CoCoST: A Computational Cost Sensitive Classifier

Liyun Li*, Umut Topkara*, Baris Coskun† and Nasir Memon*
*Computer Science and Engineering Dept.
†Electrical and Computer Engineering Dept.
Polytechnic Institute of NYU
6 Metrotech Center, Brooklyn, NY
{liyun, topkara, baris}@isis.poly.edu , memon@nyu.edu

***Abstract*—Computational cost of classification is as important as accuracy in on-line classification systems. The computational cost is usually dominated by the cost of computing implicit features of the raw input data. Very few efforts have been made to design classifiers which perform effectively with limited computational power; Instead, feature selection is usually employed as a pre-processing step to reduce the cost of running traditional classifiers. We present CoCoST, a novel and effective approach for building classifiers which achieve state-of-the-art classification accuracy, while keeping the expected computational cost of classification low, even without feature selection. CoCost employs a wide range of novel cost-aware decision trees, each of which is tuned to specialize in classifying instances from a subset of the input space, and judiciously consults them depending on the input instance in accordance with a cost-aware meta-classifier. Experimental results on a network flow detection application show that, our approach can achieve better accuracy than classifiers such as SVM and random forests, while achieving 75%-90% reduction in the computational costs.**

***Keywords*-Cost Efficient Decision Tree, Suppressed Cost, Inverse-Boosting, Meta-Classifier**

I. Introduction

Machine learning classifiers are widely used in on-line real-time scenarios, such as medical diagnosis [1], network intrusion detection [2][3][4][5], spam filters [6], and recommender systems [7][8]. A major concern for these classification tasks, besides accuracy, is the computational cost of making a decision, i.e. price, CPU time, storage, and/or power use. In most of these applications, the online computational cost, or the test cost, is much more important than the model building cost(training cost) because the classifiers can be built offline while classification tasks must be accomplished online. In this paper, we present an effective machine learning classifier, CoCoST, which can achieve not only high classification accuracy, but also low expected classification computational cost, thereby enabling effective use of classifiers for the above practical scenarios. The basic idea is that CoCost employs a wide range of novel cost-aware decision trees, each of which is tuned to specialize in classifying instances from a subset of the input space. For instance, some trees are specialized in classifying easy-to-classify instances without using costly features, whereas some trees focus on hard-to-classify instances without any computational concerns. Finally, CoCost judiciously consults these trees depending on the input instance in accordance with a cost-aware meta-classifier.

The computational cost of running a classifier online has two major contributors: i) information extraction from the unseen test instance, e.g. lab tests, parsing, remote database lookups, etc. ii) running the decision function or algorithm using the test and training information. Pre-computing models or summaries of the training information helps reduce the cost of the latter contributor. Since the unseen instances do not yield to pre-computation as much, feature extraction remains as the bottleneck of computational cost of on-line classification. In order to fit the feature extraction costs to a computational budget, it is common to employ feature selection which chooses a suitable subset of the features to be used with a traditional classification method. The number of features or the total budget of the feature extraction is an adjustable parameter in feature selection which we can use to tune the trade-off between the accuracy and cost of classification. As the number of selected features gets smaller, the computational cost of feature extraction gets lower, and depending on the information content of the dismissed features, classification accuracy gets lower as well.

Traditional classification algorithms commonly require the extraction of a fixed set of features for all the training and testing instances, and in order to operate under computational budget limitations, they need to employ feature selection to reduce the size of this feature set. CoCoST adapts an alternative approach to cost budgeting: CoCoST examines the complete set of features for all instances during classifier training, which results in a classifier that judiciously chooses a unique set of features to be extracted for each unique test instance, thereby minimizes the expected cost of classification while still achieving the best accuracy. Instead of following the one-size-fits-all feature extraction, CoCoST creates a specific feature set for each test instance; note that the union of all the feature sets in CoCost may cover all of the initial set of features for the given data. CoCost achieves this unique ability by employing a range of decision trees under the control of a meta-classifier and

extracting the required features as the tree nodes are visited during classification. Some of the paths from the root to the decision leaves do not contain all the features, and the instances that are labeled using such decision paths will have lower classification cost. As a result, CoCoST can achieve high classification accuracy and still keep the expected computational cost for each instance low. In particular, we tested our algorithm under a practical scenario of network flow data type detection [9]. Our classifier achieves the accuracy of the state-of-the-art SVM classifier with only 10%-25% of its cost.

Prior Work:
Decision trees are often associated with costs. There are generally two kinds of cost in decision tree classifiers, misclassification cost and test cost. Misclassification cost occurs where there is penalty for making mistakes in classification. Test cost refers to the cost incurred by feature extraction and measurement, and is usually measured in computing time, storage or power use. In particular, we are interested in building a decision tree that has the smallest expected computational test cost.

Earlier work [10][11][12] focused on minimizing the error weighted by misclassification cost, where the cost refers to the penalty incurred by misclassification. There is also research work [13][14][1] aiming to construct tree classifiers with low worst or average testing cost. Most of them are based on single ID-3 like tree classifiers. The heuristic is usually in the form of the entropy gain over the feature cost.

One limitation in these existing test cost efficient trees is that, in building the tree, it is not possible to change the sensitivity to cost because the frequency information of each node(the fraction of instances in each node) is not taken into consideration. However in practice, we might want the tree features to be cheap at the root where all the instances are tested, while we are not very sensitive to cost when at the bottom level where only a small portion of the instances are tested. To solve this problem, we originated the "Suppressed Cost" heuristic, where we can change the sensitivity to cost through tuning one parameter. In this way, we are able to build tree classifiers that are very sensitive to cost at the root and become less sensitive as the tree grows to the bottom and the frequency of each node gets smaller.

Individual tree classifiers are not very accurate and are susceptible to problems such as overfitting [15]. Various techniques are developed to improve the accuracy of tree classifiers [16]. AdaBoost [17], is a meta-algorithm that is constructed from various individual trees. The idea behind the adaptive boosting technique is to combine diversified weak hypotheses into a more accurate hypothesis. Breiman originated the meta-classifier Random Forest [18], which performs better than Adaboost on noisy data by utilizing bagging data and random feature selection. Both of these meta-classifiers are trying to combine different weak classifiers into a more robust and accurate hypothesis. There are also research papers that try to combine tree classifiers with genetic algorithms [13], so that a more diversified pool of potential good decision trees could be obtained.

Contributions:
The main contribution of this paper is three-fold. Firstly, by taking into consideration the frequency of each node(i.e. nodes with fewer data instances passing through them are allowed to contain more expensive features), and looking forward in building a decision tree, we propose a new decision tree algorithm, called one step look-ahead tree with 'Suppressed Cost'(abbreviated as 'LASC Tree' in the rest of paper), as our base weak classifier. Experimental results show that, by taking into account the frequency information of each node and expanding the search space by looking ahead, an increase in efficiency can be expected with high probability. Secondly, to train different tree classifiers specialized at different target classes, we extend the technique of inverse boosting by taking the test cost into account, which actually provides inverse boosting with a new meaning for practice. Inverse boosting provide specialized trees which tend to use cheap features to classify the easy instances. Finally, by combining inverse-boosted trees and standard Adaboost trees using stacked generalization, where the decisions of each tree on some validation data become features for the new meta-classifier, we obtain a meta-classifier with high accuracy and low cost. This technique of building meta-classifiers over various weak classifiers that are specialized at different targets, provides a general approach to construct better meta-classifiers.

The rest of the paper is organized as follows: In section II, we describe some existing cost efficient decision tree classifiers. In section III, we discuss in detail how to build the base classifiers using the suppressed cost, how to boost and inverse-boost those base classifiers to get specialized trees, and finally the way we combine those classifiers into a meta-classifier. In section IV, experimental results are presented and we compare performances of different base classifiers and meta-classifiers. Conclusions and limitations are discussed in section V.

II. Preliminaries and Background

Decision tree classifiers (DTC) are unstable and diverse classifiers that are easy to implement [19][20]. The 'support' of a decision tree, where we call 'frequency', is defined as the expected fraction of instances coming to a node. For convenience and without loss of generality, in this paper, we only discuss binary split decision tree because it is proved that any decision tree with multiple split can be uniquely transformed to an equivalent binary split decision tree [20]. In this section, we first describe feature computational cost and the expected computational cost for a decision tree. Then we give a brief description to the cost-efficient variants of the C4.5 trees by Quinlan[16].

A. Feature Computational Cost

Computational cost associated with the features can be measured in computing power, memory occupation and cpu time. It is common that a more computationally expensive feature usually provides more discrimination power in classifying ambiguous cases than a cheap feature[1]. The question is whether the high computational price paid for this expensive feature justifies the gain in discrimination power[1]. In reality, asymmetry usually exists in the instances. Some instances are easy to classify while others might be hard to differentiate. This phenomenon illuminates that we can choose cheap features to classify the easy instances, and put more computing power into those ambiguous instances where expensive features provide more discrimination information. With this approach, tree classifiers have an implicit advantage as decision trees are naturally hierarchical. The decisions given by a tree classifier are incremental while other state-of-the-art classifiers, such as Support Vector Machine(SVM), gives a decision only after calculating all the feature values of each instance. In this sense, tree classifiers are internally more efficient while still maintaining reasonable accuracy.

The expected testing cost for a decision tree can be defined as:

$$Cost(T) = \frac{\sum_{i=1}^{N} PathCost(i)}{N} \tag{1}$$

where $PathCost(i)$ is the total computational cost of one instance incurred along its path to the leaf and N is the total number of instances. If all the feature costs are independent and there is no overlapping of the feature costs, the $PathCost(i)$ can be written as:

$$PathCost(i) = \sum_{j \in \{x_{k_j}\}} c(j) \tag{2}$$

where $c(j)$ is computational cost associated with feature j, and the sum is taken over all the k_j features $x_{k_1}, ..., x_{k_j}$ along the path.

B. Cost Efficient Variants of C4.5 Trees

The ID3 and its successor C4.5 tree families[21][16] take an information theoretic approach to choose the feature for splitting. They use Shannon's Entropy, defined as: $H = -\sum_i p_i log_2 p_i$ to design the heuristic. In ID3, the feature that yields the **maximum** information-gain between the parent and its children, defined as: $I_k(C_k; X_k) = H(C_k) - H(C_k|X_k)$, is chosen as the feature in that node. In C4.5, the GainRatio, which is the entropy gain normalized by the entropy of the feature, is used as the split criteria. In both ID3 and C4.5, the trees are built in a top-down approach.

There are variants of C4.5 trees which take attribute costs into consideration. Examples are EG2[14], CS-ID3[21], and IDX[21]. Most of these cost efficient tree classifiers apply a greedy algorithm and use heuristics to find the optimal feature to use at each node. The feature that gives the **maximum** heuristic value is chosen as the feature of the current node. The EG2 uses heuristic:

$$\frac{2^{\triangle I_i} - 1}{(c(i)+1)^{\omega}}, \tag{3}$$

where $c(i)$ is the cost of the feature, ω is a constant parameter and $\triangle I_i$ is the entropy gain the same as the entropy gain in C4.5. The heuristic used in building CS-ID3 tree is: $\frac{\triangle {I_i}^2}{c(i)}$. And the IDX tree uses a similar heuristic: $\frac{\triangle I_i}{c(i)}$.

One common limitation among those trees is that they do not take the expected number of instances at each node into consideration. But in reality, it is highly possible that we have different cost preferences at nodes with different sizes. At the root node(i.e. the biggest node), where each unseen instance needs to be tested against the root feature, we want the root feature to be very cheap and efficient. At a deep level node, where few instances enter, we can tolerate using features with expensive cost for a better discrimination power. Therefore, we need a new heuristic, which takes into account both the original feature cost and the frequency information (how many of the examples are using this feature). Then using this heuristic, we can build better cost efficient tree classifiers. In the next section, we will present a new heuristic using our originated "Suppressed Cost", where the sensitivity of the cost is suppressed as the tree grows to the bottom leaves.

C. Bagging and Boosting

As individual trees may not be accurate, meta-classifiers are built to improve the generalization accuracy. Among those meta-classifiers, bagging and boosting are the most popular and effective ones. They are meta-classifiers constructed from a pool of individual classifiers[22]. Brieman's Random Forest[18] applies the bagging method, where each individual tree is constructed from N examples sampled with replacement from the original training data. Because of randomization, it is possible that the same instance might be sampled more than once in building the decision tree. A decision forest is the combination of these decision trees, and the final decision for one unseen instance is the class that wins the most votes from the unit vote of the individual trees. Random Forest has many advantages, such as robustness and insensitivity to overfitting[18]. And the accuracy on those out-of-bag data gives an unbiased estimation of the prediction accuracy.

Another meta-classifier is boosting forest[17], which Breiman called 'the best off-the-shelf classifier in the world'. Initially in boosting, every instance has an equal weight. After a classifier is built in one iteration, the weight for each instance is updated and in the next iteration the classifier will be constructed from the new distribution of

data. Misclassified instances get more weight in the next iteration and the correctly classified instances are given less weight. Boosting is very successful in binary classification, and can also be applied to multiple classes.

III. CoCoST: Building the Universal Classifier from the Specialized

CoCost is a novel cost-sensitive classifier that combines cost-aware decision trees, each of which specializes in cost-efficiently classifying a subset of the input space, with a cost-aware meta-classifier to achieve high accuracy and low expected classification cost on the overall input space.

There are three key concepts in building CoCoST classifier: **Suppressed Cost**, **Inverse Boosting** and **Stacked-Generalization** for the meta-learning. We describe each concept in detail in this section.

A. The Base Classifier: LASC Tree–Look Ahead Tree with Suppressed Cost

LASC trees have a greedy tree construction algorithm like most traditional decision trees. There are two discriminating properties in LASC trees as the name implies: i) Suppressed Cost Heuristic, ii) Look Ahead Entropy Gain. Supressed Cost brings into consideration the estimate of how often a decision tree node will be visited during classification of a large number of test instances. It is gathered from the number of training instances that are being considered during the recursive greedy tree construction step. Look Ahead Entropy Gain enables us to find possible combination of features that provides the largest discriminating power.

The suppressed cost heuristic is defined as:

$$H = \frac{\triangle I}{freq^{\alpha}C + (1 - freq^{\alpha})} \tag{4}$$

.

Here, $\triangle I$ is the entropy gain of the possible component that is being considered, and C is the normalized cost of extracting it. $freq$ is the fraction of training instances that have followed path from root of the decision tree to the current node. The exponent α is the parameter for controlling the sensitivity of the heuristic to feature extraction cost. We will first discuss how Suppressed Heuristic works, and then describe how Look Ahead Entropy is used to improve the trees.

Suppressed Cost

The most important feature for our tree construction heuristic is that its sensitivity to feature extraction cost is "suppressed" when we are adding nodes to the tree that are far from the root node. It is built on the observation that the expected number of instances that will be processed by an internal node decreases as we move down the tree, hence more expensive features are allowable.

In Equation 4, for any fixed α, when we are near the root level and $freq$ is almost 1, $freq^{\alpha}C$ will be dominating while $1 - freq^{\alpha}$ will be near zero. Therefore, the most efficient combination of features will be chosen at the root levels. As the tree grows, the instances remaining at the newly-grown nodes are fewer. The $freq^{\alpha}$ will become smaller, resulting in less influence from the feature extraction cost on the choice of features. Meanwhile, the item $1-freq^{\alpha}$ will get larger. Noting that this item is a constant with respect to the cost, the larger this item is, the less sensitive the heuristic will be. At the bottom nodes towards the leaves where $freq$ is almost zero, the Suppressed Cost heuristic will just select the features that provide the biggest entropy gain, i.e. the features and/or leaves that provide the largest discriminating power, which is commonly used in construction of ordinary decision trees.

The parameter α is to control the speed that the effect of cost is "suppressed". The smaller this parameter is, the slower the sensitivity to cost is 'suppressed'. Take two extreme cases for example. If α is very small, say 0.01, even a very small frequency value will still yield a big value of the item $freq^{\alpha}$, which makes the item $freq^{\alpha}C$ more important than the item $1-freq^{\alpha}$ and results in a very sensitive choice of features due to the cost C. If α is large, say 10, even the frequency is near 1, the exponent item $freq^{\alpha}$ will be very small and cause the effect of the cost C to be much less significant. It has been observed that, for a large α, after the root node, the heuristic H will choose the features that give the maximum information gain because the denominator will be almost constant.

$\triangle I$: The Look Ahead Entropy Gain for all possible components in building LASC

In C4.5, the calculation of entropy gain in the heuristic only involves one node. However in LASC, as we enlarged our search space significantly by exhaustively looking ahead one step, the possible component chosen by the heuristic may consist of two levels of nodes, as shown in Figure 1. If the current frequency is so small that there is no much statistical meaning of the entropy gain, a leaf node could also be chosen, which corresponds to the case e) in Figure 1

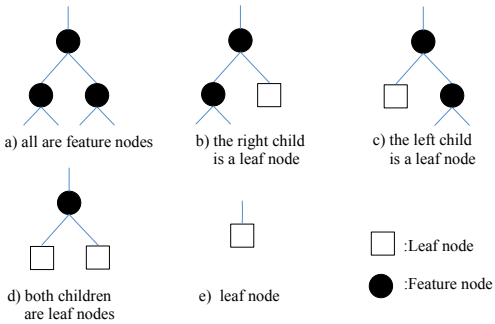

Figure 1. Five Possible Components of the One Step Look-Ahead Tree with Suppressed Cost(LASC Tree)

The tree is grown recursively adding the "fittest" component which has the largest heuristic value(H value) among the five types of possible components. To prevent overfitting for individual trees, pre-pruning is performed when necessary so that the tree stops growing if there are too few instances.

Experiments show that the look ahead enables us to find good combinations of two features that might not be found by a node-by-node search. The price we pay is that the complexity of searching the component is now $O(n^3)$ with respect to the number of features. The advantage of looking ahead is discussed in [23]. It is worth mentioning that unless we take feature extraction cost into consideration, even though it is highly possible that looking ahead will yield better trees (better accuracy and smaller size), it may also result in trees with pathology (bigger tree and lower accuracy).

B. Building the Specialized: Boosting and Inverse Boosting

The idea of boosting is to iteratively focus on the "hard" instances, which cannot be correctly labeled by the classifiers in hand, so that a lower error rate will be achieved if all the classifiers of the iterations are combined. When decision trees are used as base-classifiers in boosting, these highly specialized trees tend to have larger size, and they incur larger test costs as well, when they make use of expensive features.

Inverse boosting was originally invented to create ensembles of classifiers with high diversity[24], [25]. However, when combined with cost efficient classifiers, inverse boosting can be utilized to obtain more specialized classifiers. We propose the technique of inverse-boosting to train classifiers that are more specialized on the "easy" part of the data. Since this relieves the tree construction from the obligation to correctly classify harder instances, inverse-boosted trees are smaller, and they make errors by bending the stick to the other side. This is especially useful in practice when there are asymmetries in the difficulty to classify instances. Here by asymmetry, we mean that some instances are more difficult to classify than other instances. Note that the difficulty for classification might be typical among instances of specific classes, or it could be for instances that are near a decision boundary; The algorithm does not treat the two cases separately.

The idea of inverse-boosting is to iteratively generate computationally cheaper classifiers specialized on the "easy" instances, i.e. the instances that have already been classified correctly by the classifier at the previous iteration.

In standard boosting, given the weights $D_t(i), i = 1...M$ and the error rate ϵ_t of the current classifier, the new weight is updated as:

$$D_{t+1}(i) = \frac{D_t(i)exp(-\beta_i I(y_t(i) = h_t(\vec{X}_i))}{Z_t} \qquad (5)$$

where the indicator I selects those correctly labeled instances from the previous iteration,

$$I(y_t(i) = h_t(\vec{X}_i)) = \begin{cases} 1, & \text{if } y_t(i) = h_t(\vec{X}_i) \\ -1, & \text{otherwise} \end{cases} \qquad (6)$$

and $-\beta$ reduces their weight at each iteration so that the classifier focuses on incorrectly labeled instances.

$$\beta_t = \frac{1}{2} log \frac{1 - \epsilon_t}{\epsilon_t}. \qquad (7)$$

Here Z_t is the normalizing vector. In inverse-boostig, we want to focus on the 'easy' part of the data. Therefore, the new weight for each instance is obtained by simply changing the sign in front of β:

$$D_{t+1}(i) = \frac{D_t(i)exp(\beta_i I(y_t(i) = h_t(\vec{X}_i))}{Z_t} \qquad (8)$$

Note that Equation 8 is not the only way to reassign the weights in inverse-boosting, and it does not have the statistical meanings of the Equation 5 in standard boosting. An empirical result is presented in the next section that shows how accuracy of inverse-boosting converges with different choices of re-weighting functions.

C. Combine the Specialized into a Meta-Classifier

CoCost uses a cost-sensitive meta-classifier to combine the classification power of different trees which are specialized at different types of instance. As mentioned in the previous section, we can create such trees from boosting and inverse-boosting LASC trees. These inverse-boosted trees tend to be inexpensive and accurate at the "easy" part of the data, and the standard boosted trees will be more expensive as they are trained to focus at the "hard" instances. CoCost judiciously combines these specialized LASC trees to construct a more accurate meta-classifier that works well on all types of instances.

The way we generate the meta-classifier is to use the 'Stacked Generalization' method. The decision of each boosted and inverse-boosted tree on the validation data becomes a new feature in the meta-classifier, and the computational cost of each new feature is the expected computational cost of each tree. Eventually in the meta-classifier, each feature variable is actually one boosted tree. Note that the number of "features"(or trees) that the meta-classifier processes might be arbitrarily large, since it is simply the number of base-classifiers that we build.

CoCost's way of building the meta-classifier is also to use the cost sensitive LASC tree (with α=1). The LASC tree meta-calssifier allows us to choose a unique set of specialized trees in order to identify each test instance. In the next section, we present experimental results that demonstrate the advantages of using CoCost's meta-classifier.

IV. Experimental Results

We test our algorithm in a network flow detection scenario. The objective is to classify the underlying network traffic data type from the statistical features of the network package payload. The advantage of avoiding looking into the headers is that it is easy to forge the file headers but it is almost impossible to change the underlying statistical characteristics of the file content. The only way to obscure the statistical features of the file is to encrypt.

A. Data and features

There are eight data types in this application–TXT, BMP, WAV, JPG, MP3, MPG, ZIP and ENC. The feature set we used comes from a Network Abuse Detection system called Nabs. The features are generally the statistical characteristics of the network flows, such as mean, variance, and entropy. A detailed feature description can be found in [9]. Noting that in Nabs, the features are extracted from a chunk of 16Kbytes data randomly sampled from the network flow. We expanded the 22 features by randomly sampling chunks of 2Kb, 4Kb, 8Kb, and 16Kb from the network flow and recorded the average computational cost for the features in terms of computing time. The computational cost for these features are shown in Appendix A. In particular, there are overlapping costs in different features. All the frequency domain features (labeled by an '*' in TableIV) require the computation of an FFT from the sampled data. This implies that, if one of the frequency domain features is calculated and the FFT result is calculated and stored, then the cost for other frequency domain features will be lowered greatly. This important character of the feature costs give tree & forest classifiers an extra advantage. Trees give decisions incrementally and the overlapping of the feature costs decreases the average cost of one instance.

The raw data we used consists of files we arbitrarily downloaded from the internet. There are 1200 files, each type with 150 files. We choose an uniform distribution of the data types because we do not differentiate between the importance of various data types and assume that they are equally important. The only constraint on these files is that the file should be bigger than 50Kbytes, which makes the random sampling of different data chunks meaningful. We put 2/3 of these files as training files where training instances are extracted, while the remaining 1/3 are used as testing files. We extracted 2400 data instances from 800 files, each type with 100 different files. The remaining 400 test files are used to provide 1200 testing instances. The reason for choosing 2400 instances for training is that the learning curve (Figure 2) shows that the accuracy (on the same testing data) gets saturated when it is more than around 1500 training instances. And we use more than 1500 instances because we can reserve a proportion of the training data as validation data in building a meta-classifier.

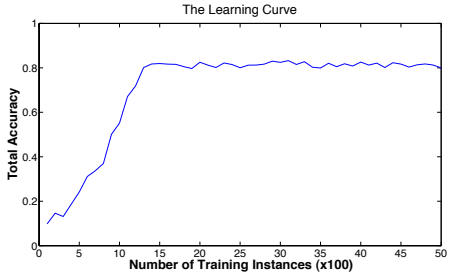

Figure 2. The Learning Curve: Number of Instances Versus Accuracy Using the C4.5 Classifier

B. Performance of different base classifiers

1) Performance Comparison on Random Feature Selection: First, we compare the performance of our base classifiers, the LASC Tree with Suppressed Cost with the C4.5 trees and the state-of-the-art SVM classifiers. We use the public SVM library[26] and choose the RBF kernel. To get a general view of the possible performance of the tree classifiers, we randomly pick up a subset of all the features, and build each classifier from these features. The parameters for SVM are chosen using a 10-fold cross validation on the training set. The performance of a classifier is the expected computational cost of one instance, and the weighted accuracies on all the target types. We do not take model building time into consideration because the training is performed offline.

As shown in Figure 3, 5 to 88 different features are randomly chosen from all the 88 features, and to generate more diverse tree classifiers, we injected randomness into these trees as follows: at each node of any tree, only a random half of all the selected features are available for building the tree. At each feature set, the plus points are the performances of C4.5 trees while the star points are the performances of LASC Trees with suppressed cost. Note that α is set to a median sensitivity of 1, and the effect of this parameter is discussed later. It is obvious that through building trees computational cost efficiently, we can get much lower cost at the same accuracies. The only drawback is that LASC Tree cannot get the highest accuracy 88.75% achieved by SVM using all the features at a very expensive cost around 200ms.

2) Performance Comparison on Deterministic Feature Selection: The performance of random feature selection provides a picture that through cost efficiently building classifiers we can save significant computational cost. However, it is notable that the number of all the possible feature combinations out of 88, $\sum_{i=1}^{88} C_{88}^{i}$,is an astronomical number, which we cannot cover all the possibilities using random feature selections. Therefore, we need to deterministically choose a set of features from the 88 features.

With SFS(Sequential Forward Feature Selection)[27], we can choose the features with most discrimination informa-

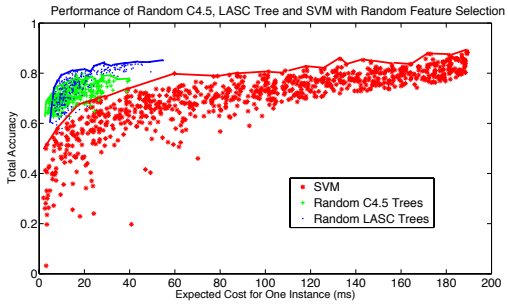

Figure 3. Performance of base classifiers with Random Feature Selection. The lines are envelop of the performance points of the three classifiers. It shows that LASC Tree with Suppressed Cost is more cost-efficient than C4.5 and SVM

tion. Taking the cost of each feature into consideration, the feature is re-ordered by its discrimination score over its cost. Table I provides the performance of the LASC Tree and other classifiers at different subsets of features.

It can be seen from the table that among all the classifiers, the LASC Tree usually gives the best performance but the difference in accuracy diminishes as bigger feature sets are used. CS-ID3 and EG2 trees perform favorably over IDX trees on larger feature sets, and CS-ID3 and IDX generate identical trees when using 6 and 7 selected features because the two similar heuristic function $\frac{\triangle I_i}{C_i}$ and $\frac{\triangle I_i^2}{C_i}$ choose the same features with certain feature sets.

C. Performance of CoCoST Versus Random Forest

In this section we compare the performance of the CoCoST with the Random Forest. We used the LASC Tree as the base classifier for both Random Forest and CoCoST. To make the Random Forest more efficient, short-cutting is applied in the unit vote. For example, if among 5 trees, the first 3 of them agree on the same decision, there is no need to calculate the cost for the other two and the cost is saved.

As the number of trees increases, the accuracy of Random Forest quickly converges and the cost incurred is larger. Efficiency of the Random Forest decreases quickly as the number of the tree members in the forest increases. With many trees in the forest, the accuracy saturates to the limit but the cost increases approximately linearly. Also with different α values, the accuracy of the forest increases as α increases. We tried Random Forests that consist of different vales of the sensitivity parameter α. The best result is recorded as the benchmark for evaluating CoCoST.

1) Changing the Sensitivity to Cost by Tuning α: The parameter α in our base classifier controls the trade-off between accuracy and cost. It determines how fast the sensitivity to cost is "suppressed" as the tree grows.Table II shows how the accuracy and cost will change as the parameter α changes. The classifiers are built on the set of 14 features selected by SFS. As α increases, the tree tends to ignore the cost and just picks up the features that provide more

Table II
CHOOSING DIFFERENT VALUES OF α WILL CAUSE DIFFERENT COST SENSITIVITY SUPPRESSION

α	Accuracy(%)	Cost(ms
0.1	77.97%	3.115
0.2	78.24%	4.241
0.4	79.13%	4.940
0.8	81.29%	6.558
0.95	81.78%	6.384
1.0	82.51%	7.625
1.5	83.34%	11.58
2.0	83.93%	18.55
4.0	84.73%	33.10

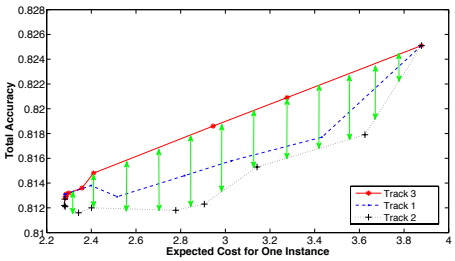

Figure 4. Individual Tree Performance of Inverse Boosted Trees

significant entropy gain. Therefore, with big α values, such as 2 or 4, the accuracy increases along with the cost. On the other hand, when α is small, say 0.1, we obtain a tree that is very cheap but has a lower accuracy. A good point for this cost accuracy trade-off is to set α to 1 when we want to use single tree as the classifier.

2) Performance of CoCoST: For CoCoST meta-classifier, we boosted and inverse-boosted 10 trees for each α appearing in Table II. Figure 4 shows the performance of inverse boosted trees with different re-assigning weights. The individual boosted tree's total accuracy actually decreases as we put more focus on the misclassified examples.

Depending on how we update the inverse-boosting weights, the inverse-boosted tree points are plotted in different tracks in the picture, each track representing a path of inverse-boosted trees using a different weighting function. Track 1 is the inverse-boosting we suggested in Section 3, while for Track 2 and 3 we tried the square and the square root of the reweighting function in Track 1. It can be seen that the only difference of these tracks is the speed they converge. The arrows in the graph shows that difference of accuracies at same costs will finally decrease to a very small value.

Because we only want trees that are specialized in a local part of the data, we discarded the first 9 trees and only kept the last tree in building the meta-classifier. Finally we have 16 candidate trees, boosted and inverse-boosted by the 8 different values of α. Their decisions on the validation data become the new training data on the meta-classifier, and the expected cost of each tree is now treated as the new price

Table I
PERFORMANCE OF DIFFERENT BASE CLASSIFIERS ON DIFFERENT SIZES OF FEATURES SETS CHOSE BY SEQUENTIAL FORWARD FEATURE SELECTION

Buffer Size(KB)		2KB	2KB,4KB	2KB,4KB and 8KB			2KB,4KB, 8KB and 16KB		
No. Of Features		6	7	14	21	28	35	42	88
Accuracy(%) Cost(ms)	LASC Tree(α=1)	80.45% 3.167	81.44% 3.279	82.51% 3.878	82.70% 8.852	83.73% 13.61	83.89% 23.89	84.57% 32.73	85.01% 58.27
	SVM	78.26% 5.470	79.34% 6.449	79.59% 12.60	81.44% 20.35	82.95% 32.91	84.76% 74.32	87.71% 145.5	88.75% 189.2
	IDX	80.37% 3.367	80.61% 3.199	80.53% 4.295	82.11% 8.918	82.23% 14.37	82.74$ 26.37	82.98% 38.26	83.22% 60.42
	CS-ID3	80.37% 3.367	80.61% 3.199	81.95% 4.208	82.03% 8.753	82.87% 12.91	83.19% 23.37	83.36% 35.85	83.76% 54.83
	EG2	78.35% 3.098	80.54% 4.929	81.43% 5.133	82.12% 10.23	82.85% 13.77	82.96% 28.84	83.32% 37.60	83.89% 67.99
	DL8	80.50% 19.16	81.82% 5.655	83.58% 9.170	—	—	—	—	—
	C4.5	79.50% 19.34	80.14% 5.197	83.58% 9.170	82.64% 14.38	83.15% 27.01	83.27% 34.94	83.95% 34.94	84.55% 72.12

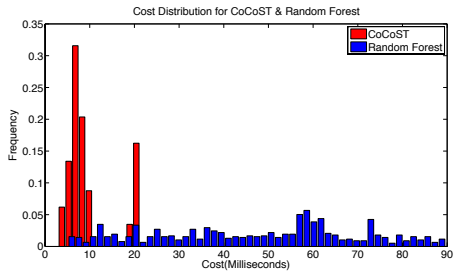

Figure 5. Compare the Frequency-Cost distribution between CoCoST and Random Forest. Noting that the distribution for SVM is only one bar as all the instances have the same cost

of the features. We again use our base learner to build the meta-classifier and set the α to 1 in this meta learning.

The tree structure of the meta-classifier is plotted in Figure 6. The accuracy of the classifier 89.51% with an expected cost for each instance 14.63, which is more than 10 times cheaper than the SVM classifier with an accuracy of 88.75%. It is also much better than the most accurate Random Forest we have ever obtained. The comparison is shown in Table III.

The final CoCoST meta-classifier expressed our idea that each tree is specialized at its own local part of the data, and combining them we can get very good results. The raw types(TXT,BMP and WAV) can be determined by the cheap inverse-boosted trees, while the compressed types(ZIP and ENC) are usually decided by two of the specialized boosted trees. Also the CoCoST has nice properties of generating classification rules that are interpretable. For example, the left most TXT decision presents the rule that if two inverse-boosted trees, which are already cheap and accurate at the raw types, agree that it is a TXT instance, then it is precise enough to just give an decision of TXT and there is no need to classify the instance using other trees. As we build the meta-classifier also cost-efficiently, it is shown in the graph that larger α values only appear at the bottom. For example,

Table III
COMPARISON AMONG COCOST, SVM, AND THE BEST RANDOM FOREST(α=1,CONSISTING OF 20 TREES)

	CoCoST	SVM	Random Forest
Accuracy	89.51%	88.75%	86.53%
Cost	14.63	189.2	40.11

the largest α values in the structure are 1.5 and 2.0. And they only appear after a series of trees with smaller α values have been tested. These trees are selected to classify the classify the most "confusing" part of the data.

3) Cost Distribution: CoCoST versus Random Forest: Noting that the averaged cost does not tell the whole story, the frequency distributions for each testing instances in CoCoST and the best Random Forest are plotted in Figure 5. It shows that CoCoST has a distribution of two peaks, where the easy and difficult instances are clustered. However, for Random Forest, the distribution is more like uniform because there is no mechanism allocating the cost. This graph explains why CoCoST performs much more efficient than ordinary classifiers. Because we do not know where the incoming instance may lie in the spectrum, Random Forest is more probable to incur more cost in classifying this instance. However, if we use CoCoST, the possible testing cost for the instance can only be around the two peaks of the cost distribution.

V. CONCLUSION

CoCoST is a very efficient tool in predictions where feature computational cost is an important concern. Because of the asymmetric in the data, it is possible to design classifiers that are only accurate at the clustering parts of the data. The asymmetry in the data lies in the fact that some instances are associated with the features of less computational cost while some instances require the information provided exclusively by the expensive features. Combining those diversified and specialized classifiers cost-efficiently into a meta-classifier, we can generate an accuracy

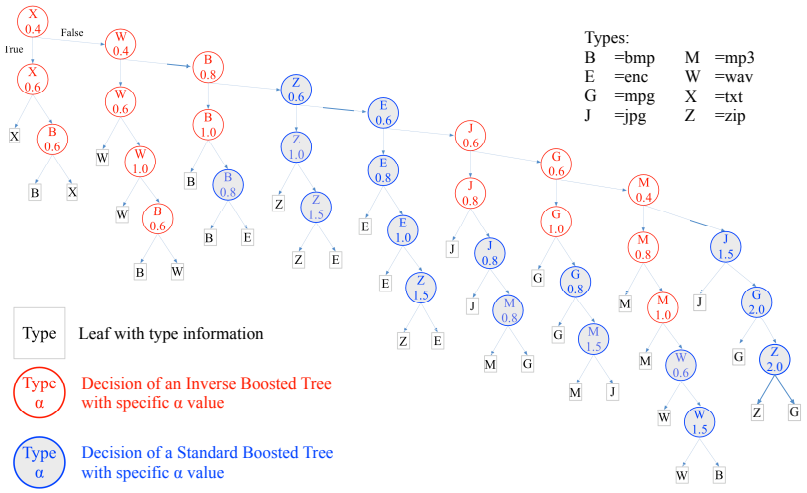

Figure 6. Structure of the CoCoST meta-classifier

almost the same as the SVM classifier using all the features, while the expected computational cost is only 10% of SVM.

The main limitation of the CoCoST classifier is the asymmetry requirement in the data. If the instances are associated with features that are of the same computational cost, it is possible that that we cannot construct classifiers of different prices and specialities. A further investigation of the measurement of the asymmetry between classification difficulty and cost will be performed. Also noting that each individual tree is only for binary decisions(such like if the decision is TXT), there is still space to cut the cost. And we leave this as the future work.

VI. Appendix A

Table IV shows the costs of the features. The frequency features are calculated from a 1024-point FFT of the original data flow samples. These frequency features have overlapping costs. Once one of the frequency feature is calculated, the cost of all the others is decreased to 0.2.

Table IV
Computational Prices(milliseconds) for Each Feature with Different Buffer Size

Feature	16K	8K	4K	2K
Entropy	2.381	1.521	0.734	0.802
Mean	0.548	0.271	0.217	0.094
Variance	3.589	1.824	0.904	0.661
Autocorrelation	42.27	18.37	10.46	5.358
Mean, Var, Power and Skewness in the first $\frac{1}{4}$ frequency band*	1.614	0.939	1.002	1.543
Mean, Var, Power and Skewness in the second $\frac{1}{4}$ frequency band*	1.251	0.778	0.912	0.979
Mean, Var, Power and Skewness in the third $\frac{1}{4}$ frequency band*	1.097	1.007	0.907	0.967
Mean, Var, Power and Skewness in the fourth $\frac{1}{4}$ frequency band*	1.591	1.204	1.193	1.185
Skewness	13.94	6.668	3.558	1.536
Kurtosis	13.203	6.491	3.746	1.671

Acknowledgment

The authors would like to thank Lisa Hellerstein for helpful discussions. Also we extend our thanks to the three anonymous reviewers for their constructive feedback.

References

[1] A. Kapoor and R. Greiner, "Learning and classifying under hard budgets," *Machine Learning: ECML 2005*, pp. 170–181, 2005. [Online]. Available: http://dx.doi.org/10.1007/11564096_20

[2] T. Abbes, "Protocol analysis in intrusion detection using decision tree," in *In Proc. ITCC04*, 2004, pp. 404–408.

[3] G. Stein, B. Chen, A. S. Wu, and K. A. Hua, "Decision tree classifier for network intrusion detection with ga-based feature selection," in *ACM-SE 43: Proceedings of the 43rd annual Southeast regional conference*. New York, NY, USA: ACM, 2005, pp. 136–141.

[4] C. Kruegel and T. Toth, "Using decision trees to improve signature-based intrusion detection," in *In Proceedings of the 6th International Workshop on the Recent Advances in Intrusion Detection (RAID2003), LNCS v. 2820*. Springer Verlag, 2003, pp. 173–191.

[5] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan, "Cost-based modeling for fraud and intrusion detection: Results from the jam project," in *In Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*. IEEE Computer Press, 2000, pp. 130–144.

[6] E. K. J. Alspector, "Svm-based filtering of e-mail spam with content-specific misclassification costs," in *In Proceedings of the Workshop on Text Mining (TextDM2001)*, 2001.

[7] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: an open architecture for collaborative filtering of netnews," in *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work*. New York, NY, USA: ACM, 1994, pp. 175–186.

[8] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems a survey of the state-of-the-art and possible extensions," *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, no. 6, pp. 734–749, 2005.

[9] M. K. Kulesh Shanmugasundaram and N. Memon, "Nabs: A system for detecting resource abuses via characterization of flow content type," *Computer Security Application Conference, Annual*, vol. 0, pp. 316–325, December 2004.

[10] C. X. Ling, Q. Yang, J. Wang, and S. Zhang, "Decision trees with minimal costs," in *ICML '04: Proceedings of the twenty-first international conference on Machine learning*. New York, NY, USA: ACM, 2004, p. 69.

[11] C. Elkan, "The foundations of cost-sensitive learning," in *In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001, pp. 973–978.

[12] C. Drummond and R. C. Holte, "Exploiting the cost (in)sensitivity of decision tree splitting criteria," in *In Proceedings of the Seventeenth International Conference on Machine Learning*. Morgan Kaufmann, 2000, pp. 239–246.

[13] A. B. B. G. C. Anna Marconato, Michele Gubian and D. Petri, "Accurate and resource-aware classification based on measurement data," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 9, pp. 2044–2051, September 2008.

[14] M. Nunez, "The use of background knowledge in decision tree induction," *Machine Learning*, vol. 6, no. 3, pp. 231–250, May 1991.

[15] C. Schaffer, "When does overfitting decrease prediction accuracy in induced decision trees and rule sets," in *In Machine Learning, EWSL-91*. Springer-Verlag, 1991, pp. 192–205.

[16] J. R. Quinlan, "Bagging, boosting, and c4.5," in *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*. AAAI Press, 1996, pp. 725–730.

[17] R. E. S. Yoav Freud, "A decision-theoretic generalization of on-line learning and an applicatio to boostin," *Computationa Learning Theory*, vol. 904, pp. 23–37, 1995.

[18] E. Leo Breiman, "Random forest," *Machine Learning*, vol. 45, no. 1, pp. 5–32, October 2001.

[19] S. K. Murthy, "Automatic construction of decision trees from data: A multi-disciplinary survey," *Data Mining and Knowledge Discovery*, vol. 2, pp. 345–389, 1998.

[20] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 21, no. 3, pp. 660–674, 1991.

[21] J. C. S. Ming Tan, "Two case studies in cost-sensitive concept acquisition," in *In Proceedings of the Eighth National Conference on Artificial Intelligence*, 1990.

[22] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees," in *Bagging, boosting, and randomization. Machine Learning*, 2000, pp. 139–157.

[23] S. Esmeir and S. Markovitch, "Lookahead-based algorithms for anytime induction of decision trees," in *In ICML04*. Morgan Kaufmann, 2004, pp. 257–264.

[24] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," vol. 55, pp. 119–139, 1997.

[25] L. I. Kuncheva and C. J. Whitaker, "Using diversity with three variants of boosting: Aggressive, conservative, and inverse," pp. 81–90, 2002.

[26] C. W. Hsu, C. C. Chang, and C. J. Lin, "A practical guide to support vector classification," Taipei, Tech. Rep., 2003. [Online]. Available: http://www.csie.ntu.edu.tw/˜cjlin/papers/guide/guide.pdf

[27] P. Somol, P. Pudil, J. Novovičová, and P. Paclík, "Adaptive floating search methods in feature selection," *Pattern Recogn. Lett.*, vol. 20, no. 11-13, pp. 1157–1163, 1999.