Public Review for

A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification

Nigel Williams, Sebastian Zander, and Grenville Armitrage

This work is a a nice empirical study of the use of main-stream machine learning algorithms for the classification of network traffic. As the title suggests, it is a preliminary study, and it does a good job of filling that role.

An important role of this work is to show the need for thorough comparisons between the plethora of proposed solutions for traffic classification. The machine learning techniques and their use is carefully explained that it can also serve as quick primer on supervised learning. Certainly there are other learning algorithms, other features, other performance measures, different aproaches to traffic classification, and (in general) more research that could be done. This paper is a good first attempt to create discussion and inspire future research in this direction.

Public review written by

Michalis Faloutsos

UC Riverside, USA

acm sigcomm

A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification

Nigel Williams, Sebastian Zander, Grenville Armitage
Centre for Advanced Internet Architectures (CAIA)
Swinburne University of Technology
Melbourne, Australia
+61 3 9214 {4837, 4835, 8373}
{niwilliams,szander,garmitage}@swin.edu.au

ABSTRACT

The identification of network applications through observation of associated packet traffic flows is vital to the areas of network management and surveillance. Currently popular methods such as port number and payload-based identification exhibit a number of shortfalls. An alternative is to use machine learning (ML) techniques and identify network applications based on per-flow statistics, derived from payload-independent features such as packet length and inter-arrival time distributions. The performance impact of feature set reduction, using Consistency-based and Correlation-based feature selection, is demonstrated on Naïve Bayes, C4.5, Bayesian Network and Naïve Bayes Tree algorithms. We then show that it is useful to differentiate algorithms based on computational performance rather than classification accuracy alone, as although classification accuracy between the algorithms is similar, computational performance can differ significantly.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations - *Network monitoring*; C.4 [**Performance of Systems**]: Measurement Techniques

General Terms: Algorithms, Measurement

Keywords: Traffic Classification, Machine Learning

1. INTRODUCTION

There is a growing need for accurate and timely identification of networked applications based on direct observation of associated traffic flows. Also referred to as 'classification', application identification is used for trend analyses (estimating capacity demand trends for network planning), adaptive network-based Quality of Service (QoS) marking of traffic, dynamic access control (adaptive firewalls that detect forbidden applications or attacks) or lawful interception.

Classification based on well-known TCP or UDP ports is becoming increasingly less effective – growing numbers of networked applications are port-agile (allocating dynamic ports as needed), end users are deliberately using non-standard ports to hide their traffic, and use of network address port translation (NAPT) is widespread (for example a large amount of peer-to-peer file sharing traffic is using non-default ports [1]).

Payload-based classification relies on some knowledge about the payload formats for every application of interest: protocol decoding requires knowing and decoding the payload format while signature matching relies on knowledge of at least some characteristic patterns in the payload. This approach is limited by the fact that classification rules must be updated whenever an application implements even a trivial protocol change, and privacy laws and encryption can effectively make the payload inaccessible.

Machine learning (ML) [2] techniques provide a promising alternative in classifying flows based on application protocol (payload) independent statistical features such as packet length and inter-arrival times. Each traffic flow is characterised by the same set of features but with different feature values. A ML classifier is built by training on a representative set of flow instances where the network applications are known. The built classifier can be used to determine the class of unknown flows.

Much of the existing research focuses on the achievable accuracy (classification accuracy) of different machine learning algorithms. The studies have shown that a number of different algorithms are able to achieve high classification accuracy. The effect of using different sets of statistical features on the same dataset has seen little investigation. Additionally, as different (in some cases private) network traces have been used with different features, direct comparisons between studies are difficult.

There have been no comparisons of the relative speed of classification (computational performance) for different algorithms when classifying IP traffic flows. However, within a practical IP traffic classification system, considerations as to computational performance and the type and number of statistics calculated are vitally important.

In this paper we attempt to provide some insight into these aspects of ML traffic classification. We define 22 practical flow features for use within IP traffic classification, and further reduce the number of features using Correlation-based and Consistency-based feature reduction algorithms. We confirm that a similar level of classification accuracy can be obtained when using several different algorithms with the same set of features and training/testing data. We then differentiate the algorithms on the basis of computational performance. Our key findings are:

- Feature reduction greatly reduces the number of features needed to identify traffic flows and hence greatly improves computational performance
- While feature reduction greatly improves performance it does not severely reduce the classification accuracy.
- Given the same features and flow trace, we find that different ML algorithms (Bayes Net, Naïve Bayes Tree and C4.5) provide very similar classification accuracy.

- However, the different algorithms show significant differences in their computational performance (build time and classification speed).

The paper is structured as follows. Section 2 summarises key machine learning concepts. Section 3 discusses related work, while Section 4 outlines our approach, including details on algorithms, features and datasets. Section 5 presents our main findings and in Section 6 we conclude and discuss future work.

2. MACHINE LEARNING

2.1 Machine Learning Concepts

We use machine learning algorithms to map *instances* of network traffic flows into different network traffic *classes*. Each flow is described by a set of statistical *features* and associated feature values. A feature is a descriptive statistic that can be calculated from one or more packets – such as mean packet length or the standard deviation of inter-arrival times. Each traffic flow is characterised by the same set of features, though each will exhibit different feature values depending on the network traffic class to which it belongs.

ML algorithms that have been used for IP traffic classification generally fall into the categories of being *supervised* or *unsupervised*. Unsupervised (or clustering) algorithms group traffic flows into different clusters according to similarities in the feature values. These clusters are not pre-defined and the algorithm itself determines their number and statistical nature. For supervised algorithms the class of each traffic flow must be known before learning. A classification model is built using a *training set* of example instances that represent each class. The model is then able to predict class membership for new instances by examining the feature values of unknown flows.

2.2 Feature Reduction

As previously stated, features are any statistics that can be calculated from the information at hand (in our case packets within a flow). Standard deviation of packet length, Fourier transform of packet inter-arrival times and the initial TCP window size are all valid features. As network flows can be bi-directional, features can also be calculated for both directions of the flow.

In practical IP classification tasks we need to decide which features are most useful given a set of working constraints. For instance calculating Fourier transform statistics for thousands of simultaneous flows may not be feasible. In addition, the representative quality of a feature set greatly influences the effectiveness of ML algorithms. Training a classifier using the maximum number of features obtainable is not always the best option, as irrelevant or redundant features can negatively influence algorithm performance.

The process of carefully selecting the number and type of features used to train the ML algorithm can be automated through the use of feature selection algorithms. Feature selection algorithms are broadly categorised into the *filter* or *wrapper* model. Filter model algorithms rely on a certain metric to rate and select subsets of features. The wrapper method evaluates the performance of different features using specific ML algorithms, hence produces feature subsets 'tailored' to the algorithm used.

2.3 Evaluation Techniques

Central to evaluating the performance of supervised learning algorithms is the notion of training and testing datasets. The training set contains examples of network flows from different classes (network applications) and is used to build the classification model. The testing set represents the unknown network traffic that we wish to classify. The flows in both the training and testing sets are labelled with the appropriate class a-priori. As we know the class of each flow within the datasets we are able to evaluate the performance of the classifier by comparing the predicted class against the known class.

To test and evaluate the algorithms we use *k*-fold cross validation. In this process the data set is divided into *k* subsets. Each time, one of the *k* subsets is used as the test set and the other *k*-1 subsets form the training set. Performance statistics are calculated across all *k* trials. This provides a good indication of how well the classifier will perform on unseen data. We use k=10 and compute three standard metrics:

- Accuracy: the percentage of correctly classified instances over the total number of instances.
- Precision: the number of class members classified correctly over the total number of instances classified as class members.
- Recall (or true positive rate): the number of class members classified correctly over the total number of class members.

In this paper we refer to the combination of accuracy, precision and recall using the term *classification accuracy*.

We use the term *computational performance* to describe two additional metrics: build time and classification speed. Build time refers to the time (in seconds) required to train a classifier on a given dataset. Classification speed describes the number of classification that can be performed each second.

2.4 Sampling Flow Data

System memory usage increases with the number of instances in the training/testing set [3], as does CPU usage. As the public trace files used in this study contain millions of network flows, we perform flow sampling to limit the number of flows and therefore the memory and CPU time required for training and testing (see Section 4.3).

We sample an equal number of traffic flows for each of the network application classes. Thus class prior probabilities are equally weighted. Although this may negatively impact algorithms that rely on prior class probabilities, it prevents algorithms from optimising towards a numerically superior class when training (leading to overly optimistic results). Furthermore, this allows us to evaluate the accuracy of ML algorithms based on feature characteristics without localising to particular trace-dependent traffic mixes (different locations can be biased towards different traffic classes).

3. RELATED WORK

The Expectation Maximization (EM) algorithm was used by McGregor et al. [4] to cluster flows described by features such as packet length, inter-arrival time and flow duration. Classification of traffic into generic groups (such as *bulk-transfer*, for instance) was found to be achievable.

Dunnigan and Ostrouchov [5] use principal component analysis (PCA) for the purpose of Intrusion Detection. They find that network flows show consistent statistical patterns and can be detected when running on default and non-default ports.

We have proposed an approach for identifying different network applications based on greedy forward feature search and EM in [6]. We show that a variety of applications can be separated into an arbitrary number of clusters.

Roughan et al. [7] use nearest neighbour (NN) and linear discriminate analysis (LDA) to map different applications to different QoS classes (such as *interactive* and *transactional*). They demonstrated that supervised ML algorithms are also able to separate traffic into classes, with encouraging accuracy.

Moore and Zuev [3] used a supervised Naive Bayes classifier and 248 flow features to differentiate between different application types. Among these were packet length and inter-arrival times, in addition to numerous TCP header derived features. Correlation-based feature selection was used to identify 'stronger' features, and showed that only a small subset of fewer than 20 features is required for accurate classification.

Karagiannis et al. [8] have developed a method that characterises host behaviour on different levels to classify traffic into different application types.

Recently Bernaille et al. [9] used a Simple K-Means clustering algorithm to perform classification using only the first five packets of the flow.

Lim et al. [10] conducted an extensive survey of 33 algorithms across 32 diverse datasets. They find that algorithms show similar classification accuracy but quite different training performance (for a given dataset and complementary features). They recommend that users select algorithms based on criteria such as model interpretability or training time. Classification speed of the algorithms was not compared.

4. EXPERIMENTAL APPROACH

As the main focus of this study is to demonstrate the benefit of using computational performance as a metric when choosing an ML algorithm to implement, we use a single dataset and fixed algorithm configurations. varying only the feature set used in training.

In the following sections we detail the machine learning and feature selection algorithms used in the paper. The IP traffic dataset, traffic classes, flow and feature definitions are also explained.

4.1 Machine Learning Algorithms

We use the following supervised algorithms that have been implemented in the Weka [20] ML suite:

- Bayesian Network
- C4.5 Decision Tree
- Naïve Bayes
- Naïve Bayes Tree

The algorithms used in this study are simple to implement and have either few or no parameters to be tuned. They also produce classifications models that can be more easily interpreted. Thus algorithms such as Support Vector Machines or Neural Networks were not included.

The algorithms used in this investigation are briefly described in the following paragraphs, with extended descriptions in Appendix A.

Naive-Bayes (NBD, NBK) is based on the Bayesian theorem [11]. This classification technique analyses the relationship between each attribute and the class for each instance to derive a conditional probability for the relationships between the attribute values and the class. Naïve Bayesian classifiers must estimate the probabilities of a feature having a certain feature value. Continuous features can have a large (possibly infinite) number of values and the probability cannot be estimated from the frequency distribution. This can be addressed by modelling features with a continuous probability distribution or by using discretisation. We evaluate Naive Bayes using both discretisation (NBD) and kernel density estimation (NBK). Discretisation transforms the continuous features into discrete features, and a distribution model is not required. Kernel density estimation models features using multiple (Gaussian) distributions, and is generally more effective than using a single (Gaussian) distribution.

C4.5 Decision Tree (C4.5) creates a model based on a tree structure [12]. Nodes in the tree represent features, with branches representing possible values connecting features. A leaf representing the class terminates a series of nodes and branches. Determining the class of an instance is a matter of tracing the path of nodes and branches to the terminating leaf.

Bayesian Network (BayesNet) is structured as a combination of a directed acyclic graph of nodes and links, and a set of conditional probability tables [13]. Nodes represent features or classes, while links between nodes represent the relationship between them. Conditional probability tables determine the strength of the links. There is one probability table for each node (feature) that defines the probability distribution for the node given its parent nodes. If a node has no parents the probability distribution is unconditional. If a node has one or more parents the probability distribution is a conditional distribution, where the probability of each feature value depends on the values of the parents.

Naïve Bayes Tree (NBTree) is a hybrid of a decision tree classifier and a Naïve Bayes classifier [14]. Designed to allow accuracy to scale up with increasingly large training datasets, the NBTree model is a decision tree of nodes and branches with Naïve Bayes classifiers on the leaf nodes.

4.2 Feature Reduction Algorithms

We use two different algorithms to create reduced feature sets: Correlation-based Feature Selection (CFS) and Consistency-based Feature selection (CON). These algorithms evaluate different combinations of features to identify an optimal subset. The feature subsets to be evaluated are generated using different subset search techniques. We use Best First and Greedy search methods in the forward and backward directions, explained below.

Greedy search considers changes local to the current subset through the addition or removal of features. For a given 'parent' set, a greedy search examines all possible 'child' subsets through either the addition or removal of features. The child subset that shows the highest goodness measure then replaces the parent subset, and the process is repeated. The process terminates when no more improvement can be made.

Best First search is similar to greedy search in that it creates new subsets based on the addition or removal of features to the current subset. However, it has the ability to backtrack along the subset selection path to explore different possibilities when the current path no longer shows improvement. To prevent the search from backtracking through all possibilities in the feature space, a limit is placed on the number of non-improving subsets that are considered. In our evaluation we chose a limit of five.

The following brief descriptions of the feature selection algorithms are supplemented by additional details in Appendix B.

Consistency-based feature subset search [15] evaluates subsets of features simultaneously and selects the optimal subset. The optimal subset is the smallest subset of features that can

identify instances of a class as consistently as the complete feature set.

Correlation-based feature subset search [16] uses an evaluation heuristic that examines the usefulness of individual features along with the level of inter-correlation among the features. High scores are assigned to subsets containing attributes that are highly correlated with the class and have low inter-correlation with each other.

To maintain a consistent set of features when testing each of the algorithms, wrapper selection was not used (as it creates algorithm-specific optimised feature sets). It is recognised that wrapper-generated subsets provide the upper bound of accuracy for each algorithm, but do not allow direct comparison of the algorithm performance (as features are different).

4.3 Data Traces and Traffic Classes

Packet data is taken from three publicly available NLANR network traces [17], which were captured in different years and at different locations. We used four 24-hour periods of these traces (auckland-vi-20010611, auckland-vi-20010612, leipzig-ii-20030221, nzix-ii-20000706). As mentioned in Section 2.4 we use stratified sampling to obtain flow data for our dataset. 1,000 flows were randomly and independently sampled for each class and each trace. The traces were then aggregated into a single dataset containing 4,000 flows per application class. This is referred to as the 'combined' dataset. 10-fold cross-validation is used to create testing and training sets (see Section 2.3).

We chose a number of prominent applications and defined the flows based on the well-known ports: FTP-Data (port 20), Telnet (port 23), SMTP (port 25), DNS (port 53) and HTTP (port 80). We also include the multiplayer game Half-Life (port 27015). The chosen traffic classes account for a large proportion (up to 75%) of the traffic in each of the traces. The choice of six application classes creates a total of 24,000 instances in the combined dataset.

A drawback of using anonymised trace files is the lack of application layer data, making verification of the true application impossible. As we use default ports to obtain flows samples, we can expect that the majority of flows are of the intended application. We accept however that a percentage of flows on these ports will be of different applications. Despite this the number of incorrectly labelled flows is expected to be small, as port-agile applications are more often found on arbitrary ports rather than on the default port of other applications (as found for peer-to-peer traffic in [1]). Therefore the error introduced into the training and testing data is likely to be small.

4.4 Flow and Feature Definitions

We use NetMate [18] to process packet traces, classify packets to flows and compute feature values. Flows are defined by source IP and source port, destination IP and destination port and protocol.

Flows are bidirectional and the first packet seen by the classifier determines the forward direction. Flows are of limited duration. UDP flows are terminated by a flow timeout. TCP flows are terminated upon proper connection teardown (TCP state machine) or after a timeout (whichever occurs first). We use a 600 second flow timeout, the default timeout value of NeTraMet (the implementation of the IETF Realtime Traffic Flow Measurement working group's architecture) [19]. We consider only UDP and TCP flows that have at least one packet in each direction and transport at least one byte of payload. This excludes flows without payload (e.g. failed TCP connection attempts) or 'unsuccessful' flows (e.g. requests without responses).

When defining the flow features, the 'kitchen-sink' method of using as many features as possible was eschewed in favour of an economical, constraint-based approach. The main limitation in choosing features was that calculation should be realistically possible within a resource constrained IP network device.

Thus potential features needed to fit the following criteria:

- Packet payload independent
- Transport layer independent
- Context limited to a single flow (i.e. no features spanning multiple flows)
- Simple to compute

The following features were found to match the above criteria and became the base feature set for our experiments:

- Protocol
- Flow duration
- Flow volume in bytes and packets
- Packet length (minimum, mean, maximum and standard deviation)
- Inter-arrival time between packets (minimum, mean, maximum and standard deviation).

Packet lengths are based on the IP length excluding link layer overhead. Inter-arrival times have at least microsecond precision and accuracy (traces were captured using DAG cards [17]). As the traces contained both directions of the flows, features were calculated in both directions (except protocol and flow duration). This produces a total of 22 flow features, which we refer to as the 'full feature set'. A list of these features and their abbreviations can be found in Appendix C.

Our features are simple and well understood within the networking community. They represent a reasonable benchmark feature set to which more complex features might be added in the future.

5. RESULTS AND ANALYSIS

Our ultimate goal is to show the impact of feature reduction on the relative computational performance of our chosen ML algorithms. First we identify significantly reduced feature sets using CFS and Consistency subset evaluation. Then, having demonstrated that classification accuracy is not significantly degraded by the use of reduced feature sets, we compare the relative computational performance of each tested ML algorithm with and without reduced feature sets.

5.1 Feature Reduction

The two feature evaluation metrics were run on the combined dataset using the four different search methods. The resulting reduced feature sets were then used to train and test each of the algorithms using cross-validation as described in Section 2.3. We obtain an average accuracy across the algorithms for each of the search methods. We then determine the 'best' subset by comparing the average accuracy against the average accuracy across the algorithms using the full feature set.

Figure 1 plots the average accuracy for each search method using Consistency evaluation. The horizontal line represents the average achieved using the full feature set (94.13%). The number of features in the reduced feature subset is shown in each bar.

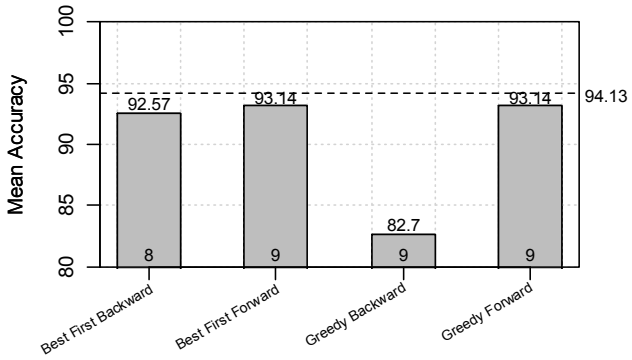

Figure 1: Consistency-generated subset accuracy according to search method

A large reduction in the feature space is achieved with relatively little change in accuracy. Two search methods (greedy forward, best first forward) produced an identical set of nine features. This set also provided the highest accuracy, and was thus determined to be the 'best' Consistency selected subset. The features selected are detailed in Table 1, referred to as CON subset in the remainder of this paper.

CFS subset evaluation produced the same seven features for each of the search methods. This is therefore the 'best' subset by default. We refer to this as the CFS subset, shown in Table 1.

Table 1: The best feature subsets according to feature reduction method

CFS subset	fpackets, maxfpktl, minfpktl, meanfpktl, stdbpktl, minbpktl, protocol
CON subset	fpackets, maxfpktl, meanbpktl, maxbpktl, minfiat, maxfiat, minbiat, maxbiat, duration

The 'best' feature sets chosen by CFS and Consistency are somewhat different, with the former relying predominantly on packet length statistics, the latter having a balance of packet length and inter-arrival time features. Only maxfpktl and fpackets are common to both reduced feature sets. Both metrics select a mixture of features calculated in the forward and backward directions.

The reduction methods provided a dramatic decrease in the number of features required, with the best subsets providing similar mean accuracies (CFS: 93.76%, CON: 93.14%). There appears to be a very good trade-off between feature space reduction and loss of accuracy.

5.2 Impact of Feature Reduction on Classification Accuracy

We examine the impact that feature reduction has on individual algorithms, in terms of accuracy, precision and recall, using the feature sets obtained in Section 5.1. Cross-validation testing is performed for each of the algorithms using the full feature set, the CFS subset and the CON subset. We obtain the overall accuracy and mean class recall/precision rates across the classes after each test. These values provide an indication as to the overall performance of the algorithm as well as the performance for individual traffic classes. Figure 2 compares the accuracy for each ML algorithm when using the CFS subset, CON subset and the full feature set.

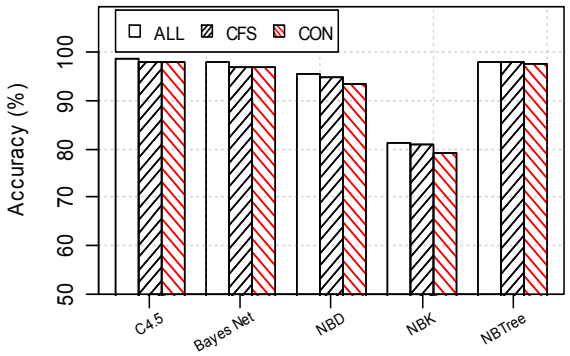

Figure 2: Accuracy of algorithms using CFS subset, CON Subset and All features.

The majority of algorithms achieve greater than 95% accuracy using the full feature set, and there is little change when using either of the reduced subsets. NBK does not perform as well as in [3], possibly due to the use of different traffic classes, features and equally weighted classes.

Figure 3 plots the relative change in accuracy for each of the algorithms compared to the accuracy using all features. The decrease in accuracy is not substantial in most cases, with the largest change (2-2.5%) occurring for NBD and NBK when using the CON subset. Excluding this case however, both subsets produce a similar change, despite the different features used in each.

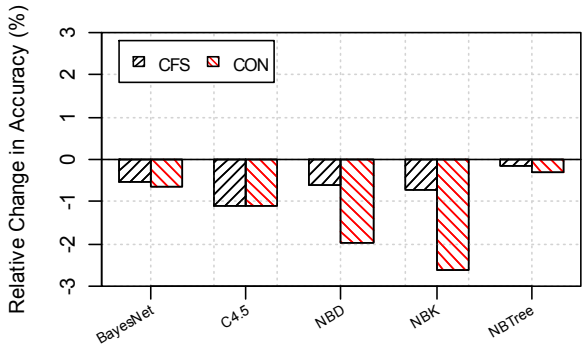

Figure 3: Relative change in accuracy depending on feature selection metric for each algorithm compared to using full feature set

Examining the mean class recall and precision rates for each of the algorithms showed a similar result to that seen with overall accuracy. The mean rates were initially high (>0.9) and remained largely unchanged when testing using the reduced feature sets.

Although the classification accuracies for each of the algorithms were quite high, they do not necessarily indicate the best possible performance for each algorithm, nor can any wider generalisation of accuracy for different traffic mixes be inferred. They do however provide an indication as to the changes in classification accuracy that might be expected when using reduced sets of features more appropriate for use within operationally deployed, high-speed, IP traffic classification systems.

Despite using subsets of differing sizes and features, each of the algorithms achieves high accuracy and mean recall/precision rates, with little variation in performance for the majority of algorithms. An interesting implication of these results is that, given our feature set and dataset, we might expect to obtain similar levels of classification accuracy from a number of

different algorithms. Though ours is a preliminary evaluation, a more extensive study [10] reached similar conclusions.

5.3 Comparing Algorithm Computational Performance

It is clearly difficult to convincingly differentiate ML algorithms (and feature reduction techniques) on the basis of their achievable accuracy, recall and precision. We therefore focus on the build time and classification speed of the algorithms when using each of the feature sets. Computational performance is particularly important when considering real-time classification of potentially thousands of simultaneous networks flows.

Tests were performed on an otherwise unloaded 3.4GHz Pentium 4 workstation running SUSE Linux 9.3. It is important to note that we have measured the performance of concrete implementations (found in WEKA) as opposed to theoretically investigating the complexity of the algorithms. This practical approach was taken to obtain some tangible numbers with which some preliminary performance comparisons could be made.

Figure 4 shows the normalised classification speed for the algorithms when tested with each of the feature sets. A value of 1 represents the fastest classification speed (54,700 classifications per second on our test platform).

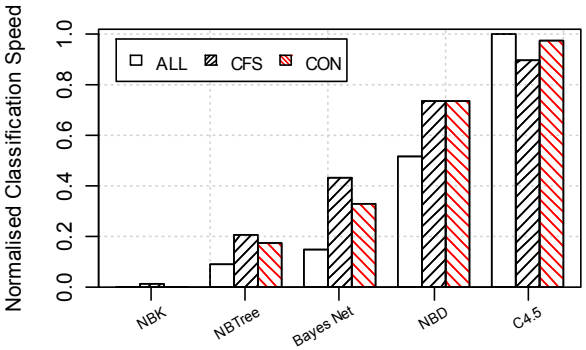

Figure 4: Normalised classification speed of the algorithms for each feature set

Although there was little separating the results when considering accuracy, classification speed shows significant differences. C4.5 is the fastest algorithm when using any of the feature sets, although the difference is less pronounced when using the CFS subset and CON subset. Using the smaller subsets provides noticeable speed increases for all algorithms except C4.5.

There are a number of factors that may have caused the reduction of classification speed for C4.5 when using the smaller subsets. It appears that decreasing the features available during training has produced a larger decision tree (more tests and nodes), thus slightly lengthening the classification time. The difference is relatively minor however, and in the context of a classification system benefits might be seen in reducing the number of features that need to be calculated.

Figure 5 compares the normalised build time for each of the algorithms when using the different feature sets. A value of 1 represents the slowest build time (1266 seconds on our test platform).

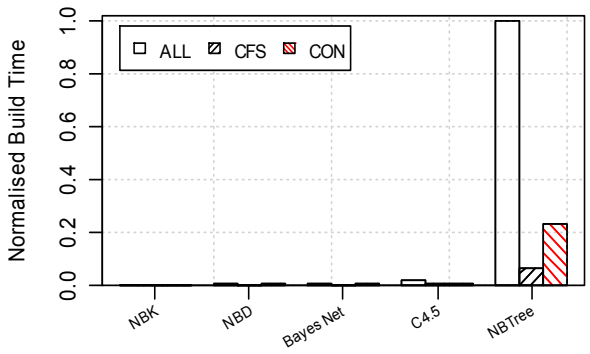

Figure 5: Normalised build time for each algorithm and feature set

It is immediately clear that NBTree takes substantially longer to build than the remaining algorithms. Also quite clear is the substantial drop in build time when using the reduced feature sets (though still much slower than the other algorithms).

Figure 6 provides a closer view of the algorithms excluding NBTree. Higher values represent lengthier build time.

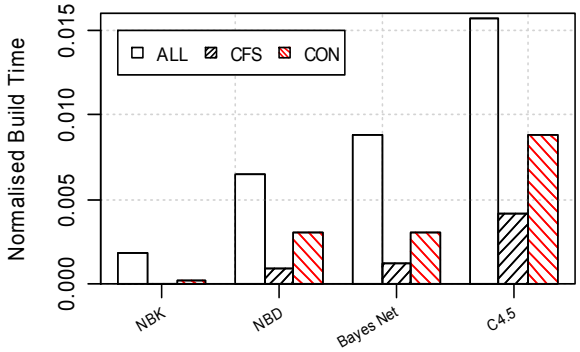

Figure 6: Normalised build time for each algorithm and feature set except NBTree

Moore and Zuev [3] found that NBK classifiers could be built quickly, and this is also the case here. NBK builds quickly as this only involves storing feature distributions (classification is slow however as probability estimates must be calculated for all distinct feature values). Bayes Net and NBD have comparable build times, while C4.5 is slower.

Overall the reduced feature sets allow for large decreases in the time taken to build a classifier. The different mix of features within the reduced subsets now appear to have some impact – the subset relying predominantly on packet length features (CFS subset) builds much faster than the subset using a mixture of packet lengths and inter-arrival times (CON subset).

One explanation for this behaviour is that there are many more possible values for inter-arrival times compared to packet length statistics (packet length statistics are more discrete). This produces a wider distribution of feature values that requires finer quantisation when training, with the increased computation leading to longer build times.

These preliminary results show that when using our features, there is a large difference in the computational performance between the algorithms tested. The C4.5 algorithm is significantly faster in terms of classification speed and appears to be the best suited for real-time classification tasks.

6. CONCLUSIONS AND FUTURE WORK

Traffic classification has a vital role in tasks as wide ranging as trend analyses, adaptive network-based QoS marking of traffic, dynamic access control and lawful interception. Traditionally performed using port and payload based analysis, recent years have seen an increased interest in the development of machine learning techniques for classification.

Much of this existing research focuses on the achievable accuracy (classification accuracy) of different machine learning algorithms. These experiments have used different (thus not comparable) datasets and features. The process of defining appropriate features, performing feature selection and the influence of this on classification and computation performance has not been studied.

In this paper we recognise that real-time traffic classifiers will operate under constraints, which limit the number and type of features that can be calculated. On this basis we define 22 flow features that are simple to compute and are well understood within the networking community. We evaluate the classification accuracy and computational performance of C4.5, Bayes Network, Naïve Bayes and Naïve Bayes Tree algorithms using the 22 features and with two additional reduced feature sets.

We find that the feature reduction techniques are able to greatly reduce the feature space, while only minimally impacting classification accuracy and at the same time significantly increasing computation performance. When using the CFS and Consistency selected subsets, only small decreases in accuracy (on average <1%) were observed for each of the algorithms. We also found that the majority of algorithms achieved similar levels of classification accuracy given our feature space and dataset, making differentiation of them using standard evaluation metrics such as accuracy, recall and precision difficult.

We find that better differentiation of algorithms can be obtained by examining computational performance metrics such as build time and classification speed. In comparing the classification speed, we find that C4.5 is able to identify network flows faster than the remaining algorithms. We found NBK to have the slowest classification speed followed by NBTree, Bayes Net, NBD and C4.5.

Build time found NBTree to be slowest by a considerable margin. The remaining algorithms were more even, with NBK building a classifier the fastest, followed by NBD, Bayes Net and C4.5.

As this paper represents a preliminary investigation, there are a number of potential avenues for further work, such as an in-depth evaluation of as to why different algorithms exhibit different classification accuracy and computational performance. In a wider context, investigating the robustness of ML classification (for instance training on a data from one location and classifying data from other locations) and a comparison between ML and non-ML techniques on an identical dataset would also be valuable. We would also like to explore different methods for sampling and constructing training datasets.

7. ACKNOWLEDGEMENTS

This paper has been made possible in part by a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley. We also thank the anonymous reviewers for their constructive comments.

8. REFERENCES

[1] T. Karagiannis, A. Broido, N. Brownlee, kc claffy, "Is P2P dying or just hiding?", In *Proceedings of Globecom*, November/December 2004.

[2] T. M. Mitchell, "*Machine Learning*", McGraw-Hill Education (ISE Editions), December 1997.

[3] A. W. Moore, D. Zuev, "Internet Traffic Classification Using Bayesian Analysis Techniques", in *Proceedings of ACM SIGMETRICS*, Banff, Canada, June 2005.

[4] A. McGregor, M. Hall, P. Lorier, J. Brunskill, "Flow Clustering Using Machine Learning Techniques", *Passive & Active Measurement Workshop*, France, April 2004.

[5] T. Dunnigan, G. Ostrouchov, "Flow Characterization for Intrusion Detection", Technical Report, Oak Ridge National Laboratory, November 2000.

[6] S. Zander, T.T.T. Nguyen, G. Armitage, "Automated Traffic Classification and Application Identification using Machine Learning", in *Proceedings of IEEE LCN*, Australia, November 2005.

[7] M. Roughan, S. Sen, O. Spatscheck, N. Duffield, "Class-of-Service Mapping for QoS: A statistical signature-based approach to IP traffic classification", in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, Italy, 2004.

[8] T. Karagiannis, K. Papagiannaki, M. Faloutsos, "BLINC: Multilevel Traffic Classification in the Dark", in *Proceedings of ACM SIGCOMM*, USA, August 2005.

[9] L. Bernaille, R. Teuxeira, I. Akodkenou, A. Soule, K. Salamatian, "Traffic Classification on the Fly", *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 2, April 2006.

[10] T. Lim, W. Loh, Y. Shih, "A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New Classification Algorithms", *Machine Learning*, volume 40, pp. 203-229, Kluwer Academic Publishers, Boston, 2000.

[11] G. H. John, P. Langley, "Estimating Continuous Distributions in Bayesian Classifiers", in *Proceedings of 11th Conference on Uncertainty in Artificial Intelligence*, pp. 338-345, Morgan Kaufman, San Mateo, 1995.

[12] R. Kohavi and J. R. Quinlan, Will Klosgen and Jan M. Zytkow, editors, "*Decision-tree discovery*", in Handbook of Data Mining and Knowledge Discovery, pp. 267-276, Oxford University Press, 2002.

[13] R. Bouckaert, "Bayesian Network Classifiers in Weka", Technical Report, Department of Computer Science, Waikato University, Hamilton, NZ 2005.

[14] R. Kohavi, "Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid", in *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, 1996.

[15] M. Dash, H. Liu, "Consistency-based Search in Feature Selection", *Artificial Intelligence*, vol. 151, issue 1-2, pp. 155-176, 2003.

[16] M. Hall, "Correlation-based Feature Selection for Machine Learning", PhD Diss. Department of Computer Science, Waikato University, Hamilton, NZ, 1998.

[17] NLANR traces: http://pma.nlanr.net/Special/ (viewed August 2006).

[18] NetMate, http://sourceforge.net/projects/netmate-meter/ (viewed August 2006).

[19] N. Brownlee, "NeTraMet & NeMaC Reference Manual", University of Auckland, http://www.auckland.ac.nz/net/Accounting/ntmref.pdf, June 1999.
[20] Waikato Environment for Knowledge Analysis (WEKA) 3.4.4, http://www.cs.waikato.ac.nz/ml/weka/ (viewed August 2006).

9. APPENDIX

9.1 Appendix A: Machine Learning Algorithms

9.1.1 Naïve Bayes

Naive-Bayes is based on the Bayesian theorem [11]. This classification technique analyses the relationship between each attribute and the class for each instance to derive a conditional probability for the relationships between the attribute values and the class. We assume that X is a vector of instances where each instances is described by attributes $\{X_1,...,X_k\}$ and a random variable C denoting the class of an instance. Let x be a particular instance and c be a particular class.

Using Naive-Bayes for classification is a fairly simple process. During training, the probability of each class is computed by counting how many times it occurs in the training dataset. This is called the prior probability $P(C=c)$. In addition to the prior probability, the algorithm also computes the probability for the instance x given c. Under the assumption that the attributes are independent this probability becomes the product of the probabilities of each single attribute. Surprisingly Naive Bayes has achieved good results in many cases even when this assumption is violated.

The probability that an instance x belongs to a class c can be computed by combining the prior probability and the probability from each attribute's density function using the Bayes formula:

$$P(C=c \mid X=x) = \frac{P(C=c)\prod_i P(X_i=x_i \mid C=c)}{P(X=x)} . \quad (1)$$

The denominator is invariant across classes and only necessary as a normalising constant (scaling factor). It can be computed as the sum of all joint probabilities of the enumerator:

$$P(X=x) = \sum_j P(C_j)P(X=x \mid C_j) . \quad (2)$$

Equation 1 is only applicable if the attributes X_i are qualitative (nominal). A qualitative attribute takes a small number of values. The probabilities can then be estimated from the frequencies of the instances in the training set. Quantitative attributes can have a large number (possibly infinite) of values and the probability cannot be estimated from the frequency distribution. This can be addressed by modelling attributes with a continuous probability distribution or by using discretisation. We evaluate Naive Bayes using both discretisation (NBD) and kernel density estimation (NBK). Discretisation transforms the continuous features into discrete features, and a distribution model is not required. Kernel density estimation models features using multiple (Gaussian) distributions, and is generally more effective than using a single (Gaussian) distribution.

9.1.2 C4.5 Decision Tree

The C4.5 algorithm [12] creates a model based on a tree structure. Nodes in the tree represent features, with branches representing possible values connecting features. A leaf representing the class terminates a series of nodes and branches. Determining the class of an instance is a matter of tracing the path of nodes and branches to the terminating leaf. C4.5 uses the 'divide and conquer' method to construct a tree from a set S of training instances. If all instances in S belong to the same class, the decision tree is a leaf labelled with that class. Otherwise the algorithm uses a test to divide S into several non-trivial partitions. Each of the partitions becomes a child node of the current node and the tests separating S is assigned to the branches.

C4.5 uses two types of tests each involving only a single attribute A. For discrete attributes the test is A=? with one outcome for each value of A. For numeric attributes the test is $A \le \theta$ where θ is a constant threshold. Possible threshold values are found by sorting the distinct values of A that appear in S and then identifying a threshold between each pair of adjacent values. For each attribute a test set is generated. To find the optimal partitions of S C4.5 relies on greedy search and in each step selects the test set that maximizes the entropy based gain ratio splitting criterion (see [12]).

The divide and conquer approach partitions until every leaf contains instances from only one class or further partition is not possible e.g. because two instances have the same features but different class. If there are no conflicting cases the tree will correctly classify all training instances. However, this over-fitting decreases the prediction accuracy on unseen instances.

C4.5 attempts to avoid over-fitting by removing some structure from the tree after it has been built. Pruning is based on estimated true error rates. After building a classifier the ratio of misclassified instances and total instances can be viewed as the real error. However this error is minimised as the classifier was constructed specifically for the training instances. Instead of using the real error the C4.5 pruning algorithm uses a more conservative estimate, which is the upper limit of a confidence interval constructed around the real error probability. With a given confidence CF the real error will be below the upper limit with 1-CF. C4.5 uses subtree replacement or subtree raising to prune the tree as long as the estimated error can be decreased.

In our test the confidence level is 0.25 and the minimum number of instances per leaf is set to two. We use subtree replacement and subtree raising when pruning.

9.1.3 Bayesian Networks

A Bayesian Network is a combination of a directed acyclic graph of nodes and links, and a set of conditional probability tables. Nodes represent features or classes, while links between nodes represent the relationship between them.

Conditional probability tables determine the strength of the links. There is one probability table for each node (feature) that defines the probability distribution for the node given its parent nodes. If a node has no parents the probability distribution is unconditional. If a node has one or more parents the probability distribution is a conditional distribution where the probability of each feature value depends on the values of the parents.

Learning in a Bayesian network is a two-stage process. First the network structure B_s is formed (structure learning) and then probability tables B_p are estimated (probability distribution estimation).

We use a local score metric to form the initial structure and refine the structure using K2 search and the Bayesian Metric [20]. An estimation algorithm is used to create the conditional probability tables for the Bayesian Network. We use the Simple Estimator, which estimates probabilities directly from the dataset [13]. The simple estimator calculates class membership probabilities for each instance, as well as the conditional

probability of each node given its parent node in the Bayes network structure.

There are various other combinations of structure learning and search technique that can be used to create Bayesian Networks.

9.1.4 Naïve Bayes Tree

The NBTree [14] is a hybrid of a decision tree classifier and a Naïve Bayes classifier. Designed to allow accuracy to scale up with increasingly large training datasets, the NBTree algorithm has been found to have higher accuracy than C4.5 or Naïve Bayes on certain datasets. The NBTree model is best described as a decision tree of nodes and branches with Bayes classifiers on the leaf nodes.

As with other tree-based classifiers, NBTree spans out with branches and nodes. Given a node with a set of instances the algorithm evaluates the 'utility' of a split for each attribute. If the highest utility among all attributes is significantly better than the utility of the current node the instances will be divided based on that attribute. Threshold splits using entropy minimisation are used for continuous attributes while discrete attributes are split into all possible values. If there is no split that provides a significantly better utility a Naïve Bayes classifier will be created for the current node.

The utility of a node is computed by discretising the data and performing 5-fold cross validation to estimate the accuracy using Naïve Bayes. The utility of a split is the weighted sum of the utility of the nodes, where the weights are proportional to the number of instances in each node. A split is considered to be significant if the relative (not the absolute) error reduction is greater than 5% and there are at least 30 instances in the node.

9.2 Appendix B: Subset Evaluation Metrics

9.2.1 Consistency

The consistency-based subset search algorithm evaluates subsets of features simultaneously and selects the optimal subset. The optimal subset is the smallest subset of features that can identify instances of a class as consistently as the complete feature set.

To determine the consistency of a subset, the combination of feature values representing a class are given a pattern label. All instances of a given pattern should thus represent the same class. If two instances of the same pattern represent different classes, then that pattern is deemed to be inconsistent. The overall inconsistency of a pattern p is:

$$IC(p) = n_p - c_p \tag{3}$$

where n_p is the number of instances of the pattern and c_p the number of instances of the majority class of the n_p instances. The overall inconsistency of a feature subset S is the ratio of the sum of all the pattern inconsistencies to the sum of all the pattern instances n_S:

$$IR(S) = \frac{\sum_p IC(p)}{n_s}. \tag{4}$$

The entire feature set is considered to have the lowest inconsistency rate, and the subset most similar or equal to this is considered the optimal subset.

9.2.2 Correlation-based Feature Selection

The CFS algorithm uses an evaluation heuristic that examines the usefulness of individual features along with the level of inter-correlation among the features. High scores are assigned to subsets containing attributes that are highly correlated with the class and have low inter-correlation with each other.

Conditional entropy is used to provide a measure of the correlation between features and class and between features. If $H(X)$ is the entropy of a feature X and $H(X|Y)$ the entropy of a feature X given the occurrence of feature Y the correlation between two features X and Y can then be calculated using the symmetrical uncertainty:

$$C(X \mid Y) = \frac{H(X) - H(X \mid Y)}{H(Y)}. \tag{5}$$

The class of an instance is considered to be a feature. The goodness of a subset is then determined as:

$$G_{subset} = \frac{k\overline{r_{ci}}}{\sqrt{k + k(k-1)\overline{r_{ii}}}} \tag{6}$$

where k is the number of features in a subset, $\overline{r_{ci}}$ the mean feature correlation with the class and $\overline{r_{ii}}$ the mean feature correlation. The feature-class and feature-feature correlations are the symmetrical uncertainty coefficients (Equation 5).

9.3 Appendix C: Table of Features

Feature Description	Abbreviation
Minimum forward packet length	minfpktl
Mean forward packet length	meanfpktl
Maximum forward packet length	maxfpktl
Standard deviation of forward packet length	stdfpktl
Minimum backward packet length	minbpktl
Mean backward packet length	meanbpktl
Maximum backward packet length	maxbpktl
Standard deviation of backward packet length	stdbpktl
Minimum forward inter-arrival time	minfiat
Mean forward inter-arrival time	meanfiat
Maximum forward inter-arrival time	maxfiat
Standard deviation of forward inter-arrival times	stdfiat
Minimum backward inter-arrival time	minbiat
Mean backward inter-arrival time	meanbiat
Maximum backward inter-arrival time	maxbiat
Standard deviation of backward inter-arrival times	stdbiat
Protocol	protocol
Duration of the flow	duration
Number of packets in forward direction	fpackets
Number of bytes in forward direction	fbytes
Number of packets in backward direction	bpackets
Number of bytes in backward direction	bbytes