Approximate Decision Tree Induction over Approximately Engineered Data Features

Dominik Ślęzak[1(✉)] and Agnieszka Chądzyńska-Krasowska[2]

[1] Institute of Informatics, University of Warsaw, Warsaw, Poland
slezak@mimuw.edu.pl
[2] Polish-Japanese Academy of Information Technology, Warsaw, Poland

Abstract. We propose a simple SQL-based decision tree induction algorithm which makes its heuristic choices how to split the data basing on the results of automatically generated analytical queries. We run this algorithm using standard SQL and the approximate SQL engine which works on granulated data summaries. We compare the accuracy of trees obtained in these two modes on the real-world dataset provided to participants of the Suspicious Network Event Recognition competition organized at IEEE BigData 2019. We investigate whether trees induced using approximate SQL queries – although execution of such queries is incomparably faster – may yield poorer accuracy than in the standard scenario. Next, we investigate features – inputs to the decision tree induction algorithm – derived using SQL from a bigger associated data table which was provided in the aforementioned competition too. As before, we run standard and approximate SQL, although again, that latter mode needs to be checked with respect to the accuracy of trees learnt over the data with approximately extracted features.

Keywords: SQL-based decision tree induction · SQL-based feature engineering · Approximate SQL engines · Granulated data summarization · Big data analytics · Cybersecurity analytics

1 Introduction

Every typical KDD process consists of several stages, such as data preparation, attribute construction and selection, decision model induction and more [8,21]. Given the growing sizes of data required to be mined, there are a number of approaches attempting to utilize higher-level interfaces to data storage and data processing systems instead of operating directly on raw data sources. With this respect, employment of relational database systems and SQL is one of intensively examined opportunities [13,22].

In our research, we often refer to KDD methods based on standard SQL queries supported by most of database vendors. We rewrite some of algorithms which are well-known in the KDD domain to illustrate how basic SQL procedures can replace lower-level computations. This way KDD solutions can gain important data management and computational scalability features of modern database systems. Moreover, users who are familiar with SQL can easily introduce changes into previous implementations, at the level which is specific to declarative rather than imperative languages.

R. Bello et al. (Eds.): IJCRS 2020, LNAI 12179, pp. 376–384, 2020.
https://doi.org/10.1007/978-3-030-52705-1_28

In this paper, for the purpose of illustrating the role that database systems can play in KDD, we introduce a very simple new version of SQL-based decision tree induction algorithm. This particular algorithm is dedicated to datasets with numeric attributes and binary decisions. It makes its heuristic "attribute greater/lower than value" split choices by basing on the results of automatically generated aggregate queries. Such approach is surely not novel [7,10]. Still, our contribution is twofold. First, we run our algorithm using one of approximate SQL engines available in the market [17], in order to verify whether decision trees constructed using approximate queries may yield poorer accuracy than while basing on classical exact SQL. Second, given the multi-table characteristics of the considered real-world dataset [5], we investigate whether newly engineered attributes – added to the main training table by executing analytical SQL statements over another available data table – could be derived using approximate queries instead of exact ones, with no harm to the efficiency of further learning mechanisms.

Both above aspects reflect the same challenge, although they refer to different KDD stages – attribute engineering (exemplified by SQL-based usage of one-to-many relation between data tables) and decision model construction (exemplified by our decision tree induction algorithm). The question is whether approximate SQL – which can be orders of magnitude faster than exact SQL over big datasets – is able to drive KDD processes accurately enough, so acceleration is achieved without losing too much quality. Indeed, one could suspect that the quality of the aforementioned splits made during decision tree construction is potentially worse if their heuristic evaluation relies on not-fully-precise calculations over the training data. Analogously, one may be afraid of using imprecisely derived values of newly created attributes as the input to any machine learning algorithm, no matter whether that algorithm itself is based on exact or approximate computations. Our goal is to illustrate to what extent such worries are justified.

The rest of the paper is structured as follows. Section 2 refers to some related works. Section 3 describes the dataset used in our studies. Section 4 outlines the proposed decision tree induction algorithm. Section 5 reports our experimental results in four modes: running our algorithm using classical or approximate SQL, over the data derived using classical or approximate SQL. Section 6 concludes our work.

2 Related Work

Let us begin with the literature on SQL-based machine learning/data mining. We have already cited papers [13,22] (related to SVM and k-NN methods) and [7,10] (related to decision tree induction). For further research in this field we refer to [3,11,14,15] (feature selection, data clustering, association rules and more details about decision trees). An interesting additional aspect of applying SQL in KDD corresponds to relational – single-table or multi-table – feature/attribute engineering [6,20].

We refer also to approximate query engines which become popular because of big data analytics challenges [9,12]. We work with the first-ever engine based entirely on the concept of data summarization [16,17], which was successfully deployed in industry[1]. This engine, whereby query execution operations are designed as transformations of granulated data summaries, can be used as if it was standard PostgreSQL. It delivers

[1] securityondemand.com/solutions/superscale-analytics-threat-detection/.

Table 1. SQL-based features. 'x' identifies a record for which a new feature value is calculated.

devicetype_cd(x)	select count(distinct devicetype) from localizedalerts where threatwatchalertid = x;
devicevendor_cd(x)	select count(distinct devicevendor) from localizedalerts where threatwatchalertid = x;
direction_cd(x)	select count(distinct direction) from localizedalerts where threatwatchalertid = x;
domain_cd(x)	select count(distinct domain) from localizedalerts where threatwatchalertid = x;
dstip_cd(x)	select count(distinct dstip) from localizedalerts where threatwatchalertid = x;
dstipcategory_cd(x)	select count(distinct dstipcategory) from localizedalerts where threatwatchalertid = x;
dstport_cd(x)	select count(distinct dstport) from localizedalerts where threatwatchalertid = x;
eventname_cd(x)	select count(distinct eventname) from localizedalerts where threatwatchalertid = x;
p6(x)	select count(distinct alerttype) from localizedalerts where threatwatchalertid = x;
p9(x)	select count(*) from localizedalerts where alerttype like 'Suspicious Outbound Anomaly%' and threatwatchalertid = x;
protocol_cd(x)	select count(distinct protocol) from localizedalerts where threatwatchalertid = x;
reportingdevice_cd(x)	select count(distinct reportingdevice) from localizedalerts where threatwatchalertid = x;
severity_cd(x)	select count(distinct severity) from localizedalerts where threatwatchalertid = x;
srcip_cd(x)	select count(distinct srcip) from localizedalerts where threatwatchalertid = x;
srcipcategory_cd(x)	select count(distinct srcipcategory) from localizedalerts where threatwatchalertid = x;
srcport_cd(x)	select count(distinct srcport) from localizedalerts where threatwatchalertid = x;
username_cd(x)	select count(distinct username) from localizedalerts where threatwatchalertid = x;

accurate results even for highly selective queries involving combinations of numeric and alphanumeric columns, such as those in Table 1. We will apply it for both SQL-based decision tree induction and the above-mentioned attribute engineering.

As we run our experiments on the data disclosed in an online machine learning competition, let us emphasize the importance of such events for development of both academic and commercial research. The most widely recognized platform in this area is Kaggle[2], although there are also others, such as KnowledgePit[3]. The reader can find more details about machine learning competitions held on KnowledgePit in [4,5].

3 The Data from the IEEE BigData 2019 Competition

We conduct experiments on the dataset made available at one of machine learning competitions held at IEEE BigData 2019. This competition was organized jointly by Security On-Demand (SOD)[4] and QED Software[5], at aforementioned KnowledgePit[6].

The data was provided by SOD in three tables. The first one contains nearly 60,000 records corresponding to so-called threatwatch alerts investigated by the security team at SOD in Q4 of 2018 and Q1 of 2019. Alerts are described by 61 columns and represent information that is available to security analysts during their decision-making processes. For each record, it is indicated whether the given alert was considered as *serious* by an analyst and therefore, whether the given SOD's client was notified about it.

The second table includes so-called localized alerts registered by SOD. For each record in the first table, there is a series of associated localized alerts. This table contains about 8,700,000 records described by a mixture of 20 numeric and symbolic features. It provides more detailed information about the network traffic and devices related

[2] www.kaggle.com.
[3] www.knowledgepit.ml.
[4] www.securityondemand.com.
[5] www.qed.pl.
[6] www.knowledgepit.ml/suspicious-network-event-recognition/.

to threatwatch alerts evaluated by security analysts. In particular, the severity of each localized alert is automatically assessed by expert-made heuristics designed by SOD.

The third table is an extract from raw network event logs that are continually captured by SOD using so-called collectors. This table is considerably larger than the previous ones. Its fragment disclosed to competition participants consisted of nearly 9,000,000,000 anonymized records described by 26 features. More information about this data source can be found in [16]. For more information about the discussed machine learning competition and its results we refer to [5].

In this paper, we concentrate on the two first tables. During the competition, participants did their best to utilize localized alerts to extract new attributes describing threatwatch alerts. The task was to learn – basing on the historical data labeled by SOD's analysts – how to distinguish between threatwatch alerts requiring and not requiring client notifications. Thus, aggregations derived for threatwatch alerts from their associated collections of localized alerts could be helpful.

For our experiments, we selected 8 numeric features from the first table and 17 new features generated from the second one. Our selection was based on SOD's expertise and on some of successful competition solutions. Given one-to-many relation between tables, all new features were derived using *SELECT COUNT* or *COUNT DISTINCT* queries, so they can be treated as numeric too. As a result, we obtain a dataset with nearly 60,000 records, 25 numeric columns, and the binary decision attribute (which can be referred also as the target variable) corresponding to client notifications.

The 8 original features are: *parentcategory*, *overallseverity*, *correlatedcount*, *isiptrusted*, *untrustscore*, *trustscore*, *flowscore*, *enforcementscore*. The 17 derived features are listed in Table 1 together with SQL statements executed to compute them. The meanings of columns in considered data tables are quite typical for the area of cybersecurity [1,19], although SOD's way of calculating their values is unique.

4 Naïve SQL-Based Decision Tree Induction

The aim of the algorithm introduced below is to establish a framework for investigating the quality-related differences between decision models derived using classical and approximate SQL statements. The algorithm itself is extremely simplified and we refer the reader to other aforementioned publications for more sophisticated ideas how to take advantage of relational database systems in decision tree induction [7,10,15,21]. We actually wanted to keep it so simple to concentrate mainly on the classical versus approximate SQL comparison. In future, analogous comparisons can be studied for other SQL-based machine learning implementations as well.

The algorithm works with standard tabular data input, i.e., each attribute corresponds to a separate column. The decision attribute (target variable) is declared as the binary column *DECISION*. Conditional attributes (dependent variables) are assumed to be numeric, although there is also an interesting interpretation of our algorithm for binary columns. We denote attributes-columns as *a1,...,an*, where n is the number of conditional attributes in the training set. The algorithm can be triggered with parameter $K > 0$ which stands for the maximum depth of decision tree induced from the data.

Let us recall that the considered approximate engine can be queried as if it was a typical instance of PostgreSQL, where the only difference is that results of analytical *SELECT* statements are not guaranteed to be fully precise (and on the other hand, one obtains those results incomparably faster than in the case of any standard engine because of the ability to work entirely on granulated data summaries) [16,17]. Therefore, we scripted our algorithm in standard PL/pgSQL. Moreover, it is straightforward to run it in the same way on the considered approximate engine and on classical PostgreSQL.

The algorithm is constructing a tree in a typical greedy way, whereby the heuristic binary split evaluation function is triggered recursively for every current leaf unless it satisfies one of three stop conditions illustrated in Fig. 1. At the beginning the following statement is executed:

```
SELECT DECISION, COUNT(*), AVG(a1)...AVG(an) FROM DATA GROUP BY DECISION;
```

Then, *ai* with the highest difference between its average values on records dropping into decision classes 0 and 1 is selected. Precisely, using notation in Fig. 1, we choose *ai* with the maximum ratio $|AVG0(ai) - AVG1(ai)| / (|AVG0(ai)| + |AVG1(ai)|)$, where denominator is used to compare more fairly between attributes with varying scales. New nodes are created with the cut $(AVG0(ai) + AVG1(ai))/2$, i.e., records satisfying conditions $ai < (AVG0(ai) + AVG1(ai))/2$ and $ai \geq (AVG0(ai) + AVG1(ai))/2$ are assigned to left and right nodes, respectively. The procedure is repeated with each of these nodes independently, whereby the only difference is that the previously chosen *ai*

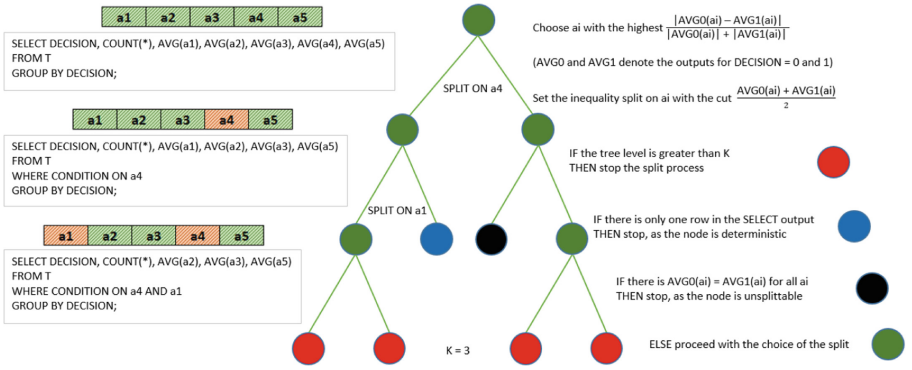

Fig. 1. High-level illustration of our naïve decision tree induction algorithm, with the maximum tree depth fixed as $K = 3$. The node statuses: Green – ready for further splits; Red – the maximum depth reached; Blue – deterministic leaf pointing at a single decision, no further splits needed; Black – available attributes do not provide sufficient discrimination between decisions, no further splits make sense. Left-side green/red statuses reflect whether particular numeric attributes have been already used in the given tree path (our naïve implementation does not allow to reuse an attribute in a path). Quantities *AVG0(ai)* and *AVG1(ai)* refer to *SELECT ... GROUP BY ...* results – they denote the average value of the *i*-th attribute in the given node, for decisions 0 and 1, respectively. The *COUNT(*)* component is used to derive decision class distributions. (Color figure online)

does not occur on the *SELECT* list any longer while the *WHERE* part is extended by the above-specified inequality conditions over column *ai*.

Surely, removing once chosen *ai* from *SELECT* lists in subsequent phases of splitting nodes is a simplification. Another simplification corresponds to one of the stop criteria – the *black* one in Fig. 1. The fact that quantities *AVG0(ai)* and *AVG1(ai)* are the same does not always mean that it is impossible to set up a useful condition on *ai*, although indeed attributes yielding larger differences between *AVG0* and *AVG1* can be regarded as more informative. Nevertheless, the whole algorithm is quite efficient, as it executes only a single SQL query per node. The *blue* stop criterion in Fig. 1 is particularly elegant, as a single-tuple output of the considered *SELECT* statement means that the corresponding node drops fully into one of decision classes.

Let us also note that the attribute choice criterion driven by the above queries neglects probabilities of decision classes in particular tree nodes. Indeed, quantities *AVG0(ai)* and *AVG1(ai)* are simply compared to each other, no matter how many records were taken into account while deriving them. It may happen that *AVG0(ai)* is the average value of attribute *ai* calculated on, e.g., 100 records with *DECISION* = 0 while there is only one record satisfying condition *DECISION* = 1. Then, the "greater/lower than value" split on *ai* is fixed as a completely non-weighted mean of *AVG0(ai)* and *AVG1(ai)*. This kind of *Bayesian* approach to decision tree induction was first proposed in [2] (whereby yet another SQL-based data mining methodology was employed) and further formalized with respect to arbitrary feature-based data partitions in [18] (whereby a decision tree induces a special case of data partition).

5 Experimental Results

As mentioned in Sect. 1, we report experiments conducted in four (two times two) following modes: running our naïve decision tree induction algorithm using two variants of SQL, i.e., classical PostgreSQL versus approximate engine [16,17], and over two versions of the considered dataset, whereby features displayed in Table 1 were computed using – again – classical or approximate *SELECT* statements.

More precisely, in its both versions, the dataset discussed in the end of Sect. 3 has nearly 60,000 records (corresponding to the same set of threatwatch alerts investigated by SOD's analysts), 25 numeric attributes and the same binary decision. The only difference between these two versions is the way of creating 17 new features from the associated data table that stores localized alerts. That table was actually loaded both into PostgreSQL and the considered approximate query engine, which internally replaced its 8,700,000 records with far lower number of multidimensional data summaries.

Experimental results are summarized in Table 2. The difference between two above data versions is indicated by the column *new columns*. Both those datasets were loaded into both PostgreSQL and our approximate engine, in order to run two considered variants of decision tree induction – indicated by column *tree induction*. For example, combination *approximate-exact* means that a tree was learnt using classical SQL but on the dataset with 17 features calculated using approximate SQL.

Outcomes for three different maximum tree depth levels K are presented. The number of leaves grows when longer root-to-leaf paths are allowed, although it can be noticed that the algorithm introduced in Sect. 4 tends to produce more compact trees when working with approximate SQL (for both *approximate* and *exact* versions of new features). The last two columns of Table 2 report additionally average intensities of occurrence of original and derived attributes in tree paths.

Table 2. Characteristics of decision trees induced using different settings of our procedure.

New columns	Tree induction	K level	R score	# of nodes	Derived	Original
Exact	Exact	5	0.251	45	0.195	0.102
Exact	Approximate	5	0.324	63	0.063	0.492
Approximate	Exact	5	0.181	59	0.162	0.266
Approximate	Approximate	5	0.288	63	0.080	0.469
Exact	Exact	10	0.408	473	0.259	0.153
Exact	Approximate	10	0.394	875	0.190	0.586
Approximate	Exact	10	0.369	771	0.276	0.324
Approximate	Approximate	10	0.332	805	0.195	0.553
Exact	Exact	15	0.559	2573	0.271	0.165
Exact	Approximate	15	0.454	3339	0.210	0.595
Approximate	Exact	15	0.421	2793	0.295	0.334
Approximate	Approximate	15	0.349	2599	0.210	0.561

It is also important to evaluate the quality of induced trees. Herein, we follow the aforementioned approach which was developed in [2,18] to assess data partitions (induced by subsets of attributes or collections of root-to-leaf tree paths) with respect to the level of information that they provide about decisions. The considered methodology is based on the following *relative information gain* measure *R(tree)* =

$$\sum_{leaves} \max_j \frac{\#\ of\ records\ in\ leaf\ with\ DECISION = j}{\#\ of\ records\ in\ dataset\ with\ DECISION = j} - 1 \tag{1}$$

Measure R has values ranging from 0 to 1, whereby equality *R(tree)* = 1 holds, if and only if all tree leaves are deterministic (i.e. they support single decision classes). Moreover, R is monotonic – splitting any leaf onto two new leaves cannot decrease its value – and it is generally perceived as a good indicator in the case of analyzing highly imbalanced datasets, such as the one disclosed in the considered machine learning competition at the IEEE BigData 2019 conference.

6 Conclusions

We introduced a naïve SQL-based decision tree induction algorithm, with the aim to compare classical PostgreSQL and the approximate query engine working on granulated data summaries with respect to the quality of trees derived from the data.

Our experiments focused on real-world dataset made publicly available in frame of the Suspicious Network Event Recognition competition held at the IEEE BigData 2019 conference. In particular, we studied two out of three data tables disclosed in the competition and additionally, we utilized the considered approximate engine to investigate opportunities of approximate-SQL-driven feature engineering.

In future, besides improvements of the above-mentioned algorithm, we intend to extend our decision-tree-related research onto the third data source associated with the discussed machine learning competition. Given its huge volume, this data source needs approximate analytical methods to the highest extent. Let us also point out that the experimental framework developed in this paper can serve as a useful environment for testing enhancements of our approximate engine and other analogous solutions.

References

1. Garcia-Teodoro, P., Díaz-Verdejo, J.E., Maciá-Fernández, G., Vázquez, E.: Anomaly-based network intrusion detection: techniques, system challenges. Comput. Secur. **28**(1–2), 18–28 (2009)
2. Grant, A., et al.: Examination of routine practice patterns in the hospital information data warehouse: use of OLAP and rough set analysis with clinician feedback. In: AMIA 2001, p. 916 (2001)
3. Hu, X., Lin, T.Y., Han, J.: A new rough sets model based on database systems. Fundam. Inform. **59**(2–3), 135–152 (2004)
4. Janusz, A., Grad, Ł., Grzegorowski, M.: Clash royale challenge: how to select training decks for win-rate prediction. In: FedCSIS 2019, pp. 3–6 (2019)
5. Janusz, A., Kałuża, D., Chądzyńska-Krasowska, A., Konarski, B., Holland, J., Ślęzak, D.: IEEE BigData 2019 cup: suspicious network event recognition. In: IEEE BigData (2019)
6. Kobdani, H., Schütze, H., Burkovski, A., Kessler, W., Heidemann, G.: Relational feature engineering of natural language processing. In: CIKM 2010, pp. 1705–1708 (2010)
7. Kowalski, M., Stawicki, S.: SQL-based heuristics for selected KDD tasks over large data sets. In: FedCSIS 2012, pp. 303–310 (2012)
8. Kurgan, L.A., Musílek, P.: A survey of knowledge discovery and data mining process models. Knowl. Eng. Rev. **21**(1), 1–24 (2006)
9. Mozafari, B., Niu, N.: A handbook for building an approximate query engine. IEEE Data Eng. Bull. **38**(3), 3–29 (2015)
10. Nguyen, H.S., Nguyen, S.H.: Fast split selection method and its application in decision tree construction from large databases. Int. J. Hybrid Intell. Syst. **2**(2), 149–160 (2005)
11. Ordonez, C., Cereghini, P.: SQLEM: fast clustering in SQL using the EM algorithm. In: SIGMOD 2000, pp. 559–570 (2000)
12. Orr, L., Suciu, D., Balazińska, M.: Probabilistic database summarization for interactive data exploration. PVLDB **10**(10), 1154–1165 (2017)
13. Rüping, S.: Support vector machines in relational databases. In: Lee, S.W., Verri, A. (eds.) SVM 2002. LNCS, vol. 2388, pp. 310–320. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45665-1_24
14. Sarawagi, S., Thomas, S., Agrawal, R.: Integrating association rule mining with relational database systems: alternatives and implications. Data Min. Knowl. Discov. **4**(2–3), 89–125 (2000)
15. Sattler, K., Dunemann, O.: SQL database primitives for decision tree classifiers. In: CIKM 2001, pp. 379–386 (2001)

16. Ślęzak, D., Chądzyńska-Krasowska, A., Holland, J., Synak, P., Glick, R., Perkowski, M.: Scalable cyber-security analytics with a new summary-based approximate query engine. In: IEEE BigData 2017, pp. 1840–1849 (2017)
17. Ślęzak, D., Glick, R., Betliński, P., Synak, P.: A new approximate query engine based on intelligent capture and fast transformations of granulated data summaries. J. Intell. Inf. Syst. **50**(2), 385–414 (2018)
18. Ślęzak, D., Ziarko, W.: The investigation of the bayesian rough set model. Int. J. Approx. Reason. **40**(1–2), 81–91 (2005)
19. Sommer, R., Paxson, V.: Outside the closed world: on using machine learning for network intrusion detection. In: IEEE S&P 2010, pp. 305–316 (2010)
20. Wróblewski, J.: Analyzing relational databases using rough set based methods. In: IPMU 2000, vol. 1, pp. 256–262 (2000)
21. Wróblewski, J., Stawicki, S.: SQL-based KDD with infobright's RDBMS: attributes, reducts, trees. In: Kryszkiewicz, M., Cornelis, C., Ciucci, D., Medina-Moreno, J., Motoda, H., Raś, Z.W. (eds.) Rough Sets and Intelligent Systems Paradigms. LNCS, vol. 8537, pp. 28–41. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08729-0_3
22. Yao, B., Li, F., Kumar, P.: K nearest neighbor queries and kNN-joins in large relational databases (almost) for free. In: ICDE 2010, pp. 4–15 (2010)