Differential Privacy for Decision Trees

Stefan Hanisch
stefan.hanisch@tu-braunschweig.de
Seminar Attacks against Machine Learning
Institute of System Security

***Abstract*—When public available data sets contain sensitive information about individuals, security measures are essentially. This paper provides an overview of traditional techniques with their corresponding weaknesses and introduces differential privacy as a mathematical substantiated framework to guarantee privacy. After a mathematical derivation, real world examples illustrate the technique and point out possible disadvantages. To extract general relationships out of noisy data, machine learning algorithms are introduced with respect to the tradeoff between the number of queries and accurate results. Finally, an own implementation of a random private decision tree and forest is presented, applied on a real data set and compared with non-private decision trees and forests.**

I. Introduction

The objective of public available data sets containing records about individuals is to provide high quality data to extract useful information and the protection of the privacy of any individual simultaneously. The U.S. Health Insurance Portability and Accountability Act (HIPAA) dictates, in the context of medicine data sets, the censoring of a set of attributes to ensure privacy. That the removal of identifiers (name, social security number, etc. [6]) is not enough was shown by the MIT computer science student Latanya Sweeney, who merged a public health data set with a purchasable voter roll [3] and could identify the record of the governor and his private health data by a little amount of background knowledge[1]. [10]

Another approach to deal with this problem is the k-anonymity technique, which requires the removal of identifiers and pretends the existence of at least k records 'for each combination of key attributes' like job, address, age, gender [6]. This can be done by averaging similar continuous values like age or by generalizing ZIP codes to 38xxx. If each group, of at least k records with the same combination of key attributes, contains miscellaneous outcome variables (like income or present diseases), it is more difficult for an adversary to determine the outcome variable for a specific person.

Example: Two people live in the region of the ZIP code 38xxx, are approximately 50 years old and are diagnosed with two different diseases. As shown in Table 1, for each combination of the key attributes age and ZIP code exist at least two records in the data set (2-anonymity). If the adversary has no background knowledge, he cannot determine the disease of a specific person. If he has background knowledge of a specific person (e.g. information about previous journeys to south America, which is known for yellow fever), he can estimate which disease is more likely (in the case of a journey to south africa: the yellow fever is more likely).

[1] 'Only six people in Cambridge shared his birth date, only three of them men, and of them, only he lived in his ZIP code' [3]

TABLE I: Example for 2-anonymity and 2-diversity

Age	ZIP code	Disease
50	38xxx	Yellow fever
50	38xxx	Dementia

If a data set exists wherein all individuals within a group share the same disease, and the group is k-anonymity, the adversary can be sure about the diagnosis (e.g. a data set, within all individuals in the ZIP code 38xxx and an age equals to 50 has dementia. If the adversary knows that a specific individual lives in this ZIP code and is 50 years old, he can conclude that the specific individual has also dementia).

Techniques like l-diversity (e.g. a k-anonymity data set with at least l different diseases for each group of equal characteristics [13]) or t-closeness (which also deals with the information gain with respect to the distribution of the diseases [11]) deal with this problem. In case that an adversary has information about all individuals in the data set except one, he can conclude the correct outcome of an individual. [10]

Therefore, this techniques are not appropriate to guarantee privacy.

The approach to deal with privacy, discussed in this paper, is the differential privacy framework, that can guarantee privacy by adding a specific amount of noise to the outcome of queries to protect individuals at the expense of accuracy.

The objective of public available data sets, e.g. in the context of medicine, is to protect individuals by permit scientists to extract the general interdependencies for research purposes simultaneously. In this context, machine learning is an often-used instrument for this task, to generate an abstract model out of the underlying data, regardless of individual data records.

Machine learning models like decision trees (which allow an extensive visualization of internal decisions to determine the outcome) are used to get an insight of the interdependencies, but are not the best models for predictions to new data instances. Therefore, models like random forests (which combine several decision trees) can compensate possible weaknesses of decision trees and usually allow better predictions.

To evaluate the usefulness of decision tree based machine learning models on differential privacy protected data sets, an own implementation of a private random decision tree classifier is introduced and extended to an ensemble method (private random forest). Afterwards, the accuracies of both methods (with their corresponding non-private equivalents) are evaluated.

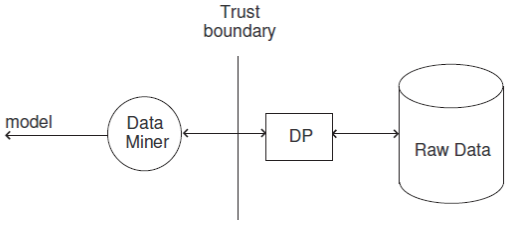
II. Differential Privacy

In the first main part, the differential privacy framework is introduced, followed by an example, which shows also the disadvantages of this technique and a feasible countermeasure.

Differential privacy allows data owners to provide publicly accessible interfaces to databases, which contain sensitive information about individuals, by modifying the outcome of a query with a specific amount of noise, so that an adversary cannot determine if a specific individual is in the data set or not. With this technique, data owners can provide third persons access to a private record set and guarantee privacy at the same time. [8]

Figure 1 visualizes the two components around the so called 'trust boundary'. Differential privacy is applied to the outcome of queries on the side of the data owner. The owner specifies a privacy budget which determines the amount of noise. The data miner on the other side can use and ration this budget to query the database. Every result contains an amount of noise, but the amount is depending on the budget used by a query. He can perform a small number of queries with a small amount of noise, or consequently several queries with a large amount of noise. [8]

Fig. 1: Data mining with a differential privacy access interface [8]



The objective of differential privacy is to minimize the negative effects for any individual when the data set is public available. [1] When the noisy outcome of a function does not allow conclusions about the existence of an individual in a record set, this objective is satisfied, independently 'on an adversary's background knowledge or computation power'. [8]

A. The Framework

Definition 1: The random function $\tilde{f}$ represents a random variable $\tilde{f}(D)$ for a given dataset D. For example, $\tilde{f}$ can be an aggregate function that sums up values of an attribute in the dataset D plus a given amount of noise from a distribution function. [10]

Definition 2: The distance function d, given two datasets D and D', specifies the lower bound of steps needed to transform D into D'. This can be done by modifying a record, or by insertions and deletions. If D and D' contains the same records, except one, $d(D, D')$ is equal to 1. In this case, D and D' are so called neighboring datasets. [10]

Definition 3: A function that 'takes a dataset as input' is called query f. 'The answer to the query f is denoted $f(D)$'. [10]

Definition 4: The random function $\tilde{f}$ guarantees (ϵ, δ)-differential privacy, given the datasets D and D' with $d(D, D') = 1$ and S in the range of $\tilde{f}$, so that

$$Pr(\tilde{f}(D) \in S) \leq \delta + e^{\epsilon} Pr(\tilde{f}(D') \in S) \quad [4]$$

The parameter ϵ specifies 'the level of privacy' [8] and is 'called the privacy budget' [10]. The lower ϵ, the lower the difference of the two sides of the inequation and the stronger the privacy. [10]
Usually, the values for ϵ are smaller than one. [8]

The number e represents the Euler's number (≈ 2.7182) and is used, because it pertains to the Laplacian distribution, which allows to shift the mean and to restrict the probabilites to given boundaries. [1]

Bambauer et al. show in [1] a different point of view: e^{ϵ} determindes the ratio of probabilities, that the two neighboring data sets returns the same outcome, as shown in the following inequation:

$$\frac{1}{e^{\epsilon}} \leq \frac{Pr(Responsе = r|D)}{Pr(Responsе = r|D')} \leq e^{\epsilon} \quad [5]$$

The parameter δ 'represents the probability that a [...] [function's] output varies by more than a factor of e^{ϵ}', because the function returns a sample in the dataset indepently with probability δ. Therewith, $n\delta$ records are released. To avoid information 'leakage, δ must be smaller than $1/n$ '. It also applies that smaller values for δ, in conjunction with small values of ϵ, result in higher privacy. [10]

In this paper, the focus is on $(\epsilon, 0)$-differential privacy, so called ϵ-differential privacy, which protects the privacy of an individual better than (ϵ, δ)-differential privacy. [5]

The privacy budget for consecutive queries 'can be accumulated to provide a differential privacy bound over all the queries'. The more queries are done, the higher are the costs. If the privacy budget is exhausted, additional queries will not be answered. It is also possible to 'bound the overall privacy budget (over all data consumers)' to 'deal with collusion between adversaries'. [8]

The adjustment of the output of a query f, depends on the noise which is necessary to achieve e-differential privacy, is determined by the sensitivity.

Definition 5: The sensitivity of a query f is

$$S(f, ||.||) = \max_{d(D,D')=1} ||f(D) - f(D')|| \quad [4]$$

The sensitivity measures the maximal influence of an instance on the output whether it is in the relevant set for the output or not. [8]

Another approach is the *exponential mechanism*, which uses a score function with a probability distribution to select instances for the relevant set. [8]

Ji et al. show in [10] the relation between the exponential mechanism and the Laplacian mechanism, which results in answers based on the same distrubtion, but the exponential menchasim only needs one half of the the privacy budget.

Theorem 1: The random function $\tilde{f}$

$$\tilde{f}(D) = f(D) + Laplace(S(f)/\epsilon)$$

guarantees ϵ-differential privacy. [8]

This implies that the smaller ϵ, the bigger the amount of noise which is added to the outcome of the query, and vice versa. [9]

The maximum influence of an instance for a count query is 1. Therefore, the amount of noise which is added to the outcome to guarantee differential privacy is

$$\tilde{f}(D) = f(D) + Laplace(1/\epsilon) \quad [8]$$

This means also that the amount of noise only depends on ϵ. This means, that the bigger the dataset D is, the lower is the relativle amount of noise. [8]

An alternative version is provided in the paper by Ji et al. in [10], which applies Gaussian noise to the output of a query f, instead of Laplacian noise. Therewith, a large amount of noise is less likely, but 'it only preservesϵ, δ-differential privacy for some $\delta > 0$, which is weaker than e-differential privacy'.

Theorem 2: If k ϵ-differential privacy queries $\tilde{f}$ are applied consecutively, the overall usage of the privacy budget is $\sum_k \epsilon_k$. Jagannathan et al. describe this in [9] as the *composition theorem*.

As described under definition 4, the costs of consecutive queries are totaled up. Therefore, the number of sequential queries needed by a learning algorithm affects the accuracy of the learner. This concerns decision trees which queries the database several times on each decision point. [9]

Theorem 3: If k ϵ-differential privacy queries $\tilde{f}$ are applied in each case on disjoint subsets of D, the overall used privacy budget is ϵ. This affects the leafs of a decision tree with disjoint subsets and is called the '*parallel composition theorem*'. [9]

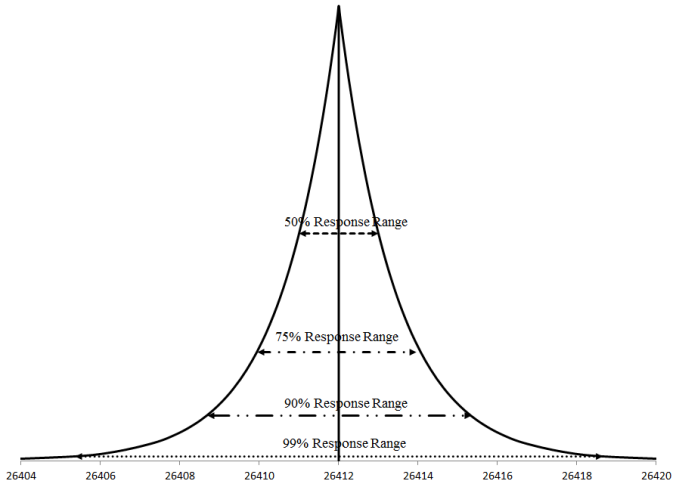
B. Examples and critique

Imagine a fictional public database D contains the assets of all inhabitants in Washington. If the correct answer for the number of millionaires is 26,412 and the database provides differential privacy, the output fluctuates around the true value. If an adversary does not know the wealth of Bill Gates, but knows the wealth of all other 26,411 millionaires (which represents the data set D' with $d(D, D') = 1$), the probabilities of results for a query, given the parameter $\epsilon = ln(2)$, is shown in Figure 2.

In this case, the adversary can not determine if Bill Gates is a millionaire, but simultaneously data miners are able to recieve answers that fluctuates around the true value. [1]

The amount of noise in this case is small, because the maximum sensitivity for count queries is 1. The situation is quite different for other kind of queries like $sum(.)$ or $average(.)$. Baumbauer et al. provides in [1] an example for a case with a significantly higher influence of an individuum to the outcome: Booneville, Kentucky is a small town with a low average income of $23,426 and approximately 100 residents. If a billionaire moves to Booneville, his income has a very strong influence to the average income, which must be compensate by

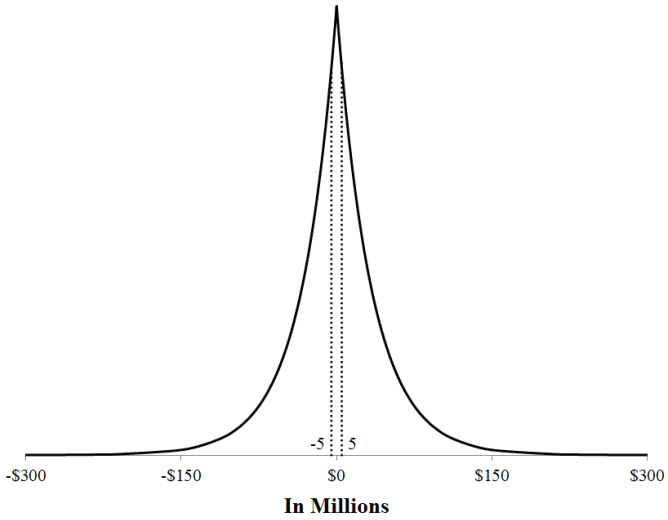
Fig. 2: Possible results for a count query with $\epsilon = ln(2)$ [1]



the differential privacy database. To protect the billionaire, ϵ is set to 0.5. In this example, the adversary knows the existence of the billionaire, but not his place of resident.

The least income of a resident in Booneville is $0. This results in the sensitivity of $1 billion. To satisfy ϵ-differential privacy, the noise must have a standard deviation of 48 million. Only with a probability of 3%, the noise added answer is in the range $\pm$ $ 1 million. Figure 3 visualizes the probability distribution. [1]

Fig. 3: Possible results for an average query with $\epsilon = 0.5$ and a sensitivity of $1 billion [1]



But on the other hand, differential privacy should not only protect individuals within the data set, but also individuals outside. As shown in definition 4, the amount of noise only depends on ϵ and on the sensitivity. If this database protects absent individuals, the amount of noise is independent from the underlying data and the query. [1]

Image the billionare could move to one of two possible towns like Booneville with at least one inidividual without an income and no billionaire up to now, in each case. If the

adversary queries both databases, he can estimate in which town the billionare most likely has moved.

But the authors of [1] go way beyond this. Because differentially privacy is by definition (if it also protects absent individuals) query- and data-independent, the noise should also add to queries, which could not include a specific individual. For example, if the billionare is male, also queries which only consider women, must be altered, because the only missing information for an adversary can be the billionares' sex.

Actually, the '*Census Bureau*' provides a differential privacy database containing individuals of Booneville. The average income of $21,907 is modified by noise in the range of $\pm$11,247. Outcomes of queries within this range are very unlikely for the existence og a billionaire in this town. This information can used by an adversary to exclude this town out of all possible places of the residence for the billionaire. [1]

Bambauer (who coauthored the paper [1] with the examples shown above) and Muralidhar provides a solution in [2] for the problem with the big sensitivity because of one outlier in the data set (the exemplary billionaire). They propose to use the median, instead of the mean. With the median, which is robust against outlier, the sensitivity is not skewed by the influence of the one billionaire in this example. If the median is used, the outcome of the noisy queries does not give a hint whether one billionaire is in the data set or not.

Another weakness of differential privacy is that it only focus on one individual, not on groups. But in practice, this is not a big problem, because the access to the database is published for research purposes and statistics. Therefore, it is sufficient to protect only individuals by preserve the fundamental circumstances simultaneously. [1]

III. Differential Privacy for Decision Trees

This chapter concern to the theorems 2 (composition theorem) and 3 (parallel composition theorem). As a short recap, in accordance with the composition theorem, the used privacy budget ϵ over the full record set is summed up. The parallel composition theorem specifies that several queries on disjoint subsets only use ϵ-differential privacy. [9]

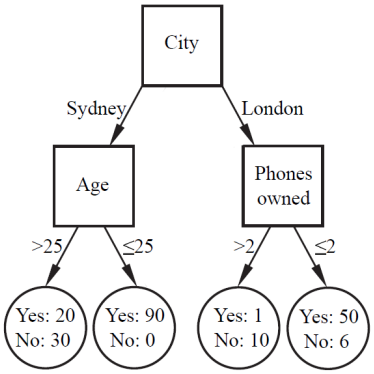
This part introduces the application of decision trees for differentially private databases and deals with the strengths and weaknesses.

The standard non-privacy ID3 decision tree algorithm builds up a tree structure (as shown in Figure 4), based on decision points, so called nodes (which represents the attributes) and the concrete values on the edges to the next decision point. This works recursively until all records correspondig to a node share the same class value.

A. Private ID3 Trees

The selection of an attribute for a decision point is based on the information gain with respect to the entropy regarding the class attribute. For each new decision point, the algorithm queries the database for each attribute to measure the influence on the class label, to find the attribute, that divides the records best. This results in many noisy queries, which affects (on the root node) each record and yields, with respect to the

Fig. 4: Structure of a decision tree [7]



composition theorem, to a usage of $k\epsilon$ of the privacy budget, for k queries. Therefore, ϵ must be very small to perform all queries that are needed, which results in strong privacy and inaccurate outcomes. Jagannathan et al. demonstrates in [9] also how an ID3 decision tree can be modified for differential privacy to use the privacy budget more economical, by querying the database only count queries for the leaf nodes with sensitivity of 1 by a query of the size of records in the leaf, and another query that asks for the number of records for each class in the leaf. These results can be used in the following steps (to decide the class label), instead of additional queries. For the entropy estimation, additional queries with the sensitivity of 1 are needed.

The queries on the same level contains disjoint subsets and can use the parallel composition theorem. Queries on different levels of the tree are added up. Therefore, for k queries, the privacy budget $k\epsilon$ is used, which results in so much noise, that the results are nearly useless. In [9], this algorithm is applied to a few record sets (shown in Figure 5). The accuracy of the private ID3 decision tree is like a naïve classifier, which labels all test instances with the most likely class. In one case, the private ID3 tree was even worse than a naïve classifier (in the penultimate column).

Fig. 5: Accuracy of private ID3 decision trees for $\epsilon = 0.5$ [9]

Data set	# rows	# queries	Accuracy original **ID3**	Accuracy Most Freq Class	Accuracy Private **ID3**
Nursery	12960	36	98.19%	33%	46%
Cong. Votes	435	136	94.48%	61%	40%
Mushroom	8124	253	100%	52%	56%

B. Private Random Decision Trees

Another approach described in [9] by Jagannathan et al. is the '*private random decision tree algorithm*', which chooses the attributes for the decision points randomly out of the set of available attributes recursively for a given height, completely without access to the data. The chosen attributes along a path in the tree are unique, but in parallel paths, the same attributes are possible (the algorithm selects for each node an unused attribute for a specific path at random).

If all discrete attributes and the corresponding values are known, this can be applied easily. More difficult are continuous

attributes. A split at random is possible, but a better approach gives the exponential mechanism as mentioned in definition 5. This mechanism returns instances, based on a probability distribution, more likely, if their attributes are good split points. [8]

Also, all leaves are at the same height of the tree and aim at disjoint subsets. Therefore, this tree only uses count queries with a sensitivity of 1 and an overall privacy budget usage of ϵ, instead of $k\epsilon$ like in the case of the private ID3 algorithm, which results in a smaller number of queries with higher accuracy. The representation of this random decision tree is the tree itself and a leaf vector of the size $m * t$ with m leaf nodes and t class labels. [9]

If several random decision trees are combined in an ensemble of n trees, the used privacy budget is summed up and noise in the amount of $Laplacian(n/\epsilon)$ is added to each query in the leaf vectors. [9]

Jagannathan et al. presents in [9] that the private random decision tree can achieve a higher accuracy score in comparison with the private ID3 decision tree. They applied both algorithms to the mushroom data set, public available on the 'UCI Machine Learning Repository'.

For the ID3 implementation, their accuracy, based on ϵ can be seen in Figure 6. Consider the inverse ϵ values on the abscissa. It can be seen, that the accuracy of the ID3 algorithm decreases strongly for smaller values of ϵ.

Fig. 6: Accuracy of private ID3 decision trees applied on the ICU mushroom data set [9]

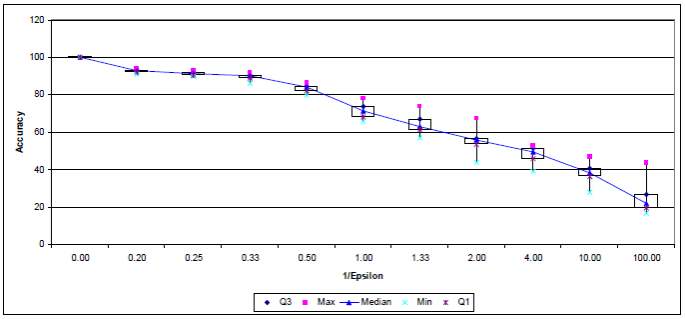


Figure 7 shows the accuracy of the private random decision tree algorithm for the same data set. This technique has a good accuracy on the test set until ϵ is set to 0.01, equals to $1/\epsilon = 100$.

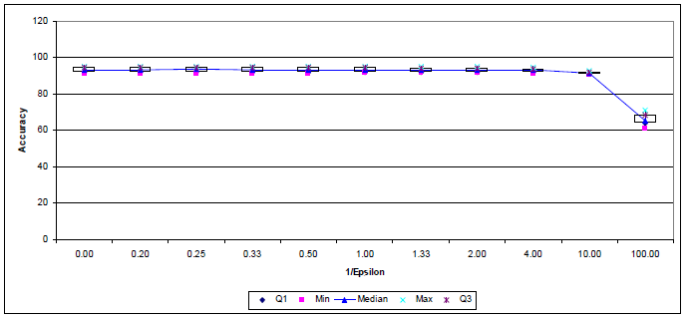
IV. Own Implementation of a Private Random Decision Tree

This chapter deals with an own implementation of a database procedure for differential privacy queries in Transact-SQL (T-SQL) and a private random decision tree algorithm in Python, which is extended to an ensemble method to combine several tree instances to a private random forest.

A. Differential Private Count Procedure

The procedure, attached in Appendix A, allows count queries (which are necessary for decision trees) on arbitrary tables and outcome variables. It takes the names for the table and the

Fig. 7: Accuracy of private random decision trees applied on the ICU mushroom data set [9]



outcome attribute, a condition and a parameter for the ϵ value as input. Afterwards, a random number u by the uniform distribution in the interval $[0, 1]$ is generated and shifted by -0.5 to get values in the range of $[-0.5, 0.5]$ to facilitate a noise above and below the true value.

With respect to this random number, the result of the user defined query is manipulated by Laplacian noise, by the formula

$$\mu - b * sign(u) * log(1 - 2|u|) \quad [4]$$

The variable μ represents the true outcome of the count function, and b is equal to $1/\epsilon$.

B. Differential Private Random Decision Tree

The tree structure consists of nodes for attributes and edges for the specific (here: discrete) values for the attribute. The tree is build recursively until the maximum depth is reached or all attributes are used along a path in the tree. In this case, the node remains as a leaf node and executes the SQL condition to determine the most common class label for all instances that match the condition.

The root node (Appendix B) is an instance within the RandomDecisionTree class (Appendix C), which enhances the node structure by methods for visualization and prediction. To classify new data instances, the associated path can be traversed with respect to the values of the new instance and the corresponding label is returned. To consider combinations of values that were not in the training set, the a priori probability for each class is stored in the root node.

In this implementation, only a learner for a two-class problem is implemented, but it is also possible to store not only the more likely class in the leaf node, but also the noised amount. This provides the opportunity for later updates, when more data is collected.

C. Differential Private Random Forest

To prevent overfitting of a single decision tree, often several weak learners are combined, which can classify new instances and do a majority vote. This is implemented in the RandomForest class (Appendix D), which trains several private random decision trees. In this case, the queries are not disjoint anymore, because several trees query the same data. Therefore, the privacy budget is divided by the number of the trees, which

leads to a bigger amount of noise to each query. At prediction time, it performs a majority vote and return the most likely class label. It also allows to specify the number and the depth of the underlying private random trees.

D. Evaluation

To evaluate the implementation, the census income data set, provided by the ICU Machine Learning Repository [12] is introduced. Afterwards, non-private decision trees and random forests (by the Python scikit-learn library) are applied on the data set to show the fundamental behavior on non-private data. Subsequently, the own implementations of the private random decision tree and forests are applied and compared with the non-private techniques.

1) The Data Set: The census income data set contains in the original 14 attributes of categorical and continuous values. The data is about individuals (and their corresponding attributes like age, work class, native country, or relationship) from 1994, to determine, if an individual has an income greater than $50.000 per year or below. The implementation above only regards categorical attributes. The usage of a large range of continuous (or near-continuous) attributes (like age or working hours per week) as discrete attributes leads to huge tree structures and computation times, which cannot be handled with current systems, not even by rented cloud computing instances at Amazon Web Services. Therefore, the attributes are grouped into several, nearly same sized, bins. For example, the attribute age contains after the transformation process the values '<=37' or '>=38'.

Also, the number of records for each class label is adjusted to nearly the same quantity. Therewith, the accuracy of the implementation can easily compared with a naïve classifier, which labels all data with the most likely class. In the original data set, the income was not equal distributed. Approximately 75% of the people in 1994 had an income below $50.000. So, a naïve classifier has also an accuracy of 75%, which is, in general, difficult to beat. Therefore, it is necessary to delete instances to get a nearly equal distributed data set. Figure 8 shows a small sample of the data set with the remaining attributes.

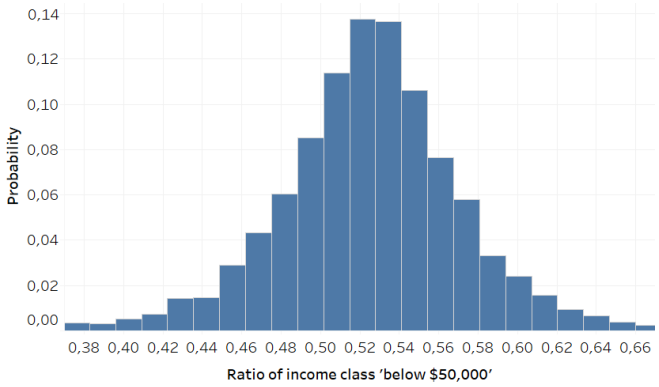
Fig. 8: Sample of the modified ICU census data set

age	workclass	education_level	relationship	race	sex	hours_per_week	income
<=37	Private	Masters	Not-in-family	Asian-Pac-Islander	Male	<=36	0
<=37	Private	HS_or_below	Not-in-family	White	Male	37-50	0
>=38	Private	HS_or_below	Not-in-family	Black	Female	<=36	0
>=38	Private	Bachelors	Not-in-family	White	Male	>=51	1
<=37	Private	HS_or_below	Own-child	White	Male	37-50	0

Figure 9 shows the distribution of the outcome of several calls of the Laplacian count query with $\epsilon = 0.001$. The Laplacian distribution is identifiable, the accuracy of a naïve classifier fluctuates around 52.6%.

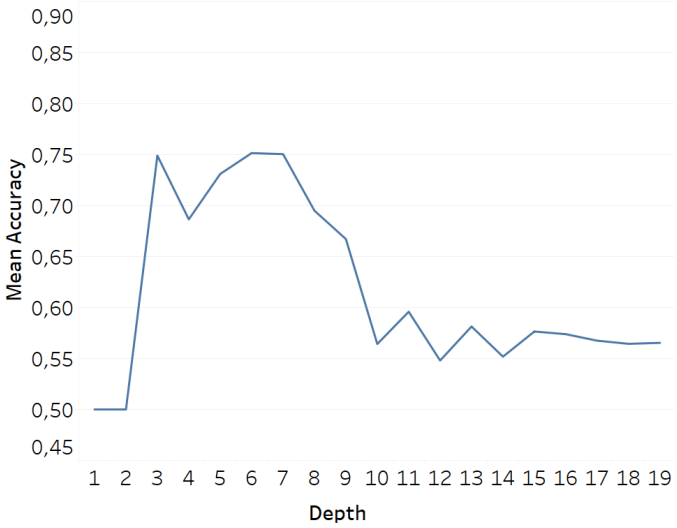
For testing purposes, the data set is split into a traing and a testing set.

Fig. 9: Results of the Laplacian count query about the ratio of the income classes



2) Non-private Decision Tree: To proof the existence of general interdependencies in the modified data set, a standard non-private decision tree with several depths is applied on the data set without Laplacian noise. The accuracy (shown is Figure 10) for depths greater than 3 are always above the naïve classifier, the best accuracy is about 75%. The decreasing accuracy for a bigger number of decision points can be explained by overfitting to the training set, which leads to a lower accuracy on the testing set.

Fig. 10: Accuracy of a non-private decision tree without Laplacian noise



3) Non-private Random Forest: Random Forests combine several decision trees and are able, within certain limits, to compensate the effect of overfitting. The mean accuracy with respect to the parameters depth and number of underlying trees can be seen in the heatmap in Figure 11. The highest mean accuracy is 80.17% and therewith above the mean accuracy of a single decision tree. The lowest accuracy is 63.99%, which is also above the lowest accuracy of the decision tree. The highest accuracy of a random forest on the data set is 86.6% (not in the illustration) for a number of trees equals to 7 and a depth of 1. Figure 12 demonstrates the influence of the depth

Fig. 11: Mean accuracy of a non-private random forest without Laplacian noise

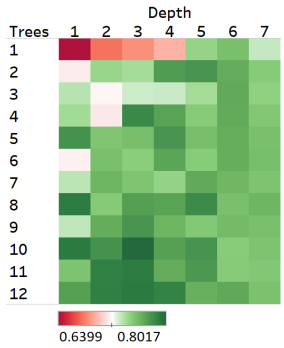
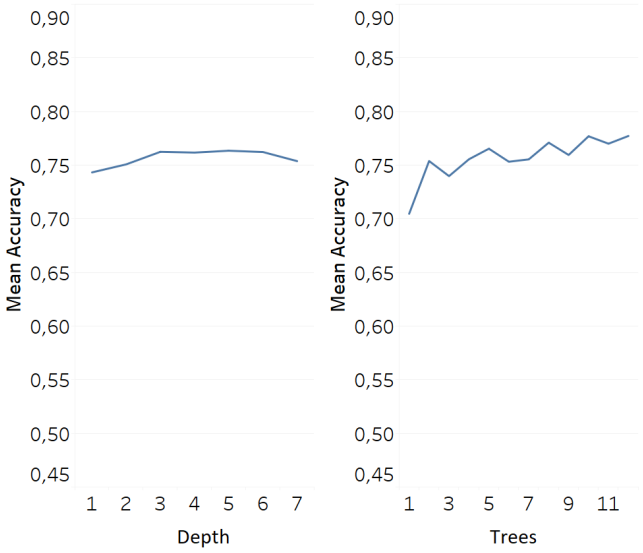


Fig. 12: Influence of the depth and the number of trees to the mean accuracy of non-private random forests without Laplacian noise

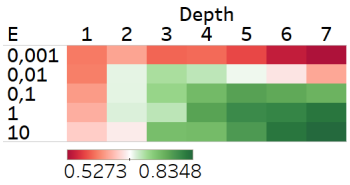


and the number of trees to the mean accuracy separately. The number of trees is positive correlated to the accuracy, the depth has one global maximum and is decreasing for higher values.

For the following illustrations, a maximal depth of seven is considered, because the modified data set has only seven different attributes plus the outcome attribute. The private random decision tree chooses for each level of the tree one attribute and is consequently limited to a depth of seven. To enhance comparability, all illustrations show the accuracy on the axis of ordinates in the interval $[0.45, 0.9]$.

4) Private Random Decision Tree: The accuracy of private random decision trees, trained on the Laplacian count query, is visualized in Figure 13. An increasing ϵ leads to more accurate results, an increasing depth is also advantageous. The highest mean accuracy of 83.48% is above the highest accuracy of the non-private decision tree (75%), but below the highest overall accuracy of the non-private random forest (86.6%). For $\epsilon = 0.001$, the accuracy is mostly not better than a naïve classifier. The variance of the accuracy over several simulations (visualized in Figure 14) is decreasing for an increasing depth,

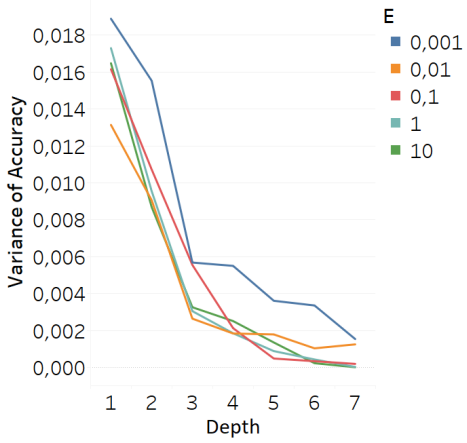
Fig. 13: Mean accuracy of a private random decision tree with Laplacian noise



for all values for ϵ. Because the tree chooses the attributes at random, the information gain with respect to the outcome variable can, e.g. for a depth of one, vary widely, if an attribute is selected with a little usefulness for the classification.

For a maximal depth of seven, the variance converges to zero, because all attributes are checked. This result is independent of the order of attributes in the tree. Therefore, all tree instances with a maximum depth check the same attributes, only the Laplacian noise causes a variance greater than zero.

Fig. 14: Variance of accuracy of a private random decision tree with Laplacian noise

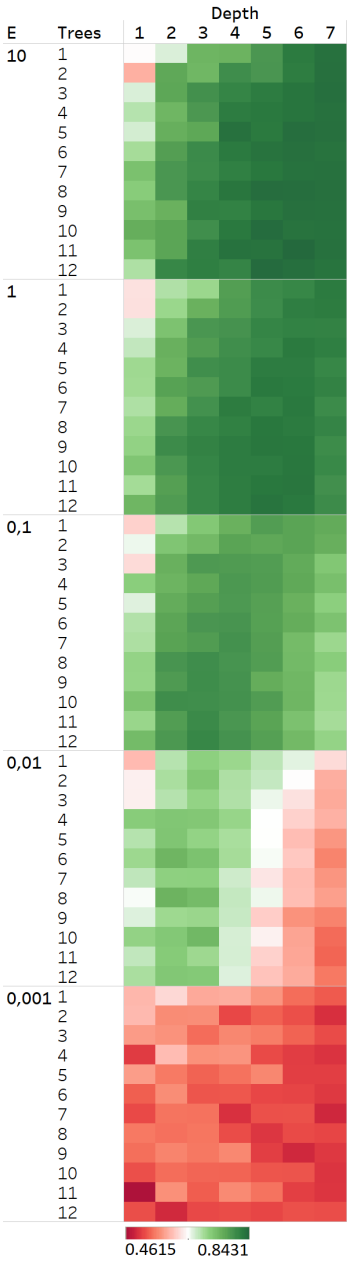


5) Private Random Forest: To improve a single private random decision tree, several trees are combined to a private random forest. There is a chance, that several trees can compensate a very inaccurate query result (with a big amount of noise), but the privacy budget is divided by the number of trees, therefore all queries have in general a bigger amount of noise, which is a tradeoff. The heatmap in Figure 15 shows the mean accuracy for random forests over several depths, internal trees and values for ϵ. For an ϵ between 1 and 10, a higher depth leads to a higher mean accuracy. For a decreasing ϵ, this is quite different. The area in the heatmap of a high mean accuracy moves to smaller depths for a smaller ϵ. This can be explained with an overfitting to noisy data, which reflects the interdependencies in an inaccurate way.

For $\epsilon = 0.001$, the mean accuracy for some instances is below the naïve classifier. These results are practically useless, if the overall a priori distribution of the classes are known.

As shown on the non-private random forest, the accuracy is positively correlated with the number of trees. This does not confirm here. As shown above, there is a tradeoff between

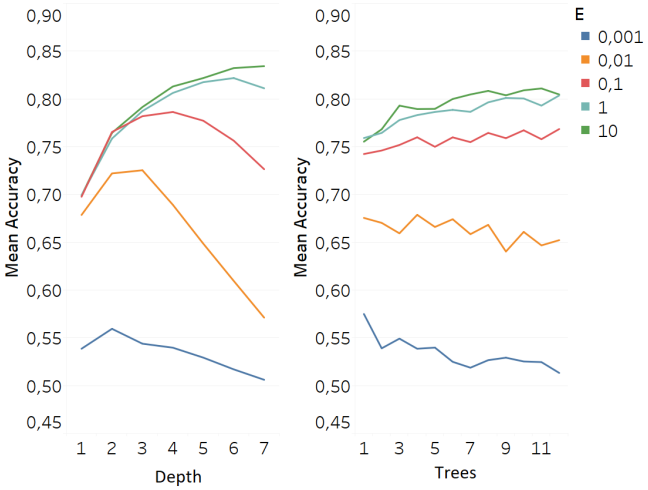
Fig. 15: Mean accuracy of a private random forest



better accuracy because of a better generalization (in case of the non-private random forests without Laplacian noise) and a possibly worse accuracy in cause of the smaller privacy budget for each underlying tree in the private random forest on noisy data. Both effects seem to compensate each other, in the considered range of trees.

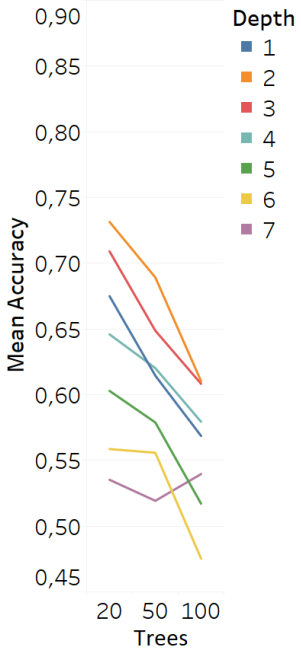
This can also be seen in Figure 16, which visualizes the influence of the depth and the number of trees separately, with respect to different values for ϵ. A high value for ϵ means a small amount of noise, which behaves like a non-private random forest on data without noise: an increasing number of underlying trees and depth increases the mean accuracy. For small values of ϵ, neither a higher depth, nor a higher number of underlying trees are conducive. The interrelationship between an increasing number of trees and a decreasing mean accuracy

Fig. 16: Influence of the depth and the number of trees to the mean accuracy with Laplacian noise



is also identifiable for simulations with a higher number of trees, as shown in Figure 17. The series for the depth of seven is conspicuous, because only in this case, the accuracy seems to increase for a higher number of trees. This can be explained with a too small amount of simulation results. A single accuracy result for a private random forest with 100 underlying trees, each with a depth of seven, needs several hours of computation. Fore more precise statements, more simulations are needed.

Fig. 17: Mean accuracy of a private random forest for $\epsilon = 0.01$



V. Discussion

All accuracy curves show the influence of an increasing ϵ to the predictions of a classifier. Because the noise is data-independent, learning algorithms on a larger data set could achieve a higher accuracy, because the relative amount of noise, for a given ϵ, is decreasing. Probably, the absolute amount of noise is not strong enough to pretend another class distribution, e.g. if the ratio between two classes are 10% and 90%, the returned class label, in spite of the Laplacian noise, can be correct. But as explained in [1], the aim of differential privacy is to protect individuals, not larger groups of individuals.

Further improvements for the implementation of the private random decision tree classifier (which also affects the superior random forest classifier) are also suitable. In this implementation, the classifier can only deal with categorical attributes. If the training data contains continuous data, the algorithm treats each possible value differently and without connection to each other. This results in an enormous tree, because it creates a child node for each singe value. If several continuous attributes are contained and randomly selected consecutively, the runtime strongly increases and so the variance of the predictions, which results unavoidable in a low accuracy, because of a strong overfitting. This problem also occurs on noisy and not noisy data.

VI. Conclusion

Overall, differential privacy is appropriate to protect individuals. Private random decision trees and forests work well by protecting them and allow the data miner the extraction of general interdependencies in the data simultaneously.

On the other hand, as shown in the billionaire example, in chapter 3, the protection of the existence of the billionaire in the data set is problematic, if mean values are used. Using the median instead of the mean can solve this problem. [2] This solves also the problem, if the billionaire is not present in the data set.

References

[1] J. Bambauer, K. Muralidhar, *A Response to the Criticisms of Fool's Gold: An Illustrated Critique of Differential Privacy*. https://blogs.harvard.edu/infolaw/2016/05/17/diffensive-privacy/ , 2016.

[2] J. Bambauer, K. Muralidhar and R. Sarathy, *Fool's Gold: an Illustrated Critique of Differential Privacy*. In 16 Vanderbilt Journal of Entertainment & Technology Law, 2014 (Forthcoming); Arizona Legal Studies Discussion Paper No. 13-47, 2013.

[3] D. Barth-Jones, *The "Re-identification" of Governor William Weld's Medical Information: A Critical Re-examination of Health Data Identification Risks and Privacy Protections, Then and Now*, 2012.

[4] E. Chen, *Laplacian random number generator*, 2007. http://de.mathworks.com/matlabcentral/fileexchange/13705-laplacian-random-number-generator?focused=5085033&tab=function

[5] A. De, *Lower bounds in differential privacy*. arXiv:1107.2183 [cs.CR], 2008.

[6] J. Domingo-Ferrer and V. Torra, *A Critique of k-Anonymity and Some of Its Enhancements* . In The Third International Conference on Availability, Reliability and Security, 2008.

[7] S. Flechter and M. Islam, *Decision Tree Classification with differential Privacy: A Survey*. In ACM Computing Surveys, Vol. X, No. Y, Article Z. arXiv:1611.01919 [cs.DB], 2016.

[8] A. Friedman and A. Schuster, *Data Mining with Differential Privacy*. In KDD, 2010.

[9] G. Jagannathan, K. Pillaipakkamnatt and R. Wright, *A Practical Differentially Private Random Decision Tree Classifier*. In Transactions on Data Privacy 5, pages 273-295, 2012.

[10] Z. Ji, Z. Lipton and C. Elkan, *Differential Privacy and Machine Learning: a Survey and Review*. arXiv:1412.7584 [cs.LG], 2014.

[11] N. Li, T. Li and S. Venkatasubramanian, *t-closeness: Privacy Beyond k-Anonymity and l-Diversity*. In IEEE 23rd International Conference on Data Engineering, 2007.

[12] M. Lichman, *UCI Machine Learning Repository*, 2007. https://archive.ics.uci.edu/ml/datasets/census+income.

[13] A. Machanavajjhala, J. Gehrke and D. Kifer, *l-Diversity: Privacy Beyond k-Anonymity* In Journal ACM Transactions on Knowledge Discovery from Data (TKDD), Volume 1 Issue 1 March 2007, Article No. 3.

VII. Appendix

A: Laplacian count T-SQL procedure

```
CREATE PROCEDURE laplacianCount
  -- name of the table, which is affected by
  -- the query
  @table VARCHAR(MAX),
  -- class attribute
  @attribute VARCHAR(MAX),
  -- epsilon
  @e FLOAT,
  --condition along the path in the tree
  @condition VARCHAR(MAX)
AS
BEGIN
  -- generate a random number by the uniform
  -- distribution between 0 and 1
  -- and shift it by - 0.5 to get values in
  -- the range of -0.5 and 0.5
  DECLARE @random VARCHAR(12)
  SET @random = ROUND(rand() - 0.5 , 10)

  -- b is the inverse of the epsilon parameter
  DECLARE @b VARCHAR(20)
  SET @b = ROUND( 1 / @e , 20)

  -- if @random is equal to 0.5, the
  -- argument of the log function is equals
  -- to 0 and undefined. While @random is
  -- equals to 0.5, new random numbers
  -- are generated
  WHILE (ABS(CAST(@random AS FLOAT)) = 0.5)
    BEGIN
      SET @random = ROUND(rand() - 0.5 , 10)
    END

  -- Execute the query and add Laplacian
  -- noise to the outcome. The noisy
  -- result is rounded and casted to an
  -- integer to get a natural number
  -- for the count query

  -- The formula is: quantity - (1 / epsilon
  -- * sign (@random *
  -- log ( 1 - 2 * Abs(@random))))
  DECLARE @query VARCHAR(MAX)
  SET @query = 'SELECT
  CAST(
    ROUND
      (COUNT(*) - (' + @b + '* sign(' +
          @random + ')) * LOG(1-2*ABS(' +
          @random + '))
    ,0)
     AS int) As quantity
  FROM ' + @table + '
  WHERE ' + @condition

  EXEC(@query)
END
```

B: Class Node

```
import random
import pandas as pd

# This class represtens the recursive tree
# structure.
#
# Attributes:
# - attributes: The full list of available
# attributes from which the decision
# trees is able to choose from
# - max_depth: The depth of each decision tree
# - e: The epsilon parameter for the Laplacian
# distribution
# - conn: Connection object to
# the database
# - table: Underlying table for the queries
# - a_priori: Pointer to the root node for
# updating the number
# of records for each class
# - condition: Relevant for leaf nodes.
# Contains the path (attributes and values)
# in a sql query string
# - value: The value the attribute of the
# parent node.
# This allows a faster searching the tree
# structure
#
class Node():
   def __init__(self, attributes, max_depth,
     e, conn, table, a_priori,
     condition='', value=None):

     # create a new instance, for deletion
     # purposes.
     # the chosen attribute is removed
     # from the list of available attributes
       self.attributes = attributes.copy()
       self.children = dict()
       self.attribute = None
       self.value = value
       self.label = "" # class label
       self.condition = condition #SQL query
       self.conn = conn
       self.e = e
       self.table = table

       # add recursively new levels while
     # the max_depth
       # is > 0 and attributes are available.
       if max_depth > 0 and
       len(self.attributes) > 0:

       # Select a random attribute
       self.attribute = random.choice(list(
              self.attributes.items()))[0]

       values = self.attributes.pop(
              self.attribute, None)

       # Create a child node for each
       # distinct value
       # of the chosen attribute
       for value in values:
         condition = self.update_condition(
           self.condition, self.attribute,
           value)
         child = Node(self.attributes,
           max_depth - 1,
           e, conn, table, a_priori,
           condition, value)

         self.children[value] = child
     else: #leaf
       self.train(a_priori)

   # this updates the SQL condition
   def update_condition(self, parent_condition,
     additional_condition, value):
     condition = parent_condition
     # concatenate the new condition to the
     # existing one
     if condition != "":
       condition = condition + " AND "
     try: #try, if value is an integer
       condition = condition +
           str(additional_condition) +
         " LIKE '" + str(int(value)) + "' "
     except:
       condition = condition +
           str(additional_condition) +
         " LIKE '" + str(value) + "' "
     return condition

   # train the model (2-class-problem) by
   # executing the SQL condition
   def train(self, a_priori):
     query0 = "EXEC laplacianCount @table = '"
        + self.table +  "', @attribute =
            'income',
       @e = " + str(self.e)
       + ", @condition = '"
       + self.condition.replace("'", "''")
       + " AND income = 0'"

     query1 = "EXEC laplacianCount @table = '"
        + self.table + "', @attribute =
            'income',
       @e = " + str(self.e)
       + ", @condition = '"
       + self.condition.replace("'", "''")
       + " AND income = 1'"

     # number of records for the
     # specific classes
     quantity0 = pd.read_sql(query0,
         self.conn).values[0][0]
     quantity1 = pd.read_sql(query1,
         self.conn).values[0][0]
     # determine the label for the leaf
     if quantity0 >= quantity1:
       self.label = 0
     else:
       self.label = 1

     # update a priori probability of the
     # root node
     a_priori[0] += quantity0
     a_priori[1] += quantity1
```

C: Class RandomDecisionTree

```
import pandas as pd
import numpy as np

# This class represents the random decision
# tree object, which contains the
# root node and functions for visualization
# and preditcion.
# Because the random forest class uses
# threads to call this class,
# he parameters for the constructor
# are passed within a list structure.
#
# To make predictions for completely
# unknown instances, the tree collects
# the quantity of instances for each
# class and returns, if no prediction
# can be made, with the most likely
# class based on the a priori probability.
class RandomDecisionTree(object):
  def __init__(self, inputs):
    max_depth = inputs[0]
    attributes = inputs[1]
    e = inputs[2]
    conn = inputs[3]
    table = inputs[4]

    # 2 possible classes
    self.a_priori = list()
    self.a_priori.append(0)
    self.a_priori.append(0)

    # Create the root node
    self.root = Node(attributes,
      max_depth, e, conn,
      table, self.a_priori)

  # Visualize the tree structure
  def visualise(self):
    self.__traverse_for_visualisation__
      (self.root, 0)

  def __traverse_for_visualisation__
      (self, node, level):

    if node:
      print(str(level) + ' '
        + str(node.attribute))

      #iterate over all children
      if len(node.children) > 0:
        for key, value in
          node.children.items():

          self.__traverse_for_visualisation__
          (node.children[key], level + 1)
    else: #leaf
      print(node.condition + ", label: "
        + str(node.label))

  # predict the class
  # for a data instance
  # df: DataFrame
  def predict(self, df):
    predictions = list()
    row = df
    node = self.root
    while len(node.children) > 0:
      value = row.loc[node.attribute]
      # check if key in dict
      if value in node.children:
        node = node.children[value]
      else:
        # value combination
        # not in training set
        predictions.append(
          self.get_class_a_priori())
            break

    # no approproate
    # instance in training set
    if node.label == None:
      predictions.append(
        self.get_class_a_priori())
    else:
      predictions.append(node.label)
    return predictions[0]

  # return the most likely class
  def get_class_a_priori(self):
    if self.a_priori[0] > self.a_priori[1]:
      return 0
    else:
      return 1
```

D: Class RandomForest

```
import pandas as pd
import threading
import queue

# This class represents an ensemble
# structe of private random decision trees.
# It allows the parallel creation and
# training of the contained trees.
#
# Parameters for constructor:
# - number_trees: Number of random decision
# trees, which should be used
# - max_depth: The depth of each decision tree
# - attributes: The full list of available
# attributes from which the decision
# trees can choose from
# - e: the epsilon parameter
# for the Laplacian distribution
# - conn: connection object
# for the database
# - table: underlying table for the queries

class RandomForest(object):
  def __init__(self, number_trees,
    max_depth, attributes, e,
    conn, table):

    #list for the trees
    self.trees = list()

    # the more trees are used, the higher
    # the noise
    self.e_per_tree = e / number_trees

    # Queue for the output of the parallel
        threads
    que = queue.Queue()
    for i in range(0,number_trees):
      thr = threading.Thread(
        target = lambda q,
        arg : q.put(RandomDecisionTree(arg)),
        args = (que,
        [max_depth,attributes,
        self.e_per_tree,conn[i],
        table, '', None]))

      thr.start()
      thr.join()
    while len(self.trees) < number_trees:
      # wait until all trees are build
      # and trained
      self.trees.append(que.get())

  # This function accepts a
  # pandas DataFrame object
  # with the instances for predictions
  # and returns a list with the
  # estimated classes.
  def predict(self, df):
    predictions = list()
    for (idx, row) in df.iterrows():
      # the variable vote collects
      # the votes of all trees.
      # This can easily adapt to
      # parallel execution.
      vote = 0
      for tree in self.trees:
        # prediction for a specific
        # instance for each tree
        vote += tree.predict(row)

    # Compute prediction
    # If vote is greater than the half
    # number of trees,
    # more of the half trees voted for
    # class 1.
    # Otherwise the prediction is 0.
    # It is also possible to use a weighted
    # majority vote
    # or use the posterior probability over
    # all known instances
    if vote > len(self.trees) / 2:
      predictions.append(1)
    else:
      predictions.append(0)
    return pd.Series(predictions,
        dtype='int64')
```