# Similarity Forests

Saket Sathe
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
ssathe@us.ibm.com

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
charu@us.ibm.com

## ABSTRACT

Random forests are among the most successful methods used in data mining because of their extraordinary accuracy and effectiveness. However, their use is primarily limited to multidimensional data because they sample features from the original data set. In this paper, we propose a method for extending random forests to work with any arbitrary set of data objects, as long as similarities can be computed among the data objects. Furthermore, since it is understood that similarity computation between all $O(n^2)$ pairs of $n$ objects might be expensive, our method computes only a very small fraction of the $O(n^2)$ pairwise similarities between objects to construct the forests. Our results show that the proposed similarity forest approach is very efficient and accurate on a wide variety of data sets. Therefore, this paper significantly extends the applicability of random forest methods to arbitrary data domains. Furthermore, the approach even outperforms traditional random forests on multidimensional data. We show that similarity forests are robust to the noisy similarity values that are ubiquitous in real-world applications. In many practical settings, the similarity values between objects are incompletely specified because of the difficulty in collecting such values. Similarity forests can be used in such cases with straightforward modifications.

## CCS CONCEPTS

•**Computing methodologies →Classification and regression trees;** *Supervised learning by classification;* •**Information systems** →*Data mining;*

## KEYWORDS

data mining; classification; random forests

## 1 INTRODUCTION

Random forests are among the most successful classifiers because of their tremendous accuracy and robustness in various domains. A recent study [8] evaluated 179 classifiers from 17 families on the *entire UCI collection of data sets*, and concluded that random forests were the best performing classifier among these families, and in most cases, their performance was better than other classifiers in a statistically significant way. In fact, multiple third-party implementations of random forests were tested by this study and virtually all implementations provided better performance than multiple implementations of other classifiers. The only other classifier that sometimes outperformed the random forest on a modest number of data sets was the support-vector machine; however, even in this case, the performance of random forests was superior in the aggregate. These results suggest that the random forest classifier is generally quite robust across a wide variety of settings, and is a good choice to use as a default classifier in the absence of domain-specific information.

Random forests are naturally designed to work with multidimensional data because they sample features from the original space in order to create the decision trees that form the ensemble components of the forest. In many applications, such as time-series data, discrete sequences, or graphs, a multidimensional representation might not exist. However, in such domains, the problem of similarity function computation is well studied, and it may be possible to compute similarities between objects. A further complication is that similarities are often incompletely specified when given by domain experts. In all these cases, one cannot use traditional random forests, which are inherently designed to work with multidimensional data sets.

A key problem is that similarity computations are often quite expensive in many domains [15], and it may not be feasible to compute all pairwise similarities from a computational point of view. Therefore, it is important to be able to construct the random forests efficiently while computing an extremely small subset of the pairwise similarities. After all, the number of pairwise similarities scales up quadratically with the number of points. Therefore, any method that requires the entire similarity matrix between pairs of objects to be specified up front is bound to face computational and storage challenges. Furthermore, in cases where all pairwise similarities are not specified (or even computable), this type of approach can still be used, albeit in a limited way.

One possible solution to this problem is to extract a multidimensional embedding from the similarity matrix. As we will discuss later, this approach is not only outrageously expensive (because the extraction of the embedding might require $O(n^3)$ time), but it also requires $O(n^2)$ space to materialize the similarity matrix. Just to provide an idea of the space required, a data set containing a million points would require space of the order of $10^{12}$, which is already in the tera-byte order. On the other hand, the approach discussed in this paper requires no more than $O(n \cdot \log(n))$ time and $O(n)$ space for each component of the ensemble. It is also difficult to use traditional classification methods (like SVMs) when the similarities between pairs of data objects are incompletely specified. We show how our approach can be extended to such difficult cases.

*KDD '17, August 13-17, 2017, Halifax, NS, Canada*
© 2017 ACM. 978-1-4503-4887-4/17/08…$15.00
DOI: 10.1145/3097983.3098046

In this paper, we propose the construction of random forests directly with similarity matrices *without assuming access to an explicit multidimensional representation of the data*. We refer to this approach as *SimForest*, which corresponds the fact that it is a *similarity forest*. The basic idea behind this approach is to assume that the data is embedded in some theoretical multidimensional space, and then use the similarities for computing the coordinates of data points along 1-dimensional projections. These 1-dimensional projections are created by sampling pairs of points belonging to different classes, and then for each pair projecting the remaining points along the theoretical multidimensional direction representing the line joining the pair. As we will show, this projection can be performed by using only the pairwise similarities. An important advantage of this approach is that by sampling pairs of points belonging to different classes, one often ends up selecting discriminating directions at least in a randomized sense. In other words, the bias characteristics of the individual components of the similarity forest are often very promising. In combination with the inherent robustness of an ensemble-centric approach, high-quality results are obtained. Therefore, much like the support vector machine, which uses pairwise similarities in order to perform the prediction (without explicitly materializing an embedding), we are able to use similarities in order to construct a random forest without materializing an embedding. However, as discussed in [8], random forests often have an advantage over the SVM in many data domains. The similarity forest approach extends these advantages to cases where traditional random forests cannot be used.

An additional advantage of the *SimForest* approach is that it can be naturally extended to settings where all pairs of similarity values are not available. This can happen in many cases where the number of data points is large, and the entire $O(n^2)$ matrix of similarity values cannot be reasonably materialized. Furthermore, the approach can be used even in domains (e.g., text) where the multidimensional representation is available, but traditional decision trees do not always enable good splits because of feature sparsity; in such cases, similarities can provide better splits.

This paper is organized as follows. The remainder of this section discusses the related work. In the next section, we will introduce the basic notations and definitions. We will also discuss the motivation and the basic idea underlying our approach. The *SimForest* algorithm is presented in section 3. The different variations and extensions of the *SimForest* approach are also discussed in this section. The experimental results are discussed in section 4. The conclusions are presented in section 5.

### 1.1 Related Work

A detailed survey of various data classification and ensemble algorithms is provided in [1]. A popular ensemble method is that of random forests. Random forests require the availability of a multidimensional representation up front in order to perform the classification. The support vector machine is one of the few classifiers that can work with similarity values in order to provide a classification with the use of the kernel trick [6, 7]. The kernel trick is equivalent to the use of a linear SVM on the transformed data, in a form that is computationally feasible [9, 19]. In general, it is impossible to fully materialize pairwise similarity matrices to perform transformations especially if the data size is large. This is the reason that the kernel trick is preferred over explicit transformation.

A feature bagging variant of the decision tree was first proposed in [11], and subsequently generalized to the present form of the random forest by Breiman [4, 5]. Interestingly, the work in [5, 17] shows that one can extract pairwise similarities from a random forest by counting the frequency with which two instances end up in the same leaf node. In this paper, we achieve the reverse, where we use the pairwise similarities for constructing the random forest. The work in [13, 18] constructs a rotation forest, in which a rotated axis system is constructed up front; the rotation directions are constructed using a combination of attribute partitioning and PCA. Several methods have been proposed for constructing oblique decision trees with [10] or without [12] the use of pairwise directions. However, these methods still need the original multidimensional feature space. The close relationship between random forests and kernel methods is provided in [16]. An extensive study showing the robustness of random forests is available in [8]. In this paper, we propose the use of a similarity-based approach for the direct construction of random forests, while materializing only a very small fraction of the pairwise similarities between objects.

## 2 DEFINITIONS AND MOTIVATION

We assume that we have a set of $n$ objects denoted by $O_1 \dots O_n$. These $n$ objects can be of any data type, and in most cases, we do not even need to know the specific object type, as long as it is possible to compute a similarity value between them. The similarity between the objects $O_i$ and $O_j$ is denoted by $S_{ij}$, and it may not necessarily be available up front to the algorithm. For example, in the time-series domain, a domain-specific similarity function may be used. In such cases, the similarity between objects are computed on the fly as needed by the algorithm; the algorithm design needs to minimize such computation. It is assumed that the labels are drawn from a set of $c$ different possibilities denoted by $L = \{1 \dots c\}$. It is assumed that the $i$th object $O_i$ in the training data is labeled as $l_i \in L$. We state the problem as follows:

Problem 1 (Similarity Forest). *Given a set of $n$ objects $O_1 \dots O_n$, which are labeled $l_1 \dots l_n$, construct a random forest with the use of only computed pairwise similarities between these objects and no access to the multidimensional representations (if any) of these objects.*

The resulting model is used for classification of objects with unknown labels. For a given test object $O$ with computable similarities with other training objects, a principled method must be proposed in order to perform the classification. As we will show later in this paper, the testing process uses similar principles as the training phase.

For now, we assume that the similarity between any pair of objects can be computed, although we will discuss later how cases of incompletely available similarities can be effectively addressed. Furthermore, in some cases, one might be able to compute distances rather than similarities. For example, one might use distance functions such as the dynamic time-warping distance or edit distance in some domains. In such cases, the similarities can be computed from distances using the cosine law or (as an approximation) the distances can be used in lieu of the similarities with few implementation changes. These variations will also be discussed.

## 2.1 SimForest: The Motivation

Even though a multidimensional representation is not provided for the data objects, the basic idea is to assume that a multidimensional space exists in which the objects can be embedded. It is noteworthy that one can use kernel methods to extract a multidimensional representation of the objects, if all $O(n^2)$ pairs of similarities between the objects are provided. Such an approach can potentially use $O(n^2)$ space and $O(n^3)$ time, which is not practical for most real-world settings. Aside from this fact, all $O(n^2)$ pairs of similarities are not required to construct a similarity forest. This is because a similarity forest gains large accuracy advantages from the diversity of its individual components. Each ensemble component can often be constructed using a small number of pairwise similarities that are selected in randomized fashion. This is an important advantage of the approach when the similarity computations between pairs of objects are expensive and need to be performed on the fly.

Another point to keep in mind is that explicit embedding extractions do not always work well when the underlying space is very high-dimensional, and when individual dimensions capture little information. One advantage of directly constructing the random forest with similarities is that one can choose to select randomized splits that are biased towards more discriminative directions. Such an approach is able to retain excellent bias characteristics of the individual ensemble components without compromising too much on accuracy.

Let us assume that the objects $O_1 \ldots O_n$ can be theoretically embedded in some multidimensional space as the points $\overline{X_1} \ldots \overline{X_n}$. However, we do not assume that we explicitly know the representation of $\overline{X_1} \ldots \overline{X_n}$; instead, we will work only with similarities between pairs of objects. In fact, we do not even assume that we know the dimensionality of the embedded data points $\overline{X_1} \ldots \overline{X_n}$, because they are not required for constructing the similarity forest. Just as a random forest samples features from a multidimensional data set, the similarity-based approach samples pairs of objects in order to define directions in the multidimensional space. For example, sampling the object $O_i$ and $O_j$ results in the vector direction from $\overline{X_i}$ to $\overline{X_j}$. The data objects are then split into two groups using a hyperplane perpendicular to this direction. As we will show later, we do not need to explicitly project the data points along this direction, but the split can be fully executed only using pairwise similarities. This approach is applied recursively to construct each ensemble component of the similarity forest. During the testing phase, the traversal of the tree also requires such similarity computations. It is noteworthy that if the pairs of sampled objects $\overline{X_i}$ and $\overline{X_j}$ are chosen to belong to different classes, the randomized split directions will naturally tend to be more discriminating. The number of pairs of data objects that are required to be sampled at each node is analogous to the number of features that are sampled by the traditional random forest algorithm. Our experimental results show that a small number of pairs such as 2 or 3 can achieve a high level of accuracy.

## 3 THE SIMFOREST ALGORITHM

In order to describe the *SimForest* algorithm, we will denote the theoretical embeddings by $\overline{X_1} \ldots \overline{X_n}$, although none of the algorithmic steps require the knowledge of these embeddings. Therefore, whenever we talk about "directions" in the data space, we are effectively talking about the theoretical embedding $\overline{X_1} \ldots \overline{X_n}$. The approach constructs each decision tree in the similarity forest in top-down fashion by using recursive splits. This is similar to the traditional random forests approach. Like traditional random forests, which sample features for splitting, the *SimForest* algorithm samples directions from the original space. The notion of using multivariate directions in the data space for splitting is also used in traditional random forests [18]. However, in order to create a similarity-based approach, the *SimForest* method uses a completely different solution by defining the random directions in terms of *pairs of objects* to enable similarity-based splitting.

These pairs of objects are sampled from among the objects present at the node, which is to be split. One of our interesting and surprising discoveries was that a proper sampling of pairs of objects even improves the performance of *multidimensional* random forests. In other words, even if actual multidimensional representations of data points were given, it is still more accurate to use this approach over traditional random forests. Specifically, at each node, $r$ pairs of objects are sampled in order to define $r$ different random directions. The pairs of objects can be sampled in one of two ways:

(1) The pairs are selected randomly without paying any special attention to the class label.
(2) The pairs are always selected randomly to belong to different classes. In other words, the first object in the pair is selected randomly, whereas the second is selected from the objects belonging to a different class.

The second approach tends to lead to more discriminative splits and therefore reduces the height of the tree that is constructed. In our experiments, we always chose this option because it provided better results. In order to retain diversity, the value of $r$ to be used should be small. Furthermore, using small values of $r$ is beneficial for the efficiency of the method, and one can easily ensure accuracy by increasing the number of trees in the forest. In our experiments, we found that it was best to set the value of $r$ to 1.

Next, we describe the process of evaluating the quality of a split used by the *SimForest* algorithm. Consider a pair of objects $O_i$ and $O_j$ that define the direction of the split. We would like to project each data point along the direction *from* $O_i$ to $O_j$. Therefore, an object that is identical to $O_i$ would obtain a coordinate of 0, and $O_j$ should obtain a positive coordinate. Consider an object $O_k$ that needs to be projected along this direction. As discussed earlier, the theoretical multidimensional embeddings of $O_i$, $O_j$, and $O_k$ are, respectively, $\overline{X_i}$, $\overline{X_j}$ and $\overline{X_k}$. Then, the unit direction of the embedding is given by $\frac{\overline{X_j}-\overline{X_i}}{||\overline{X_j}-\overline{X_i}||}$. The projection $P(\overline{X_k})$ of $\overline{X_k}$ on this direction is therefore given by the dot product of $\overline{X_k} - \overline{X_i}$ on this unit direction. In other words, we have the following:

$$P(\overline{X_k}) = (\overline{X_k} - \overline{X_i}) \cdot \frac{\overline{X_j} - \overline{X_i}}{||\overline{X_j} - \overline{X_i}||} \tag{1}$$

We will show that this entire projection along the 1-dimensional line from $\overline{X_i}$ to $\overline{X_j}$ can be computed only in terms of similarities between pairs of objects. By expanding the product in the numerator of the aforementioned expression, we obtain the following:

$$\begin{aligned} P(\overline{X_k}) &= \frac{\overline{X_k}\cdot\overline{X_j} - \overline{X_k}\cdot\overline{X_i} - \overline{X_i}\cdot\overline{X_j} + \overline{X_i}\cdot\overline{X_i}}{||\overline{X_j} - \overline{X_i}||} \\ &= \frac{S_{kj} - S_{ki} - S_{ij} + S_{ii}}{||\overline{X_j} - \overline{X_i}||} \end{aligned}$$

Here, $S_{ij}$ represents the similarity between objects $O_i$ and $O_j$. The main problem at this point is the denominator, which is not expressed as similarities. It is possible to also express the denominator in terms of similarities by using the following:

$$\begin{aligned} ||\overline{X_j} - \overline{X_i}|| &= \sqrt{||X_j||^2 + ||\overline{X_i}||^2 - 2\overline{X_i}\cdot\overline{X_j}} \\ &= \sqrt{S_{ii} + S_{jj} - 2S_{ij}} \end{aligned}$$

However, in practice, it is not necessary to explicitly compute the denominator because (for a given pair) the denominator is independent of the specific data point $O_k$ being projected. In other words, the denominator changes the embedded representation of each data point by only a constant factor, and the *ordering* of the projected points along the 1-dimensional line is not affected, whether or not one includes the denominator in the computation. In a decision tree, a split is computed by examining all $(n-1)$ possible partitions induced by this ordering. The $(n-1)$ potential split points are midway between the $(n-1)$ pairs of adjacent points in this aforementioned ordering. Therefore, we can compute the projection to within a constant of proportionality to achieve the same result. The corresponding projection $P(\overline{X_k})$ may be expressed as follows:

$$P(\overline{X_k}) \propto S_{kj} - S_{ki} - S_{ij} + S_{ii} \tag{2}$$

Furthermore, since the $i$th and $j$th objects are already fixed over all projections computed at the splitting node, the last two terms are constant for all data points within the node. In other words, we can write the following:

$$P(\overline{X_k}) \propto S_{kj} - S_{ki} + C \tag{3}$$

Here, $C$ is a constant that does not affect the relative ordering of the points in the projection along the line from $\overline{X_i}$ to $\overline{X_j}$. One can view $S_{kj} - S_{ki}$ as a *scaled and translated proxy* for the projection of $O_k$ on the line joining $O_i$ to $O_j$. This proxy is used in lieu of the actual projection both in the training and in the testing phase. *Therefore, in order to compute a split for the direction defined by the pair $(O_i, O_j)$, we need to sort the data objects $\{O_k\}_{k=1}^{n}$ in order of $(S_{kj} - S_{ki})$ and use it to evaluate various splitting points.*

Subsequently, the splitting point is chosen such that it minimizes the weighted Gini index of the children nodes. Specifically, if $p_1 \ldots p_c$ are the fractions of data points belonging to the $c$ different classes in node $N$, then the Gini index of that node is given by the following:

$$G(N) = 1 - \sum_{i=1}^{c} p_i^2 \tag{4}$$

Then, if the node is split into two children $N_1$ and $N_2$, with $n_1$ and $n_2$ points, respectively, the weighted Gini quality $GQ(N_1, N_2)$ of the children nodes is given by the following:

$$GQ(N_1, N_2) = \frac{n_1 G(N_1) + n_2 G(N_2)}{n_1 + n_2} \tag{5}$$

When $r$ different pairs of points are used at a node to define possible split directions, all $(n-1)$ possible splits along these $r$ different directions are tested, and the best split is retained. In similarity forests, small values of $r$ are typically used in order to ensure diversity. This is particularly important in cases where the pairs are chosen from different classes.

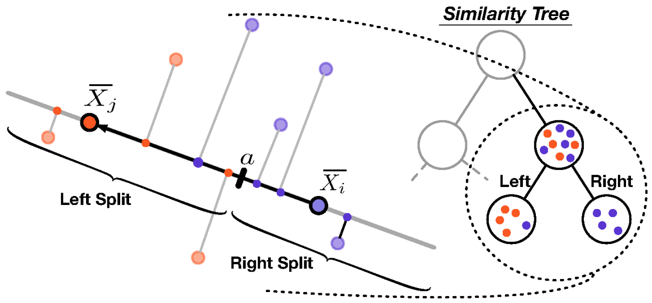


**Figure 1: Conceptual illustration of splitting for similarity tree construction.**

A conceptual illustration of the splitting process is shown in Figure 1. Each decision tree in the similarity forest is constructed recursively until the leaf nodes purely belong to a specific class. Although the construction of a decision tree often has a pruning phase in which the lower nodes of the tree are removed (to reduce overfitting), this is not done in all types of random forests, including the similarity forest. We construct the tree to full height without pruning, until each node only contains instances belonging to a particular class.

After the decision tree has been constructed, the relevant pairs of objects and the split points are stored at each node. These are required for the testing phase. It is important to note that the pair of objects $(O_i, O_j)$ is *ordered* (for consistency with the testing phase) because the projection $S_{ij} - S_{ii}$ of object $O_i$ is always negative, and the projection $S_{jj} - S_{ji}$ of object $O_j$ is always positive. This holds true for most reasonable similarity functions in which the absolute magnitude of self-similarity $S_{ii}$ is greater than the absolute magnitude of any cross-object similarity $S_{ij}$. This sign convention is caused by the fact that the direction of the projection vector is from $O_i$ to $O_j$.

The testing phase uses an identical approach to project the test points on the line defined by the pairs of training points at each node. If $(O_i, O_j)$ is the defining pair of objects for a given node, then the value of $S_{kj} - S_{ki}$ is computed for the test object $O_k$. The stored split point (say $a$) is used to determine which path in the decision tree to follow depending on whether or not $S_{kj} - S_{ki} \leq a$. This step is performed for each node on the path of the tree, until the object $O_k$ is assigned to a leaf node. The label of the leaf node is reported as the prediction.

## 3.1 Implicit Assumptions

One of the implicit assumptions behind this approach is that the data points can be embedded in a multidimensional space. In order for this fact to be true, the $n \times n$ similarity matrix can be expressed as $DD^T$, where each row of $D$ contains one of the embedded points $\overline{X_i}$. Any matrix that can be expressed as $DD^T$ needs to be positive-semidefinite [2]. Many natural similarity functions satisfy this condition. However, *in practice*, one can still use this approach in a heuristic manner even if the underlying similarity matrix is not positive semi-definite. In particular, the use of the difference in similarities of pairs of objects (belonging to different classes) has a very natural and intuitive tendency to partition them into different classes. Therefore, even though the approach was *derived* assuming the existence of a multidimensional embedding, it can be used in virtually any setting.

## 3.2 Computational Complexity

One of the attractive features of this approach is its excellent computational complexity. Explicitly extracting a multidimensional embedding from the data might require $O(n^2)$ space and $O(n^3)$ time. Furthermore, at least $O(n^2)$ similarity computations will be required if the embedding is explicitly extracted. This can be prohibitive even when only a few thousand points are present in the data. A single split of *SimForest* requires time that is linear in the number of points *at the splitting node* in the tree. Since a decision tree strictly partitions the data points, the time required to construct *an entire level* of the decision tree is linear in the number of points. Assuming that the height of the tree is $O(\log(n))$, the time required to construct the entire decision tree is $O(n \cdot \log(n))$. Since a random forest typically contains a constant number of decision trees, the overall running time is still $O(n \cdot \log(n))$. The testing procedure requires $O(\log(n))$ time. This efficiency makes the approach an effective alternative to other competing methods.

## 3.3 Distances Instead of Similarities

In some settings, it is easier to compute distances rather than similarities. For example, in the sequence domain, one might naturally use the edit distance or the dynamic time-warping distance. In such cases, there are two ways in which one can convert distances to similarities:

(1) **Exact approach (computationally intensive):** One can use the *cosine law* to convert distances to similarities. Let $U$ be an $n \times n$ matrix of 1s, and $I$ be an $n \times n$ identity matrix. Let $\Delta$ be the $n \times n$ matrix of squared distances. According to the cosine law, a mean-centered similarity matrix $S$ is constructed from the distance matrix $\Delta$ using the following rule [2]:

$$S = -\frac{1}{2}(I - U/n)\Delta(I - U/n) \tag{6}$$

The main disadvantage of this approach is that it requires the $O(n^2)$ distance matrix up front, which causes challenges from the point of view of computational and space efficiency.

(2) **Approximate approach (computationally efficient):** This approach uses the squared distances between points in lieu of the similarities. As we will show, this is equivalent to using the differences in similarities, if we make the *normalization* assumption that all points have the same squared norm. This is equivalent to saying that the self-similarity $\overline{X_i} \cdot \overline{X_i}$ of each object $O_i$ is the same. In many domains, this is a reasonable assumption to make. For example, in the text domain, the cosine similarity between pairs of documents always lies in $(0, 1)$ and the self-similarity of a document to itself is always 1. This is because the cosine is a *normalized* similarity function. Similar normalizations are reasonable to assume in many data domains. Such an assumption provides an efficient heuristic approach to perform the splitting in cases where the affinities between objects are expressed in terms of distances rather than similarities.

Consider a setting in which the data points $O_i$ and $O_j$ are the pair of objects defining the direction along which all points are embedded. Let $D(O_i, O_j)$ be the distance between data points $O_i$ and $O_j$. Then, the difference in squared distances of point $O_k$ with respect to $O_i$ and $O_j$ is defined as follows:

$$\begin{aligned}
&D(O_k, O_i)^2 - D(O_k, O_j)^2 = \\
&= ||\overline{X_k} - \overline{X_i}||^2 - ||\overline{X_k} - \overline{X_j}||^2 \\
&= 2\overline{X_k} \cdot \overline{X_j} - 2\overline{X_k} \cdot \overline{X_j} + \underbrace{(||\overline{X_i}||^2 - ||\overline{X_j}||^2)}_{\text{Zero (Because of Normalization)}} \\
&\propto S_{kj} - S_{ki}
\end{aligned}$$

This is the same expression that we used for the case of similarities. Note that the expression $(||\overline{X_i}||^2 - ||\overline{X_j}||^2)$ is zero because it is assumed that all embedded points have the same self-similarity. Therefore, with this assumption *there is no difference* between using the squared distances or the similarities for splitting. This makes the approach quite general in many settings, because one can use either distances or similarities between objects, depending on what is available in the specific domain at hand.

## 3.4 Incompletely Specified Similarities

*SimForest* can also be used in cases where only a subset of pairwise similarities is observed. This is a particularly common problem in settings where similarity estimation is expensive. For example, if similarities between pairs of objects (such as images) are obtained using crowd-sourcing, a cost may be incurred for each pairwise similarity. Obviously, all pairwise similarities will be impossible to collect in a large data set.

This situation creates a problem from an algorithmic perspective. The problem is that not all objects can be partitioned between a pair of nodes at each split point because the projection of data points along arbitrary directions may require unobserved similarities to be available. The first algorithmic modification to address this issue is that the selected pairs of objects $(O_i, O_j)$ for splitting should always be such that similarity between them is observed. However, this still does not solve the problem that the similarity of an arbitrary point $O_k$ in that node may have unobserved similarities with either $O_i$ or $O_j$. In such cases, since $O_k$ cannot be assigned to one of the child nodes, we allow it to stay at the current node as its final destination. The class label of an internal node is defined by its majority class. Therefore, unlike the case of fully observed similarities, internal nodes also have a label.

When test instances are classified, the same procedure is applied in computing $S_{kj} - S_{ki}$ (when similarity is available) and assigning the point $O_k$ to a child node, depending on the relationship of $S_{kj} - S_{ki}$ to the split point. Otherwise, if these similarities are not available, the point $O_k$ is assigned to an internal node. The label of the corresponding node is reported as the prediction.

## 4 EXPERIMENTAL RESULTS

In this section we present extensive experimental results comparing *SimForest* with other competitive techniques. We start by describing the data sets in Section 4.1, followed by a description of the evaluation methodology in Section 4.2. Next, we extensively test *SimForest*'s resistance to noisy similarities in Section 4.3. In Section 4.4 we evaluate how missing similarity values effect the accuracy of *SimForest*. Lastly, in Section 4.5, we show results on multidimensional data.

### 4.1 Data Sets

We used several classification data sets for our experiments. The important characteristics of these data sets are given in Table 1. The data sets are of various sizes and have varying numbers of features. All of the data sets are obtained from the LibSVM Website repository of classification data sets[1]. All data sets were normalized to zero mean and unit variance.

**Table 1: Summary of the Data Sets.**

| Data set | Points | Features | Classes |
|---|---|---|---|
| Heart | 226 | 13 | 2 |
| Ionosphere | 295 | 34 | 2 |
| Breast-Cancer | 573 | 10 | 2 |
| Australian | 579 | 14 | 2 |
| Diabetes | 645 | 8 | 2 |
| German-Numer | 840 | 24 | 2 |
| SVMGuide3 | 1035 | 22 | 2 |
| a1a | 32,240 | 123 | 2 |
| Madelon | 2200 | 500 | 2 |
| Splice | 2975 | 500 | 2 |
| Mushrooms | 6824 | 112 | 2 |

### 4.2 Evaluation Strategy

For every experiment we perform a 80-20 split of the data. 80% data is used for training the classifiers, while 20% data is used for testing its performance. In many experiments the training data is further divided into portions for model building and parameter tuning, in which 10% is used for tuning. Model performance is measured using *accuracy*. Accuracy is measured as the percentage of correctly classified test data points.

[1]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

### 4.3 Performance with Noisy Similarity Matrices

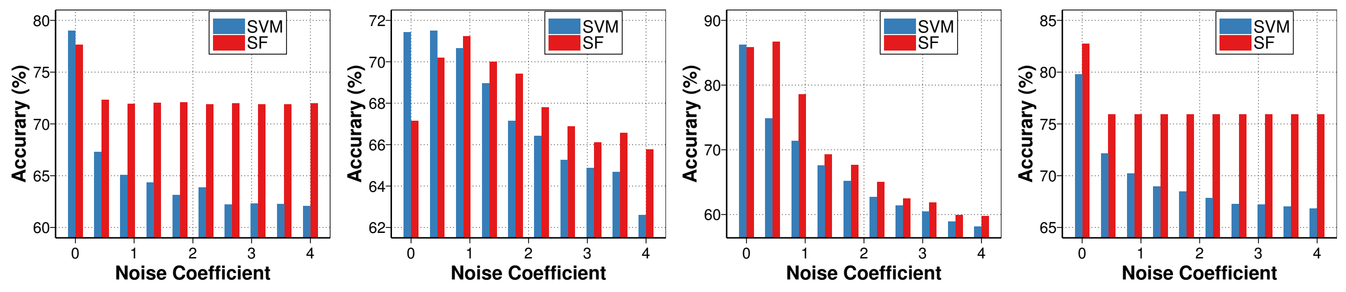
The primary use case of *SimForest* is one in which only the similarities between objects are available, and the objects might not be multidimensional. In such cases, a traditional random forest cannot even be used, and the natural competitor to our approach is a kernel support vector machine. Therefore, this section will show results with respect to this natural competitor.

In many real settings, similarity matrices are noisy. For example, consider the use of crowdsourcing to measure image similarity, where a human crowdsourcer is tasked with assigning similarity to a pair of images. In such cases, it is natural for errors to occur in the similarity, which are partly due to error in judgment or even due to the inherent bias of different workers. In this section, we will show that *SimForest* is highly resistant to such noisy similarities, as compared to well-known classifiers, such as SVMs.

A key point here is in the availability of noisy similarity matrices, and that of testing the effect of a specific level of noise. Therefore, our approach was to use multidimensional data to construct the similarity matrices (with added noise), but not to assume access to the multidimensional features for building the classification model (either *SimForest* or baselines). This was achieved by constructing two types of similarity matrices, corresponding to the cosine similarity and the Gaussian RBF similarity. Note that both these similarity matrices are naturally used as kernels in support vector machines, and therefore the SVM approach[2] is well suited to these types of input similarity matrices. Noise was added[3] to the similarity matrices with the use of a *noise coefficient* $\alpha$. For each similarity value, a noise value that was uniformly distributed between $[0, \alpha]$ was added. To mitigate the effect of random variations, ten different instantiations of the noise were generated, and the averaged results were reported. The bandwidth of the RBF kernel for computing the similarity matrix was selected by picking a point of optimum performance for the SVM (without added noise), but it was fed into both classifiers. This actually puts *SimForest* at a disadvantage because the similarity matrix that is given as part of the input data is biased towards working well with the (baseline) SVM. Note that the assumption here is that the similarity matrix is a part of the input data, and the analyst has no say in how it is chosen, computed, or obtained.

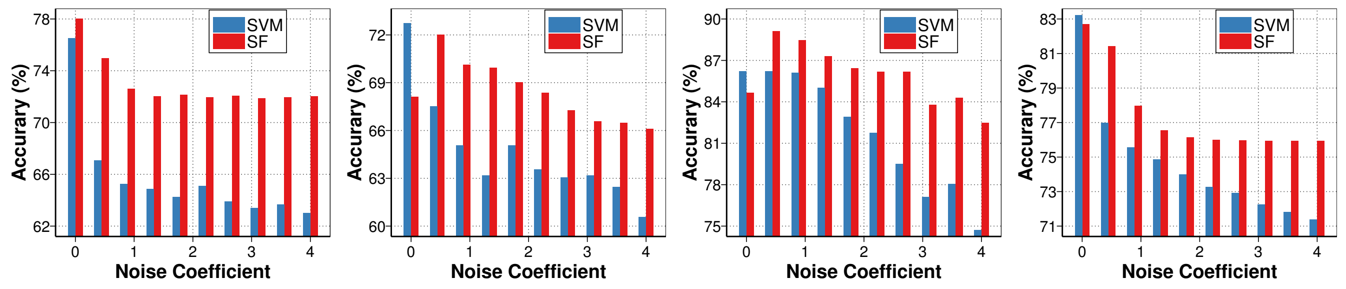
The similarity matrices were generated for six data sets, which were Ionosphere, Heart, German-Numer, Diabetes, Australian, and a1a. A *SimForest* with 100 similarity trees is used for this experiment. In Table 2, we illustrate the accuracy obtained over various data sets with both types of similarity matrices at $\alpha = 2.5$. In all the data sets, *SimForest* has superior performance as compared to SVM. The best performance improvement of 14.7% and 12.6% is observed in the German-Numer data set for RBF kernel and the Ionosphere data set for the cosine similarity, respectively. Overall, these experiments demonstrate that *SimForest* is highly noise resistant and can consistently produce superior results over

[2]In fact, *scikit-learn* has an option to input the similarity matrix rather than the multidimensional features in its SVM implementation. The details are available at http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. We use this off-the-shelf implementation.
[3]It is assumed that the same noise is added to the $(i, j)$th and $(j, i)$th entries because the similarity matrix is symmetric.



(a) German-Numer (b) Diabetes (c) Australian (d) a1a

**Figure 2: RBF Kernel: Comparing noise sensitivity of *SimForest* (SF) and SVM (best viewed in color).**



(a) German-Numer (b) Diabetes (c) Australian (d) a1a

**Figure 3: Cosine Similarity: Comparing noise sensitivity of *SimForest* (SF) and SVM (best viewed in color).**

state-of-the-art methods like support vector machines. Most importantly, *SimForest* represents a natural generalization of the idea of random forests to the setting where only similarity matrices are available.

**Table 2: Accuracy observed at a fixed value of the noise coefficient ($\alpha = 2.5$).**

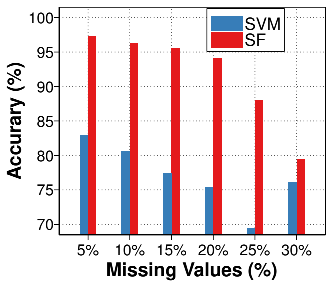
| Data set | RBF Kernel | | Cosine Similarity | |
|---|---|---|---|---|
| | *SimForest* | SVM | *SimForest* | SVM |
| Heart | **68.14** | 67.22 | **72.96** | 68.70 |
| Ionosphere | **71.69** | 68.59 | **74.22** | 65.91 |
| Australian | **61.88** | 61.23 | **85.79** | 79.49 |
| Diabetes | **68.44** | 64.87 | **67.33** | 62.85 |
| German-Numer | **72.00** | 62.80 | **71.85** | 65.40 |
| a1a | **75.94** | 67.81 | **75.97** | 72.88 |

Further experiments were performed to compare the noise resistance of the proposed *SimForest* approach to SVMs, both of which can exclusively operate only on similarity matrices. The idea is to increase the noise coefficient $\alpha$ to examine the impact on the accuracy. Figures 2 and 3 show the comparison between *SimForest* and SVM for the RBF kernel and cosine similarity, respectively. For both types of similarity measures, *SimForest* exhibits extraordinary ruggedness to noise. One reason for this ruggedness is that SVMs attempt to maximize the inter-class margin by using a few select points known as *support vectors*. In other words, SVMs ignore points other than support vectors for prediction. It is particularly noteworthy that misclassified points and noise are usually support vectors that occur on the wrong side of the decision boundary. Therefore, addition of noise adversely affects the capability of the SVM to choose reliable support vectors, which results in a poor decision boundary. On the other hand, *SimForest* is inherently noise tolerant because of its robust approach in averaging out the noise. This results in higher overall accuracy of the *SimForest*.

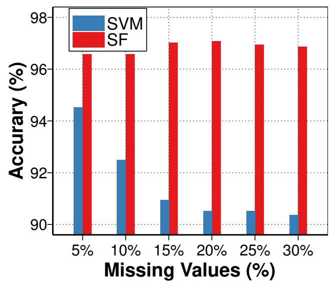
The choice of the similarity matrix (cosine or RBF) does have some effect on the relative performance of the two methods, although the effect for high amounts of noise is similar. In all the plots shown in Figures 2 and 3, the *SimForest* classifier does much better than SVM when the value of $\alpha$ was set to large values. It is noteworthy that the similarity matrices in real applications are often highly noisy, which could be an artifact of the subjective way in which these matrices are extracted.

## 4.4 Handling Missing Similarities

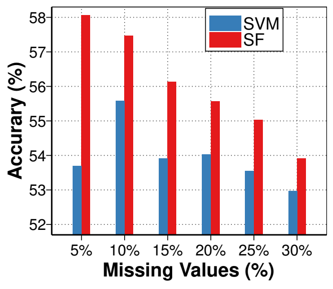
Aside from the noise, an additional problem occurs when many entries of the similarity matrix are missing. The presence of missing entries is very natural when such matrices are derived from user feedback. Manual feedback is tedious and crowd-sourced feedback (e.g., *Amazon Mechanical Turk*) is expensive. Therefore, it is impractical to assume that all pairwise similarities will be available. Furthermore, even if the pairwise similarities can be computed, they are computationally expensive to evaluate. This approach is not feasible when the number of points is very large. For example, if a



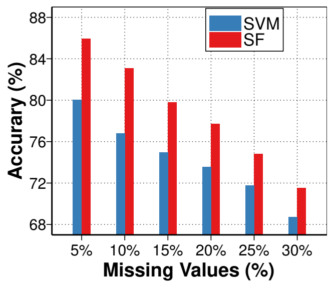
(a) Ionosphere



(b) Breast-Cancer

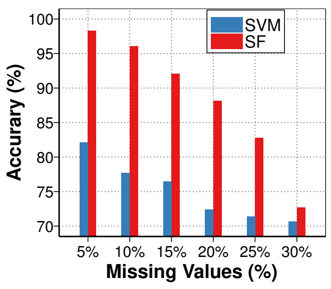


(c) Madelon

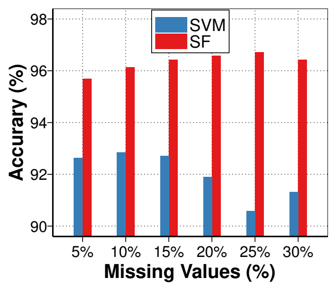


(d) Splice

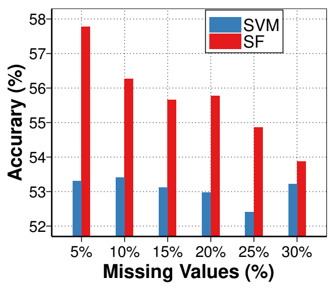
**Figure 4: RBF Kernel: Impact of missing similarity values on *SimForest* (SF) and SVM (best viewed in color).**



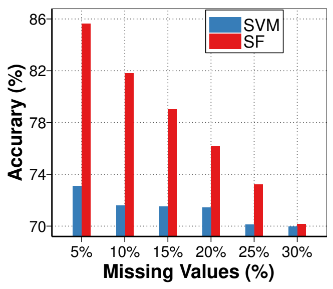
(a) Ionosphere



(b) Breast-Cancer



(c) Madelon



(d) Splice

**Figure 5: Cosine Similarity: Impact of missing similarity values on *SimForest* (SF) and SVM (best viewed in color).**

matrix has $10^6$ training points, one can hardly be expected to compute all $10^{12}$ similarity values between training pairs. At the end, one is often left to using simple extrapolation rules, such as the column-wise mean for imputation, when using a similarity-based classifier like an SVM. Very few classifiers have the ability to learn from incomplete data, particularly when the data is not specified in multidimensional form, but in the form of highly incomplete similarity matrices. A fundamental advantage of *SimForest* is that it can be built on similarity matrices with missing values (as discussed in section 3.4), without having to explicitly impute these values.

**Table 3: Accuracy observed when 15% of similarity values are missing.**

| Data set | RBF Kernel | | Cosine Similarity | |
|---|---|---|---|---|
| | *SimForest* | SVM | *SimForest* | SVM |
| German-Numer | **72.25** | 67.60 | **72.05** | 64.25 |
| Australian | **90.14** | 83.47 | **89.63** | 86.23 |
| Ionosphere | **95.49** | 77.46 | **92.11** | 76.47 |
| Breast-Cancer | **97.00** | 90.94 | **96.42** | 92.07 |
| Madelon | **56.11** | 53.91 | **55.65** | 53.11 |
| Splice | **79.81** | 74.94 | **79.02** | 71.53 |

As in the previous section, we construct two different types of similarity matrices with the cosine and RBF similarity functions, and assume that only the similarity matrices are available without access to the multidimensional attributes. However, in this case, we drop a certain fraction of the similarity values, and allow access to only the retained values. The SVM model cannot work directly with missing values in the similarity matrix. Therefore, when such values were required by the SVM, the column-wise mean of the similarity matrix is used instead. In the case of the *SimForest*, the missing-value approach in Section 3.4 was used. The experiments were repeated for 10 different random samples of the missing entries for all methods and data sets, and the performance values were averaged to give stable results. Table 3 shows the results for the case when 15% of the entries are missing. The experiments are performed on the German-Numer, Australian, Ionosphere, Breast-Cancer, Madelon, and Splice data sets. The average improvement of *SimForest* across all data sets was 9.23% for the RBF kernel, while it was 9.42% for the cosine similarity. Thus, *SimForest* not only works in the presence of missing values without the need for imputation, but is also remarkably accurate when similarities are incompletely specified.

How do the fraction of the missing values affect the results? We show the sensitivity of *SimForest* to an increasing percentage of missing values. The sensitivity results are shown on four data sets, which are Ionosphere, Breast-Cancer, Madelon, and Splice. These results are shown in Figures 4 and 5 for the RBF kernel and cosine similarity, respectively. In both cases, it is evident that the *SimForest* approach is far more tolerant to missing values than SVMs. As the fraction of missing values increases, the *SimForest* approach performs much better than SVMs.

**Table 4: Accuracy on multi-dimensional data sets.**

| Data set | Heart | Ionosphere | Breast-Cancer | German-Numer | SVMGuide3 | a1a | Mushrooms |
|---|---|---|---|---|---|---|---|
| *SimForest* | **83.33** | **100.00** | **96.35** | **78.50** | **90.24** | 82.81 | **100.00** |
| Random Forest | 79.62 | 94.36 | **96.35** | 77.00 | 87.80 | 82.63 | **100.00** |
| SVM | 81.48 | 87.32 | **96.35** | 76.00 | 58.53 | **83.42** | **100.00** |

## 4.5 Multidimensional Data

Next, we illustrate the performance of *SimForest* on multidimensional data, and we assume that access to the multidimensional attributes is available. Although the *SimForest* approach is designed for settings in which only similarities are available, it is easy to use it in cases where multidimensional features are available. There are no required changes to the implementation to adapt it to the multidimensional case, other than the fact that we explicitly compute the similarities (as needed) using the features. An additional advantage is that we can draw from a greater choice of similarity functions. We used the dot products between multidimensional features as the similarity values in the *SimForest*. Note that one can use other types of similarity values as well, although we used the simplest option in this case. To ensure comparability, we also used the dot product as the similarity for the SVM. A key difference in the comparison here is that we can now also include a traditional random forest because the underlying multidimensional feature representation is available. In this experiment, we include the traditional random forest with splits on individual attributes. The number of features sampled at each node for the traditional random forest is chosen as the square root of the data dimensionality, and 100 trees are grown for both random forest and *SimForest*. For the SVM, the value of the slack penalty was tuned using cross-validation. The results are shown in Table 4. Observe that *SimForest* is highly competitive to Random Forest and SVMs, even though *SimForest* was not originally designed for a multidimensional setting. The random forest, however, comes very close to matching the accuracy of *SimForest*. This is hardly surprising because the random forest is conceptually related to the *SimForest*, and is one of the best available classifiers. The key takeaway from the results on multidimensional data is not necessarily to advocate the use of *SimForest* in the multidimensional setting, but to emphasize the fact that it squeezes the most that is possible from a particular similarity matrix. This is evidenced by the fact that even direct access to the actual features does not help a traditional random forest win over *SimForest*. This indirectly suggests that the performance of *SimForest* is formidably strong for the case where one does not have access to the multidimensional features, and only the similarity matrices are available.

## 5 CONCLUSIONS AND SUMMARY

In this paper, we presented a method for constructing random forests from arbitrary object data types by using a similarity-centric approach to decision-tree construction. Our approach provides the ability to incorporate domain-specific similarity functions within the mining process. Only a small fraction of the pairwise similarities are computed, and therefore all pairwise similarities do not need to be materialized. Even where many of the pairwise similarities are missing, one can use such an approach for prediction. Although the approach is naturally designed for the case where the input is specified in the form of similarity matrices, it can be used for multidimensional data by using the derived similarity matrices of these objects. In these settings, the technique is competitive with off-the-shelf methods, which shows that *SimForest* gets the most out of any particular similarity function for the purposes of classification. The experimental results show that our approach has significant advantages over the baseline techniques of support vector machines and random forests, which are state-of-the-art methods for classification.

## REFERENCES

[1] C. Aggarwal. Data Classification: Algorithms and Applications, *CRC Press*, 2014.
[2] C. Aggarwal. Data Mining: The Textbook, *Springer*, 2016.
[3] C. Aggarwal and S. Sathe. Outlier Ensembles: An Introduction, *Springer*, 2017.
[4] L. Brieman. Random Forests. *Journal Machine Learning archive*, 45(1), pp. 5–32, 2001.
[5] L. Brieman and A. Cutler. Random Forests Manual v4.0, *Technical Report, UC Berkeley*, 2003.
[6] C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2), pp. 121–167, 1998.
[7] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3), pp. 273–297, 1995.
[8] M. Fernandez-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?. *The Journal of Machine Learning Research*, 15(1), pp. 3133–3181, 2014.
[9] A. Graf, A. Smola, and S. Borer. Classification in a normalized feature space using support vector machines. *IEEE Transactions on Neural Networks*, 14(3), 2003.
[10] G. E. Hinton and M. Revow. Using pairs of data points to define splits for decision trees. In *NIPS* pp. 507-513, 1996.
[11] T. K. Ho. Random decision forests. *Third International Conference on Document Analysis and Recognition*, 1995. Extended version appears in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8), pp. 832–844, 1998.
[12] M. Kretowski. An evolutionary algorithm for oblique decision tree induction. In *International Conference on Artificial Intelligence and Soft Computing*, pp. 432-437, 2004.
[13] L. Kuncheva and J. Rodrguez. An experimental study on rotation forest ensembles. In *Multiple Classifier Systems*, pp. 459–468, 2007.
[14] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation Forest. *ICDM Conference*, 2008. Extended version appears as "Isolation-based Anomaly Detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1), 3, 2012.
[15] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb), pp. 419–444, 2002.
[16] E. Scornet. Random forests and kernel methods. *IEEE Transactions on Information Theory*, 62(3), pp. 1485-1500, 2016.
[17] Y. Qi, J. Klein-Seetharaman, and Z. Bar-Joseph. Random forest similarity for protein-protein interaction prediction from multiple sources. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, 2004.
[18] J. Rodriguez, L. Kuncheva, and C. Alonso. Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10), pp. 1619–1630, 2006.
[19] B. Schölkopf, A. Smola, K. Muller. Kernel principal component analysis. *International Conference on Artificial Neural Networks*, pp. 583–588, 1997.