Classification of Examples by Multiple Agents with Private Features

Peter Woo Tae Kim – advised by Pragnesh Jay Modi,
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
pmodi@cs.cmu.edu, pk@andrew.cmu.edu

Abstract

We consider the problem of classification where relevant features are distributed among a set of agents and cannot be centralized, for example due to privacy restrictions. Accurate prediction of the output class is difficult for an isolated single agent because the target concept may involve features to which the agent does not have access. To increase prediction accuracy, a learning algorithm is required in which agents collaborate to classify new examples, while preserving the privacy of their local features. We formalize this problem as the distributed classification task. We introduce a novel distributed decision-tree inspired algorithm for such tasks named DDT. One of the key ideas in DDT is that agents can communicate the information gain of a private feature without revealing the semantics of the feature or its actual value. We present empirical results in a calendar management domain where software assistant agents classify new meetings as "likely to be difficult to schedule" using private features such as each attendee's willingness to attend the meeting. We show empirically that our approach outperforms a single agent learner and performs as good as a centralized learner with hypothetical access to all the features.

1. Introduction

In many multiagent domains where classification tasks arise, agents have private features they are not willing to reveal to other agents or humans. For example a personal assistant agent who assists a human user in managing his or her calendar may have information about the user's personal preferences such as which meetings are important to the user or which other people are important to meet with. In order for such an agent to be an effective assistant to a human user, these preferences must be kept private by the agent.

In a conventional classification task, accurate prediction of the class of a new unlabeled example requires access to all features that are relevant to its classification. In distributed multiagent domains with privacy concerns, this assumption is inappropriate. For example in the distributed meeting scheduling problem [7], multiple agents perform a potentially complex negotiation in order to discover a mutually agreed time for the meeting. Consider the task of predicting whether a given meeting will be successfully scheduled. The outcome of the distributed scheduling process is either a "success" in which a start time for the meeting was agreed to by all attendees, or "failure" in which no start time could be agreed to after some finite amount of effort after which the attendees give up. The outcome of the process depends on the values of private features of each attendee, such as each attendee's current calendar density, their willingness to bump prior scheduled meetings in favor of the new meeting, and each attendees personal importance for participating in the meeting. Indeed, the correct target concept for a particular set of agents involve features belonging to different agents so that no single agent may be able to learn it individually.

In this paper, we consider the following problem: How can a set of agents collaborate to accurately classify a new example when the set of input features is distributed among them and must be kept private? We formalize this problem as *the distributed classification task*. We are motivated by the domain of personal assistant agents which negotiate in order to manage calendars on the behalf of their human users. Although the goal of each agent is to serve the interests of its user, the agents of different users can collaborate in order to perform distributed classification tasks, at least to the extent that such collaboration benefits their respective users.

We introduce a distributed learning algorithm for the distributed classification problem named DDT. DDT contains several key ideas. First, DDT is inspired by centralized decision tree algorithms, specifically the ID3[6], in which measuring the information gain of a feature over a given training set is a core operation used to build a decision tree. The DDT algorithm exploits the fact that information gain of a particular feature can be computed by the agent who has access to the values of that feature; the values of the rest of the features are not needed. In DDT, agents communicate the information gain of their local features to make predictions but without revealing the semantics of the features or their values.

Second, DDT is a lazy learning algorithm in which the agents communicate information gain about their private features *at prediction time*. There is no training period and no explicit model is learned. This overcomes a key difficulty of the distributed classification task in which a single agent could not interpret or apply a learned model such as a decision tree when the model involves features it does not have access to.

Finally, DDT performs a parallelized breadth-first search through the space of decision trees by having each agent maintain a list of possible paths through some decision

tree. This list is dynamically ordered according to a given inductive bias such as a preference for shorter decision trees. This form of search allows agents to communicate asynchronously while obtaining prediction accuracy comparable to a centralized decision tree learner.

We present empirical results in a calendar management domain using the CMRadar simulator [1]. In our experiments, the simulator is populated with a set of agents each of which represents an imaginary user with a given set of preferences and given calendar density. The agents engage in a scheduling process for a new meeting and log the outcome of the process. Each scheduling episode is a training example. Then, as new meetings arise, agents predict if the meeting is likely to be successfully scheduled using the prior episodes as training data. Accuracy of the prediction is verified by engaging in the scheduling process for the new meeting and logging its outcome. We show empirically that DDT outperforms a single agent learner and performs as good as a centralized learner with hypothetical access to all the features.

2. The Distributed Classification Task

A common method of knowledge representation for the classification task is to represent examples as feature vectors with discrete or numeric values. The input to a classification task consists of a set of pre-classified training examples E, with each example described by (the value of) a vector of features A. The goal is to construct a mapping from feature values to classes. We adopt and modify this knowledge representation scheme for our work because it has two distinct advantages; First, it has been shown that the feature-value formalism is an extremely general way of representing knowledge and can be applied to a variety of domains; Second, there are wide varieties of existing techniques and algorithms that apply to this type of representation. This allows us to take advantage of these existing techniques when devising new algorithms.

We formulate the *distributed classification task* as follows**:** Given a collection α of n agents, we assume the feature vector A is divided into (not necessarily disjoint) subsets, $A_i \subseteq A$ ($\cup A_i = A$), $i \in 1...n$. Each $agent_i \in \alpha$ knows the classification of every training example, but has access only to A_i from each training example. We will use the notation E_i to denote the projection of training examples E onto A_i.

In this way, each agent has only a local, partial view of the training experience. We will assume that each training example has a unique id known to all. This is so agents can communicate about individual training examples by id only. This is a reasonable assumption for many domains, including meeting scheduling in which each meeting can be assigned a unique id. Finally and importantly, we assume that agents are not willing to share the values of their local features with other agents. The goal of the agents is the same as in the centralized task: to accurately predict the class of new unseen example.

3. Algorithm

3.1 Decision Tree Learning

Decision tree learning is one of the most practical methods for classification from labeled training examples. We give a brief introduction to decision tree learning and refer the reader to [1] for a more detailed explanation. Decision tree learners perform a search through the space of decision trees by recursively choosing an feature on which to partition the training examples. The "best" feature on which to partition the examples is judged by the one that provides maximum information gain. One way to define information gain is the reduction in entropy of a set of examples, E, when split on a given feature a. Entropy is given by

$$Entropy(E) = \sum_{i=1}^{c} -p_i log_2 p_i$$

where p_i is the proportion of E belonging to class i. Information gain, the reduction in entropy, is given by

$$Gain(E, a) = Entropy(E) - \sum_{v \in values(a)} (|E_v| / |E|)\, Entropy(E_v)$$

By recursively choosing the feature with maximum information gain at each stage, the learner performs a greedy search for the best decision tree. We desire a group of agents to perform this search in a distributed, asynchronous manner without having to share their entire local dataset with one another.

3.2 DDT

The key idea behind the DDT algorithm is to realize that information gain of a particular feature can be computed by the agent who has access to the values of that feature; the values of the rest of the features are not needed. Furthermore, the information gain measure is a highly compact summarization of each agent's local view of the training set. Agents can use this measure to communicate about their local data in an indirect way and thus perform a distributed search for the best decision tree.

DDT is a lazy learning algorithm in that each agent stores its training data for prediction. There is no training stage and no explicit hypothesis is learned. At prediction time, the agents collectively and asynchronously determine a path through an implicit decision tree. The path is determined using the training data and the feature values of the test example. The leaf of this path is used to classify the test example. The benefits of lazy decision tree learning are described in [9]. The DDT algorithm is depicted in Figure 1 and is described next.

Let a set of training examples E and an unclassified test example e, be given. Each $agent_i \in \alpha$ begins by choosing

the local feature with maximum information gain over E_i. From its local point of view, this is the best choice for the root of the tree. It then partitions E_i on this feature and then creates a tuple we call a TreePath. A TreePath has two fields, a list of real numbers and a set of example ids. Intuitively, it is called a TreePath because it holds the information corresponding to a particular path through some decision tree. Each real number in the first field corresponds to the information gain of the feature that was used to partition the examples at a particular node along the path. The second field holds the set of example ids at leaf of this path. For example, suppose $agent_i$ chooses the feature $a_1 \in A_i$ because it has maximum information gain

```
Given:
   - Ei, training examples with feature vector Ai
   - e, unclassified example to be labeled
   - Q, an empty data structure for holding a sorted list of
     TreePaths
initialize
   a  ← feature with max info gain over Ei
   gain ← info gain of a over Ei
   v  ← value of a in e
   [ids] ← list of examples in Ei with value v for a insert in order(Q,
   TreePath:([gain], [ids])
when received(TreePath T))
   insert_in_order(Q, T)
procedure make_prediction()
   TreePath:([infogains], [ids]) ← pop(Q)
   a ←feature with max info gain over [ids]
   gain ← info gain of a over [ids]
   v ← value of a in e
   if gain = = 0
      return the most common class in [ids]
   else
      [new ids] ← list of examples in [ids] with value v for a
      broadcast TreePath:([infogains,gain], [new ids]) to all agents
      make_prediction()
```

Figure 1

over E_i, equal to say, 0.24 (see Figure 2). Suppose a_1 takes on the value v_1 in e, and $agent_i$ finds that the examples 1,2,8,9 and 11 in E_i are the ones that have value v_1 for a_1. It then creates the following TreePath: ((0.24) (1,2,8,9,11)). This is a path of depth one, since the examples were separated on only one feature. Figure 2 shows that as additional features are used to further partition the examples, the list of information gains becomes longer and the set of example ids becomes smaller.

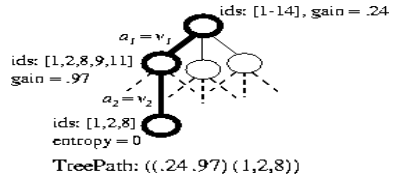

Figure 2

Every agent goes through the above process and broadcasts their TreePath to the rest of the agents. As agents receive TreePaths from others, they order them according to a greedy "best info gain first" heuristic. This ordering is shown in Figure 3. The best TreePath received by each agent is then deepened by one level in the same manner as above, except instead of computing information gain over all the examples in E_i, it is computed over the list of example ids in the TreePath. This new TreePath is then broadcast to all agents and the process repeats. The process terminates whenever the best TreePath available to an agent cannot be extended because no feature available to it provides positive information gain. Since the set of example ids for a TreePath is always getting smaller as the TreePath is extended and no TreePath is ever made shorter, it follows each agent will eventually terminate.

Intuitively, DDT is performing a parallelized, breadth-first search for the best path through some decision tree. This is in contrast to the depth-first greedy strategy of most centralized decision tree learners. However, the inductive bias of DDT is similar in that it prefers shorter paths with higher information gain features at the top. We now highlight two important properties of this algorithm.

- Agents never share their local feature values. In fact, agents never even reveal the identities of their local features. This satisfies our requirement of privacy. As an aside, note that this property also provides flexibility in that an agent can locally, autonomously decide what features are relevant for a given learning task.
- DDT is asynchronous. This means that the group prediction can continue if some agent is slow in communicating. This is important when agents are operating in environments where perfect synchronous communication cannot be assured.

```
//This procedure compares two TreePaths.
//Returns true if T1 is better than T2.
procedure best_info_gain_first(T1,T2)
gains1 ← list of information gains in T1
gains2 ← list of information gains in T2
for i in 1 to min(length of gains1, length of gains2)
   if gains1[i] > gains2[i]
      return TRUE
   else if gains2[i] > gains1[i]
      return FALSE
if length of gains1 < length of gains2
   return TRUE
else if length of gains2 < length of gains1
   return FALSE
else return EQUAL
```

Figure 3

4. The CMRadar Simulator

4.1 The CMRadar project

We evaluate our techniques in the context of the CMRadar Project [1] whose goal is to develop personalized assistant agents that are able to make people more efficient by automating many routine everyday tasks such as scheduling of meetings. Importantly, CMRadar is developed as an agent that also interacts with other users or agents. Maintaining privacy during such interactions is an important consideration.

The motivation in the CMRadar project is to develop an end-to-end system for use by real users to obtain data to facilitate learning. However due to the difficulties associated with such a deployment and the time required to overcome such difficulties, in the meantime we have also built a simulator testbed to enable parallel development of learning and scheduling techniques. In this section we describe this testbed in more detail and in the following section, we present the empirical results.

4.2. Experimental Testbed

The CMRadar agents live in a simulated distributed environment and are able to pass simulated email messages between them. In initialization phase, we generate a set of CMRadar agents with calendars of a given density. Each agent's calendar has 50 timeslots to simulate a 5 day, 10-hr/day work week. The number of attendees for each meeting is chosen according to a distribution in which meetings of more people are less likely than meetings with fewer people, and every meeting has at least two attendees.

Each training example is generated by one run in which agents employ a multiagent scheduling protocol, described next, to schedule a new meeting M_{m+1}. The attendees of meeting M_{m+1} are chosen to be a random subset of the agents, with the size of the meeting as an input parameter.

4.3 Agent Strategies

The multiagent scheduling protocol employed by CMRadar agents proceed in a sequence of rounds. In each round, an initiator sends a message to the other attendees with a proposed start time for a new meeting. The attendees may accept or reject the proposal. The protocol continues in rounds until an agreement is reached ("success") or the initiator has no more values to propose or a max time elapses ("failure").

Each attendee decides whether to accept or reject the proposal using a *scheduling strategy*. A scheduling strategy is a decision procedure that is given to an agent that it uses to determine whether to accept a given meeting proposal or to reject a meeting proposal. The idea is that this strategy reflects its user's preferences about which meetings are important to schedule. We discuss the details of these strategies next.

4.3.1 Features. Each agent's scheduling strategy is based on the values of the following set of private features.

- Schedule density (SD): This feature represents the current density of an agent's schedule. It has values **low** (less than 40%), **medium** (between 40% ~ 70%), and **high** (greater than 70%).
- Attendee importance (AI): For given meeting M, this is a set of features that represent for each of the other attendees of M, the user's preference for meeting with that attendee in meeting M. Attendee importance has values **low**, **medium**, **high**, and **notPresent**. The **notPresent** value is a default value for the AI feature corresponding to an agent is not attendee of M.
- Subject importance (SI): This feature represents a user's importance level for the subject of a particular meeting. It has values **very high**, **high**, **medium**, **low**, and **very low**.

4.3.2 Scheduling Strategies

As mentioned, each agent's strategy is based on the values of the above set of private features. For our experiments, we populate the simulator with a set of agents that employ scheduling strategies we invented. While these strategies are made up, they intuitively reflect plausible user preferences. In any case, our goal is to demonstrate the viability of our learning approach to accurately make predictions when features are private, not necessarily to employ the most realistic scheduling strategies that users may have.

There are strategies that are common to all agents and there are strategies that are specific to each agent. Two strategies that are common to all agents are:

- if SI=verylow for the new meeting, always reject the proposal.
- If the proposed time is unoccupied in local schedule, always accept the proposal.

Strategies that vary between agents are those that are employed when a proposed time conflicts with an existing meeting. When a "new" meeting conflicts with an "old" meeting, some examples of scheduling strategies used by the agents includes:

- If the maximum AI over all attendees in new meeting is higher than maximum AI in old meeting, bump old meeting for new meeting.
- If SI of new meeting is higher than SI of old meeting, then bump old meeting for new meeting.
- Always reject new meeting.
- Always reject old meeting.

In our experiments described in the next section, we allow the agents to execute the distributed scheduling process as described above and they log the values of the features and the outcome of each scheduling episode (success or failure).

5. Experimental evaluation

5.1 Setup

We populate the CMRadar simulator with 10 agents where each assists a different imaginary human user. The CMRadar simulator tries to schedule a meeting M and each agent logs its own features. As described earlier, each agent has access to features that describe its user's current calendar and preferences. In each meeting scheduling episode, each agent logs a feature for subject importance (SI), current schedule density (SD), and 9 features for attendee importance, one for each of the other agents in the system. Thus, there are a total of 11 features that are logged at each agent for training and testing purpose.

For empirical evaluation, the performance of DDT is compared against the following approaches.

- Single Agent Average: We apply the standard ID3 decision tree learner to a single agent's features. The accuracy on a test set, averaged over all the agents, is used to measure the total correctness.
- Super Agent: In this case, ID3 is given the combined features of all agents strictly for evaluation purposes. So if agent1, agent2, and agent3 are in the meeting, then all the features from agent1, agent2, and agent3 are combined. Super agent uses these combined features for its prediction.

5.2 Empirical Result

Result is obtained with 100 training examples and 60 test examples. M is classified into 2 cases which is impossible case and confirmed case. Impossible case represents that the members of M cannot reach an agreement for a time for meeting. Unless everyone agrees on certain time for meeting, M is classified as impossible meeting. Confirmed case is where the members of meeting found an agreement for the meeting time. To measure the performance, correctness is measured. The correctness is represented as the percent of correctly predicted test example.

5.2.1 Heterogeneous Agents. In this experiment, agents used the rules that are described in section 4.3.2. The size of M is the number of attendee in the meeting. Size of M is increased from 2 to 10. Figure 4 shows the result.

When there are only two agents, it is fairly easy to find an agreement. Also, there is not much variation in features and rules. Therefore, all DDT, single agent, average, and super agents show high accuracy. As more agents are involved in determining meeting schedule, both features, and bumping rules become complicated. As result, it shows lower accuracy overall. However, accuracy drop for DDT is not as significant as single agents, because DDT can use all the information from all agents. As size of M increase to have size over 7, the accuracy begins to increase again. The reason is that now it is hard to find

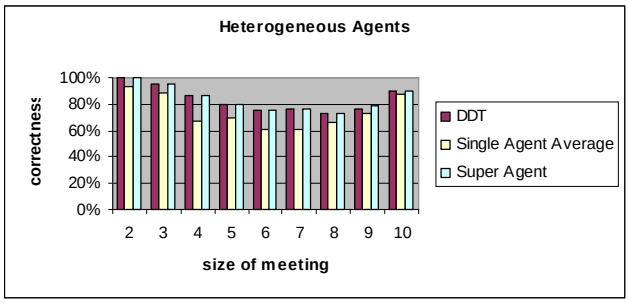

Figure 4

agreement among agents, therefore, in a lot of cases, M is simply becoming impossible

5.2.2 Low density and mixed rules. In this experiment, same rule as 4.3.2 is used. However, the schedule density of initial calendar is fixed to have low values. Figure 5 shows the result.

In this case, DDT shows overall high accuracy. The reason is that now very low subject importance is only feature that makes impossible and otherwise, it is confirmed. Therefore, DDT can predict the outcome very accurately. However, single agents do not have information for other agents, thus lower accuracy.

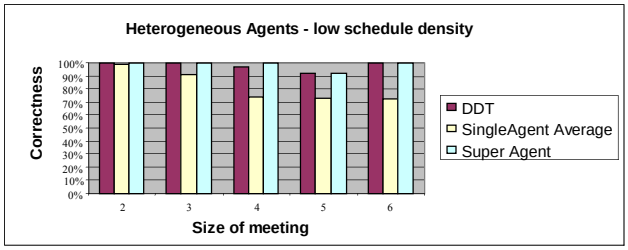

Figure 5

5.2.3 High schedule density. In this experiment, the same rule as 4.3.2 is used except that schedule density is fixed to have high values. Figure 6 shows the result.

As it is explained in section 4.3 agents uses a bumping process more if there is conflict for two different meeting times. In high schedule density, the accuracy does not show too much difference between single agents and DDT. The reason is that in high schedule density case, there is a

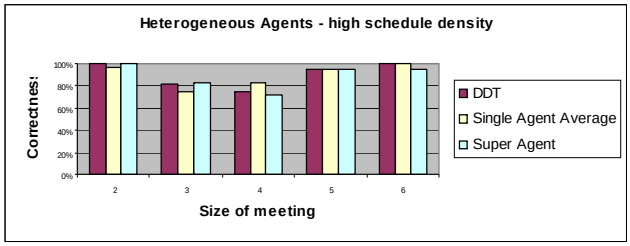

Figure 6

lot of bumping process involved; and each agents are now deciding on its own. It is much easier for single agents to predict because, if any one of agents does not agree then there will be no meeting held, and there are a lot more

cases where one or more agents that do not agree due to bumping process.

5.2.4 Effect of subject importance. Subject importance is key feature that used by every agent as first rule. To test the feature importance, the experiment is designed to have all different features for every agent, same subject importance for every agent, and all same features except subject importance. The result is shown in Figure 7. The result shows that having same subject importance improves the accuracy of single agent greatly.

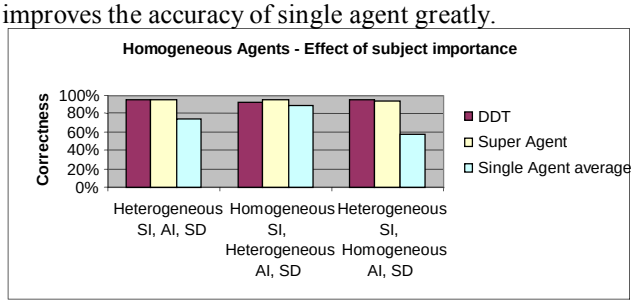

Figure 7

5.2.5 Messages required for collaboration. In this section we have examined how many bytes needs to be sent and receive to make prediction. The result is shown in Figure 8. As the size of M increase, the numbers of bytes sent and received for communication increase because there are more agents involved in prediction process. However, the average numbers of bytes sent is still fairly small. It requires less than 5000 bytes for prediction with 10 agents.

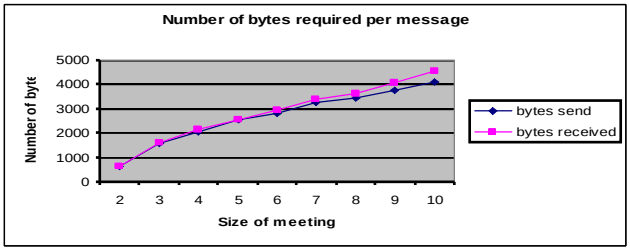

Figure 8

6. Conclusion

Motivated by the real world domain of distributed calendar management, we introduced the distributed classification task for learning in multiagent systems where privacy is a key concern. We described the DDT algorithm in which allows a set of agents to collaborate to make predictions based on private training data. The performance of DDT was compared against a single agent learner and a superagent with hypothetical access to all the features. Our results show that DDT is able to accurately learn and it is able to perform as good as the superagent and outperform a single agent learner. We conclude that DDT is useful algorithm because it allows agents to keep their private features and learn more accurately as group than any one of them could learn individually.

7. Related Work

There is significant research in the data mining community that is concerned with learning from distributed datasets [1]. This work can be classified along at least two dimensions. The first dimension concerns assumptions on how data is distributed, i.e., either horizontally or vertically. We have assumed a vertical distribution in this paper. In horizontal distribution, each agent has complete examples, but no agent has all the examples. The second dimension is related to motivation. In this paper, we have been concerned with privacy. Another motivation is efficient learning from a massive training set that is (given as or purposely) divided among a set of processors or agents [9]. While privacy and efficiency may be inter-dependant, they are fundamentally different motivations.

We highlight selected related work in data mining that assumes vertically distributed datasets and addresses privacy concerns. Vaidya and Clifton [3] discuss an approach that uses computation of scalar product to preserve privacy, but this approach is limited to two agent interactions. Kargupta et al [2] discuss an approach which leverages the Fourier representation of a decision tree, but this approach is limited to binary valued features. Agrawal and Srikant [8] look at the problem of preserving privacy by perturbing the feature values. All of these approaches are interesting avenues for addressing the distributed classification task for multiagent systems introduced in this paper.

References

1.Hillol Kargupta and Philip Chan, editors. Advances in Distributed and Parallel Knowledge Discovery. In *AAAI/ MIT Press.*, 1999.

2. H. Kargupta, B. Park, D. Hershbereger and E, Johnson, "Collective data mining: A new perspective toward distributed data mining". In Advances in Distributed and Parallel Knowledge Discovery. *AAAI/MIT Press* 1999.

3. J. Vaidya and C. Clifton, Privacy Preserving Association Rule Mining in Vertically Partitioned Data. *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.

4. P. J. Modi, M. Veloso, S. F. Smith, Jean Oh.CMRADAR: A Personal Assistant Agent for Calendar Management. In *Agent Oriented Information Systems*, (AOIS) 2004.

5. S. Sen and E. H. Durfee. A formal study of distributed meeting scheduling. In *Group Decision and Negotiation*, volume 7, pages 265–289, 1998.

6. Quinlan J. R. Induction of decision trees. *Machine Learning*, 1986

7. R. Agrawal and R. Srikant, Privacy-preserving data mining. In *Proceedings of ACM SIGMOD Conference on Management of Data.* 2000

8. Provost, F. J., & Hennessy, D. N. Scaling up: Distributed machine learning with cooperation. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. 2000
9. Friedman, J. H., Kohavi, R., & Yun, Y. Lazy decision trees. *Proceeding of the Thirteenth National Conference on Artificial Intelligence*. 1996