random Forest

Decision trees have the previously discussed issue of overfitting on statistical nuances of a particular dataset. To remedy this, one can employ a *random forest* of decision trees. Random forests employ many different decision tree classifiers to, as an ensemble, vote on data classification. To avoid overfitting and to ensure that individual decision trees are not completely identical, random forest classifiers typically resample training data with replacement to create new training sets through a process called bagging. Bagging employs bootstrap sampling, which allows training samples to be selected more than once. Furthermore, to reduce bias from highly predictive features, the "random" component of random forest will randomly select a subset of features that each decision tree will be trained on. In practice, each tree is typically trained using randomly selected features totaling the square root of the number of features.

The ultimate classification results of the optimized random forest classifier are shown in Table 36.

	Precision	Recall	F1-Score	Support
csgo	0.97	0.98	0.97	350
discord	0.72	0.59	0.65	22
hl2dm	0.71	0.55	0.62	22
minecraft	0.89	0.92	0.90	97
Netflix	0.93	0.81	0.87	16
skype	0.63	0.63	0.63	38
vimeo	0.80	0.75	0.77	16
webbrowsing	0.71	0.94	0.81	31
youtube	0.81	0.89	0.85	19
zoom	0.74	0.58	0.65	24
Accuracy	0.89			635
Macro Average	0.79	0.76	0.77	635
Weighted Average	0.89	0.89	0.89	635

Table 36: Random forest classification results

The most notable parameter to optimize in a random forest classifier is the number of decision trees, "estimators," to use. There is rarely a classification performance downside to

using a large number of estimators, but there are diminishing returns as the number of estimators increases. For the above results, 100 estimators were used in accordance with the results of the graph in Figure 11 comparing the accuracy with the number of estimators used for the classifier fitted on a 5:1 train-test split.

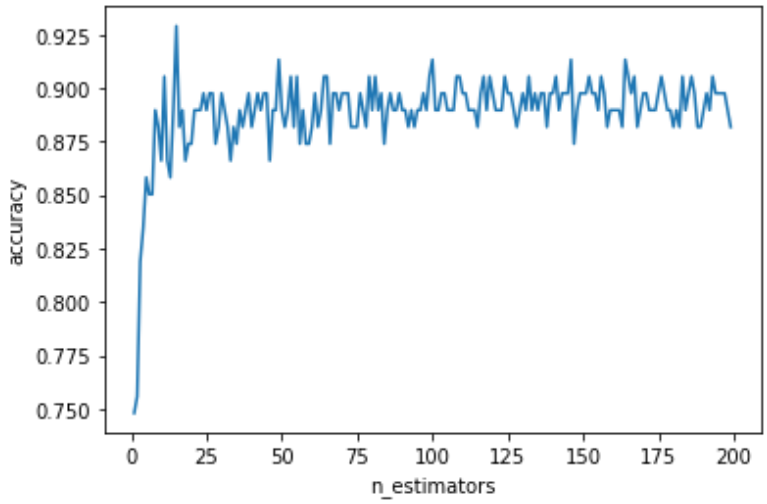

Figure 11: accuracy vs number of estimators

As Figure 11 shows, the accuracy score of the random forest classifier reaches a relatively steady state somewhere between 50 and 100 estimators, and thus 100 estimators were ultimately chosen. The accuracy results achieved at 100 estimators were approximately as good as results achieved with more estimators but did not require significantly more processing power. Other parameters optimized were those related to the performance of the individual decision trees. The optimized results varied slightly from those of the decision tree classifier section of this paper: Gini impurity was used as a purity function, and unbalanced class weights were used.

Of the paper thus far, these classification results are roughly the best according to F1 score and accuracy. It is not unexpected for a random forest classifier to perform well compared to the rest of the classifiers, particularly the single decision tree. Random forests generally perform better than any individual well-performing decision tree because they are more resistant

to overfitting. Furthermore, random forest classifiers are typically better capable of classifying completely new data of the often chaotic nature of the data in this dataset.

Confusion matrices representing the total classifications, recall per label, and results by broad category are respectively shown in Table 37, Table 38, and Table 39.

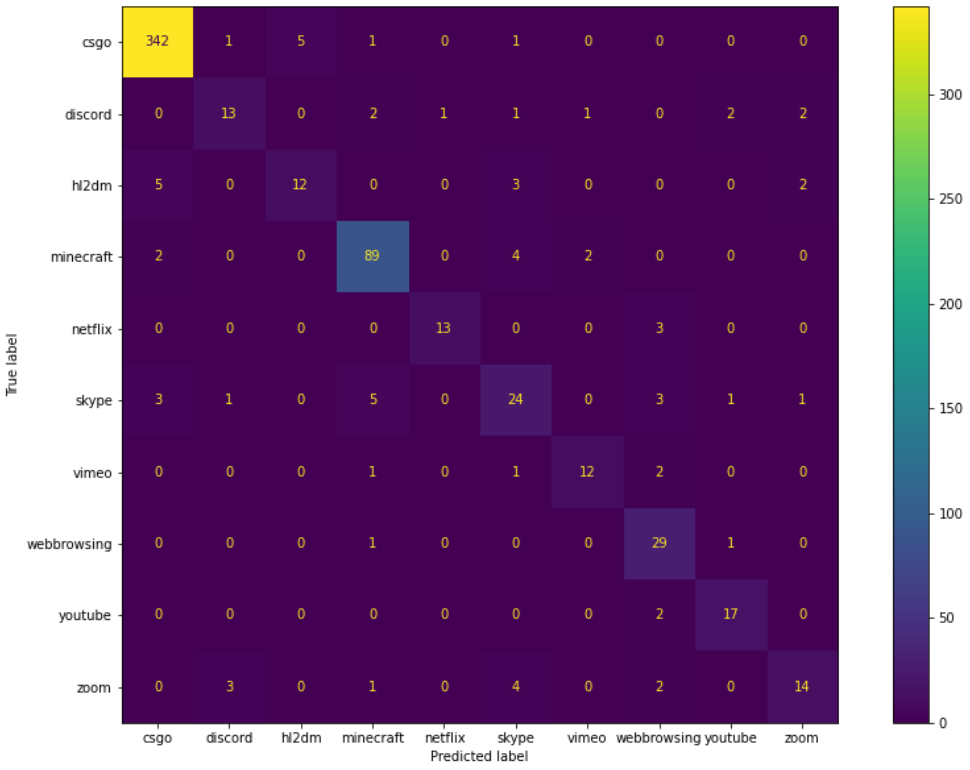

Table 37: Random forest confusion matrix

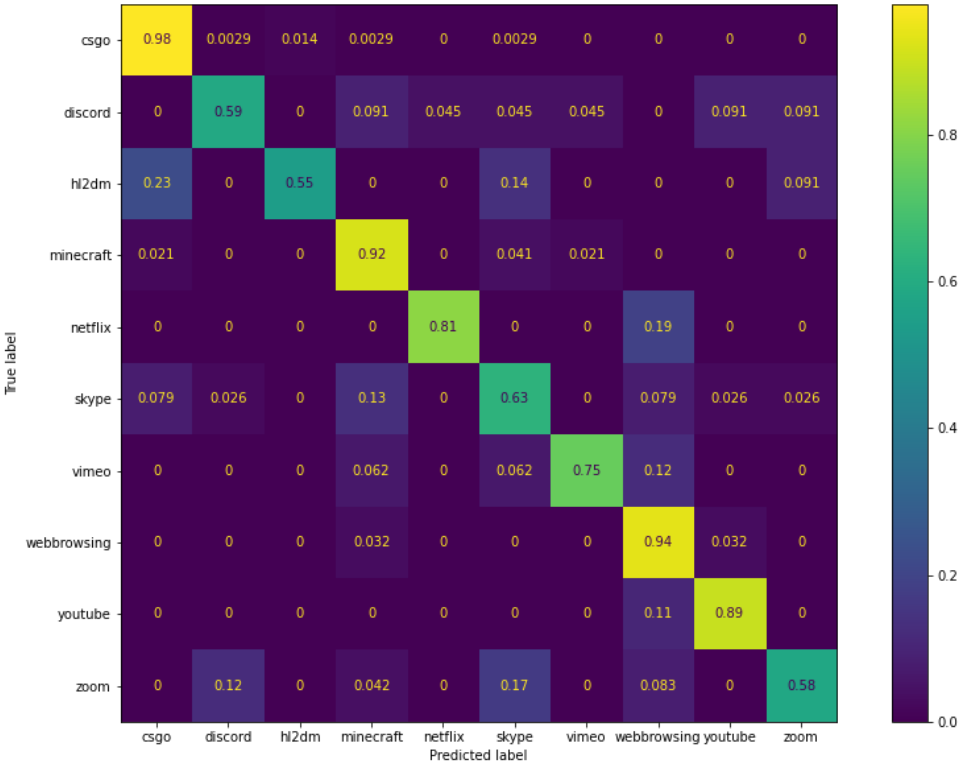

Table 38: Random forest normalized confusion matrix

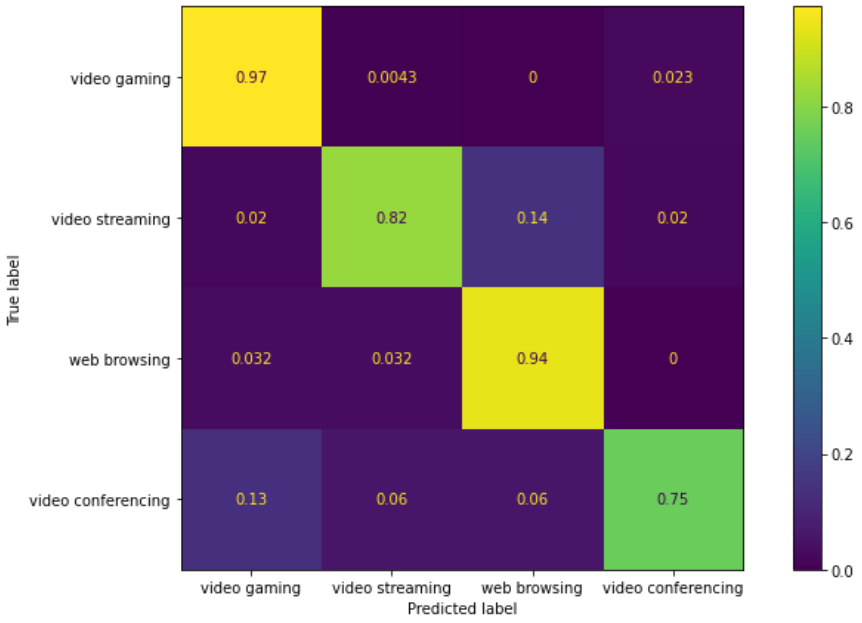

Table 39: Random forest normalized broad category confusion matrix

The most important features as measured by permutation importance for the random forest classifier are shown below.

Feature	Weight ± Std. Dev
src2dst_max_ps	0.007979 ± 0.008899
dst2src_max_piat_ms	0.006352 ± 0.003669
dst2src_mean_piat_ms	0.002992 ± 0.002974
bidirectional_fin_packets	0.002257 ± 0.003784
dst2src_stddev_piat_ms	0.001890 ± 0.003402
src2dst_duration_ms	0.001732 ± 0.004728
bidirectional_ack_packets	0.001417 ± 0.004108
bidirectional_min_ps	0.000577 ± 0.003625
bidirectional_max_ps	0.000157 ± 0.003439
bidirectional_syn_packets	0.000105 ± 0.001568

Table 40: Random forest feature importance by permutation importance

Interestingly, there is a noticeable difference between the most important features of the individual decision tree classifier as evaluated against all features and the random forest classifier's most important features. Between the two classifiers, only four of the most important features are shared, including bidirectional_min_ps, max_piat_ms, src2dst_duration_ms, and src2dst_max_ps. A question that may arise is, if a random forest is an ensemble of decision trees, are its important features valuable to a decision tree classifier? To test this, a decision tree classifier was trained, optimized, and evaluated using only the ten features shown in Table 40. While the results were good, they were not significantly better or different from either the decision tree classifier using all features or the feature-tuned decision tree classifier, and as such, their results are not included for discussion.

Multilayer Perceptron

Increasingly available and of interest to the general public are neural networks. Neural networks are composed of "layers" of "neurons" that process input data in the pursuit of performing accurate classification. *Layers* in a neural network appear as an input layer (the inputs to the model), an output layer (the classification/regression ultimately determined), and intermediary "hidden" layers populated by neurons whose purpose is to perform activation functions on the inputs they receive before passing them onto the next layer. Modern deep

learning is built on the concept of neural networks using many hidden layers (hence being called "deep") and is most often preferred in the context of extremely large and complex datasets. Multilayer perceptron is one of the simplest popular neural network models. The classification results for an optimized multilayer perceptron classifier are shown below:

	Precision	Recall	F1-Score	Support
csgo	0.96	0.98	0.97	350
discord	0.61	0.64	0.62	22
hl2dm	0.65	0.59	0.62	22
minecraft	0.90	0.93	0.91	97
netflix	0.77	0.62	0.69	16
skype	0.68	0.68	0.68	38
vimeo	0.79	0.69	0.73	16
webbrowsing	0.76	0.81	0.78	31
youtube	0.75	0.79	0.77	19
zoom	0.79	0.62	0.70	24
Accuracy	0.88			635
Macro Average	0.77	0.73	0.75	635
Weighted Average	0.88	0.88	0.88	635

Table 41: MLP classification results

The sklearn library implementation of the multilayer perceptron model's optimized results used a weight optimization solver parameter of "adam" (a stochastic gradient-based optimizer), an activation function of "relu" (a Rectified Linear Unit function of $f(x) = \max(0, x)$), and a maximum number of epochs ("max_iter") of 800 was chosen based on early stop analysis to avoid common overfitting issues and to preserve performance. The network architecture included a single hidden layer with 68 neurons. A single hidden layer was chosen because it performed the best, as will be discussed later. A graph showing the error rate of the classifier by the maximum number of epochs used by the stochastic solver is shown in Figure 12 below. Below that, Table 42 compares error rate with hidden layer size (neuron count in the first hidden layer).

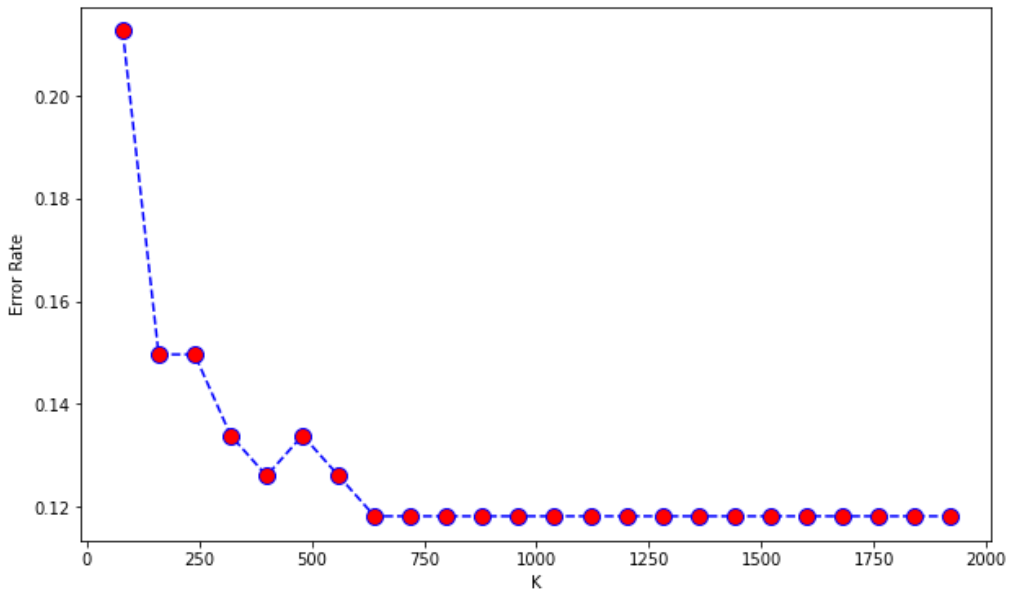

Figure 12: Error rate vs epoch count (K)

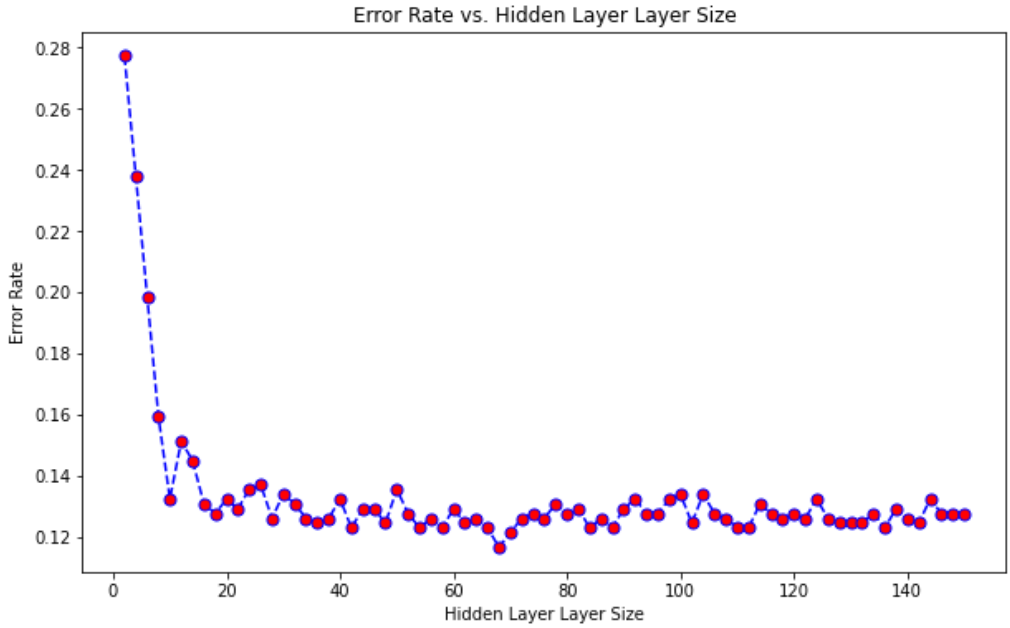

Table 42: Error rate vs hidden layer size (neuron count)

In distinguishing a simple neural network from one engaged in "deep learning," the number of hidden layers used by the classifier is typically considered. "Wide" neural networks contain few layers, whereas deep neural networks contain many. This classifier performed best with a single hidden layer and thus cannot be considered to have engaged in deep learning. This

is not unexpected for a small dataset containing a limited number of features. The purpose of having many layers in a neural network is to identify more detailed and more nuanced relationships in data. Small, non-complex, linearly separable datasets (as shown to generally be the case by the performance of a linear SVM classifier discussed below) are not expected to require more than a single hidden layer to perform well. Adding additional layers of the same or different sizes at best (using a similar number of neurons per layer) had no impact on the classifier's performance, and at worst (using a significantly higher or lower number of neurons per layer) reduced performance.

One indicator that the dataset is linearly separable can be shown in the performance of a linear SVM classifier. Without going into great detail, a linear SVM classifier with an optimized C value achieves an accuracy/weighted average recall of 87%, macro average precision of 72%, macro average recall of 70%, macro average F1 score of .71, weighted average precision of 87%, and a weighted average F1 score of .87. These are very adequate results.

Confusion matrices representing the total classifications, recall per label, and results by broad category are respectively shown in Tables 43, 44, and 45.

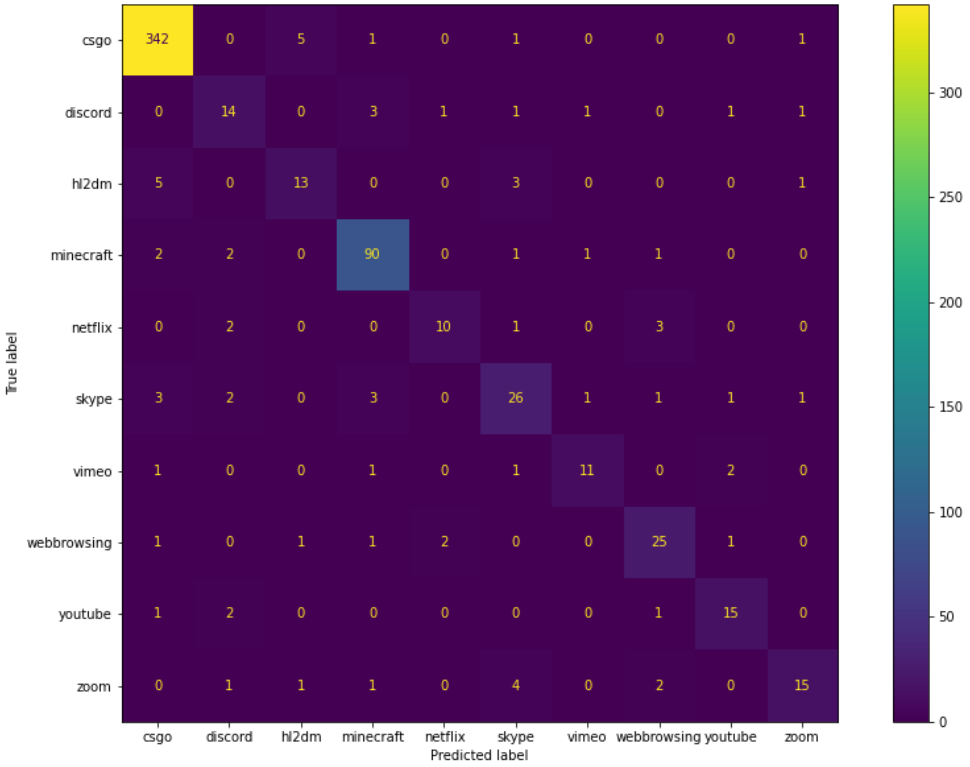

Table 43: MLP confusion matrix

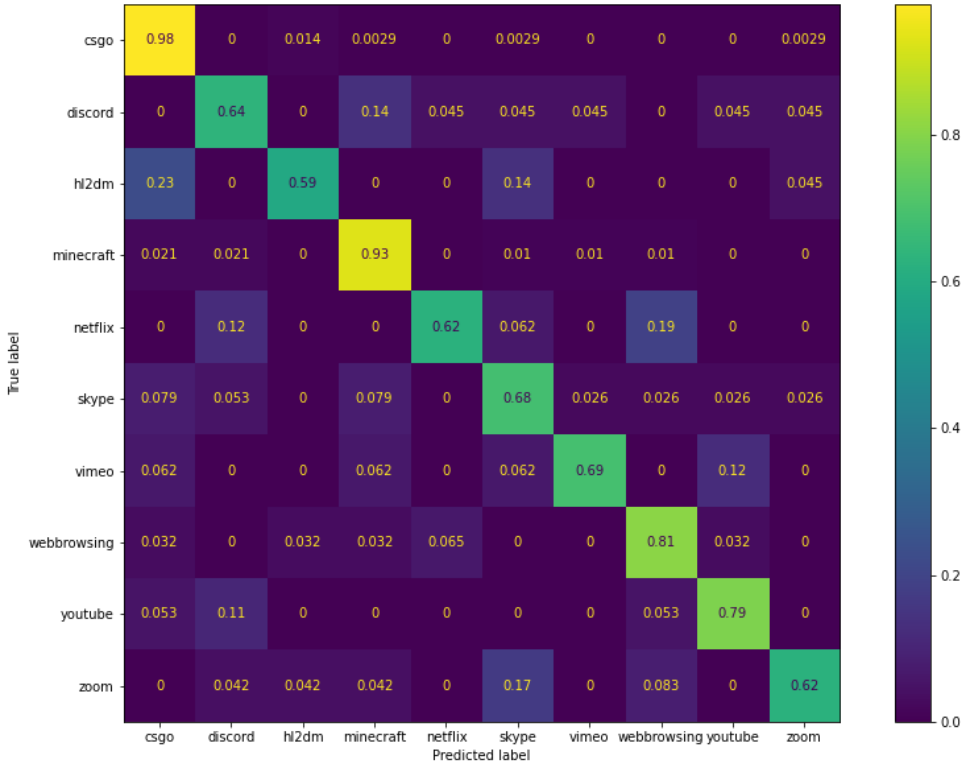

Table 44: MLP normalized confusion matrix

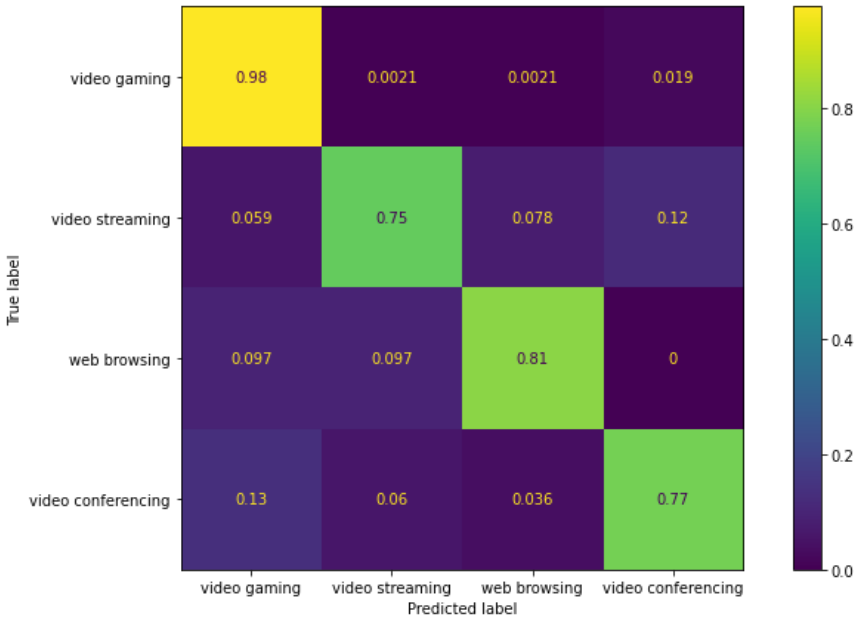

Table 45: MLP normalized broad category confusion matrix

Perhaps the most notable trait of the multi-layer perceptron classifier in this context is its generally very good performance despite the relatively limited dataset. While it does not perform as well as a random forest, it, by its and other neural network algorithms' nature, would almost certainly be better able to handle a larger quantity of more complex data. It would not be at all remarkable if this or another neural network classifier would quickly outpace the others used in this paper in a larger production network with a more elaborate and all-encompassing data labeling process.

Whereas classical machine learning models have mature, well-established, easily-accessible methods of identifying feature importance, neural network models remain somewhat immature on the subject. There are cutting-edge solutions and projects that seek to remedy this issue, but they often vary in implementation and do not always have universally accepted wisdom regarding their usage. This paper does not seek to explore neural networks, or, by extension, deep learning, in great detail, but only to touch upon its performance and usefulness, even in its simplest of forms. As such, feature importance is not discussed.

SECTION 4: CONCLUSIONS

In order of macro F1 score, the best-performing classifiers were random forest (.77), multi-layer perceptron (.75), SVM (.74), the feature-optimized decision tree (.73), the "all features" decision tree (.72), k-NN (.71), the feature-selected naïve Bayes classifier (.52), and nearest centroid (.45). The macro F1 score is likely the best, most useful metric because of the imbalanced nature of the dataset produced from a network traffic capture.

If looking at accuracy, the order of performance is the same, with the most accurate classifier, random forest, achieving 89% accuracy, and the least accurate classifier, nearest centroid, achieving 67% accuracy. This is significantly better than random guessing between ten classes (which would result in 10% accuracy scores for all classifiers). Even using macro average recall, scores are always better than random guessing, with the lowest scoring classifier, nearest centroid, achieving 52% macro average recall.

In all likelihood, the scores shown in this paper would all be improved if the dataset were larger and presented more opportunities for trends to make themselves known. Still, despite the limited dataset, the scores are very workable, and classifiers achieving these scores could prove useful in identifying network traffic. While not all the applications used in this paper have perfect, common means of identification using tools easily incorporated into a workflow, it would not be difficult to employ a similar machine learning process against traffic automatically labeled based on deep packet inspection tools. Then, statistical similarities between applications could be identified to enhance application identification beyond what deep packet inspection can provide.

One potentially useful output of this paper is a list of the most commonly important features of the dataset. While the dataset used is indeed limited, because of the variety of

applications used, its features could potentially prove useful if more broadly applied. Whether or not all these features are generally useful or are just specifically useful on this dataset would require further study. A table summarizing ten of the most common important shared features as derived from the permutation importance of the classical models is shown in Table 46.

Feature	Count	Classifiers
src2dst_max_ps	6	Random Forest, Decision Trees (all features), SVM, k-NN, Nearest Centroid, Optimized Naive Bayes
dst2src_max_piat_ms	5	Nearest Centroid, k-NN, SVM, Decision Tree (all features), Random Forest
bidirectional_max_ps	4	Nearest Centroid, Optimized Naive Bayes, SVM, Random Forest
dst2src_rst_packets	4	Nearest Centroid, k-NN, Optimized Naive Bayes, SVM
src2dst_stddev_ps	4	k-NN, Optimized Naive Bayes, SVM, Decision Tree (all features)
bidirectional_max_piat_ms	3	Nearest Centroid, k-NN, SVM
dst2src_stddev_ps	3	Nearest Centroid, SVM, Decision Tree (all features)
src2dst_duration_ms	3	Nearest Centroid, Decision Tree (all features), Random Forest
src2dst_max_piat_ms	3	Nearest Centroid, k-NN, SVM
bidirectional_mean_ps	2	k-NN, Optimized Naive Bayes

Table 46: Most common important features by classifier(s)

Another useful output is an aggregate comparison of broad category performance for the classical models in this paper (nearest centroid, k-NN, best-performing naïve Bayes, SVM, best-performing decision tree, and random forest). This comparison is useful in identifying how applications are generally misclassified in machine learning. The table below averages the results of the broad category confusion matrices for this purpose. Based on its results, it can be inferred that video gaming traffic is most commonly misclassified as video streaming traffic, video streaming traffic is most commonly misclassified as video conferencing traffic, web browsing traffic is most commonly misclassified as video streaming traffic, and video conferencing traffic is most commonly misclassified as video gaming traffic.

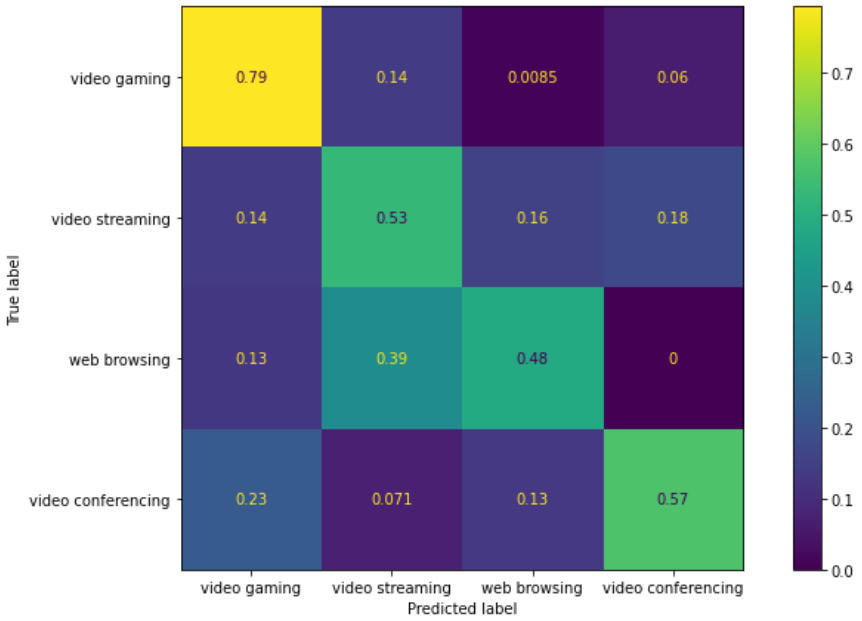

Table 47: Averaged broad category normalized confusion matrix

There are many opportunities for future work based on the results of this paper. Simply performing much the same testing on a significantly larger dataset would almost certainly be very informative. Expanding the number of applications identified to more accurately cover the scope of typical traffic on enterprise networks could reveal traffic identification opportunities and challenges. Additionally, while network traffic is almost inherently imbalanced, taking further steps to balance the dataset without compromising its legitimacy could reveal new strengths, weaknesses, and common properties of machine learning in traffic classification. A promising workflow that could be easily implemented today in a regular enterprise network is to perform network captures and then use application information derived from deep packet inspection tools, like NFStream, as the basis for training a machine learning classifier to gain enhanced visibility into a computer network.

References

Al-Obaidy, F., Momtahen, S., Hossain, M. F., & Mohammadi, F. (2019, May). Encrypted traffic classification based ml for identifying different social media applications. In *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)* (pp. 1-5). IEEE.

Aouini, Z., & Pekar, A. (2022). NFStream: A flexible network data analysis framework. *Computer Networks*, *204*, 108719.

CANIV TECH. (n.d.). *Rethink network operations with predictive analytics*. Retrieved March 1, 2023, from http://caniv-tech.com/.

Hajjar, A., Khalife, J., & Díaz-Verdejo, J. (2015). Network traffic application identification based on message size analysis. *Journal of Network and Computer Applications*, *58*, 130-143.

Moore, A. W., & Zuev, D. (2005, June). Internet traffic classification using bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (pp. 50-60).

Ntop/NDPI: Open source deep packet inspection software toolkit. (n.d.). GitHub. Retrieved March 1, 2023, from https://github.com/ntop/nDPI.

NFStream: flexible network data analysis framework. (n.d.). NFStream. Retrieved March 1, 2023, from https://www.nfstream.org/.

Palo Alto Networks. (n.d.). *Wildfire inline ML.* Retrieved March 23, 2023, from https://docs.paloaltonetworks.com/pan-os/10-1/pan-os-admin/threat-prevention/wildfire-inline-ml.

Weiss, G. M. (2013). Foundations of imbalanced learning. *Imbalanced Learning: Foundations, Algorithms, and Applications*, 13-41.

Yamansavascilar, B., Guvensan, M. A., Yavuz, A. G., & Karsligil, M. E. (2017, January). Application identification via network traffic classification. In *2017 International Conference on Computing, Networking and Communications (ICNC)* (pp. 843-848). IEEE.

Zander, S., Nguyen, T., & Armitage, G. (2005, November). Automated traffic classification and application identification using machine learning. In *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05) l* (pp. 250-257). IEEE.

Zhang, J., Xiang, Y., Zhou, W., & Wang, Y. (2013). Unsupervised traffic classification using flow statistical properties and IP packet payload. *Journal of Computer and System Sciences*, *79*(5), 573-585.