Random Intersection Trees

Rajen Dinesh Shah and Nicolai Meinshausen
Statistical Laboratory, University of Cambridge
Department of Statistics, University of Oxford
r.shah@statslab.cam.ac.uk and meinshausen@stats.ox.ac.uk

September 30, 2013

Abstract

Finding interactions between variables in large and high-dimensional datasets is often a serious computational challenge. Most approaches build up interaction sets incrementally, adding variables in a greedy fashion. The drawback is that potentially informative high-order interactions may be overlooked. Here, we propose at an alternative approach for classification problems with binary predictor variables, called *Random Intersection Trees*. It works by starting with a maximal interaction that includes all variables, and then gradually removing variables if they fail to appear in randomly chosen observations of a class of interest. We show that informative interactions are retained with high probability, and the computational complexity of our procedure is of order p^κ, where p is the number of predictor variables. The value of κ that can reach values as low as 1 for very sparse data; in many more general settings, it will still beat the exponent s obtained when using a brute force search constrained to order s interactions. In addition, by using some new ideas based on min-wise hash schemes, we are able to further reduce the computational cost. Interactions found by our algorithm can be used for predictive modelling in various forms, but they are also often of interest in their own right as useful characterisations of what distinguishes a certain class from others.

Key words: High-dimensional classification, Interactions, Min-wise hashing, Sparse data.

1 Introduction

In this paper, we consider classification with high-dimensional binary predictors. We suppose we have data that can be written in the form (Y_i, X_i) for observations $i = 1, \ldots, n$; Y_i is the class label and $X_i \subseteq \{1, \ldots, p\}$ is the set of active predictors for observations i (out of a total of p predictors). An important example of this type of problem is that of text classification, where then X_i is the set of frequently appearing words (in a suitable sense) for document i, and Y_i indicates whether the document belongs to a certain class. In this case, the dimension p can be of the order of several thousand or more. More generally, if data with continuous predictors are available, they can be converted to binary format by choosing various split-points, and then reporting whether or not each variable exceeds each of these thresholds.

Our aim here is to develop methodology that can discover important interaction terms in the data without requiring that any of their lower order interactions are also informative. More precisely, we are interested in finding subsets $S \subseteq \{1, \ldots, p\}$ of all predictor variables that occur more often for observations in a class of interest than for other observations. We will use the terms "leaf nodes", "rules", "patterns" and "interactions" interchangeably to describe such subsets S. For simplicity, suppose there are only two classes, the set of labels being $\{0, 1\}$. The case with more than two classes can be dealt with using one-versus-one, or one-versus-all strategies. Given a pair of thresholds, $0 \leq \theta_0 < \theta_1 \leq 1$, our goal is to find all sets S (or as many as possible), for which

$$\mathbb{P}_n(S \subseteq X|Y = 1) \geq \theta_1 \quad \text{and} \quad \mathbb{P}_n(S \subseteq X|Y = 0) \leq \theta_0. \tag{1.1}$$

Here and throughout the paper, we use the subscript n to indicate that the probabilities are empirical probabilities. For example, for $c \in \{0, 1\}$,

$$\mathbb{P}_n(S \subseteq X|Y = c) := \frac{1}{|C_c|} \sum_{i \in C_c} \mathbb{1}_{\{S \subseteq X_i\}},$$

where we have denoted the set of observations in class c by C_c. Of course, one would also be interested in sets S which satisfy a version of (1.1) with classes 1 and 0 interchanged, but we will only consider (1.1) for simplicity.

The interaction terms uncovered can be used in various ways. For example, they can be built into tree-based methods, or form new features in linear or logistic regression models. The interactions may also be of interest in their own right, as they can characterise distinctions between classes in a simple and interpretable way. These potentially high-order interactions that our method aims to target would be very difficult to discover using existing methods, as we now explain.

A pure brute force search examines each potential interaction S of a given size to check whether it fulfils (1.1). Restricting the order of interactions to size s, the computational complexity scales as p^s, rendering problems with even moderate values of p infeasible.

Instead of searching through every possible interaction, tree-based methods build up interactions incrementally. A typical tree classifier such as *CART* [Breiman et al., 1984] works by building a decision tree greedily from root node to the leaves; see also Loh and Shih [1997]. The feature space is recursively partitioned based on the variable whose presence or absence best distinguishes the classes. The myopic nature of this strategy makes it a computationally feasible approach, even for very large problems. The downside is that it produces rather unstable results: small changes in the data can lead to very different partitions being produced at the leaf nodes. Moreover, because of the incremental way in which interactions are constructed, the success of this strategy in recovering an important interaction S rests on at least some of its lower order interactions being informative for distinguishing the classes.

Approaches based on tree ensembles can somewhat alleviate the problem of tree instability; *Random Forests* [Breiman, 2001] is a prominent example. Here the data with which the decision trees are constructed is sampled with replacement from the original data. Further randomness is introduced by randomising over the subset of variables considered for each split in the construction of the trees. While the results of *Random Forests* are very complex and hard to interpret, one can examine what are known as variable importance measures. These aim to quantify the marginal or pairwise importance of predictor variables [Strobl et al., 2008]. Though such measures can be useful, checking through all possible high-order interactions is too cumbersome, and so these may fail to be highlighted.

More recently, there has been interest in algorithms that start from deep splits or leaf nodes in trees and then try to build a simpler model out of many thousands of these leaves by regularisation and dimension reduction. Examples include *Rule Ensembles* [Friedman and Popescu, 2008], *Node Harvest* [Meinshausen, 2010] and the general framework of *Decision Lists* [Marchand and Sokolova, 2006, Rivest, 1987]. Though these methods have been demonstrated to improve on *Random Forests* in some situations, they nevertheless crucially rely on a good initial basis of leaf nodes. These bases are usually generated by tree ensemble methods and so, if the base trees miss some important splits, they would also be absent in the results of these derivative algorithms.

A complementary approach has developed in data mining under the name of frequent itemset search, starting with the *Apriori* algorithm [Agrawal et al., 1994], which has since then developed into many improved and more specialised forms. The starting point for these was "market basket analysis", where the shopping behaviour of customers is analysed and the goal is to identify baskets that are often bought together. Many algorithms have been proposed that aim to improve on *Apriori* in terms of memory requirements and speed, such as the *FP-growth* [Han et al., 2000] and *H-mine* [Pei et al., 2001] algorithms. While generally very successful, all these methods are only computationally feasible in large-scale settings if among the itemsets of low size, there are many that are infrequent, and so using the principle that subsets of frequent itemsets are also frequent, the search space can be greatly reduced. However, if small itemsets all have roughly the same frequency, these methods cannot greatly improve over a brute force search.

We now give a simple example where tree-based approaches and those based on the *Apriori* algorithm will struggle. Let $Z = (Z_1, \dots, Z_p) \in \{0,1\}^p$ be a random variable with p independent components each having a Bernoulli(1/2)-distribution. We take X to be the set of active entries $\{k : Z_k = 1\}$. Suppose the response $Y \in \{0,1\}$ is determined by an interaction between the first two variables such that $Y = \mathbb{1}_{\{Z_1+Z_2 \neq 1\}}$. Then none of the variables have a marginal effect as Y is independent of Z_k for all $k = 1, \dots, p$. In this case, when using trees or the *Apriori* algorithm, one would have to search among $O(p^2)$ potential interactions to find the interaction pattern $\{1, 2\}$.

This paper looks at a new way to discover interactions, which we call *Random Intersection Trees*. Rather than searching through potential interactions directly, our method works by looking for collections of observations whose common active variables together form informative interactions. We present a basic version of the *Random Intersection Trees* algorithm in the following section. This approach allows for computationally feasible discovery of interactions in settings where most existing procedures would perform poorly. Bounds on the complexity of our algorithm are given in Section 3. For example, our results yield that in the scenario discussed in the previous paragraph, the order of computational complexity of our method is at most $o(p^\kappa)$ for any $\kappa > 1$. In Section 4, we propose some modifications of our basic method to reduce its computational cost, based on min-wise hash schemes. Some numerical examples are given in Section 5. We conclude with a brief discussion in Section 6, and all technical proofs are collected in the appendix.

2 Random Intersection Trees

Our method searches for important interactions by looking at intersections of randomly chosen observations from class 1. We start with the full set of variables as an interaction and then iteratively prune away variables to make the interaction smaller. At each iteration, we just keep variables in the interaction that are present in a new randomly chosen observation of class 1. All variables in the interaction that are not present in the chosen observation are removed. Then we repeat with a new randomly chosen observation until an interaction of the desired size emerges. If a pattern S has high prevalence in class 1, i.e. $\mathbb{P}_n(X = S|Y = 1)$ is large, it will be included in the observations chosen with high probability. Thus, provided the overall process is repeated often enough, S is likely to be retained in some of the final intersections. On the other hand, elements in S^c, the complement of S in $\{1, \ldots, p\}$, are unlikely to be present in all the observations being intersected. Thus of those intersections which contain S, there is a good chance that at least one of them is exactly S. Arranging the procedure in a tree-type search makes performing the intersections more computationally efficient; details are given in the following section. One would then consider each of these intersections as possible solutions of (1.1), checking whether their prevalence among class 0 is below θ_0.

It may at first seem strange that in the above, class 0 plays a part in the procedure only at the very end. One might expect that many candidate interactions could be generated that have high prevalence in both classes 1 and 0 and thus would not be useful for distinguishing between classes. In Section 4, we do present an improved version of our algorithm that makes use of class 0 at an earlier stage. However, in the sparse setting we are considering here, interactions with high prevalence in either class would typically be rather few in number. Thus even if all interactions with high prevalence in class 1, and not necessarily low prevalence in class 0, were generated by the procedure outlined above, this would be a manageable number of candidate sets. Note that the assumptions that allow this to happen certainly do not trivialise the problem: even if, given all solutions to the first equation in (1.1), it is easy to uncover those interactions that additionally satisfy the second equation, the first part of the task is still very challenging.

To describe the details of our algorithm, we first define some terms associated with trees that will be needed later. Recall that a tree is a pair (N, E) of nodes and edges forming a connected acyclic (undirected) graph. We will always assume (with no loss of generality) that $N = \{1, \ldots, |N|\}$. A *rooted tree* is the directed acyclic graph obtained from a tree by designating one node as root and directing all edges away from this root.

Let α and β be two nodes in a rooted tree, with β not the root node. If $(\alpha, \beta) \in E$, β is said to be the *child* of α, and α, the *parent* of β. We will denote by $\mathrm{ch}(\alpha)$, the set of children of a node α. Since we are only considering rooted trees here as opposed to general directed graphs, we will differ with convention slightly and will use $\mathrm{pa}(\beta)$ to mean the unique parent of β. Thus here, $\mathrm{pa}(\beta)$ is a node itself, whereas $\mathrm{ch}(\alpha)$ is a set of nodes.

If $\alpha \neq \beta$ lies on the unique path from the root to β, we say α is an *ancestor* of β, and β is a *descendant* of α. We denote the sets of all ancestors and descendants of α by $\mathrm{an}(\alpha)$ and $\mathrm{de}(\alpha)$ respectively. The *depth* of α, denoted $\mathrm{depth}(\alpha)$, is the number of ancestors of α: $\mathrm{depth}(\alpha) = |\mathrm{an}(\alpha)|$. In particular, the depth of the root node is 0. The *depth* (also known as the *height*) of a rooted tree is the length of the longest path, or equivalently, the greatest number of ancestors of any particular node. By *level* d of the tree, we will mean the set of nodes with depth d.

We will say an indexing of the nodes is *chronological* if, for every parent and child pair, larger indices are assigned to the child than the parent. In particular, the root node will be 1. Note that both depth-first and breadth-first indexing methods are chronological in this way.

Algorithm 1 describes a basic version of the *Random Intersection Trees* procedure. The reason for allowing random choices of children is for the proof of Theorem 1, where we can randomly choose the number of children to be in $\{b, b + 1\}$ for a suitable integer value b.

Algorithm 1 A basic version of Random Intersection Trees

```
for tree $m = 1$ to $M$ do
    Let $m$ be a rooted tree of depth $D$, with each node $j$ in levels $0, \dots, D-1$ having $B_j$ children, where the $B_j$ are i.i.d. with a pre-specified distribution. Denote by $J$ the total number of nodes in the tree, and index the nodes chronologically. For each of the nodes $j = 1, \dots, J$, let $i(j)$ be an independently and uniformly chosen index in the set of class 1 observations $\{i : Y_i = 1\}$.
    Set $S_1 = X_{i(1)}$.
    for node $j = 2$ to $J$ do
        Set $S_j = X_{i(j)} \cap S_{\mathrm{pa}(j)}$.
    end for
    Denote the collection of resulting sets from all nodes at depth $d$, for $d = 1, \dots, D$, by $L_{d,m} = \{S_j : \text{depth}(j) = d\}$.
end for
return candidate set of interactions $L_D := \bigcup_{m=1}^{M} L_{D,m}$.
```

Although we have allowed the number of children of each non-leaf node in the trees to be random, in practice we would take this as a fixed number.

Looking at the innermost for-loop, we see that each node in each tree is associated with a randomly drawn observation from class 1. For every tree, we visit each non-root node in turn, and compute the intersection of the observation assigned to it, and all those assigned to its ancestors. Because of the way the nodes are indexed, parents are always visited before their children, and this intersection can simply be computed as $S_j = X_{i(j)} \cap S_{\mathrm{pa}(j)}$. This is crucial to reducing the computational complexity of the procedure, as we shall see in the next section.

Each of the sets assigned to the leaf nodes of each of the trees yields a collection of potential candidate interactions, L_D. One could then proceed to test these as potential solutions to (1.1); we present a more efficient approach in Section 4, where we build this testing step into the construction of the trees.

An illustration of this improved algorithm applied to the Tic-Tac-Toe data discussed in Section 5 is given in Figure 1. Observations here correspond to winning endgame positions, coded such that the data is binary. Class labels record which player (black or white) won the game, and the goal is to infer the interactions (corresponding to positions of a few counters) that lead to a win for each player. In this example, the root node contains a randomly drawn final win-state for black (class 1). This corresponds to S_1 in our algorithm. For each other node j, we draw a new random observation $i(j)$ from all class 1 observations. The randomly chosen additional black-win state $X_{i(j)}$ is shown along the edge from its parent node. The new intersection, S_j, is the intersection of the interaction in the parent node and the new set $X_{i(j)}$; it is shown in the corresponding node. The early stopping added in the improved algorithm also allows it to run until the algorithm has terminated in all nodes. Thus no prior specification of the tree depth will be necessary in practice, as will be shown in Section 4.

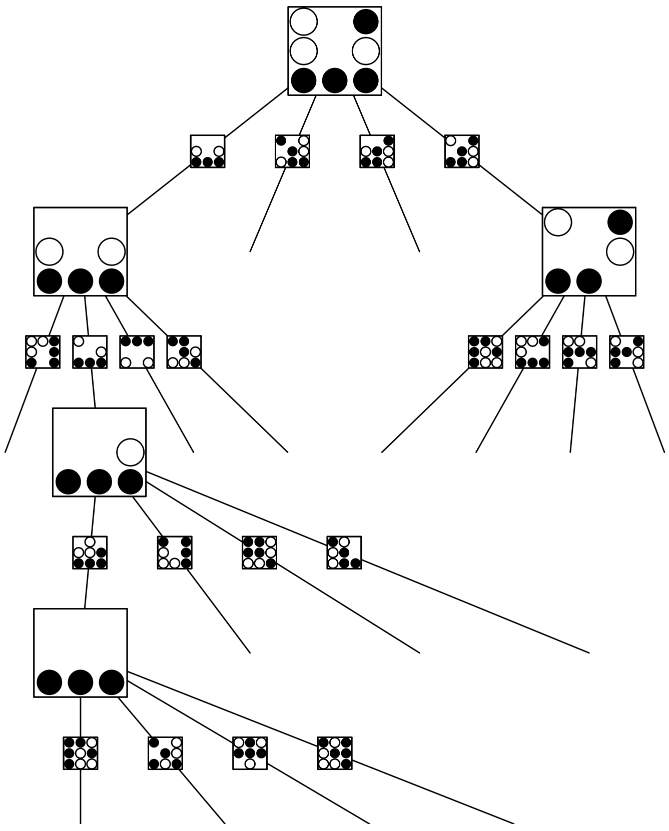

Figure 1: *An intersection tree for the Tic-Tac-Toe game dataset. Given winning positions of the black player, we intersect them randomly to produce the interactions (corresponding to positions of black or white stones) that are responsible for wins. Starting with a randomly chosen class 1 (black wins) observation at the root node, $B = 4$ randomly chosen class 1 observations are intersected with the pattern. These randomly chosen observations are shown along the edges and the resulting intersections S_j as the nodes in the next layer of the tree. Nodes are only shown if the corresponding patterns S_j have an estimated prevalence among class 0 below a set threshold; the branching of the tree terminates for all other nodes. The algorithm continues until all resulting S_j corresponding to the leaf nodes have prevalence among class 0 exceeding the threshold. Here, one of the winning states for black is filtered out after three intersections.*

3 Computational complexity

How many trees do we have to compute to have a very high probability of finding an interesting interaction S that fulfils (1.1)? And what is the required size of these trees? If the interaction is not associated with a main effect, most approaches like trees and association rules would require of order $p^{|S|}$ searches. In this section, we show that in many settings, *Random Intersection Trees* improves on this complexity. We consider a single interaction S of size $s = |S|$, and examine the computational cost for returning S as one of the candidate interactions, with a given probability. We will see that this depends critically on three factors:

- **Prevalence** $\theta_1 := \mathbb{P}_n(S \subseteq X|Y = 1)$ of the interaction pattern. If the pattern S in question appears frequently in class 1, the search is more efficient.
- **Sparsity** $\delta_k := \mathbb{P}_n(k \in X|Y = 1)$ of the predictor variables $k = 1, \ldots, p$. If δ_k is very low for many k (and sparsity of predictors consequently high), computation of the intersections is much cheaper, and so overall computational cost is greatly reduced. Indeed, for a fixed tree m, consider a node j with depth $d < D$. We have that

  $$\mathbb{E}(|S_j|) = \sum_{k=1}^{p} \delta_k^{d+1}.$$

  Thus, for $j' \in \mathrm{ch}(j)$, computation of $S_{j'}$ requires on average at most

  $$O\left(\log(p) \sum_{k=1}^{p} \delta_k^{d+1}\right)$$

  operations. This is because in order to compute the intersection, one can check whether each member of S_j is in $X_{i(j')}$, and each such check is $O(\log(p))$ if the sets X_i are ordered so a binary search can be used. If we compare this to the $O(p)$ computations required to calculate each of the S_j if no tree structure were used, we see that large efficiency gains are possible when $d \geq 1$ if many variables are sparse. For intersections with the root node, the tree structure offers no advantage, and in practice, branching the tree only after level 1 (so the root node has only one child), is more efficient, though this modification does not improve the order of complexity.
- **Independence of** S: Define $\nu := \max_{k \in S^c} \mathbb{P}_n(k \in X|S \subseteq X, Y = 1)$. If ν is low, less computational effort is required to recover S. Note that if, for some $k \in S^c$, $\mathbb{P}_n(k \in X|S \subseteq X) = 1$, interest would centre on $S \cup \{k\}$ rather than S itself. Indeed, if S satisfied (1.1), so would $S \cup \{k\}$. In general, if ν is large, the search will tend to find sets containing S, though not necessarily S itself.

With the assumptions that $\theta_1 > 0$ and $\nu < 1$, we can give a bound on the computational complexity of the basic version of *Random Intersection Trees* introduced in the previous section.

Let us define

$$C(M, D, F_B)$$

to be the expected number of computations required to perform all the intersections in the algorithm when M trees of depth D are created and the distribution of the branching factors B_j is F_B.

Theorem 1. *Given $\eta, \epsilon \in (0,1]$, there exist choices of M, D and F_B such that the set L_D returned by Algorithm 1 contains S with probability at least $1-\eta$, and*

$$C(M, D, F_B) = O\left[\log(1/\eta)\frac{\log^2(p)}{\epsilon}\left\{p + \sum_{k:\,(1+\epsilon)\delta_k > \theta_1} p^{\frac{\log\{(1+\epsilon)\delta_k/\theta_1\}}{\log(1/\nu)}}\right\}\right]. \tag{3.1}$$

As a function of the number of variables p, there is a contribution of $p\log^2(p)$ and an additional contribution in the brackets that depends on the sparsity δ_k of each variable. Sparse variables do not contribute to this sum, which can be $O(1)$ if sparsity among variables is high enough. This would yield a computational complexity with order bounded above by $o(p^\kappa)$ for any $\kappa > 1$, compared to the corresponding complexity of p^s for a brute force search. In most interesting settings, however, we would not achieve a nearly linear scaling in complexity, but would hope to still be faster than a brute force search.

Before discussing the result further, we comment briefly on the values of M, D and the distribution of the B_j, that yield (3.1). From the proof, it follows that there exist choices of M and D that give (3.1) that satisfy

$$M \le \frac{(1+2\epsilon)\log(1/\eta)}{2\epsilon\theta_1},$$
$$D \le \frac{\log\{p(1+2\epsilon)\}}{\log(1/\nu)}.$$

The random number B_j used in the proof takes just one of two consecutive integers (essentially to avoid the discretisation effect when being restricted to integers), and $\mathbb{E}(B_j) \le (1+\epsilon)/\theta_1$. Though the optimal choices of parameters for the theorem depend on the unknown ν and the minimising ϵ, which will in turn depend on ν, the functional relationships given above still provide rough qualitative guidelines for good choices for these parameters in practice.

Using the values of M, D and B_j necessary to guarantee that the set S is with high probability in the set L_D, we can also obtain a bound on the expected number of candidate interaction sets in L_D. This will in turn bound the expected number of "false positives" returned. The expected number of returned sets is bounded by

$$\mathbb{E}(|L_D|) \le M\mathbb{E}(B_j)^D \le \frac{\log(1/\eta)}{\epsilon}\left\{\frac{(1+2\epsilon)p}{\nu}\right\}^{\frac{\log(1+\epsilon)/\theta_1}{\log(1/\nu)}}$$

The value of ϵ can be chosen to minimise the bound above, but its value here and in the computational complexity bound of Theorem 1 have to be the same, as they are linked to the choice of the branching factor used when building the trees. We see that in many situations, we can expect the bound above to be very much lower than the $O(p^s)$ sets a complete list of s-way interactions would contain. Note that if s were known, the relevant quantity to consider would be

$$\mathbb{E}(|\{S' \in L_D : |S'| = s\}|),$$

which is likely to be much less than $\mathbb{E}(|L_D|)$. Even if s were unknown, one would only be interested in the expected number of non-empty sets in L_D, a quantity which may well also be substantially lower than the derived bound on $\mathbb{E}(|L_D|)$.

The influence of sparsity on computational complexity. It is interesting to make the influence of the sparsity of individual variables, δ_k, on the overall computational complexity, more explicit. We have the following corollary to Theorem 1.

Corollary 2. *Define β by $\nu = \theta_1^\beta$. Suppose that $\gamma, \alpha^\star, \alpha_\star$ are such that $\alpha^\star > \alpha_\star$, and*

$$\begin{aligned} \delta_k &\leq \theta_1^{1-\alpha^\star} && \text{for all } k \in \{1, \dots, p\} \\ \delta_k &> \theta_1^{1-\alpha_\star} && \text{for at most } p^\gamma \text{ variables.} \end{aligned}$$

Given $\eta \in (0,1]$, there exist choices of M, D and F_B such that the set L_D returned by Algorithm 1 contains S with probability at least $1-\eta$, and

$$C(M, D, F_B) = o\big(p^\kappa\big) \qquad \text{for any } \kappa > \max\left\{\frac{\alpha^\star}{\beta} + \gamma,\ \left\lceil\frac{\alpha_\star}{\beta}\right\rceil_+ + 1\right\}. \tag{3.2}$$

The implication of Corollary 2 is most apparent if we take $\gamma = 1$ as we can then set $\alpha_\star = 0$. In this case,

$$\alpha^\star = 1 - \frac{\log(\max_k \delta_k)}{\log(\theta_1)}.$$

We can then bound the computational complexity by

$$o\big(p^\kappa\big) \qquad \text{for any } \kappa > 1 + \frac{\log(1/\theta_1) - \log(1/\max_k \delta_k)}{\log(1/\nu)}. \tag{3.3}$$

The fraction on the right-hand side is a function of the prevalence of the pattern S, θ_1, the maximum sparsity of the variables, and the maximum sparsity of the variables in S^c, conditional on the presence of S. As long as this fraction is less than 1, the computational complexity is guaranteed to be better than a brute force search with the knowledge that $s = 2$, and the relative advantage grows for larger sizes of the pattern.

Independent noise variables. To gain further insight, we consider the special case where variables in S^c are independent of S (conditional on being in class 1), in the sense that for all $k \in S^c$,

$$\mathbb{P}_n(k \in X | S \subseteq X, Y = 1) = \mathbb{P}_n(k \in X | Y = 1) = \delta_k. \tag{3.4}$$

Corollary 3. *Assume* (3.4) *and that $\delta_k < 1$ for all k. Given $\eta \in (0,1]$, there exist choices of M, D and F_B such that the set L_D returned by Algorithm 1 contains S with probability at least $1-\eta$, and*

$$C(M, D, F_B) = o(p^\kappa) \qquad \text{for any } \kappa > \frac{\log(1/\theta_1)}{\log(1/\max_k \delta_k)}. \tag{3.5}$$

We see that the computational complexity is approximately linear in p if the prevalence of the pattern S is as high as the prevalence of the least sparse predictor variables. This is the case in the example mentioned in the introduction, where $\theta = \delta_k = 1/2$.

We can also consider the situation where in addition to the independence (3.4), all variables have the same sparsity δ. If the prevalence θ_1 of S is only as high as that of a random occurrence of two independent predictor variables, we get $\kappa > 2$ and the computational complexity is approximately quadratic in p. In this case, the algorithm would not yield a computational advantage over brute force search if looking for patterns of size 2. This is to be expected since *every* pattern S of size 2 would have the same prevalence in this scenario, and so there is nothing special about a pattern S of size 2 with prevalence δ^2, and in general no hope of beating the complexity p^s of a brute force search. However, the bound in (3.5) is independent of s. Thus provided the prevalence, θ_1, drops more slowly than the rate δ^s, at which every pattern of size S would occur randomly among independent predictor variables, our results show that *Random Intersection Trees* is still to be preferred over a brute force search.

4 Early stopping using min-wise hashing

While Algorithm 1 is computationally attractive, the following observation suggests that further improvements are possible. Suppose that, for a particular tree, we have just computed the intersection S_j corresponding to a node j at depth $d < D$. If

$$\mathbb{P}_n(S_j \subseteq X | Y = 0) > \theta_0,$$

then since for all $j' \in \text{de}(j)$, $S_{j'} \subseteq S_j$, we also have

$$\mathbb{P}_n(S_{j'} \subseteq X | Y = 0) > \theta_0.$$

Thus no intersection sets corresponding to descendants of j have any hope of yielding solutions to (1.1), and so all further associated computations are wasted.

In view of this, one option would be to compute the quantity $\mathbb{P}_n(S_j \subseteq X|Y = 0)$ at each node j as the algorithm progresses, and if this exceeds the threshold θ_0, not visit any descendants j' of j for computation of $S_{j'}$. This could be prohibitively costly, though, as it would require a pass over all observations in class 0, for each node of each tree. One could work with a subsample of the observations, but if θ_0 is low, the subsample size may need to be fairly large in order to estimate the probabilities to a sufficient degree of accuracy.

Instead, we propose a fast approximation, using some ideas based on min-wise hashing [Broder et al., 1998, Cohen et al., 2001, Datar and Muthukrishnan, 2002] applied to the columns of the data-matrix. We describe the scheme by leaving aside the conditioning on $Y = 0$, which can be added at the end by restricting to observations in class 0. Consider taking a random permutation σ of all observations $\{1, \dots, n\}$. Let $h_\sigma(k)$ be the minimal value ι such that variable k is active in observation $\sigma(\iota)$:

$$h_\sigma(k) = \min\{\iota' : k \in X_{\sigma(\iota')}\}.$$

It is well known [Broder et al., 1998] that the probability that $h_\sigma(k)$ and $h_\sigma(k')$ agree for two variables k, k' under a random permutation σ is identical to the Jaccard-index for the two sets $I_k = \{i : k \in X_i\}$ and $I_{k'} = \{i : k' \in X_i\}$, that is

$$\mathbb{P}_\sigma(h_\sigma(k) = h_\sigma(k')) = \frac{|I_k \cap I_{k'}|}{|I_k \cup I_{k'}|}.$$

Here the subscript σ indicates that the probability is with respect to a random permutation σ of the observations. A min-wise hash scheme is typically used to estimate the Jaccard-index by approximating the probability on the left-hand side of the equation above.

Now,

$$\begin{aligned}\mathbb{P}_n(S \subseteq X) = \mathbb{P}_n(k \in X \text{ for all } k \in S) \\ = \mathbb{P}_n(k \in X \text{ for all } k \in S \,|\, \exists\, k' \in S \text{ such that } k' \in X) \\ \times \mathbb{P}_n(\exists k \in S \text{ such that } k \in X).\end{aligned}$$

Let us denote the first and second terms on the right-hand side by $\pi_1(S)$ and $\pi_2(S)$ respectively. Note that $\pi_1(S)$ is equal to the probability that all variables $k \in S$ have the same min-wise hash value $h_\sigma(k)$:

$$\pi_1(S) = \mathbb{P}_\sigma(\exists\, \iota : h_\sigma(k) = \iota \text{ for all } k \in S). \tag{4.1}$$

Turning now to $\pi_2(S)$, observe that

$$\mathbb{E}_\sigma(\min_{k \in S} h_\sigma(k)) = \frac{n+1}{\pi_2(S)n+1}, \tag{4.2}$$

and so

$$\pi_2(S) = \frac{n+1}{n}\left\{\frac{1}{\mathbb{E}_\sigma(\min_{k\in S} h_\sigma(k))} - \frac{1}{n+1}\right\}. \tag{4.3}$$

A derivation of (4.2) is given in the appendix.

Equations (4.1) and (4.3) provide the basis for an estimator of $\mathbb{P}_n(S \subseteq X)$. First we generate L random permutations of $\{1, \dots, n\}$: $\sigma_1, \dots, \sigma_L$. We then use these to create an $L \times p$ matrix H whose entries are given by

$$H_{lk} = h_{\sigma_l}(k).$$

Now we estimate $\pi_1(S)$ and $\pi_2(S)$ by their respective finite-sample approximations, $\hat{\pi}_1(S)$ and $\hat{\pi}_2(S)$:

$$\hat{\pi}_1(L; S, H) := \tfrac{1}{L}\sum_{l=1}^{L} \mathbb{1}_{\{H_{lk} = H_{lk'} \text{ for all } k,k' \in S\}}$$

$$\hat{\pi}_2(L; S, H) := \frac{n+1}{n}\left\{\frac{1}{\frac{1}{L}\sum_{l=1}^{L}\min_{k\in S} H_{lk}} - \frac{1}{n+1}\right\}.$$

Finally, we estimate $\mathbb{P}_n(S \subseteq X)$ by

$$\hat{\mathbb{P}}_n(L; S, H) := \hat{\pi}_1(L; S, H) \cdot \hat{\pi}_2(L; S, H). \tag{4.4}$$

To our knowledge, this use of min-wise hashing techniques, and in particular the estimator $\hat{\pi}_2(L; S, H)$, is new. The estimator enjoys reduced variance compared to that which would be obtained using subsampling, as the following theorem shows.

Theorem 4. *For $\hat{\mathbb{P}}_n(L; S, H)$, $\pi_1(S)$ and $\pi_2(S)$ defined as in (4.4), (4.1), and (4.3) respectively, as $L \to \infty$, we have*

$$\sqrt{L}(\hat{\mathbb{P}}_n(L; S, H) - \mathbb{P}_n(S \subseteq X)) \xrightarrow{d} N(0, \pi_2(S)^2\pi_1(S)(1 - \pi_1(S)\pi_2(S))(1 + \epsilon(n))), \tag{4.5}$$

where

$$\epsilon(n) = \frac{1}{n}\frac{n^{-1} - \pi_2^2 - 2\pi_2 n^{-1}}{\pi_2(\pi_2 + 2n^{-1})(1 + n^{-1})} = O(n^{-1}). \tag{4.6}$$

A derivation is given in the appendix. If we tried to estimate $\pi_1\pi_2$ by evaluating the prevalence of S on a subset of the data of size L, the corresponding estimator multiplied by $\sqrt{L}$ would have variance

$$\pi_2(S)\pi_1(S)(1-\pi_1(S)\pi_2(S)) + o_n(1),$$

where $o_n(1) \to 0$ as $n \to \infty$. Comparing this variance to the variance of the normal distribution in (4.5), we see that a factor of $\pi_2(S)$ is gained: matching the accuracy of subsampling with the min-wise hash scheme would require roughly $1/\pi_2(S)$ times as many samples. By using min-wise hashing, choosing $L = 100$ typically delivers a reasonable approximation as long as we just want to resolve values at $\theta_0 = 0.01$ and above.

An improved version of Algorithm 1, building in the ideas discussed above, is given in Algorithm 2 below. Note that $\hat{\mathbb{P}}_n(S_{\mathrm{pa}(j)}, H)$ need only be computed once for every j with the same parent.

Algorithm 2 Random Intersection Trees with early stopping

```
Compute the $L \times p$ min-wise hash matrix $H$, using only class 0 observations.
for tree $m = 1$ to $M$ do
  Let $m$ be a rooted tree of depth $D$, with each node $j$ in levels $0, \dots, D-1$ having $B_j$
  children, where the $B_j$ are i.i.d. with a pre-specified distribution. Denote by $J$ the total
  number of nodes in the tree, and index the nodes chronologically. For each of the nodes
  $j = 1, \dots, J$, let $i(j)$ be an independently and uniformly chosen index in the set of class 1
  observations $\{i : Y_i = 1\}$.
  Set $S_1 = X_{i(1)}$.
  for node $j = 2$ to $J$ do
    if $\hat{\mathbb{P}}_n(S_{\mathrm{pa}(j)}, H) \le \theta_0$ then
      Set $S_j = X_{i(j)} \cap S_{\mathrm{pa}(j)}$.
    end if
  end for
  Denote the collection of resulting sets of all nodes at depth $d$, for $d = 1, \dots, D$, by
  $L_{d,m} = \{S_j : \ \mathrm{depth}(j) = d\}$.
end for
return $L_D := \bigcup_{m=1}^{M} L_{D,m}$.
```

Early stopping decreases the computational cost of the algorithm as many nodes in the trees generated may not need to have their associated intersections calculated. In addition, the set of candidate intersections L_D will be smaller but the chance of it containing interesting intersections would not decrease by much. These gains comes at a small price, since the min-wise hash matrix H must be computed, and the computational effort going into this will in turn determine the quality of the approximation in (4.4). We have previously shown the complexity bounds in the absence of early stopping and thus avoided the difficulty of making this trade-off explicit. We will use the improved version of *Random Intersection Trees* with early stopping in all the practical examples to follow, taking small values of L in the range of a (few) hundred permutations.

The depth D of the tree is still given explicitly in Algorithm 2. An interesting modification creates the tree recursively. Starting with the root node, B children are added to all leaf nodes of the tree in which the early stopping citerion has not been triggered yet. When the algorithm terminates, all intersections in the leaf nodes of the final tree are collected.

5 Numerical Examples

In this section, we give two numerical examples to provide further insight into the performance of our method. The first is about learning the winning combinations for the well-known game Tic-Tac-Toe. This example serves to illustrate how *Random Intersection Trees* can succeed in finding interesting interactions when other methods fail. The second example concerns text classification. Specifically, we want to find simple characterisations (using only a few words, or word-stems in this case) for classes within a large corpus in a large-scale text analysis application.

5.1 Tic-Tac-Toe endgame prediction

The Tic-Tac-Toe endgame dataset [Matheus and Rendell, 1989, Aha et al., 1991] contains all possible winning end states of the game Tic-Tac-Toe, along with which player (white or black) has won for each of these. There are just under 1000 possible such end states, and our goal is to learn the rules that determine which player wins from a randomly chosen subset of these. We use half of the observations for training, and the other half for testing.

There are 9 variables in the original dataset which can take the values 'black', 'white' or 'blank'. These can trivially be transformed into a set of twice as many binary variables where the first block of variables encodes presence of black and the second block encodes presence of white.

Two properties of this dataset that make it particularly interesting for us here are:

- The presence of interactions is obvious by the nature of the game.
- There are only very weak marginal effects. Knowing that the upper right corner is occupied by a black stone is only very weakly informative about the winner of the game. Greedy searches by trees fail in the presence of many added noise variables and linear models do not work well at all.

We apply *Random Intersection Trees* to finding patterns that indicate a black win (class 1), and also patterns that indicate a white win (class 0). We use the early stopping modifications proposed in Section 4, and create two min-wise hash tables from the available observations in each of the classes, taking $L = 200$. Figure 1 shows how the individual Intersection Trees are constructed and illustrates the use of the early stopping rule. We emphasise that we do not need to specify or know that the winning states are functions of only three variables. We let each tree run until all its branches terminate, and collect all resulting leaves.

Figure 2 illustrates the importance sampling effect of *Random Intersection Trees* when using only the training data, and adding a varying number of noise variables. When adding 100 noise variables, all 16 winning final combinations are among the 40 most frequently chosen patterns. All winning states are chosen hundreds of millions times more often than a random sampling of interactions would pick them.

As discussed in Section 1, the interactions or rules that are found could be entered into any existing aggregation method, such as *Rule Ensembles* [Friedman and Popescu, 2008] or *Decision Lists* [Marchand and Sokolova, 2006, Rivest, 1987]. Here, we consider an even simpler

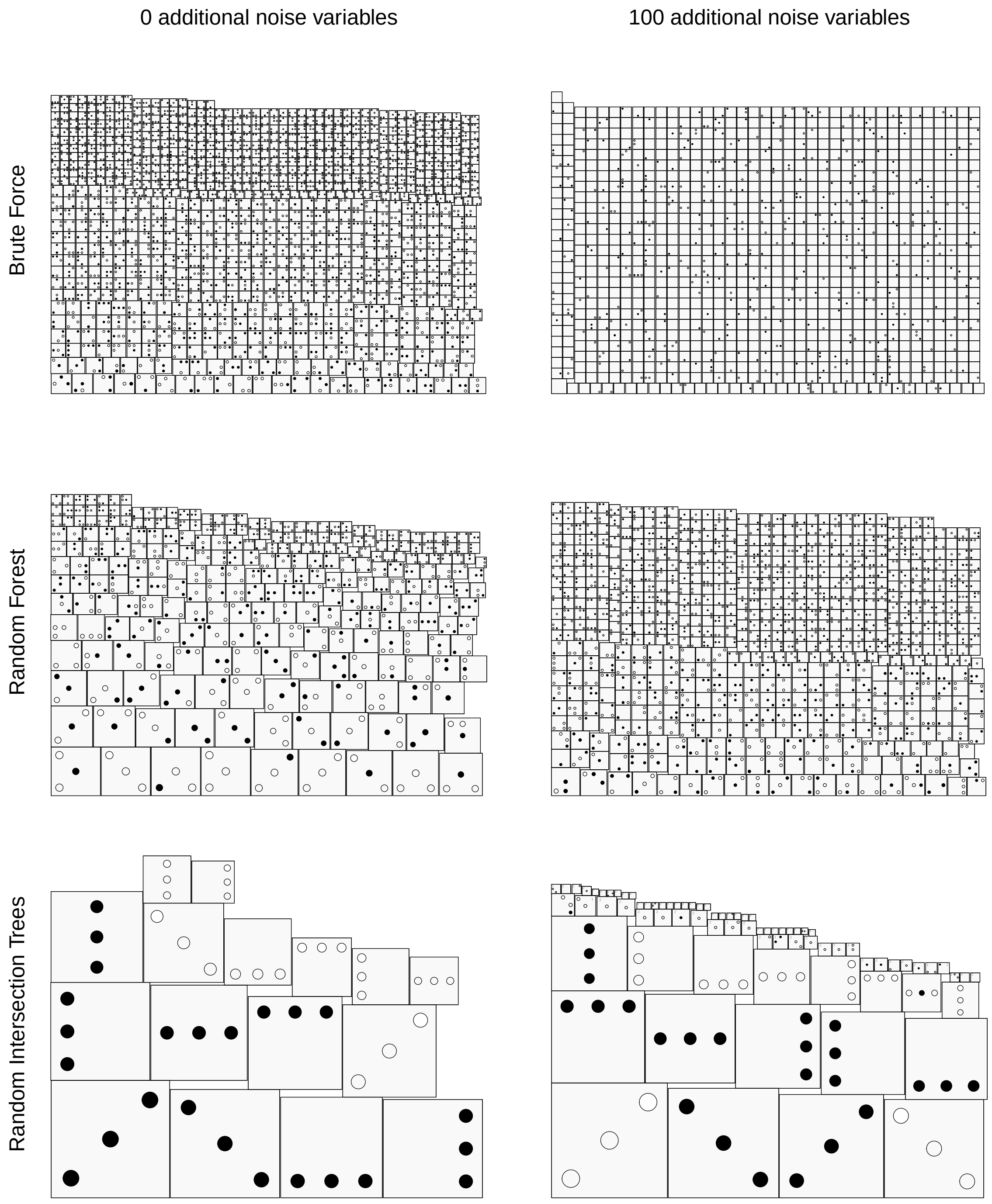

Figure 2: *Left panel: patterns that are returned by Random Intersection Trees (bottom row), Random Forests of depth 3 (middle row) and brute force search among all interactions of size 3 (top row) for the Tic-Tac-Toe data. Each pattern is scaled to make the area proportional to the empirical frequency with which each pattern is found by these search algorithms. Right panel: the same results in the case when 100 noise variables are added. Note that Random Intersection Trees were not constrained to find interactions of depth 3. In the case with noise variables, some of the patterns with the very smallest areas also contained a small number of noise variables, which are not shown. Just counting three- to five-way interactions, there are more than* 10^8 *potential interactions when 100 noise variables are added.*

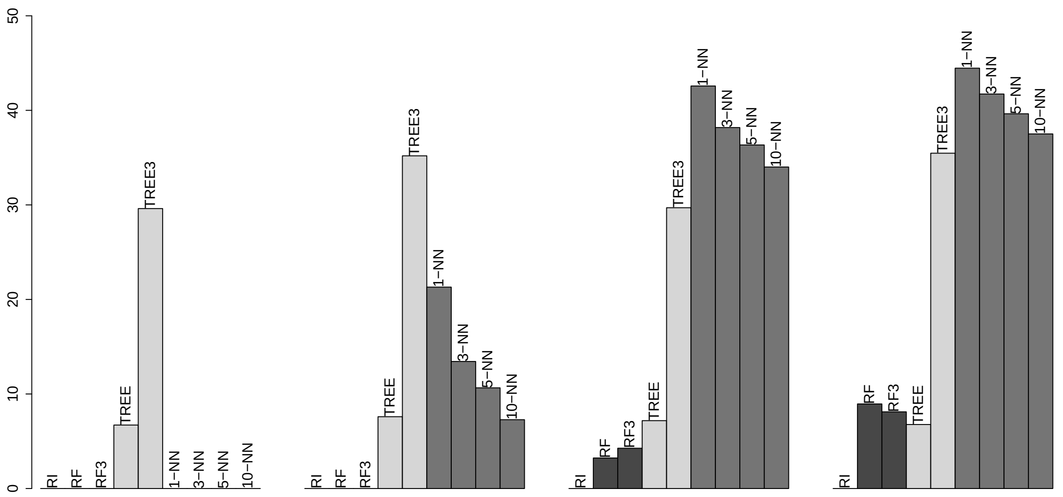

Figure 3: *From left to right: the misclassification rate (in %) on Tic-Tac-Toe data for 0, 60, 300 and 400 added noise variables. Each classifier is tuned to have equal misclassification rate in both classes. The simple classifier based on Random Intersection Trees (RI) has a misclassification rate of 0% in all cases, as the winning patterns are sampled very frequently (see Figure 2). Random Forests (RF) and Random Forests limited to depth 3 trees (RF3) are competitive but the misclassification rate increases sharply when many noise variables are added.*

aggregation method by selecting all patterns during 1000 iterations of *Random Intersection Trees* (with $B = 5$ samples as branching factor in each tree) that were selected by at least two trees. For each selected pattern, we compute the (empirical) class distributions conditional on the presence and absence of the pattern, using the training sample. That is, for each selected pattern S, we compute

$$\mathbb{P}_n(Y = 1|X \subseteq S) \text{ and } \mathbb{P}_n(Y = 1|X \nsubseteq S).$$

Then, given an observation from the test set, we classify according to the average of the log-odds of being in class 1 calculated from each of the conditional probabilities above.

Figure 3 shows the misclassification rates under situations with different numbers of added noise variables. The simple prediction based on *Random Intersection Trees* achieves perfect classification even when 400 noise variables are added. Neither k-NN nor *CART* [Breiman et al., 1984], either restricted to trees of depth 3 (TREE3) or depth chosen by cross-validation (TREE), are as successful, giving misclassification rates between 5% and 40%. Interestingly, trees of depth 3 perform much worse than deeper trees. The winning patterns are not identified in a pure form but only after some other variables have been factored in first. This also means that it is very hard to read the winning states of the trees, unlike the patterns obtained by our method. *Random Forests* also maintain a 0% misclassification rate up until about a hundred added noise variables but start to degrade in performance when further noise variables are added. It is easy to identify the noise variables from a variable importance plot [Strobl et al., 2008]. However, within the signal variables the patterns are not easy to see since each variable is approximately equally important for determining the winner (with the slight exception of the middle field in the 3×3 board which is more important than the other fields) and the nature of the interactions is thus not obvious from analysing a *Random Forest* fit.

5.2 Reuters RCV1 text classification

The Reuters RCV1 text data contain the *tf–idf* (term frequency–inverse document frequency) weighted presence of 47148 word-stems in each document; for details on the collection and processing of the original data, see Lewis et al. [2004]. Each document is assigned possibly more than one topic. Here we are interested in whether *Random Intersection Trees* is able to give a quick and accurate summary of each topic. For each topic, we seek sets of word-stems, S, whose simultaneous presence is indicative of a document falling within that topic.

To evaluate the performance of *Random Intersections*, we divide the documents into a training and test set with the first batch of 23149 documents as training and the following 30000 documents as test documents. We compare our procedure to an approach based on *Random Forests* and a simple linear method.

Random Forests and classification trees can be very time- and memory-intensive to apply on a dataset of the scale we consider here. In order to be able to compute *Random Forests*, we only consider word-stems if they appear in at least 100 documents in the training data. This leaves 2484 word-stems as predictor variables. We also only consider topics that contain at least 200 documents. To simplify the problem further, we consider a binary version of the predictor variables for all methods, using a 1 or 0 to represent whether each tf–idf value is positive or not.

Let C be the set of topics in our modified dataset. Let $Y \subseteq C$ indicate the topics that a given document belongs to. Consider a topic or class $c \in C$. Our goal is to find patterns S that maximise

$$\mathbb{P}_n(c \in Y | S \subseteq X), \tag{5.1}$$

whilst also maintaining that the prevalence of S among all observations be bounded away from 0. Specifically, we shall require that

$$\mathbb{P}_n(S \subseteq X) \geq p_c/10 \text{ where } p_c = \mathbb{P}_n(c \in Y). \tag{5.2}$$

To see how this can be cast within the framework set in (1.1), note that if $S^\star$ maximises (5.1) and $S^{\star\star}$ satisfies

$$\mathbb{P}_n(S^{\star\star} \subseteq X | Y \in c) \geq \mathbb{P}_n(S^\star \subseteq X | Y \in c) \text{ and} \tag{5.3}$$

$$\mathbb{P}_n(S^{\star\star} \subseteq X | Y \notin c) \leq \mathbb{P}_n(S^\star \subseteq X | Y \notin c), \tag{5.4}$$

then

$$\begin{aligned}\mathbb{P}_n(c \in Y | S^\star \subseteq X) &= \frac{\mathbb{P}_n(S^\star \subseteq X | c \in Y)\mathbb{P}_n(c \in Y)}{\mathbb{P}_n(S^\star \subseteq X | c \in Y)\mathbb{P}_n(c \in Y) + \mathbb{P}_n(S^\star \subseteq X | c \notin Y)\mathbb{P}_n(c \notin Y)} \\ &\leq \frac{\mathbb{P}_n(S^{\star\star} \subseteq X | c \in Y)\mathbb{P}_n(c \in Y)}{\mathbb{P}_n(S^{\star\star} \subseteq X | c \in Y)\mathbb{P}_n(c \in Y) + \mathbb{P}_n(S^{\star\star} \subseteq X | c \notin Y)\mathbb{P}_n(c \notin Y)} \\ &= \mathbb{P}_n(c \in Y | S^{\star\star}),\end{aligned}$$

whence $S^{\star\star}$ also maximises (5.1) by optimality of $S^\star$. Thus treating those documents belonging to topic c as class 1, and all others as class 0, by solving (1.1) with θ_0 and θ_1 chosen appropriately, we can obtain all solutions to (5.1).

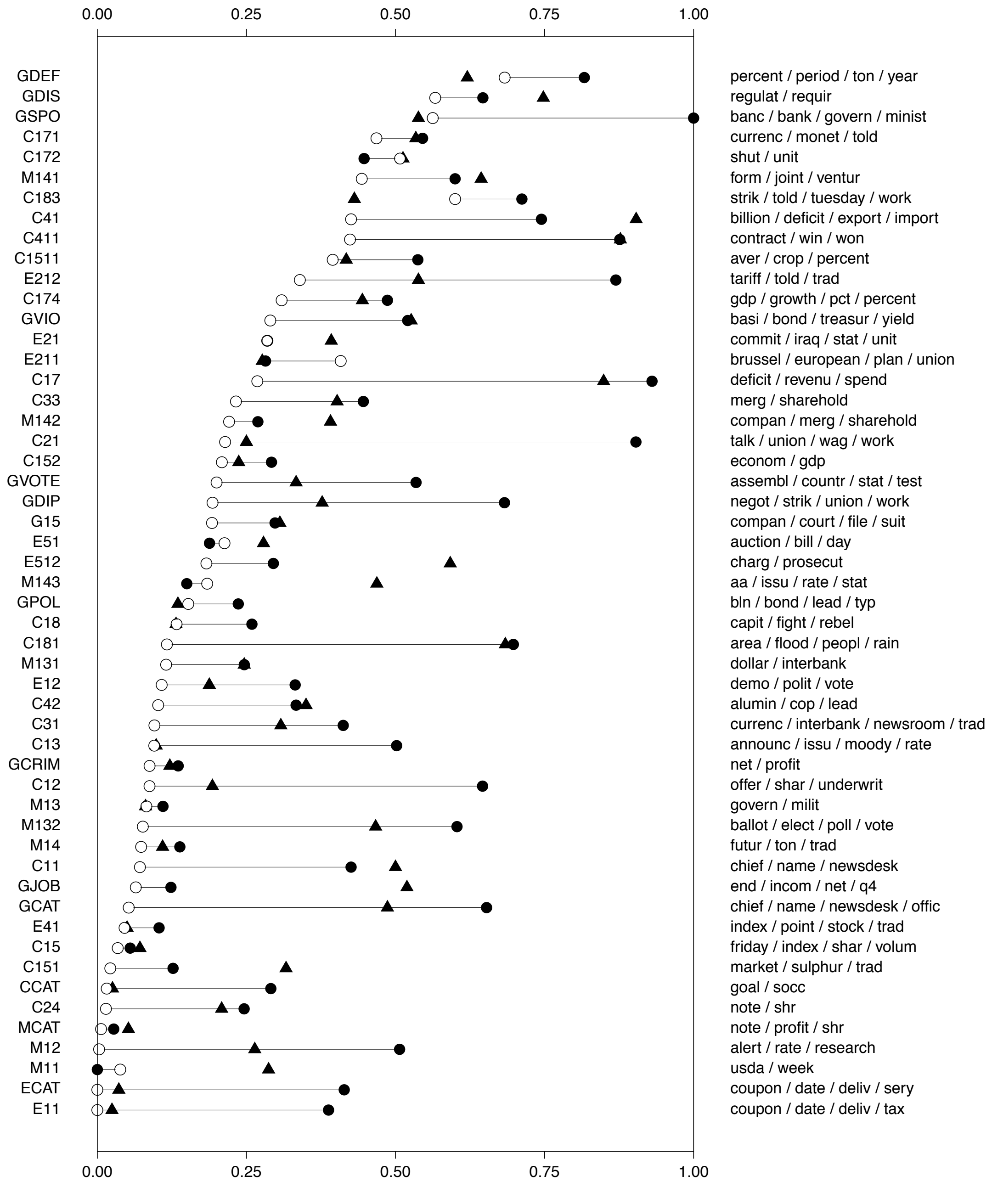

Figure 4: *The misclassification rate $\mathbb{P}_n(c \notin Y | S \subseteq X)$ on the test data for a pattern S chosen with a tree ensemble node generation mechanism (black circle), Random Intersections (white circle), and a linear method (black triangle) for topics $c \in C$ in the Reuters RCV1 text classification data. The topics are shown on the left and the word combinations chosen by Random Intersection Trees on the right.*

In view of this, we use each of the methods to search for patterns S that have high prevalence for a given topic c. We then remove all patterns that do not satisfy (5.2) on the test data. Then, from the remaining patterns, we select the one that maximises (5.1) on training data. Below, we describe specific implementation details of each of the methods under consideration.

Random Intersection Trees We create the min-wise hash table for the prevalence among all samples once, using 200 permutations with associated min-wise hash values for each word-stem. Then 1000 iterations of the tree search are performed with a cut-off value $\theta_0 = (3/20)p_c$ and all remaining patterns S with a length less or equal to 4 are retained.

Random Forests For a tree-based procedure, one approach is to fit classification trees on subsampled data and adding randomness in the variable selection as in *Random Forests* [Breiman, 2001] and then looking among all created leaf nodes for the most suitable node among all nodes created.

We generate 100 trees as in the *Random Forests* method: each is fit to subsampled training data using *CART* algorithm restricted to depth 4, and further randomness is injected by only permitting variables to be selected from a random subset of those available, for each tree. This takes on average between 90% to 110% of the computational time of a non-optimised pure R [R Development Core Team, 2005] implementation of *Random Intersection Trees* for these data. Note that this is when using the Fortran version of [Breiman, 2001] for the *Random Forests* node generation; we expect a significant speedup if Fortran or C code were used for *Random Intersection Trees*. We are currently working on such a version and plan to make it available soon. Furthermore, *Random Forests* would scale much worse if many more word-stems were included as variables.

Linear models For linear models, we fit a sparse model with at most ℓ predictors (with $\ell \leq 4$), using a logistic model with an ℓ_1-penalty [Tibshirani, 1996, Friedman et al., 2010]. We constrain the regression coefficients to be positive since we are only looking for positive associations in the two previously discussed approaches, and want to keep the same interpretability for the linear model. For each value of $\ell \leq 4$, we take S_ℓ to be the set of variables with a positive regression coefficient. We select the largest value of ℓ such that the fraction of documents attaining the maximal value is at least $p_c/10$ and select the associated pattern S_ℓ. (An alternative approach would be to retain the documents with the highest predicted value when using a sparse regression fit. This approach gave very similar results.)

After screening the candidate patterns returned by each of the methods using (5.2) on all of the topics $c \in C$, we evaluate the misclassification rate $\mathbb{P}_n(c \notin C | S \subseteq X)$ on the test data. The results for all of the topics are shown in Figure 4. The rules found with *Random Intersection Trees* have a smaller loss than those found with *Random Forests* in all but 5 of the topics. For those topics where *Random Forests* performs better, the difference in loss is typically small. Linear models achieve a smaller loss than *Random Forests* among most of the topics, but only have a smaller loss than *Random Intersection Trees* in 6 topics, performing worse in all remaining 46 topics.

6 Discussion

We have proposed *Random Intersection Trees* as an efficient way of finding interesting interactions. In contrast to more established algorithms, the patterns are not built up incrementally by adding variables to create interactions of greater and greater size. Instead we start from the full interaction $S = \{1, \dots, p\}$ and remove more and more variables from this set by taking intersections with randomly chosen observations. Arranging the search in a tree increases efficiency by exploiting sparsity in the data. For the basic version of our method (Algorithm 1), we were able to derive a bound on the computational complexity. The bound depends on (a) the prevalence or frequency with which the pattern S appears among observations in class 1, and (b) the overall sparsity of the data, with higher sparsity making it easier to detect the interaction using a given computational budget. In the best case, we can achieve an almost linear complexity bound as a function of p; more generally our complexity bound typically has a smaller exponent than that for a brute force search. Further improvements can be made by using min-wise hashing techniques to terminate parts of the search (i.e. branches of the Intersection Tree) that have no chance of leading to interesting interactions. Numerical examples illustrate the improved interaction detection power of *Random Intersection Trees* over other tree-based methods and linear models.

There are many diverse ways in which interactions that solve (1.1) can be used in further analysis. The interactions may be of interest in their own right as shown in both numerical examples. One can also simply use the search to make sure that a dataset is unlikely to have strong interactions that could otherwise have been missed. If the aim is to build a classifier, they can be added to a linear model, or built into classifiers based on tree ensembles. For the latter approach one could consider, for example, averaging predictions in a linear way or averaging log-odds as in *Random Ferns* [Bosch et al., 2007]. We believe developments along these lines will prove to be fruitful directions for future research. We also plan to generalise the idea to categorical and continuous predictor variables.

References

R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases*, volume 1215, pages 487–499, 1994.

D. Aha, D. Kibler, and M. Albert. Instance-based learning algorithms. *Machine learning*, 6: 37–66, 1991.

A. Bosch, A. Zisserman, and X. Muoz. Image classification using random forests and ferns. In *IEEE 11th International Conference on Computer Vision, 2007*, pages 1–8. IEEE, 2007.

L. Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001.

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

A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 327–336. ACM, 1998.

E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. Ullman, and C. Yang. Finding interesting associations without support pruning. *IEEE Transactions on Knowledge and Data Engineering*, 13:64–78, 2001.

M. Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. *Lecture Notes in Computer Science*, 2461:323, 2002.

J. Friedman and B. Popescu. Predictive learning via rule ensembles. *Annals of Applied Statistics*, 2:916–954, 2008.

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33:1–22, 2010.

J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, May 2000.

D. Lewis, Y. Yang, T. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

W. Loh and Y. Shih. Split selection methods for classification trees. *Statistica Sinica*, 7: 815–840, 1997.

M. Marchand and M. Sokolova. Learning with decision lists of data-dependent features. *Journal of Machine Learning Research*, 6:427, 2006.

C. Matheus and L. Rendell. Constructive induction on decision trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, volume 645650. Citeseer, 1989.

N. Meinshausen. Node harvest. *Annals of Applied Statistics*, 4:2049–2072, 2010.

J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang. H-mine: hyper-structure mining of frequent patterns in large databases. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 441–448, 2001.

R Development Core Team. *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria, 2005. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

R. Rivest. Learning decision lists. *Machine learning*, 2:229–246, 1987.

C. Strobl, A. Boulesteix, T. Kneib, T. Augustin, and A. Zeileis. Conditional variable importance for random forests. *BMC Bioinformatics*, 9:307, 2008.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1996.

7 Appendix

Proof of Theorem 1 Fix a tree $m \in \{1, \ldots, M\}$ and suppose this has node set $N = \{1, \ldots, J\}$ indexed chronologically (see Section 2). For $d \in \{1, \ldots, D\}$, define

$$N_d = \{j \in N : \text{depth}(j) = d \text{ and } S_j \supseteq S\},$$
$$W_d = |N_d|.$$

Let E be the event that S is contained in S_1, the random sample selected for the root node of tree m. Further, let $G_d(t) = \mathbb{E}(t^{W_d}|E)$, the probability generating function of W_d conditional on the event E.

We make a few simple observations from the theory of branching processes. Firstly, for $d \leq D - 1$, $G_{d+1} = G_d \circ G$ where $G := G_1$. To see this, first note that

$$W_{d+1} = \sum_{j \in N_d} \sum_{j' \in \text{ch}(j)} \mathbb{1}_{\{S \subseteq X_{i(j')}\}}.$$

Now conditional on E, the random variables $\sum_{j' \in \text{ch}(j)} \mathbb{1}_{\{S \subseteq X_{i(j')}\}}$ for $j \in N_d$, are independent of N_d. Moreover, they are independent of each other and have identical distributions equal to that of

$$\sum_{j' \in \text{ch}(1)} \mathbb{1}_{\{S \subseteq X_{i(j')}\}} = W_1.$$

This entails

$$\mathbb{E}(t^{W_{d+1}}|W_d = w, E) = \{\mathbb{E}(t^{W_1}|E)\}^w = \{G(t)\}^w.$$

Thus

$$G_{d+1}(t) = \mathbb{E}(\mathbb{E}(t^{W_{d+1}}|W_d, E)|E) = \mathbb{E}(\{G(t)\}^{W_d}|E) = G_d(G(t)),$$

as claimed.

From this we can conclude that if G has a fixed point q, then this must be a fixed point for all G_d. Since each G_d is non-decreasing on $(0, 1]$, we have that for all $d \in \mathbb{N}$, if $q' \leq q$ and $q' \in (0, 1]$, then $G_d(q') \leq q$. The relevance of these remarks will become clear from the following: for an $S' \in L_{D,m}$, we have

$$\begin{aligned}
G_D(\mathbb{P}(S' \supsetneq S|S' \supseteq S)) &= \sum_{\ell=0}^{\infty} \mathbb{P}(W_D = \ell|E)\mathbb{P}(S' \supsetneq S|S' \supseteq S)^\ell \\
&= \sum_{\ell=0}^{\infty} \mathbb{P}(\{W_D = \ell\} \cap \{S \notin L_{D,m}\}|E) \\
&= \mathbb{P}(S \notin L_{D,m}|E).
\end{aligned}$$

Thus if we can ensure that $\mathbb{P}(S' \supsetneq S|S' \supseteq S)$ is at most q, then the final probability in the above display will also be at most q. The rest of the proof proceeds with the following steps:

1. Find conditions on F_B, the distribution of the B_j, such that there exists a fixed point of G, q.

2. Find conditions on the tree depth D such that $\mathbb{P}(S' \supsetneq S|S' \supseteq S) \leq q$.

3. Given q establish conditions on M such that the overall probability of recovering S is at least $1-\eta$.

4. Given F_B, D and M, compute the expected computational cost of the algorithm.

Step 1: Let the distribution of the B_j be such that

$$B_j = \begin{cases} b & \text{with probability } 1-\alpha, \\ b+1 & \text{with probability } \alpha. \end{cases}$$

Now given a $q \in (0,1]$, we shall pick $b \in \mathbb{Z}_+$ and $\alpha \in [0,1)$ to satisfy $G(q) = q$. To this end, observe that

$$\begin{aligned} G(q) &= (1-\alpha)(1-\theta_1(1-q))^b + \alpha(1-\theta_1(1-q))^{b+1} \\ &= [1 - \{(\alpha+b) - \lfloor \alpha+b \rfloor\}\theta_1(1-q)]\{1-\theta_1(1-q)\}^{\lfloor \alpha+b \rfloor}. \end{aligned}$$

From the last displayed equation, we see that $G(q)$ varies with $\alpha+b$ continuously. Furthermore, when $\alpha+b=0$, $G(q)=1$, and by making $\alpha+b$ large, we can make $G(q)$ arbitrarily close to 0. Thus by the intermediate value theorem, for any $q \in (0,1]$, $\alpha+b$ can be chosen such that $G(q)=q$.

We now bound $\alpha+b$ from above in terms of q for use later in creating a bound on the complexity of the algorithm. We have

$$\begin{aligned} b+\alpha &= \frac{\log(q) - \log(1-\alpha\theta_1(1-q))}{\log(1-\theta_1(1-q))} + \alpha \\ &\leq \frac{-\log(q) + \log(1-\alpha\theta_1(1-q))}{\theta_1(1-q)} + \alpha \\ &\leq \frac{-\log(q)}{\theta_1(1-q)} \\ &\leq \frac{1+(1-q)/(2q)}{\theta_1}. \end{aligned} \tag{7.1}$$

In the final line, we used the inequality

$$\log(z) \geq (z-1) - \frac{(z-1)^2}{2z}, \quad 0 < z \leq 1.$$

Step 2: We now bound $\mathbb{P}(S' \supsetneq S | S' \supseteq S)$ from above in terms of D. The set S' is the intersection of $D+1$ observations selected independently of one another. In order for some $k \in S^c$ to be contained in S', it must have been present in all these $D+1$ observations. Thus by the union bound we have

$$\mathbb{P}(S' \supsetneq S | S' \supseteq S) \;\leq\; \sum_{k \in S^c} \mathbb{P}(k \in S' | S' \supseteq S) \;\leq\; p\nu^{D+1},$$

the rightmost inequality following from (A2).

To ensure this is at most q, we take

$$D = \left\lceil \frac{\log(p/q)}{\log(1/\nu)} \right\rceil - 1, \tag{7.2}$$

so

$$D \le \frac{\log(p/q)}{\log(1/\nu)}. \tag{7.3}$$

Step 3: Turning now to the probability of recovering S, we have

$$\mathbb{P}(S \in L_D) = 1 - [1 - \{1 - \mathbb{P}(S \notin L_{D,m}|E)\}\theta_1]^M.$$

Given the choices of α and b (7.1), and D (7.2), we have that $\mathbb{P}(S \notin L_{D,m}|E) \le q$. Thus taking M to be at least

$$\frac{-\log(\eta)}{(1-q)\theta_1} \ge \frac{-\log(\eta)}{\log\{1-(1-q)\theta_1\}} \tag{7.4}$$

guarantees recovery of S with probability at least $1-\eta$.

Step 4: To bound the complexity of the algorithm, observe that $\mathbb{E}(B_j) = b + \alpha$, so

$$\begin{aligned} C(M, D, F_B) \le& \log(p) M \sum_{k=1}^{p} [(b+\alpha)\delta_k + \cdots + \{(b+\alpha)\delta_k\}^D] \\ \le& \log(p) M D \Bigg[p + \sum_{k:(b+\alpha)\delta_k > 1} \Big\{ \big((b+\alpha)\delta_k\big)^D - 1 \Big\} \Bigg]. \end{aligned} \tag{7.5}$$

Substituting equations (7.1), (7.3) and (7.4) into the complexity bound (7.5), and writing $\epsilon = (1-q)/(2q)$ gives a bound for the computational complexity of

$$\log(p) \frac{\log(1/\eta)}{\theta_1} \frac{1+2\epsilon}{2\epsilon} \frac{\log\{p(1+2\epsilon)\}}{\log(1/\nu)} \Bigg[p + \sum_{k:(1+\epsilon)\delta_k > \theta_1} \Big\{ \big(p(1+2\epsilon)\big)^{\frac{\log\{(1+\epsilon)\delta_k/\theta_1\}}{\log(1/\nu)}} - 1 \Big\} \Bigg]. \tag{7.6}$$

Given that ϵ is bounded above, removing constant factors not depending on p, we get that the order of the computational complexity is bounded above by

$$\log(1/\eta) \frac{\log^2(p)}{\epsilon} \Bigg\{ p + \sum_{k:(1+\epsilon)\delta_k > \theta_1} \Big(p^{\frac{\log\{(1+\epsilon)\delta_k/\theta_1\}}{\log(1/\nu)}} - 1 \Big) \Bigg\}. \quad \square$$

Proof of Corollary 2 Note that

$$\sum_{k:(1+\epsilon)\delta_k > \theta_1} p^{\frac{\log((1+\epsilon)\delta_k/\theta_1)}{\log(1/\nu)}}$$

is bounded by

$$(1+\epsilon)^{\frac{\log(p)}{\log(1/\nu)}} \Big(p^{\gamma} \cdot p^{\alpha^\star/\beta} \mathbb{1}_{\{\alpha^\star/\beta > 0\}} + p \cdot p^{\alpha_\star/\beta} \mathbb{1}_{\{\alpha_\star/\beta > 0\}} \Big)$$

The result then follows from substituting into (7.6) and taking $\epsilon \propto 1/\log(p)$. $\square$

Derivation of (4.2) Writing $r = n\pi_2(S)$, we have

$$\begin{aligned}\binom{n}{r}\mathbb{E}_\sigma(\min_{k\in S} h_\sigma(k)) &= \sum_{\ell=1}^{n-r+1} \ell\binom{n-\ell}{r-1} \\ &= \sum_{\ell=1}^{n-r+1}\left\{(\ell-1)\binom{n-(\ell-1)}{r} - \ell\binom{n-\ell}{r}\right\} + \sum_{\ell=1}^{n-r+1}\binom{n-\ell+1}{r}.\end{aligned}$$

The first two terms sum to zero leaving only the final term. Thus

$$\begin{aligned}\binom{n}{r}\mathbb{E}_\sigma(\min_{k\in S} h_\sigma(k)) &= \sum_{\ell=1}^{n-r+1}\left\{\binom{n-\ell+2}{r+1} - \binom{n-\ell+1}{r+1}\right\} \\ &= \binom{n+1}{r+1}, \qquad (7.7)\end{aligned}$$

whence

$$\mathbb{E}_\sigma(\min_{k\in S} h_\sigma(k)) = \frac{n+1}{r+1}. \quad \square \qquad (7.8)$$

Proof of Theorem 4 Writing

$$\tilde{\pi}_2^{-1}(L; S, H) := \tfrac{1}{L}\sum_{l=1}^{L}\min_{k\in S} H_{lk}$$

and suppressing dependence on S and H, we have

$$\begin{aligned}\hat{\pi}_1\hat{\pi}_2 - \pi_1\pi_2 &= \frac{(n+1-\tilde{\pi}_2^{-1})\hat{\pi}_1}{n\tilde{\pi}_2^{-1}} - \pi_1\pi_2 \\ &= \frac{n+1-\tilde{\pi}_2^{-1}}{n\tilde{\pi}_2^{-1}}\left\{(\hat{\pi}_1-\pi_1) - \pi_1\frac{n\pi_2+1}{n+1-\tilde{\pi}_2^{-1}}\left(\tilde{\pi}_2^{-1} - \frac{n+1}{n\pi_2+1}\right)\right\}. \qquad (7.9)\end{aligned}$$

Consider $L \to \infty$. By the weak law of large numbers and the continuous mapping theorem, we have

$$\frac{n+1-\tilde{\pi}_2^{-1}(L)}{n\tilde{\pi}_2^{-1}(L)} \xrightarrow{p} \pi_2 \quad \text{and}$$
$$\frac{n\pi_2+1}{n+1-\tilde{\pi}_2^{-1}(L)} \xrightarrow{p} \frac{(\pi_2+n^{-1})^2}{\pi_2(1+n^{-1})}.$$

By the central limit theorem, Slutsky's lemma and Lemma 5,

$$A_L := \sqrt{L}(\hat{\pi}_1(L) - \pi_1) \xrightarrow{d} N(0, \pi_1(1-\pi_1)) \quad \text{and}$$
$$B_L := -\pi_1\frac{n\pi_2+1}{n+1-\tilde{\pi}_2^{-1}(L)} \times \sqrt{L}\left(\tilde{\pi}_2^{-1}(L) - \frac{n+1}{n\pi_2+1}\right) \xrightarrow{d} N(0, \pi_1^2(1-\pi_2)(1+\epsilon(n))),$$

with $\epsilon(n)$ defined as in (4.6).

Define $I_S := \{i : S \subseteq X\}$ and let $k \in S$. Now observe that

$$\{\exists\iota' : h_\sigma(k) = \iota' \text{ for all } k' \in S\} = \{\sigma^{-1}(h_\sigma(k)) \in I_S\} \text{ and } \{\min_{k\in S} h_\sigma(k) = \iota\}$$

are independent: in words, the distribution of $\min_{k\in S} h_\sigma(k)$ conditional on the fact that an observation index in I_S was permuted to a lower value than any in $I_k \setminus I_S$ is the same as its unconditional distribution. This implies the independence of $\tilde{\pi}_1$ and $\tilde{\pi}_2^{-1}$ and thence also that of A_L and B_L. Thus we have that for all $t_1, t_2 \in \mathbb{R}$,

$$\mathbb{E}(e^{i(t_1A_L+t_2B_L)}) = \mathbb{E}(e^{it_1A_L})\mathbb{E}(e^{it_2B_L}) \to \exp[\tfrac{1}{2}t_1^2\pi_1(1-\pi_1) + \tfrac{1}{2}t_2^2\{\pi_1^2(1-\pi_2)(1+)\}].$$

pointwise as $L \to \infty$. Returning to (7.9), by Lévy's continuity theorem we have

$$\sqrt{L}\{\hat{\pi}_1(L)\hat{\pi}_2(L) - \pi_1\pi_2\} \xrightarrow{d} N(0, \pi_2^2\pi_1(1-\pi_1\pi_2)(1+\epsilon(n))). \quad \square$$

Lemma 5. *Let $r = n\pi_2(S)$ and suppose $n \geq r+2$. Then*

$$\mathrm{Var}_\sigma(\min_{k\in S} h_\sigma(k)) = \frac{r(n+1)(n-r)}{(r+1)^2(r+2)}.$$

Proof. We have,

$$\begin{aligned}
\binom{n}{r}\mathbb{E}_\sigma\{(\min_{k\in S} h_\sigma(k))^2\} &= \sum_{\ell=1}^{n-r+1} \ell^2\binom{n-\ell}{r-1} \\
&= \sum_{\ell=1}^{n-r+1}\left\{(\ell-1)^2\binom{n-(\ell-1)}{r} - \ell^2\binom{n-\ell}{r}\right\} \\
&\quad + \sum_{\ell=1}^{n-r+1}\left\{2(\ell-1)\binom{n-(\ell-1)}{r} + \binom{n-\ell+1}{r}\right\} \\
&= 2\binom{n+1}{r+2} + \binom{n+1}{r+1},
\end{aligned}$$

where in the last line we used (7.7) and (7.8). Simplifying and using (4.2) then gives the result. $\square$