Weakly Supervised Network Traffic Classification

Rui Aguiar
Stanford University
raguiar2@stanford.edu

Swathi Iyer
Stanford University
swathii@stanford.edu

1 INTRODUCTION

Accurate network traffic classification is essential for a variety of applications, such as security, quality of service, or monitoring. Traffic classification can be applied at ingress ports to separate traffic into different flows or queues, and can therefore be handled with different policies.

Different methods have been applied to classify network traffic. The most basic is to characterize them based on its source and destination ports. However, this is not too useful when services use non-standard ports. Another method is to inspect the payload of the packet. This method also has its drawbacks, in that its computationally intensive, and also impossible when traffic is encrypted. Because of these shortcomings, traffic classification has been a problem of interest to machine learning researchers, who have applied a variety of metrics from labeled flow trace data to classify network traffic. We expand on this research, taking into account the fact that much of the network traffic data in the world is unlabeled. In our approach, we use a weakly supervised method to improve on the problem domain of labeled network traffic data classification. We a statistical approach, which relies on packet features such as packet length, arrival times, etc. In particular, we generate a set of heuristics based on traffic features to massively increase the amount of labeled data and improve traffic classification accuracy.

2 RELATED WORK

Existing literature on network traffic classification generally falls into one of a few categories: port number-based methods, deep packet inspection, and statistical classification. The advantages of using source and destination port numbers is that it can be fast and low resource-consuming. It is also supported by many network devices, and does not require the application-layer payload, which protects users' privacy. However, several applications and services do not use fixed port numbers, for which this method cannot be applied [4].

Payload-based classification using deep packet inspection works by analyzing the application-level payload of the packet for characteristics unique to certain applications. This is known as Signature Analysis, where a signature is a pattern associated with an application. For example, some applications embed certain bytes or characters into the payload that allow a classification to identify them [1]. On drawback to this method include poor performance, in having to inspect the payload of each packet that comes through. Another drawback is as a classification engine builds up a reference database of signatures to check new traffic against, it also needs to maintain and periodically update this database with new applications and new developments in protocols [4].

The statistical approach is based on statistical analyses of the features of network traffic flow, such as byte frequencies, packet sizes, or packet arrival times. This can often be faster than even a port-based method. Various statistical methods have been tried, including Naive Bayes filter, K-means, and Random Forest [4]. Success of these methods can vary based on network traffic properties. For example, some authors of a Naive Bayes-based method found that some classes of traffic had much higher accuracies than others, due to the fact that there simply weren't enough examples in certain classes [9].

3 DATASET

The dataset we are using is a combination of labeled and unlabeled data. The labeled dataset was collected at the Universidad Del Cauca by doing packet captures at various times in the day. Each IP flow 87 features, including source and destination IP addresses, ports, packet arrival times, application used on that flow, etc. There are over 1 million datapoints in this original dataset. However, we limit our use to a few tens of thousands of datapoints, to balance out the amount of unlabeled data we were able to capture for this paper using the flowmeter package, which is very computationally inefficient, sometimes taking days and tens of gigabytes of RAM to generate just a few hundred datapoints. The team collected their flow statistics using CICFlowmeter, and collected the application protocol by doing a Deep Packet Inspection with ntopng.

We collected the unlabeled data in a similar fashion, by doing packet captures at various times in the day, and collecting flow statistics using a CICFLowmeter-based python tool called Flowmeter, which allowed us capture almost all of the flow statistics available in the labeled dataset captured using CICFLowmeter. We later augment our unlabeled data using heuristics for weak supervision, so did not collect the application protocols. We collected around 20K lines of unlabeled data.

4 METHODS

4.1 Baseline

Our baseline was a simple shallow MLP classifier with one hidden layer. Our input to this classifier was a vector of different discrete and continuous numeric features of a packet flow, such as source port, destination port, average/max/min packet length, average/max/min header length, flow time, etc. We ran the baseline across about 80 different L7 protocols. We ran this baseline for about 10 epochs across 20000 captured packets using the Adam optimizer with a learning rate of 0.01. Using this classifier, we achieve around 45 percent accuracy on protocol classification, which is significantly better than random guessing given the number of classes that we have, but far from the achievable SOTA. Despite the fact that classification with neural networks has been somewhat successfully applied in network traffic classification before using 1-dimensional convolutional neural networks, this was with an order of magnitude more data than we were training our model on. Because we had a relatively small number of examples, we decided that we were mis-using deep learning and potentially overfitting to our training set or otherwise not generalizing well on the prediction task that we were attempting. This realization motivated our switch to a more shallow, interpretable model (random forest) that we had seen others have past success with in our research on this task.

4.2 Random Forest

After concluding that a deep learning approach was perhaps not appropriate for the amount of data that we were using, we tried a supervised random forest approach. We set our model parameters to a max depth of 6, and the number of estimators to be 50. We did not have a cap on the maximum number of leaf nodes. The model takes in the numeric features of the labeled dataset similar to how our deep learning approach worked, and outputs a predicted L7 protocol name (e.g. Google, HTTP, etc.). We were able to get to around 80 percent accuracy with a simple random forest model, which is a significant step up from our MLP classifier, but we believed that we could still do better by introducing weak supervision.

4.3 Random Forest with Weak Supervision

Our random forest model seemed to work well for classification, so we build on top of this one for our weakly supervised model, by extending it with the unlabeled dataset. We used wireshark and tcpdump in collection with flowmeter to collect about 20k unlabeled packets (we collected far more packets with wireshark, but flowmeter was only able to analyze about 20k in total across a period of weeks), and proceeded to use heuristics from our knowledge of networking and our labeled dataset in order to approximately guess the protocol of certian packet flows with reasonable probability.

4.4 Heuristics

We used the snorkel library to write and apply our labeling functions to our dataset. Each labeling function used some information from a packet flow to make a protocol classification decision (or abstain from one because there is not enough information to be sure). We then use a voting scheme to determine what protocol to classify a packet flow as. We had a variety of labeling functions of varying strength based off of packet information gained from both our knowledge of network traffic, and heuristics from our labeled dataset. An example labeling function is shown below, and pretty much all of our heuristics follow the similar format

```
@labeling_function()
def check_dest_port_https(x):
    src_https = x['src_port'] == 443
    dst_https = x['dst_port'] == 443
    is_https_port = src_https or dst_https
    return HTTPS if is_https_port else ABSTAIN
```

Where we assign a certain protocol class based off of some feature of the dataset, or ABSTAIN from that labeling function if there is no signal.

4.4.1 *Source and Destination Port.*

The immediately obvious heuristic to classify protocol is by using the source and destination port. Several well-known network protocols such as HTTP/S are known to use port 80/443. We performed this labeling function across several protocols to classify our unlabeled traffic, the most impactful of which was HTTP/S.

4.4.2 *Packet Size.*

Another feature we looked at was max/min/average packet size. We decided that this was an important feature after performing some analysis onto our labeled dataset, where we saw that if the average packet length was 6, the flow had a greater than ninety five percent probability of being a HTTP flow. We used both the average and max packet length across a flow to label our data.

4.4.3 *Flow Duration.*

Another heursitic that we derived from looking at our dataset was that of flow duration. We found the average flow duration for each of a list of common protocols in our labeled dataset, and then applied a window around that flow duration, and triggered this labeling function from our dataset if it was within that flow duration window. Obviously, this type of function is not as indicative of protocol as, say, port number but it was still able to help with our voting scheme for numerous examples, especially those going to nonstandard ports.

4.5 Voting Scheme

To make decisions after our heuristic functions had been applied to each flow in our dataset, we used a majority voting classifier which makes decisions according to the most frequent protocol labeled across all of our heuristic functions - that is, a majority voting protocol that can be expressed for a class c over a list L as follows:

$$\begin{cases} c & count(c) \geq (count(L)/2)\&\&(c > -1) \\ -1 & else \end{cases}$$

If any given class had the majority of votes from the heuristic functions, we classified the flow as being of that protocol. If not, we abstained from classifying that packet flow. This gave us a hit rate of around 50 percent for each of the classes, meaning that we had about 10k weakly labeled data points to augment our existing dataset with. We also experimented with a weighted voting scheme - that is, if the port is 443 or 80, we know that the protocol is HTTP/S with a very high probability, so we can just classify our network as HTTP/s without looking at the other labeling functions that give less of a guarantee. We did not find a significant performance delta between this voting scheme, so we generated our graphs and results using the simple majority voter, though different voting schemes across heuristic functions could be a good extension and source of future work to determine which labeling functions captured the strongest signals from the data.

5 RESULTS

We ran our random forest model across several different dataset sizes from a tiny size off 500 to a larger size of 30 thousand data points to understand the impact of the ratio of unlabeled data to labeled data across several different ratios. The results are outlined in Figure 1, where we benchmark the same model across augmented and non-augmented datasets.

In addition to comparing models, we thought it was important to better illustrate the learning curves of our dataset to show scalability and generalizability of our random forest model when applied with a weakly supervised augmented dataset.

Finally, we benchmarked every model across a series of trials with varying amounts of data to get the average classification accuracy and included our results in figure 3.

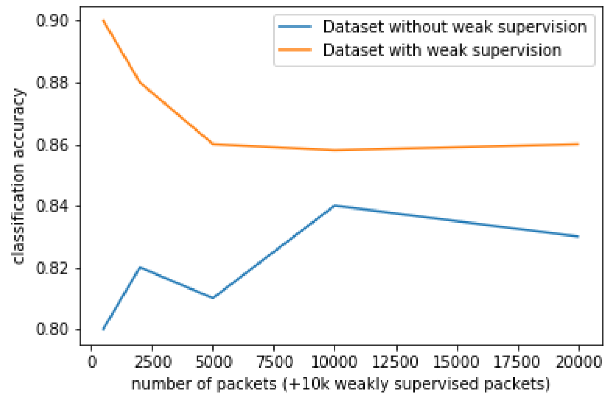

Figure 1: Performance comparison of the same classifier with weak supervision vs without

6 EVALUATION

As illustrated in figure 1, it seems that the augmentation of our weakly supervised dataset across our random forest classifier produces significantly better performance across all dataset sizes, from the unlabeled data being far larger than the amount of labeled data to a much smaller fraction of the labeled data. It seems that our heuristic labeling functions are able to actually capture a significant amount of signal from our unlabeled data, and our model can extrapolate well from this signal. If we used the full million or so lines from our dataset, assuming we have a corresponding fraction of unlabeled data, it would probably be safe to extrapolate that this type off dataset augmentation would continue to be helpful in performing protocol classification. The average accuracy of each classifier model across 5 different trials of 5 different dataset sizes is illustrated in figure 3, where we can see that the weak supervision approach outperforms both benchmarks by a significant amount.

One interesting nuance in figure 1 is that the the accuracy actually goes down as we go from a tiny dataset size off 500 labeled data points (with 100 or so in the validation set) to over 20000 labeled data points. This is likely due to the validation set size, where we end up overfitting a tiny subset of our data. In this case, the model actually is generalizing far better with more data points, though it is not reflected in an immediately obvious way with our graph.

We also performed some model analysis across our random forest classifier and weakly supervised dataset by plotting the learning curves using k-fold cross validation for our model. we can see from the learning curve in figure 2 that the generalizibilty of the model increases with the number of training examples. We can see that the training examples and cross-validation scores are beginning to converge as we

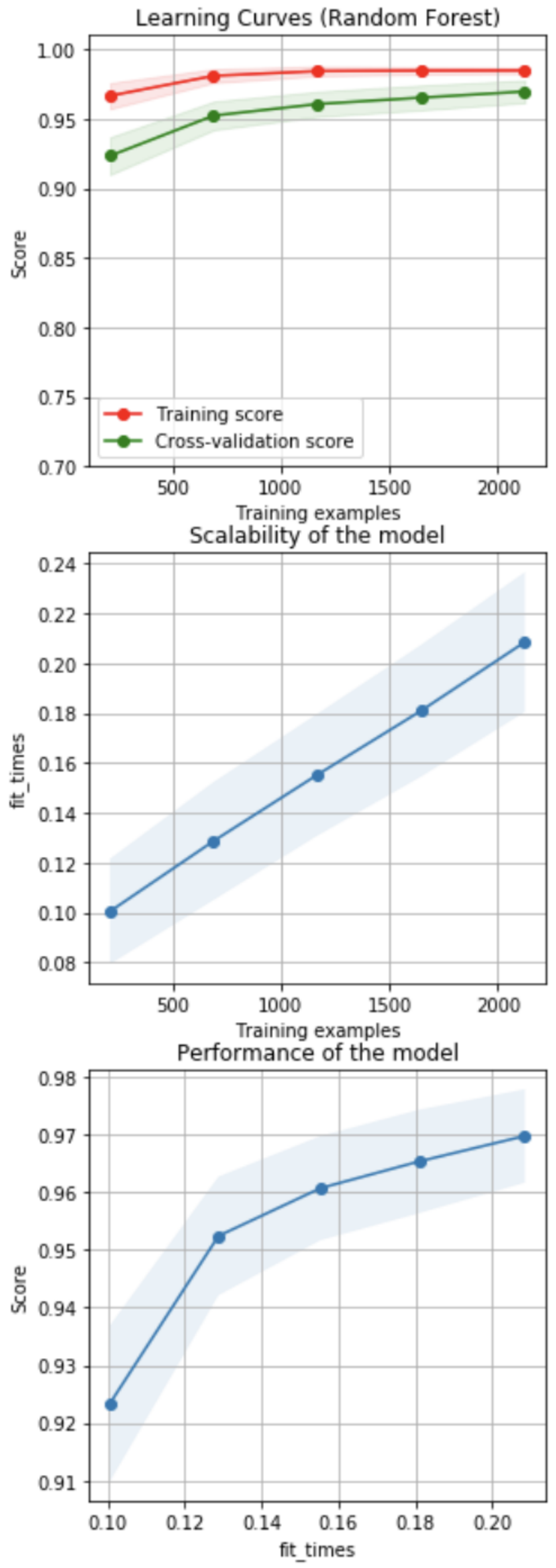

Figure 2: Learning curves of our random forest classifier with a weakly supervised dataset

Model	Accuracy
Multi-Layer Perception	0.45
Random Forest	0.82
Random Forest + weak supervision	**0.872**

Figure 3: Average classification accuracy of each of our methods

get to just a few thousand datapoints. Additionally, the scalability and performance of the model both increase with the number of training examples, indicating that adding more data (whether labeled or weakly supervised) will increase the performance of the model. Interesting followup work here would be to use a far larger dataset and see the learning curves and model generalizibility, and to compare this generalizability with a purely labeled dataset.

7 FUTURE WORK

In the future, it would be good to apply this problem framework in experimentation with different model architectures such as 1 dimensional convolutional neural networks and even just deeper MLP models to understand how deep learning can be better applied to this problem. Another piece of future work is performing more hyperparameter tuning (perhaps with a grid search) to drive accuracy as high as possible. This would also work in conjunction with a deeper understanding of the random forest model, which you could inspect to understand which features it used across the dataset to make decisions - model interpratibility in general for this problem is an interesting piece of followup work. We would also like to experiment further with heuristic selection, and see how collecting more data might affect performance. Finally, one could apply this problem at massive scale (millions or tens of millions of datapoints) with a similar ratio of unlabeled data to see how weak supervision generalizes across real traffic flows in the internet.

8 CONCLUSION

Because of the sheer quantity of unlabeled data and the ratio of unlabeled to labeled data available for any problem domain, it is a very reasonable assumption to make that unsupervised, weakly supervised and semi supervised learning will become more important as AI is applied to many more problem domains. Additionally, network traffic classification is a domain where AI methodologies including both white-box and black-box models have potential to be applicable.

In our paper, we have shown that in network traffic classification problem settings where there is a smaller quantity of labeled data, or at least a larger quantity of unlabeled data relative to the amount of labeled data, we can hand-design heuristic functions from our knowledge about networking protocols in order to actually derive a relatively strong signal from the noise of the unlabeled data. While there is still much work to be done in terms of both the actual classification methodology and approach, as well as design of heuristic functions and collection of unlabeled data at scale, we believe that we have provided a benchmark for other papers attempting similar problems in the future.

REFERENCES

[1] Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Extended independent comparison of popular deep packet inspection (dpi) tools for traffic classification.

[2] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Semi-supervised network traffic classification. *SIGMETRICS Perform. Eval. Rev.*, 35(1):369–370, June 2007.

[3] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Semi-supervised network traffic classification. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '07, page 369–370, New York, NY, USA, 2007. Association for Computing Machinery.

[4] Dr Dinesh Harkut. An overview of network traffic classification methods. 02 2015.

[5] W. John and S. Tafvelin. Heuristics to classify internet backbone traffic based on connection patterns. In *2008 International Conference on Information Networking*, pages 1–5, 2008.

[6] Antonios Minas Krasakis, Evangelos Kanoulas, and George Tsatsaronis. Semi-supervised ensemble learning with weak supervision for biomedical relationship extraction. In *AKBC*, 2019.

[7] Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel. *Proceedings of the VLDB Endowment*, 11(3):269–282, Nov 2017.

[8] Juan Rojas Meléndez, Alvaro Rendón, and Juan Corrales. Personalized service degradation policies on ott applications based on the consumption behavior of users. *Lecture Notes in Computer Science*, pages 543–557, 05 2018.

[9] Denis Zuev and Andrew W. Moore. Traffic classification using a statistical approach. In *Proceedings of the 6th International Conference on Passive and Active Network Measurement*, PAM'05, page 321–324, Berlin, Heidelberg, 2005. Springer-Verlag.