Missing or Inapplicable: Treatment of Incomplete Continuous-valued Features in Supervised Learning

Prakash Mandayam Comar* Lei Liu † Sabyasachi Saha‡ Pang-Ning Tan§
Antonio Nucci¶

January 25, 2013

Abstract

Real-world data are often riddled with data quality problems such as noise, outliers and missing values, which present significant challenges for supervised learning algorithms to effectively classify them. This paper explores the ill-effects of inapplicable features on the performance of supervised learning algorithms. In particular, we highlight the difference between missing and inapplicable feature values. We argue that the current approaches for dealing with missing values, which are mostly based on single or multiple imputation methods, are insufficient to handle inapplicable features, especially those that are continuous valued. We also illustrate how current tree-based and kernel-based classifiers can be adversely affected by the presence of such features if not handled appropriately. Finally, we propose methods to extend existing tree-based and kernel-based classifiers to deal with the inapplicable continuous-valued features.

1 Introduction

The imperfection of real-world data present significant challenges for supervised learning algorithms to accurately classify them. For example, noise due to erroneous recordings or distortions of observation values may deteriorate classification performance as it may lead to model overfitting. Another issue commonly encountered in practice is the *incomplete data* problem, where feature values are either missing at random for some observations or are unavailable for some blocks of observations [15]. The missing values are typically encoded as *null* values and are interpreted according to their context in the problem domain. For example, the null values in patient health records may indicate the absence of lab test results for costly procedures that were not performed on a patient. In survey data, null values may arise when the respondents do not provide an answer to a survey question due to privacy concerns or other reasons.

This paper considers a specific type of null value that corresponds to inapplicable feature values. A data instance is said to have an inapplicable value for a given feature if the property associated with the feature is unsuited for the given instance [11]. For example, *sales commission* is a feature that is applicable to employees working in the sales department but not to those working in other departments. Inapplicable features may also arise in survey data where only respondents who provide a specific answer to a previous question will be asked a follow-up question (e.g. a medical survey question about last surgery date is inapplicable to those who never had any surgeries). Another example is network traffic data, where a feature such as number of bytes in packet #5 is inapplicable to network flows that contain less than 5 packets.

While the distinction between missing and inapplicable feature values is well-documented in areas such as database systems [3, 2, 16, 11], paleontology [19], education [13], etc. Little work has been done by researchers in the data mining and machine learning community to treat inapplicable values differently from other types of missing values in developing supervised learning algorithms. A key distinction between missing value and inapplicable value is that, for the former, every domain value is possible, whereas for the latter, every domain value is impossible. For example, a genuinely missing sales commission value for a salesperson could be anywhere between zero and its maximum possible value whereas the sales commission for a non-sales employee should not take any of the domain values. Since conventional methods for handling incomplete data simply replace the missing values with other legitimate domain values, they are inappropriate for data sets that contain

*Michigan State University, Email: mandayam@cse.msu.edu
†Michigan State University, Email: liulei1@cse.msu.edu
‡Narus Inc., Email: ssaha@narus.com
§Michigan State University, Email: ptan@cse.msu.edu
¶Narus Inc., Email: anucci@narus.com

inapplicable feature values. Instead, we argue that the inapplicable values should be replaced by a value that does not exist within the domain of possible values. This can be done easily for nominal (categorical) features by replacing the inapplicable value with a new category. However, it is more challenging for continuous valued features as one must consider its ordinal relation as well as the addition/subtraction and multiplicative properties of the continuous domain [17]. For example, suppose the feature values for 3 data instances are $X_1 = 0$, $X_2 = 100$, and $X_3 = N/A$ (inapplicable), respectively, should $X_1 \geq X_3$, $X_2 \leq X_3$, or $X_1 < X_3 < X_2$? Furthermore, should $X_1 - X_3 = X_2 - X_3$ even though $X_1 \neq X_3$? Since many existing algorithms rely on computations of similarity between data instances, they are inadequate for handling continuous features with inapplicable values as the similarity matrix may not be positive semi-definite.

The main contributions of this paper are as follows:

1. We highlight the difference between missing and inapplicable feature values and illustrate how current tree-based and kernel-based classifiers can be adversely affected by inapplicable features if not handled appropriately.
2. We present a simple adaptation of existing tree-based method to deal with inapplicable continuous-valued features.
3. We introduce positive semi-definite kernels for data with inapplicable features and present a boosted kernel learning approach for classifying such data.

2 Decision Tree for Inapplicable Feature Values

A simple yet effective way to perform supervised classification is by using decision trees. Decision tree classifiers such as C4.5 [14] and CART [1] recursively partition the input space into rectilinear regions by considering one feature at a time. The features are selected using impurity measures such as entropy and gini index, which are defined as follows:

$$\text{(2.1)}\quad \text{Entropy} = -\sum_{i=1}^{k} p_i \log p_i, \quad \text{Gini} = 1 - \sum_{i=1}^{k} p_i^2$$

where p_i denote the proportion of training instances belonging to class i. During the tree growing process, the decision tree induction algorithm will select the feature that maximizes the gain (i.e. difference in impurity measure) as its splitting attribute. In this section, we discuss the limitations of current implementations of decision trees in dealing with inapplicable feature values and describe our proposed extensions to deal with such problems.

Current implementations of decision tree classifiers such as C4.5 [14], CART [1], and CHAID [10] do not distinguish between inapplicable and other types of missing values. They often employ data imputation methods to handle the incomplete data problem. These include *probabilistic split*, *surrogate split*, *separate class*, *grand mode/mean imputation*, *complete case* and *complete variable* methods [6]. The probabilistic split method, which is used by C4.5, routes training instances with missing values to every child of a node with different probabilities. The CART algorithm [1] uses the surrogate split strategy, where instances with missing values are routed to the child nodes based on the value of another surrogate feature, whose split most resembles the partitions made by the feature with missing values. The separate class approach was used by the CHAID algorithm [10], where a new category is introduced to represent the missing value. For continuous features, CHAID performs discretization on the features first, thus allowing the new category to be introduced as another bin or merged with other existing bins. Other strategies for dealing with missing values are based on data preprocessing, where the missing values are replaced by their overall mode (for categorical feature) or mean (for continuous feature) value. Finally, the complete case and complete variable methods simply remove the instances or variables that contain missing values before training the classifier.

In short, existing decision tree classifiers employ the same strategy to handle inapplicable values as they would for missing values. Next, we present a modified decision tree classifier that can handle inapplicable values as well as other types of missing values, for both categorical and continuous-valued features.

2.1 ModJ4.8 Decision Tree Classifier

The inapplicable feature values may contain useful information about a particular class that can be utilized to enhance the performance of a classifier. For example, an inapplicable value for sales commission may help distinguish non-sales employees from those who work in the sales department. Similarly, the absence of certain flow related features may indicate a particular type of network application or even malware attack.

In this section, we present a modified J4.8 decision tree classifier[1] to accommodate features that contain inapplicable and other missing values. The treatment of inapplicable feature values depends on the type of feature. For nominal features, similar to the approach used in CHAID [10], we introduce a separate category for the inapplicable value. A node that splits on the

[1] J4.8 is Weka's implementation of C4.5 decision tree classifier.

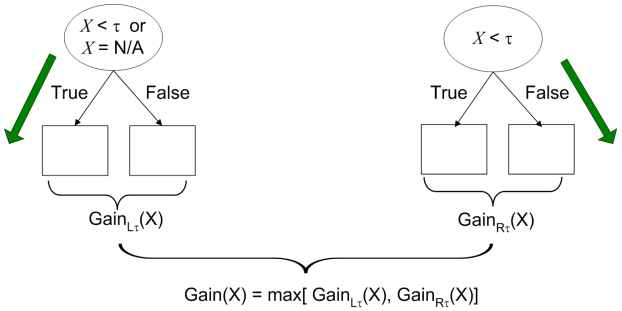


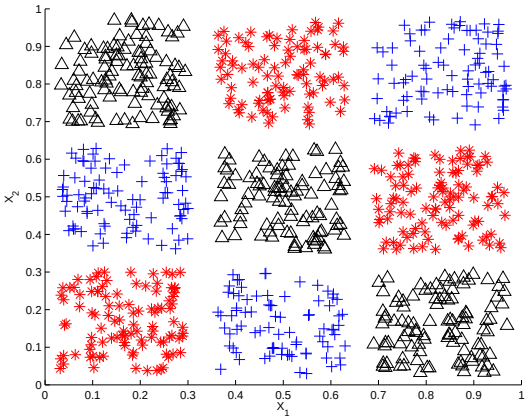
Figure 1: The modified split condition in ModJ4.8 classifier. All instances with inapplicable values (N/A) for feature X are either routed to the left branch (left figure) or to the right branch (right figure).

nominal feature will either create a separate branch for instances with inapplicable values or group them with other nominal values encountered in the data (provided it results in significant information gain).

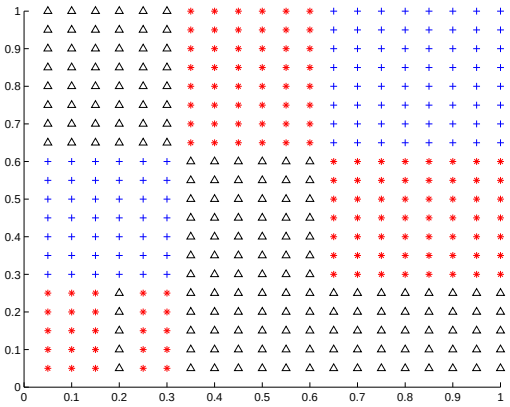
The treatment of inapplicable values present in a continuous feature is trickier. Ideally, all inapplicable values should be mapped to the same value, which must be considerably different than other legitimate domain values. As most decision tree classifiers consider only binary split condition ($X \leq \tau$) on the continuous feature X, if the inapplicable feature value is mapped to zero, instances with the inapplicable value will always be routed to the branch that corresponds to $X \leq \tau$ (if $\tau \geq 0$) and never to the branch that corresponds to $X > \tau$ even though the latter may lead to higher information gain.

Our proposed ModJ4.8 classifier assigns all instances with inapplicable values to either branch of a node once and compute their respective gains. The inapplicable instances will be routed to a child node that gives the highest gain (see Figure 1). In order to implement this, the split condition at each internal node of the tree is modified to allow a disjunctive condition of the form $(X < \tau) \vee (X = \text{N/A})$. If the split condition does not contain the second disjunct, the inapplicable instances will be routed to the right branch. As shown in Figure 1, the inapplicable instances participate in choosing the threshold τ that gives maximum gain. By participating in the process of choosing the threshold, the class information of the inapplicable values are included in the tree building process, unlike the conventional J4.8 classifier where instances with missing value do not determine the feature threshold.

2.2 Comparison between ModJ4.8 and J4.8 Classifier Figure 2(a) illustrates the synthetic data used to compare modJ4.8 against the conventional J4.8



(a) Original 2-dimensional data (before adding inapplicable values).



(b) Suboptimal decision boundary of J4.8 Classifier

Figure 2: Comparing J4.8 against modJ4.8. (a) Original 2-dimensional data. The X_2 values for all instances from the $\triangle$ class located in the upper left region and the X_1 values for instances from + class located in the bottom middle region are set to 'N/A'. (b) The suboptimal decision boundary produced by J4.8 classifier, which treats inapplicable values as missing values.

decision tree classifier [18]. We created a 2-dimensional data uniformly distributed on a unit square. We then partition the square into a 3×3 grid at thresholds 0.33 and 0.66, respectively, and assign each region to one of the three classes- $\{*, +, \triangle\}$. We then transform the data set to include inapplicable values by assigning X_2 = 'N/A' for all instances from the $\triangle$ class located in the upper left region and X_1 = 'N/A' for all instances from + class located in the bottom middle region.

With this modification, an ideal decision tree should classify all the instances with inapplicable value in X_1 as

```
x2 <= 0.6258
|   x1 <= 0.3
|   |   x2 <= 0.2997
|   |   |   x1 <= 0.2212
|   |   |   |   x1 <= 0.1965:   * (111.13/50.13)
|   |   |   |   x1 > 0.1965:    ^ (15.42/7.12)
|   |   |   x1 > 0.2212:   * (62.56/25.56)
|   |   x2 > 0.2997:       + (121.39/31.39)
|   |
|   x1 > 0.3
|   |   x2 <= 0.2997:       ^ (145.79/42.79)
|   |   x2 > 0.2997
|   |   |   x1 <= 0.6288:   ^ (97.0)
|   |   |   x1 > 0.6288:    * (105.0)
|   |   |
x2 > 0.6258
|   x1 <= 0.6288
|   |   x1 <= 0.3:   ^ (28.7/3.0)
|   |   x1 > 0.3:    * (93.0)
|   x1 > 0.6288:     + (89.0)

Confusion Matrix

     *   +   ^   <-- classified as
   296   0   4      *
     0 179  84      +
     0   0 306      ^
```

(a) Output from J4.8, which treats inapplicable as missing value

```
x2 <= 0.03,    N/A :  ^ (109.0/1.0)
x2 > 0.03
|   x1 <= 0.3,     N/A
|   |   x2 <= 0.2997
|   |   |   x1 <= 0.03,        N/A :  + (81.0/1.0)
|   |   |   x1 > 0.0319:           *  (101.0)
|   |   |
|   |   x2 > 0.2997:        + (93.0)
|   |
|   x1 > 0.3
|   |   x2 <= 0.6287
|   |   |   x2 <= 0.2997:          ^ (101.0)
|   |   |   x2 > 0.2997
|   |   |   |   x1 <= 0.6288:           ^ (97.0)
|   |   |   |   x1 > 0.6288:            *  (105.0)
|   |   |   |
|   |   x2 > 0.6287
|   |   |   x1 <= 0.6288:          * (93.0)
|   |   |   x1 > 0.6288:           + (89.0)

Confusion Matrix

      *    +    ^   <-- classified as
   299   1   0         *
     0 262   1         +
     0   0 306         ^
```

(b) Output from ModJ4.8, which treats inapplicable value differently than missing value

Figure 3: Decision trees obtained from (a) J4.8 decision tree classifier and (b) ModJ4.8 decision tree classifier.

+ and all the instances with inapplicable value in X_2 as $\triangle$. Figure 2(b) shows the decision boundary obtained from the J4.8 classifier, which uses the probabilistic split method, treating 'N/A' as missing value[2]. Although the classifier correctly identifies instances with inapplicable features that belong to the $\triangle$ class, it completely misses those that belong to the + class. Furthermore, even though the classes are well-separated, it misclassifies instances that do not contain inapplicable feature values. This is because J4.8 would propagate instances with inapplicable values to every branch of the node associated with the inapplicable feature and thus may create spurious branches that lead to misclassification of other instances[3].

On the other hand, our proposed modified J4.8 classifier distinguishes 'N/A' as a token for inapplicable value from '?' as the token for missing value. The resulting decision trees along with the confusion matrices are shown in Figure 3. Clearly, modJ4.8 gives a higher accuracy than the conventional J4.8 classifier. This is because modJ4.8 treats the inapplicable feature value as a separate category even for continuous features by allowing a disjunctive split condition. For example, the split condition at the root node is $X_2 \leq 0.03 \vee X = \text{N/A}$. This means all instances with $X_2 \leq 0.03$ or inapplicable will be routed to the left child of the node while those with $X_2 > 0.03$ will routed to its right child. This helps to isolate all the inapplicable instances from the upper left region to their correct class $\triangle$. Similarly, instances that belong to the + class in the lower middle block are subsequently separated from other classes by testing the inapplicability of feature X_1.

The proposed modification works well as long as the inapplicable values contain useful information about the target class. Unlike missing values, the inapplicable values generally do not occur at random. However, in the event that instances with inapplicable valued do not give any meaningful information about the class, the modified tree classifier should do no worse than the conventional tree classifier.

3 Kernels for Inapplicable Feature Values

Kernel-based classifiers such as support vector machine (SVM) have emerged as a popular choice for solving a variety of supervised learning tasks due to their robustness as well as higher relative accuracy compared to other traditional classifiers (including tree-based approaches). Although there has been some prior work investigating the effect of data imputation methods on SVM [12], to the best of our knowledge, none of them were designed to deal with inapplicable feature values. This section presents our proposed kernel learning method for addressing this problem.

3.1 Preliminaries Let $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \cdots (x_N, y_N)\} \in \mathcal{X} \times \mathcal{Y}$ be a collection of N training instances, where $\mathcal{X}$ denote the input domain and $\mathcal{Y} = \{+1, -1\}$ denote the set of distinct class labels. Furthermore, let $\Re_{\#} = \Re \cup \{\texttt{N/A}\}$ be the set of real num-

[2]Missing values are represented as '?' in Weka's ARFF format.

[3]As shown in Figure 2(b), the bottom left region contains a mixture of instances belonging to the classes $*$ and $\triangle$.

bers augmented with the inapplicable value (N/A). This allows us to extend the domain for input variables to $\mathcal{X} = \mathcal{R}_{\#}^{d}$. The kernel similarity between a pair of instances is a function $K : \mathcal{X} \times \mathcal{X} \rightarrow \Re$. The kernel function can be defined in the original input space or in a projected high-dimensional feature space $\Phi(\mathcal{X})$. Kernel functions that are positive semi-definite are often desirable as they form the basis for a class of large margin classifiers such as support vector machines. Popular choices of kernel functions include the linear kernel (which is the dot product between two input vectors) and non-linear kernels such as polynomial and Gaussian kernels.

3.2 Dealing with Inapplicable Feature Values

Applying the kernel function to instances with inapplicable values is non-trivial as conventional arithmetic operations such as subtraction and multiplication between two numbers in $\Re_{\#}$ may have to be re-defined to conform with their expected values. One possibility is to treat the inapplicable value # as a missing value (denoted as ? in Weka) and impute them with their global mean or mode. However, the resulting similarity values may not be consistent with their class information. As an illustration, consider the following toy example. Suppose each instance corresponds to an employee of an organization and the task is to distinguish salespersons from other employees.

Table 1: A toy example.

Employee Name	Sales commission	Class
John	\$15,000	Sales
Mary	\$10,000	Sales
Bob	N/A	Non-Sales
Lisa	N/A	Non-Sales

Imputing the N/A with the global mean (\$12,500) leads to Bob and Lisa (who are non-sales employees) to be more similar to John (a sales employee) than Mary (another sales employee). Using linear kernel as our example, the similarity between two employees can be computed as the product of their sales commission. Another strategy is to replace the inapplicable values by a constant value such as 0. This approach, though convenient, do not always agree with our expectation. For example, although similarity between John and Mary is higher than that between John and Bob (or John and Lisa), the similarity between Bob and Lisa is now equal to zero. In order to overcome this limitation, we propose to define the multiplication operator $\circledast$: $\Re_{\#} \times \Re_{\#} \rightarrow \Re$ as shown in Table 2.

For any two scalars x and y, the proposed multiplication operator assigns a maximum similarity score

Table 2: Multiplication operation $\circledast$ in $\Re_{\#}$.

x	y	$x \circledast y$
N/A	N/A	c
N/A	y	0
y	N/A	0
x	y	xy

$K(x, y) = c$ if both x and y are inapplicable and zero if only one of them is inapplicable. If both x and y are applicable, then their product reduces to the regular multiplication $x \times y$. With this definition, the similarity between Bob and Lisa is non-zero and is higher than that between John and Bob as well as John and Lisa.

The multiplication operator defined above can be used to induce an inner product between vectors.

DEFINITION 1. *Let $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \cdots x_{id})$ denote a d-dimensional feature vector, where each $x_{ik} \in \Re_{\#}$. The inner product in $\Re_{\#}$ is defined as follows:*

$$< \mathbf{x}_i, \mathbf{x}_j > \equiv \sum_k x_{ik} \circledast x_{jk} \tag{3.2}$$

where the multiplication operation follows the rules shown in Table 2.

Next, we provide a formal proof that shows the resulting dot product yields a positive semi-definite kernel.

LEMMA 3.1. *The inner product between two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ can be computed as follows:*

$$< \mathbf{x}_i, \mathbf{x}_j > = \frac{1}{2}\tilde{\boldsymbol{x}}_i^T \tilde{\boldsymbol{x}}_j + \frac{1}{2}(\tilde{\boldsymbol{x}}_i \bullet \hat{\boldsymbol{x}}_i)^T (\tilde{\boldsymbol{x}}_j \bullet \hat{\boldsymbol{x}}_j) \tag{3.3}$$

where $\bullet$ is the Hadamard product operator, $\hat{\boldsymbol{x}}_i$ and $\tilde{\boldsymbol{x}}_i$ are two vectors constructed from $\boldsymbol{x}_i \in \Re_{\#}^{d}$ as follows: $\hat{x}_{ik} = 1$ if x_{ik} = N/A and -1 otherwise; $\tilde{x}_{ik} = \sqrt{c}$ if x_{ik} = N/A and x_{ik} otherwise.

Proof. Based on the inner product definition given in Equation (3.2):

$$\begin{aligned} < \mathbf{x}_i, \mathbf{x}_j > &= \sum_k x_{ik} \circledast x_{jk} \\ &= \sum_k \tilde{x}_{ik}\tilde{x}_{jk}\frac{1 + \hat{x}_{ik}\hat{x}_{jk}}{2} \\ &= \frac{1}{2}\tilde{\mathbf{x}}_i^T\tilde{\mathbf{x}}_j + \frac{1}{2}\sum_k \tilde{x}_{ik}\tilde{x}_{jk}(\hat{x}_{ik}\hat{x}_{jk}) \\ &= \frac{1}{2}\tilde{\mathbf{x}}_i^T\tilde{\mathbf{x}}_j + \frac{1}{2}(\tilde{\mathbf{x}}_i \bullet \hat{\mathbf{x}}_i)^T(\tilde{\mathbf{x}}_j \bullet \hat{\mathbf{x}}_j) \end{aligned} \tag{3.4}$$

□

THEOREM 3.1. *The inner product given in Definition 1 induces a positive semi-definite linear kernel (Gram) matrix K, where $K_{ij} =< \mathbf{x}_i, \mathbf{x}_j >$.*

Proof. Let $\mathbf{X}$ denote an $N \times d$ data matrix, where each row corresponds to a data point. Furthermore, let $\tilde{\mathbf{X}}$ and $\hat{\mathbf{X}}$ be a pair of matrices whose i-th row corresponds to $\tilde{\mathbf{x}}_i^T$ and $\hat{\mathbf{x}}_i^T$, respectively. Following Lemma 3.1, the kernel matrix K can be computed as follows:

$$K = \frac{1}{2}\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T + \frac{1}{2}(\tilde{\mathbf{X}} \bullet \hat{\mathbf{X}})(\tilde{\mathbf{X}} \bullet \hat{\mathbf{X}})^T \tag{3.5}$$

Define the Hadamard product between $\tilde{\mathbf{X}}$ and $\hat{\mathbf{X}}$ as $\mathbf{Z}$. Then the kernel is given by

$$K = \frac{1}{2}\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T + \frac{1}{2}\mathbf{Z}\mathbf{Z}^T.$$

Furthermore, given a $d-$dimensional vector $\mathbf{v}$

$$\begin{aligned}
\mathbf{v}^T K \mathbf{v} &= \mathbf{v}^T(\frac{1}{2}\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T + \frac{1}{2}\mathbf{Z}\mathbf{Z}^T)\mathbf{v} \\
&= \frac{1}{2}\mathbf{v}^T\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T\mathbf{v} + \frac{1}{2}\mathbf{v}^T\mathbf{Z}\mathbf{Z}^T\mathbf{v} \\
&= \frac{1}{2} \parallel \tilde{\mathbf{X}}^T\mathbf{v} \parallel^2 + \frac{1}{2} \parallel \mathbf{Z}^T\mathbf{v} \parallel^2 \geq 0.
\end{aligned}$$

Thus, K must be a positive semi-definite matrix. □

The linear kernel defined in Theorem 3.1 can be extended to a polynomial kernel of degree p as follows:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \left(1 + < \mathbf{x}_i, \mathbf{x}_j >\right)^p$$

This allows us to extend the formulation to a nonlinear kernel in $\Re^d_{\#}$ space. In the next section, we present a novel algorithm to learn the kernel matrix for incomplete data. The proposed kernel is constructed as linear combination of base kernels, each of which is a rank-1 matrix constructed from the predicted outputs of tree-based classifiers.

4 Boosted Tree Kernel

The previous section presents unsupervised kernels for defining the similarity between pairs of data instances. For the nonlinear case, the appropriate choice of the kernel parameter is often tricky and poor selection may result in sub-optimal performance. In this section, we cast the kernel selection process as a learning problem, where the kernel is estimated from the available data. Here again, we are faced with the dilemma caused by inapplicable data. Towards this end, we propose an effective way to perform supervised kernel learning on data with inapplicable values.

A simple and intuitive way to learn a non-linear kernel on a given labeled data is to construct a ground truth kernel $\mathbf{G}$ from the label information as follows:

$$G(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} 1, & \text{if } y_i = y_j; \\ -1, & \text{otherwise.} \end{cases}$$

Given the ground truth kernel G, our objective is to derive a kernel K that is maximally aligned with the ground truth kernel G by minimizing the following exponential loss function:

$$\mathcal{L} = \sum_{ij} \exp\Big[- K_{ij}G_{ij}\Big] \tag{4.6}$$

The exponential loss function has been studied at length in the machine learning community. A simple yet effective way to minimize the exponential loss is by employing the boosting framework [8, 4, 5], where we impose a parametric form on the kernel,

$$K = \sum_t \alpha_t K_t$$

Here the kernel K is written as a linear combination of several "weak" kernels K_t, each of which captures the similarities among a subset of the data instances correctly. It is desirable to construct the weak kernels K_t as low rank matrices to speed up the computation time of the underlying machine learning algorithms [7, 9].

In this paper, the weak kernels are constructed as rank-1 matrices $K_t = u_t u_t^T$, where each vector $u_t \in \{-1, +1\}$ is the predicted output of the modJ4.8 decision trees discussed in the previous section. In addition to providing easy-to-compute rank-1 weak kernels, another advantage of this approach is that it can automatically handle inapplicable features. The weak kernel $K_t = u_t u_t^T$ would assign a high similarity score to pairs of data points predicted to be in the same class and low scores to pairs of data points predicted to be in different classes. Since the output of the modJ4.8 decision tree classifier is not perfect, the weak kernel K_t might not perfectly reflect the ground truth kernel G. The boosting framework [8, 4, 5] provides a systematic way to combine the weak kernels K_t to form a "strong" kernel K. Algorithm 1 presents a summary of our TreeBoost kernel learning method for inapplicable features.

The algorithm maintains a weight matrix D that reflects the discrepancy between the current estimate K and the ground truth G. The larger the discrepancy, the higher is the corresponding weight in D. D was initialized to uniform distribution in the first iteration. In each iteration, the algorithm computes a weight vector W for all the data instances as a row average of matrix D. The weight vector is used to select instances from $\mathcal{D}$ to form a training set. The training set is then used to construct a modJ4.8 decision tree u_t. The predicted output of the tree is then used to construct a rank one kernel $K_t = u_t u_t^T$. We then compute the confidence factor α_t associated with the rank one kernel

Algorithm 1 `TreeBoost Kernel`

Input: $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots (\mathbf{x}_N, y_N)\} \in \mathcal{X} \times \{+1, -1\}$

Output: K: $N \times N$ estimated kernel matrix

Initialize: T = maximum number of iterations
$W_0 = 1/N\mathbf{1}$
$G = yy^T; K = [0]_{N\times N}; D = \frac{1}{N}\exp(-K \bullet G)$

for $t = 1$ to T **do**
  Update weight for each data point
    $W_t(r) = \frac{1}{n}\sum_j D_{rj}$
  Create $\mathcal{D}_t$ by sampling with replacement from $\mathcal{D}$
    according to W_t
  Train a modJ4.8 decision tree classifier on $\mathcal{D}_t$
    $u_t : \mathcal{X} \to \{-1, +1\}$
  Construct the rank-1 kernel matrix $K_t = u_t u_t^T$
  Compute the kernel alignment error:
    $\epsilon_t = Pr_D(\mathbf{I}(K_t \neq G))$
    $\alpha_t = \frac{1}{2}\log(\frac{1-\epsilon_t}{\epsilon_t})$
  Update weight matrix:
    $D \leftarrow D \bullet \exp(-G \bullet K_t)$ and `Normalize`
  Compute $K \leftarrow K + \alpha_t K_t$
end for
`return` K

based on the kernel alignment error, ϵ_t. Finally, we adjust the weight on each kernel entry according to the exponential loss function.

The proposed TreeBoost kernel learning method has several advantages:

- It is designed for learning kernels with incomplete data.
- It explicitly learns the feature function $\phi(x)$ based on the ground truth kernel, unlike the linear and polynomial kernels described in the previous section.
- It represents the kernel as a linear combination of low rank matrices, i.e. $K = V\Lambda V^T$ where K is an $N \times N$ matrix and V is an $N \times k$ matrix ($k << N$). The low rank representation has an advantage in lowering both the storage and computational complexity [7, 9] (unlike the approach presented in [5] that requires solving a generalized eigenvalue problem, which has $O(N^3)$ complexity).

5 Experimental Evaluation

In this paper, we have proposed three simple methods for performing classification on data with inapplicable values for continuous features namely - modified J48 tree algorithm (`Mod. J48`), modified dot product based linear/polynomial kernel (`Lin.` $\circledast$) and a novel supervised tree based kernel (`T. Kernel`). In this section, we compare the proposed algorithms against two other baseline algorithms, namely, regular J48 decision tree classifier and the mean imputed linear/polynomial kernels `MI Lin.` The experimental evaluations are performed on real world data set from two different domain.

5.1 Linear Kernel We first compare the performance of the mean imputed linear/polynomial kernel and the proposed modified linear kernel. For doing this, we used the KDD CUP 2012 data set which consists of a twitter like directed social network of users and a user-item recommendation network. Each user acts on an item recommendation by either accepting (+1) or rejecting(-1) the recommendation. We selected 14 items from the population of about 6000 items and extracted the subgraph of all the user who have been recommended at least one of these 14 items. The goal is to recommend a chosen set of 10 items to users in the subgraph whose response for this chosen item is unknown. For each user in the subgraph, we compute the number of neighbors (outgoing link) who have accepted item i as a feature. In all there are 14 features for each user. These feature can take value between 0 and maximum out degree of the network. If an item is not recommended to any of the neighbors, then the feature value becomes inapplicable. The inapplicable feature value cannot be replaced with zero value, as zero value means that none of the neighbors has accepted the item. We perform a binary SVM on the above data using a simple linear/polynomial kernels. We rank order each user based on its distance from the boundary line. We flag the top 500 most positive users as the one who would accept a recommended item. We then compute the number of actual positive values in the top 500 as response rate. The results are shown in Table 3. The proposed modified linear/polynomial kernel performs better than the mean imputed kernel on 6 products and as good as mean imputed kernel on 2 other products. It performs slightly worse than the mean imputed kernel on 2 products.

5.2 J48 Trees In this section, we present the results comparing the proposed modified J48 tree from the regular J48 tree algorithm. Here we used the internet traffic flow data set where the goal is to categorize the observed flow into either malicious flow or legitimate flow. Thus it is a binary classification problem. The number of features extracted per flow varies with length of the flow. For examples, features like size of packet

Table 3: This table gives the response rate of top 500 user's accepting a recommended item. The score for top 500 users is obtained by running binary SVM using mean imputed polynomial kernel `MI Lin.` and proposed polynomial kernel for degree 1 and 2.

Product	Degree = 1		Degree = 2	
	`MI Lin.`	`Lin.` ⊛	`MI Poly.`	`Poly.`⊛
1	7.2 %	8.4 %	4.6 %	7.8 %
2	11.6 %	12 %	9.2 %	10.6 %
3	8.8 %	9.6 %	9.2 %	10.6 %
4	10.2 %	13.4 %	11.4 %	11.2 %
5	14.2 %	15.2 %	13.8 %	14.2 %
6	15.2 %	16.8 %	12.6 %	12.0 %
7	4.2 %	3.6 %	4.8 %	6.10 %
8	13.2 %	12.8 %	14 %	14.8 %
9	12.4 %	11.4 %	11.8 %	11.8 %
10	8.8 %	8.4 %	8.4 %	8.4 %

100, may not be applicable to short flows with less than 100 packets. But the size cannot be imputed with zero because, sometimes, dummy packets are inserted to avoid detection giving rise to genuine packets with zero payload size. Thus for a flow with less than k packets, the feature value of *size of packet k* is marked inapplicable instead of zero. This holds for other flow level features like inter arrival time etc. In all we have extracted 108 flow level features.

Table 4 records the F measure for each of the two flow classes on different algorithms. Notice, that the data set is skewed with legitimate flows overwhelming the malicious flows. Here, the modified J48 algorithm gives a higher F measure on malicious class than the original J48 tree algorithm, without compromising on the performance on the legitimate class.

5.3 Tree Kernel In this section, we evaluated the performance of the proposed tree kernel against the linear kernels. Here we used the same data set as described for J48 trees and the goal is to classify each malicious flow (detected using trees) into different type of malwares. In essence, we perform multi class classification on the traffic flow data set. Table 5 gives the summary of different classes in train and test set. For each class, it also lists the proportion of Inapplicable features out of 108 features extracted from the flow characteristics. Clearly, the proportion of inapplicable features varies with class. For example, out of 108 features class A has at least one inapplicable value in 76 of them. Class E has the least number of data instances with inapplicable features. We run SVM on the above data set with three different kernel. The results are summarized in Table 5. Clearly, the proposed kernel learning technique outperforms the traditional mean imputed linear kernel as well as proposed modified linear kernel

6 Conclusions

In this paper, we present arguments that highlight the need for data processing mechanism for dealing with inapplicable valued features. First we distinguish the inapplicable values from the missing values. We then highlight the deficiency of decision trees in handling the inapplicable data and propose a suitable modification for the same. Further, we utilize the modified decision trees to construct kernels on the data with inapplicable values, which are in turn used to perform multi class classification. In addition, we define a multiplication operation for dealing with inapplicable values and use this multiplication operation to define a inner product kernel on data with inapplicable values. We prove the positive semi definiteness of this new kernel and experimentally validate its superiority over regular inner product kernel for data with inapplicable values.

7 Acknowledgement

Prakash Mandayam Comar and Pang-Ning Tan's research are supported in part by Office of Naval Research grant number N00014-09-0663.

References

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.

[2] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst.*, 4:397–434, 1979.

[3] E. F. Codd. Missing information (applicable and inapplicable) in relational databases. *SIGMOD Record*, 15(4):53–78, 1986.

[4] M. Collins, R. E. Schapire, and Y. Singer. Logistic regression, adaboost and bregman distances. *Machine Learning*, 48(1-3), 2002.

[5] K. Crammer, J. Keshet, and Y. Singer. Kernel design using boosting. In *In Advances in Neural Information Processing Systems 15*, 2003.

[6] Y. Ding and J. S. Simonoff. An investigation of missing data methods for classification trees applied to binary response data. *Journal of Machine Learning Research*, 11:131–170, 2010.

[7] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *J. Mach. Learn. Res.*, 2, Mar. 2002.

[8] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1), 1997.

Table 4: This table compares the performance of the modified J48 tree against the regular J48 tree and linear kernels. The modified J48 tree gives higher F measure for both malicious and legitimate flows. Here again, the modified linear kernel performs better than the linear kernel.

Class	Train	Test	J 48	Mod J48	`Lin.` ⊛	`MI Lin.`
Malicious	500	3874	0.952	0.976	0.55	0.50
Legitimate	10000	40000	0.995	0.997	0.93	0.9

Table 5: This table records the F1 measure for different kernels. The tree based kernels involve sampling of both the data points and the features to construct trees. We therefore provide the mean and standard deviation of the F1 score.

Class	Train	Test	Prop of N/A	Tree Kernel	MI `Lin.`	`Lin.`⊛
A	300	300	70%	0.958 ± 0.008	0.3467	0.8247
B	91	211	35%	0.982 ± 0.032	0.6709	0.9952
C	41	94	55%	0.871 ± 0.131	0.6083	0.9305
D	56	132	63%	0.964 ± 0.026	0.39	0.6977
E	25	33	30%	0.846 ± 0.119	0.8529	0.9688
F	25	31	47%	0.818 ± 0.103	0.1667	0.7576
G	43	100	52%	0.966 ± 0.027	0.6753	0.7857
H	300	1200	63%	0.985 ± 0.002	0.8374	0.9593
I	300	1070	55%	0.994 ± 0.001	0.6062	0.9854
J	59	137	35%	0.885 ± 0.039	0	0.9073
K	33	78	35%	0.915 ± 0.046	0.4503	0.7368
L	25	29	55%	0.744 ± 0.107	0	0.4127
M	27	62	55%	0.951 ± 0.033	0.1151	0.7154

[9] P. Jain, B. Kulis, J. V. Davis, and I. S. Dhillon. Metric and kernel learning using a linear transformation. *Journal of Machine Learning Research*, 13:519–547, march 2012.

[10] G. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29:119–127, 1980.

[11] F. D. Linn. Missing and inapplicable values. *SIGMOD Records*, 16:18–19, September 1987.

[12] K. Pelckmans, J. D. Brabanter, J. A. K. Suykens, and B. D. Moor. Handling missing values in support vector machine classifiers. *Neural Networks,05*, 18(5-6):684–692, 2005.

[13] J. L. Peugh and C. K. Enders. Missing data in educational research: A review of reporting practices and suggestions for improvement. *Review of Educational Research*, 74(4):525–556, 2004.

[14] J. R. Quinlan. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*. Morgan Kaufmann, 1 edition, 1992.

[15] D. Rubin. Inference and missing data (with discussion). *Biometrika*, 63:581–592, 1976.

[16] Stephen and Henley. The man who wasn't there: The problem of partially missing data. *Computers Geosciences*, 31(6):780 – 785, 2005.

[17] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley Longman Publishing Co., Inc., 2005.

[18] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. 2005.

[19] R. Zaragueta-Bagils and E. Bourdon. Three-item analysis: Hierarchical representation and treatment of missing and inapplicable data. *Comptes Rendus Palevol*, 6(6-7):527 – 534, 2007.