**Project Report**

CS 371 – Dr. Silvestri, Enrico Casella

Noah T. Arragaw, Brandon T. Coty, Shane T. Flanagan

**Abstract:**

This report consists of the following sections: Introduction and motivation, proposed features, implementation, experiments, and conclusion. In the introduction and motivation section we define a packet sniffer and a machine learning model which are essential tools for monitoring and predicting network activity. In the proposed features section, we specify which features we utilize to train the machine learning models and why we decided these features would be the most useful in making our predictions. In the implementation section we go into detail about the way we utilized the packet sniffer to extract the features and identify the flows to create the dataset. We then describe how we use our dataset in the machine learning models to get the results required. In the experiments section we describe the process of taking in the data so that it only contains flows from the processes that we want. In the conclusions section we will explain our findings.

**Introduction and Motivation:**

The purpose of this project is to collect network traffic data using a packet sniffer. These data points are subsequently passed into three different machine learning models to predict the type of web traffic passing through the network. The four categories utilized in this project are web browsing, video streaming, video calling, and file downloading.

A packet sniffer is a piece of software used to analyze network traffic by intercepting data flowing through a network. While it can be used with malicious intent, it is a great tool to introduce basic networking knowledge highlighting the anatomy, networking protocols, and security features of data packets. We utilized Scapy (a python library) as our packet sniffer throughout the project.

Machine learning models are created by training an algorithm on a set of sample data. Once it is trained, the model can infer the identity of new data points relying on the patterns discovered from the training data. The three models tested in this project are the decision tree, neural network, and SVM (Support Vector Machine).

**Proposed Features:**

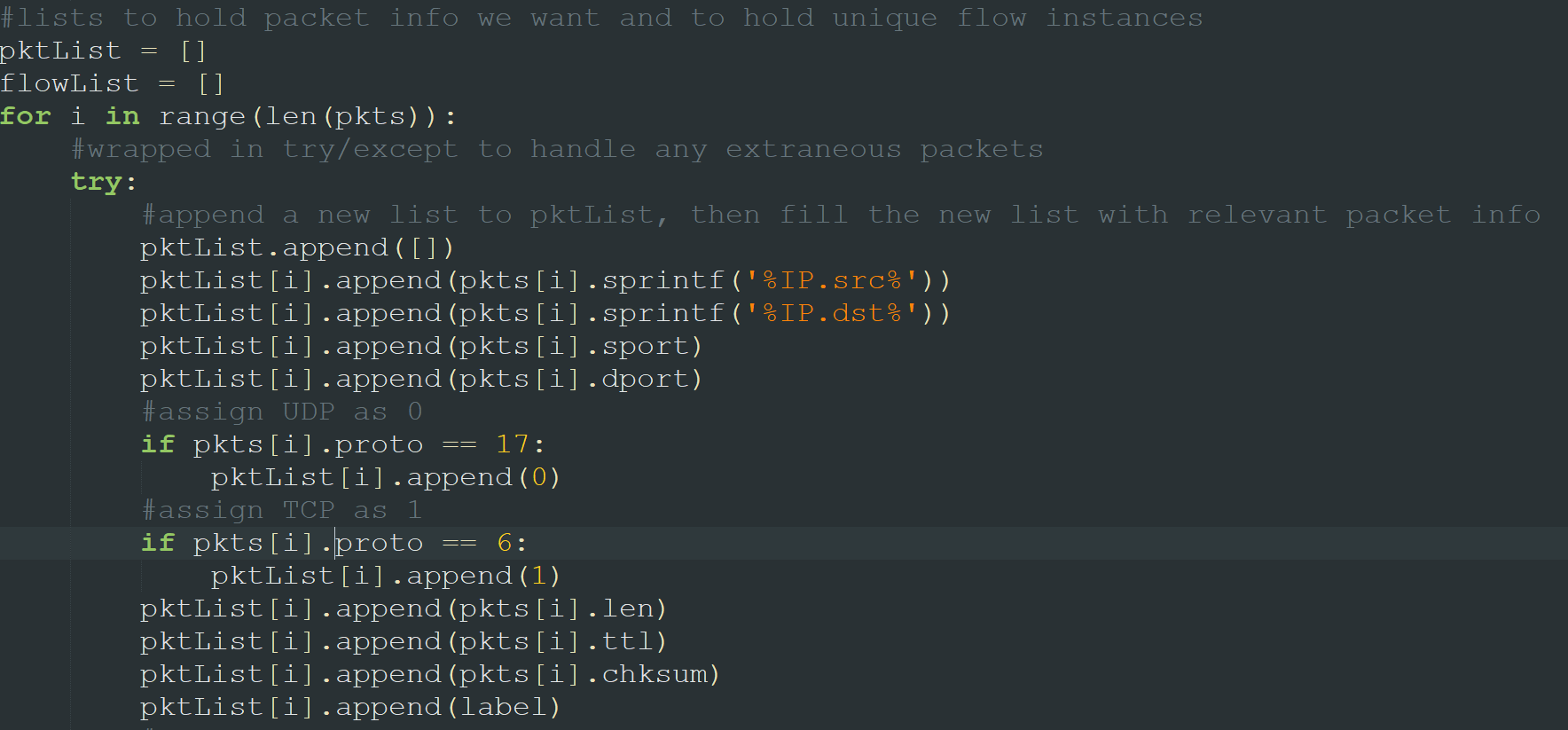
We used the protocol, packet length, checksum, and time to live (TTL) as the features in our model.

Protocol is a binary value that depicts whether a packet utilizes UDP or TCP (0 or 1 respectively). This feature was immensely useful in classifying the four web activities since both web browsing and file downloading almost exclusively use TCP while video streaming and video calling predominantly use UDP.

Packet length specifies the length (in bytes) of each packet. Analyzing this feature was helpful because each of the four types of traffic tends to have a consistent yet identifiable average packet length.

Checksum is a field used in error detection. Since the checksum appears to correlate with packet length, we felt that it would be valuable in further identifying the label of any potential packets.

Time to live (TTL) is a method of limiting a packet’s lifespan so that it doesn’t live forever on the network. We found this value to be consistent yet different between the four web activities.



This code details the process we used to make our new list of packets only including the fields we deemed valuable for training the machine learning algorithm.

**Implementation:**

Using the Scapy library, we sniff a specified number of packets (500 in the case of our final dataset) into a list. Since a lot of the packet information won’t be helpful in our machine learning models, we then iterate through the packets collected and pull out the fields we deemed necessary into a new list of packets with each of these values at a unique index. The integer values of the UDP and TCP protocols are 17 and 6 respectively, so when these values were encountered, we assigned them as 0 and 1 in the new list to be able to distinguish more easily moving forward. We ignore all the packets encountered that follow the ARP protocol since they are a part of the transport layer.

We created a “Flow” class that has a dictionary called “identifiers” with keys “IP” and “ports” that each have a list of 2 values inside of them (to hold source and destination of each). The implementation here is initializing the first packet as a new instance of the class no matter what, as we know the first packet won’t be able to be a part of a preexisting flow. Creating an instance of the class involves setting the IP destination and source of the packet as the “IP” values, and then setting the destination and source ports as the “ports” values. We also assign the ID of this first flow as 1 in an ID field in the class and add this flow to a new list to hold all our existing flows. Then, for every packet we pass in after that, we use a method called checkConnections() inside of the Flow class to compare the current packet’s source and destination IP addresses and ports to those of each flow we have so far. If the IP addresses and ports are found in a flow’s dictionary (no matter the order found) then we can say that the current packet is a part of that flow and add that flow’s ID to the list of information about that packet. If after running checkConnections() for each flow we don’t find a match, then we make a new flow using the identifiers of the current packet.

Once we have our list of packets with all our relevant information and the flow ID at the last index of each, we write these to a .CSV file with each packet on its own line. This .CSV file is then imported into the machine learning program where we specify the order of the columns in the file and which columns relate to our chosen features. Then we run the train\_test\_split() function to train the selected machine learning model (Decision tree, Neural network, or SVM). ML-skeleton.py must be executed separately for each machine learning model. Once trained, we utilize the model to predict the labels given a random selection of packet data. Following this, we calculate the accuracy, precision, recall, and F1 scores of the model. This process is repeated for a specified number of times defined by the variable testSetRange. We take the averages of the calculated values and then plot them.

**Experiments:**

To find the results for each network activity, we made sure all other foreground processes on the machine were closed. We edited the program so that we take in 500 packets at a time for each category and appended them to the .CSV file. We found that 250 for each category was a little low in terms of trying to get all the data before that cap was reached. For web browsing we opened Wikipedia and started clicking random links until we reached the 500-packet cap. For video streaming we opened a random trending video on YouTube. For video chatting we did a Facebook video call. We connected the call and closed the browser so that we did not have ads on Facebook loading in as we were taking in our packet data. Finally, for downloading we downloaded MATLAB from the UKY software page.

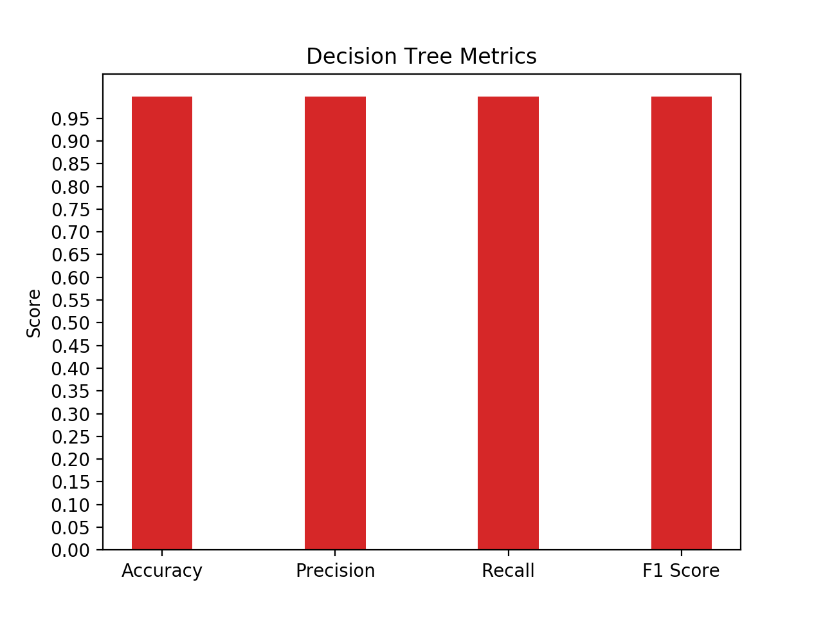
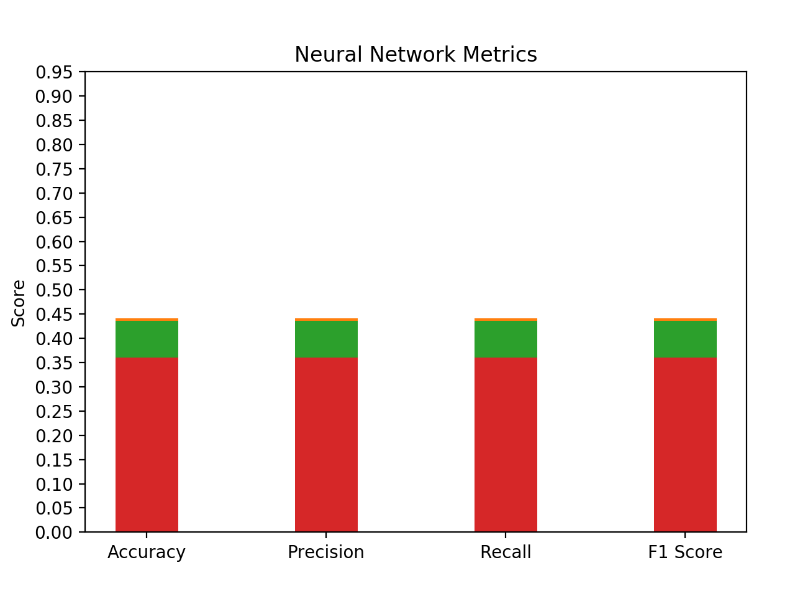
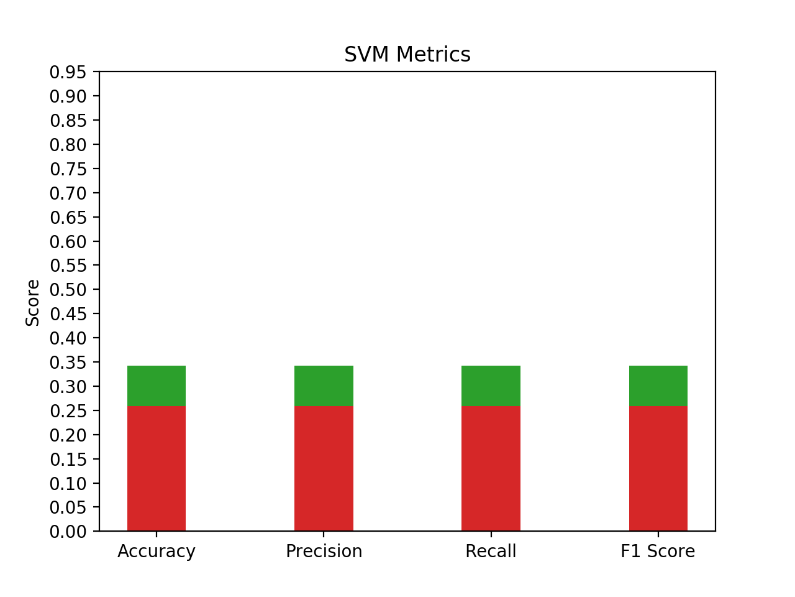
After our experiments, we graphed the data to identify trends between different network activities. One trend that we noticed was the activities that tend to use TCP have a larger packet length and checksum value on average than those that mainly use UDP. It appears that TCP uses larger packets because it is a connection-oriented and reliable service whereas UDP is unreliable. Since UDP allows for packet loss, it would be reasonable that it would utilize smaller packets.

With the time to live measurements we saw much less variance between the different activities. One observation that we made was that UDP oriented activities had larger TTL values in comparison to the TCP oriented activities. Video streaming and video chatting would likely benefit from higher TTL values because they send a higher number of low-length packets in a given timeframe. To offset the high packet-loss that is allowed when using the UDP protocol, having a higher TTL value can potentially increase the chance that the packets still make it to their destination.

One more observation is that video chatting and video streaming had a higher number of flows. Since we used YouTube and Facebook to collect the data for these activities, it is likely that the extra flows are due to processes outside of the network activity we were measuring.

**Conclusion:**

This section contains our analysis of the metrics produced by the three machine learning models. We created independent graphs for each model.

****Our decision tree model returned extremely similar (and accurate) values for each of the four required metrics. This may be attributed to the cleanliness of our collected data, but it would take more knowledge of this machine learning model to be sure. This model is by far the most successful of the three.

As with the decision tree model, the metrics produced by both the SVM and neural network models were extremely similar. The neural network was roughly 10 percentage points higher than SVM in all categories. When we produced these graphs in the machine learning program, they had some variance in color. We do not know what caused this, but it has no bearing on the validity of the data.

Initially, we took in 250 packets for each network activity. When we fed this data into the machine learning models, we found the metrics to be slightly lower across the board than they ended up being when taking in 500 packets. Even when we were taking in fewer packets, the decision tree model was still superior to the others.