Short Paper

COCOA: COrrelation COefficient-Aware Data Augmentation

Mahdi Esmailoghli
Leibniz Universität Hannover
Hannover, Germany
esmailoghli@dbs.uni-hannover.de

Jorge-Arnulfo Quiané-Ruiz
TU Berlin
Berlin, Germany
jorge.quiane@tu-berlin.de

Ziawasch Abedjan
Leibniz Universität Hannover
Hannover, Germany
abedjan@dbs.uni-hannover.de

ABSTRACT

Calculating correlation coefficients is one of the most used measures in data science. Although linear correlations are fast and easy to calculate, they lack robustness and effectiveness in the existence of non-linear associations. Rank-based coefficients such as Spearman's are more suitable. However, rank-based measures first require to sort the values and obtain the ranks, making their calculation super-linear. One of the use-cases that is affected by this is data enrichment for Machine Learning (ML) through feature extraction from large databases. Finding the most promising features from millions of candidates to increase the ML accuracy requires billions of correlation calculations. In this paper, we introduce an index structure that ensures rank-based correlation calculation in a linear time. Our solution accelerates the correlation calculation up to 500 times in the data enrichment setting.

1 INTRODUCTION

The correlation coefficient is one of the extensively used statistical measures in data science. Data scientists use the correlation coefficient to find dependencies in the data and identify possible causal relationships. In machine learning (ML) tasks, correlations can be used to evaluate the relevance of features. Correlating features expose redundancy. Thus, one can remove redundant features to avoid the curse of dimensionality. Also, a correlation between a feature and a target value can serve as a heuristic for the importance of a feature. This is exactly what filter-based feature selections are aiming for [4]: They leverage correlation to find informative and drop redundant features to achieve high accuracy in the following ML task.

Various correlation coefficients can be used to identify correlations between features in datasets. The most prominent one is the Pearson Correlation Coefficient (Pcc). Although calculating the Pcc is fairly simple and linear in the size of the compared datasets, there are several situations where a non-linear coefficient, such as the Spearman's correlation coefficient (Scc) is preferred.

Robustness. Linear correlations such as Pcc are very sensitive to outliers. As an example, Table 1 compares robustness of Pcc and Scc. Although there is no clear correlation between the two columns, the calculated Pcc identifies them as correlated, because of one outlier point (shown in red). A correlation of 1.0 is clearly misleading. Scc shows a more fitting correlation of only 0.1. This robustness makes Scc to be a better fit for noisy and dirty data such as webtables that are likely to contain outliers [7, 14].

Effectiveness. Linear correlations are not effective in capturing more complex dependencies. They are only able to find linear associations between two features. This can be useful in the case of feature selection for linear models such as linear regression. However, for complex ML models such as Random Forest or Neural Network that can train non-linear patterns, linear correlation-based feature selections can miss features with non-linear dependencies harming the achievable accuracy [23].

Table 1: Correlation in the existence of outlier.

Area (Million sq. miles)	*Calling Code*
0.29	56
0.3	90
3.8	1
0.5	51
600	9800
Pearson = 1.0	**Spearman's = 0.1**

The disadvantage of non-linear approaches, such as Scc and Kendall tau [13] is that they are rank-based and as such require a sortation of values, which poses a higher time complexity, i.e. $O(m \cdot \log(m))$ where m is the size of the variable, for calculation than their linear counterparts. The time complexity increases when we want to calculate non-linear correlation for categorical columns. The Rank-biserial correlation coefficient Rbc can be applied on one-hot encoded columns, so Rbc has a time complexity of $O(m^2)$ [7, 14]. This computation overhead negatively impacts the analysis pipeline when a large number of features have to be analyzed. For instance, to detect the redundant features, the correlation computation between each possible pair is required, which inherently leads to $O(N^2)$ correlation computations for N different features [23]. For datasets with several hundreds of potential features, the runtime overhead impedes live analysis and fast model building. For example in data enrichment [2, 17, 22, 24, 26], one aims to detect features that correlate to the given target feature from millions of extracted candidates. At this scale, runtime overhead for rank-based correlations becomes evidently a hurdle.

In this paper, we introduce a light-weight indexing structure to compute the non-linear correlation coefficient in a linear time for large-scale data enrichment tasks. It avoids the $O(m \cdot \log(m))$ sorting operation for numerical columns, benefiting the Scc calculation, and avoiding the $O(m^2)$ complexity of dealing with the one-hot encoding of categorical columns for Rbc. Our light-weight index also enables Cocoa (our system) to scale to the massive number of external tables. Furthermore, the nature of the correlation calculation also enables Cocoa to perform light-weight joins instead of full materialization of joins with candidate columns. In summary, we make three major contributions:

(1) We introduce a new index structure that enables us to compute the non-linear correlation coefficient in linear time and generalizes also for enrichment through partial joins where ranks are missing and have to be adapted.
(2) We propose algorithms that leverage our index structure to detect correlations between numerical and categorical columns to a target column in linear time.
(3) We introduce a correlation-based data enrichment solution that increases the accuracy of the ML model for a user-defined task compared to other enrichment solutions.

2 PROBLEM STATEMENT

Given an input dataset D with m rows, two explicitly selected columns q and t from D, so-called query and target columns respectively, and a corpus of external tables $T = \{T_1, ..., T_n\}$, the goal is to enrich D with the top-k_c columns from any table $T_i \in T$ that correlate with t. As we focus on regression tasks, t is a numerical column while features can be either numerical or categorical. The query column q is used to find the related tables, i.e., tables that are joinable with D on q. Ideally, q is an identity exposing column, such as name or ID. Without loss of generality, we thus assume q is selected explicitly by the user.

Depending on whether an extracted feature is numerical or categorical, one has to use Scc or Rbc, respectively. Equation 1 shows the formula of Scc where x_i and y_i are the i^{th} values in the first and second column respectively. Both columns are of size m. $R(x_i)$ represents the rank of value x_i. For instance, $R(8) = 2$ in list $[8, 11, 4, 9]$ because the value 8 is second in the sorted list $[4, 8, 9, 11]$. $\overline{R(x)}$ and $\overline{R(y)}$ represent the average of ranks in the first and the second column respectively.

$$\text{Scc} = \frac{\sum_{i=1}^{m}(R(x_i) - \overline{R(x)})(R(y_i) - \overline{R(y)})}{\sqrt{\sum_{i=1}^{m}(R(x_i) - \overline{R(x)})^2 (R(y_i) - \overline{R(y)})^2}} \quad (1)$$

The Scc calculation for numerical columns is in $O(m \cdot \log m)$ because one has to obtain the ranks through sortation.

To compute the correlation between categorical columns and t, the Rbc calculates the Scc between each one-hot feature of the categorical column and t [3, 9, 18]:

$$\text{Rbc} = \frac{M_1 - M_0}{std} \sqrt{\frac{n_1 n_0}{m^2}} \quad (2)$$

M_1 and M_0 represent the average target rank for 1s and 0s in a one-hot feature, n_0, n_1 are the number of ones and zeros in the one-hot feature, and std and m define the standard deviation of target ranks and the number of values in columns, respectively. In our case, m is the number of rows in the input dataset. The overall correlation is then the maximum Rbc between each one-hot feature and t Based on Equation 2, we can calculate the Rbc in $O(m)$ time complexity per one-hot feature. As a column can have up to m unique values, thus the overall complexity is $O(m^2)$.

If the join operation is implemented as a hash-based join, the join task has a time complexity of $O(m)$ per table. The follow-up correlation calculation per column has either a complexity of $O(m \cdot \log m)$ or $O(m^2)$ depending on whether the feature is numerical or categorical. In retrieval tasks with large databases, we can observe the runtime is dominated by the correlation calculation.

Problem We are thus looking for an index structure that allows us to calculate both Scc and Rbc in linear time. To be able to index data repositories in the scale of web tables, our index structure should be simple and light-weight.

3 COCOA SYSTEM

Figure 1 depicts the abstract view of the components designed in our system Cocoa. The main components are *Table Finder*, *Join Mapper*, and *Data Augmenter*. A user provides Cocoa with a dataset D and specifies its query column q and ML target column t. At last, it returns the top-k_c most correlating columns as the output of the system. Now, we describe each of these components.

Table Finder. This component uses q and the inverted index of the DataXformer system [1] to obtain top-k_t joinable tables.

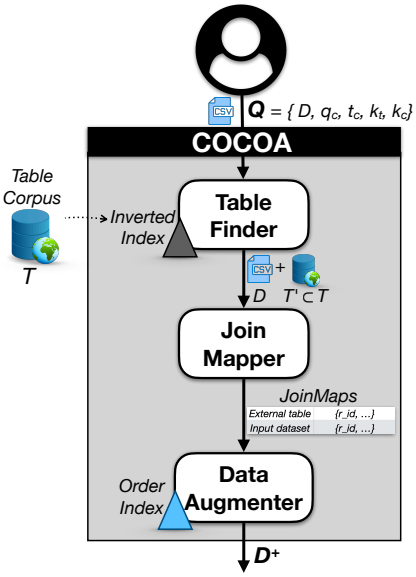


Figure 1: The overall architecture of Cocoa

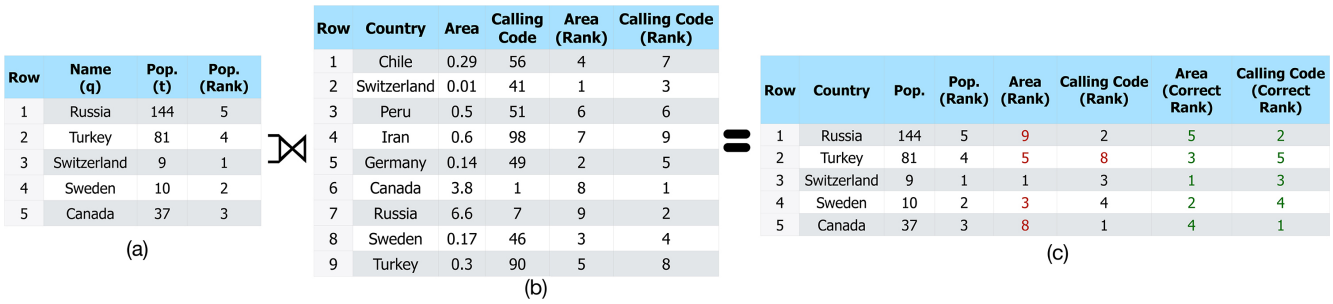
A table is joinable if it contains at least one column that overlaps values with q. For each value v_i in external tables, the inverted index lists the coordinates of its containing tables: $v_i \mapsto \{(T_{i1}, C_{i1}, P_{i1}), (T_{i2}, C_{i2}, P_{i2}), ...\}$, where T_{ij}, C_{ij}, and P_{ij} are identifiers of tables, columns, and rows in the table corpus, respectively. We store the inverted indices inside a column store and generate a SQL query to find joinable external tables in parallel and using database-level optimizations. This approach allows to push the process of finding the joinable tables down to the database itself. The SQL query calculates the overlap between each external column and q and then selects the top-k_t tables by sorting them based on the overlap score.

Running example Figure 2(a) depicts a user-provided dataset D for the task of predicting the population of a country. The column with the country names will serve as the query column q and the population as the ML target column t. Figure 2(b) depicts an external table (T_1) that contains three columns: *Country*, *Area*, and *Calling Code*. In our example, the index entry for the values "Germany" in *Country* would be: $Germany \mapsto \{(T_1, Country, 5)\}$.

Join Mapper. The *Join Mapper* receives a set of joinable tables T' as input and *virtually* joins them with the input dataset D. To enrich the input dataset with external tables, we have to apply a LEFT JOIN because we need to keep information in D and add only overlapping information from the external tables [6]. Instead of a complete materialized LEFT JOIN, our *Join Mapper* generates a lightweight JoinMap, which is inspired by P.Valduriez's "Join Indices" [19]. Using our *Order index*, we can use the light-weight join to also calculate the correlations before materializing the join. Thus we limit the join materialization for relevant columns with high correlation and further speed up the process.

Figure 3 depicts the JoinMap for the example in Figure 2. *Country* column in T_1 is overlapping with q in D. The corresponding map shows which row in *Country* has the same respective value in q. As it is shown, the value "Switzerland" in row 2 of *Country* appears in row 3 of q. In the case of duplicates in q, respective cells in the JoinMap will contain more than one value.

Data Augmenter. This component receives the JoinMaps as input and uses a novel structure, called *Order index*, to efficiently compute the Scc and Rbc. It evaluates the external columns based on their correlation with t and enriches the input dataset D with the top-k_c correlating columns to generate the final dataset D^+.



Row	Name (q)	Pop. (t)	Pop. (Rank)
1	Russia	144	5
2	Turkey	81	4
3	Switzerland	9	1
4	Sweden	10	2
5	Canada	37	3

(a)

Row	Country	Area	Calling Code	Area (Rank)	Calling Code (Rank)
1	Chile	0.29	56	4	7
2	Switzerland	0.01	41	1	3
3	Peru	0.5	51	6	6
4	Iran	0.6	98	7	9
5	Germany	0.14	49	2	5
6	Canada	3.8	1	8	1
7	Russia	6.6	7	9	2
8	Sweden	0.17	46	3	4
9	Turkey	0.3	90	5	8

(b)

Row	Country	Pop.	Pop. (Rank)	Area (Rank)	Calling Code (Rank)	Area (Correct Rank)	Calling Code (Correct Rank)
1	Russia	144	5	9	2	5	2
2	Turkey	81	4	5	8	3	5
3	Switzerland	9	1	1	3	1	3
4	Sweden	10	2	3	4	2	4
5	Canada	37	3	8	1	4	1

(c)

Figure 2: Running example: (a) Input dataset D; (b) External table T_1; (c) Joined result of D and T_1

row in C_1	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
row in q	∅	3	∅	∅	∅	5	1	4	2

Figure 3: An example of JoinMap

4 CORRELATION CALCULATION

The main goal of the *Data Augmenter* is to calculate the Scc and Rbc between each provided external column and the target column t in $O(m)$. Thus, we want to refrain from sorting the values of every potential numerical column and the generation of one-hot encodings for every categorical column during the extraction time. One might think of a simple solution that calculates the ranks and one-hot encodings offline and maintains them as part of the inverted index. However, maintaining the ranks will lead to calculation errors of the Scc when related tables are only partially joinable. Columns *Area (Rank)* and *Calling Code (Rank)* in Figure 2(c) show this error in obtaining the ranks after the partial join. Non-consecutive ranks lead to incorrect Scc. For instance, the correct Scc between *Calling Code* and t is -0.1, but using the incomplete ranks returns the Scc of $+0.117$. Furthermore, storing the one-hot encoding of all columns would unnecessarily blow up the index size. Therefore, we propose algorithms based on a new *Order index* that keeps track of the order of each value in the numerical columns and efficiently generates one-hot encodings for categorical columns.

4.1 Order Index Structure

We now introduce the simple, yet effective, *Order index* structure. Instead of the actual ranks, it stores the relative order of column values enabling the system to compute Scc and Rbc in a linear time. Keeping track of the order of values enables us to calculate the correct ranks on-demand despite partial joins. Also, keeping relative orders enables us to walk through all non-zero values in one-hot features in the right order to calculate the Rbc for categorical columns. To maintain the order information in a concise way, the *Order index* maps each pair of a table identifier T_{id} and a column identifier C_{id} to the column values as follows:

$$T_{id}, C_{id} \mapsto \{s,\ (o_1,\ ...,\ o_r),\ (b_1,\ ...,\ b_r)\} \tag{3}$$

$$s \leftarrow \alpha, \quad where \quad R(v_\alpha) = 1 \tag{4}$$

$$b_i = \begin{cases} False, & R(v_i) = R(v_{o_i}) \\ True, & R(v_i) \neq R(v_{o_i}) \end{cases} \tag{5}$$

Each entry in the *Order index* consists of the starting point s, which represents the row id of the minimum value in the column (α in Equation 4). v_i is the value located in the i^{th} row. The rank of the minimum value is always 1. In categorical columns, the minimum is the first value in the alphabetically sorted list. The second item (order list), contains a list of values where o_i is the row id of the next greater value than v_i. In case of having repetitive values, o_i is the row id of the next equal value. If v_i is the last value in the sorted list, $o_i = -1$. The item b_i denotes whether v_{o_i} is greater than its predecessor v_i or not (Equation 5). We use the same index for both numerical and categorical columns. To distinguish the numerical and categorical columns during the correlation calculation, we use a simple heuristic: a column is considered as categorical if it contains at least one non-numeric value. We store it as an additional bit per column. The *Order index* of the column *Area* in our running example in Figure 2(b) would be:

$\{2,\ (9,\ 5,\ 4,\ 6,\ 8,\ 7,\ -1,\ 1,\ 3), (T,\ T,\ T,\ T,\ T,\ T,\ \emptyset,\ T,\ T)\}$

In column *Area*, the minimum is located in row 2 (0.01), so, $s = 2$. The second element of the index represents a list of pointers to the next greater (or equal) values. The first item in this list o_1 points to row id 9 and it means that the next greater value "0.3" after the first value "0.29" is stored in the 9^{th} row. Likewise, the 9^{th} pointer o_9 in the order list is 3, which means that the next greater value "0.5" is located in row 3. Notice that as there are no repetitive values in column *Area*, all of the binary values in the third index element, except for the maximum value pointer, are *True*.

Figure 4 shows the visualized representation of the *Order index* for the column *Area* in T_1. Each square represents one value in the external column. The blue square represents the minimum value and the red one shows the maximum. Each edge that connects one square to another depicts the relative order o_i. Small numbers in the circles are the row ids of the values in the column *Area*. Each value has an outgoing edge except the maximum value. Labels on each edge represent the index values stored for the source value. For example, the outgoing edge from the 4^{th} item in the list, is $[6, T]$, which translates to $o_4 = 6$ and $b_4 = T$.

The *Order index* is the most basic index structure that supports correlation calculation in $O(m)$. A more complex B+ tree adaptation can be used if frequent corpus updates are expected. Here, we exclude the tree benefits for a more space-efficient index.

4.2 SCC for Numerical Columns

We use an example to describe how we leverage the underlying *Order index* to calculate the Scc between numerical columns of an external table and t. Figure 4 shows the *Order index* of the *Area* column and the JoinMap of the table T_1 from our running example. Assume that `p` is a pointer and in the beginning, `p` references the minimum value in the column. In each iteration, `p` traverses through the available links until it reaches the red square which means that the ranking process is finished.

Starting from the minimum value "0.01", the algorithm checks the `JoinMap` for any mapping from the value in *Area* to a value in q. There is a mapping from "0.01" to the value in the 3^{rd} row of

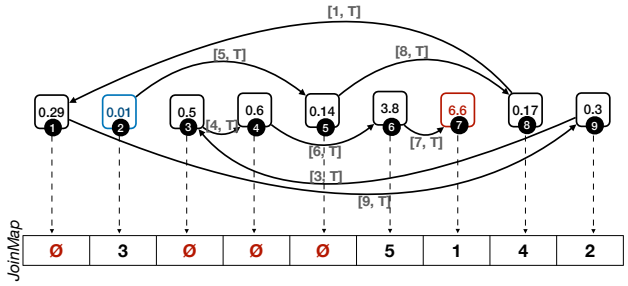


Figure 4: Visual representation of the Order index for the column *Area* from the running example in Figure 2.

q. Therefore, initial rank, which is 1, is assigned to the 3^{rd} value of *Area* (see *Area (Correct Rank)* in Figure 2(c)). Then, p will be updated to the next greater value. Here, "0.01" is linked to the 5^{th} value, which is "0.14". However, there is no map from "0.14" to any value in q. Thus, p will be updated based on the outgoing link from "0.14" to the next greater value located in the 8^{th} square with the value of "0.17". There is a mapping from the value "0.17" to the row with id 4, so the next rank, which currently is 2, will be assigned to the 4^{th} value in *Area*. This process continues until the pointer p reaches the 7^{th} and the maximum value. Since there are no outgoing links, the iteration is finished. The ranks for both *Area* and *Calling Code* columns are shown in the Figure 2(c) as *Area (Correct Rank)* and *Calling Code (Correct Rank)*, respectively.

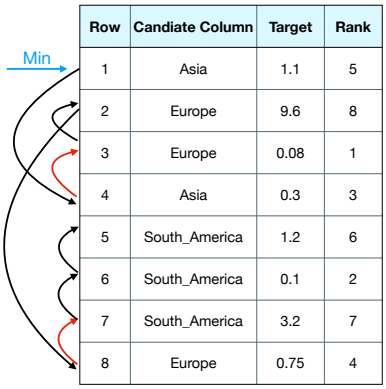
According to Equation 1, we can indeed compute the Scc in $O(m)$ by having correct ranks. Obtaining the ranks is now also possible in $O(m)$. The algorithm iterates over the values in the ordered list only once. Therefore, if the time complexity of the operations per value is constant and the path from the minimum to the maximum value is cycle-free, we conclude that the Scc calculation is done in $O(m)$. First, all operations of reading the *JoinMap*, assigning/updating current rank, and updating p to the next value are done in a constant time because the *JoinMap* and the *Order index* items are implemented using array structure and accessing the array cells is performed in a constant time using the available cell index. We also know that each o_i value, which is represented by an edge, refers to a unique row id even in the existence of repetitive values. Thus, the path from the minimum to the maximum value is cycle-free. The path length is r, i.e., the average number of rows in the external columns. External tables are often smaller than the input datasets ($r << m$), e.g., the average external table size is 10 for DWTC and 1540 for open data [26]. Thus, the complexity is typically bound by m.

4.3 RBC for Categorical Columns

To calculate the dependency between external categorical columns and t, we have to generate the one-hot features of the column and calculate the Rbc between each generated feature and the target rank column. Using our index we can avoid this and simulate the calculation in linear time. The *Order index* allows us to walk through the non-zero values in the one-hot features, one feature at a time. By iterating non-zero values of each one-hot feature consecutively we can compute the Rbc for all possible feature in one pass of reading the categorical column using the following derived Equation 6, which is obtained by replacing M_0 and n_0 in Equation 2 with $\frac{S-M_1 n_1}{n_0}$ and $m - m_1$ respectively:

$$\text{Rbc} = \frac{m \cdot s_1 - n_1 \cdot S}{std \cdot m \cdot \sqrt{n_1(m - n_1)}} \tag{6}$$

In Equation 6, S is the sum of all the ranks in the target column and s_1 the sum of the ranks where the corresponding one-hot value is one. Using Equation 6, variables m, S, and std are calculated once. To compute the Rbc between each one-hot feature and t, it is enough to compute s_1 and n_1 for each one-hot feature.



Row	Candiate Column	Target	Rank
1	Asia	1.1	5
2	Europe	9.6	8
3	Europe	0.08	1
4	Asia	0.3	3
5	South_America	1.2	6
6	South_America	0.1	2
7	South_America	3.2	7
8	Europe	0.75	4

Figure 5: Categorical candidate column example.

Cocoa uses the *Order index* to iterate over the sorted list of categorical values. It is able to compute s_1 and n_1 per one-hot feature without generating the features because the *Order index* allows to read the repetitions of the values consecutively. Once the iteration reaches a different value, b_i announces the end of the current one-hot feature. At this point, the Rbc is calculated based on Equation 6. In the end, the maximum Rbc will be reported.

Figure 5 shows an example for Rbc calculation. It depicts a categorical column "Candidate Column" and the target column "Target". We would like to compute the Rbc of all one-hot features in one pass of reading the values. The ranks of the target values are shown in the "Rank" column. The location of the minimum value is represented in blue and each link depicts the location of the next value. The red-colored links show that the next value is greater, i.e., different, than the current value. Obtaining the standard deviation of the "Rank" column and m, we only need to calculate s_1 and n_1 for each unique value.

Starting with the minimum value, as long as the followed arrow is not red, we continue reading rank values and adding them to s_1, increasing n_1 by 1 in each iteration. After reading the value in row number 1, s_1 and n_1 are equal to 5 and 1 respectively. Following the links gets us to the 4^{th} row. Reading the rank value, s_1 is increased to 5+3 = 8 and n_1 to 1+1 = 2. The next arrow is red and it means that the following value is different from the current one. Therefore, we calculate the correlation using Equation 6. Repeating this process, another correlation is calculated in row number 7. Reaching the last row, we compute the final correlation and report the maximum Rbc. Rbc calculation is done in $O(m)$ as the algorithm iterates over the ordered list only once.

5 EXPERIMENTS

We carried out a series of experiments to evaluate Cocoa with the following questions in mind: (i) What is the performance gain through *Order index* for calculating feature correlations on large corpora? (ii) How efficient is the light-weight virtual join? (iii) How scalable is Cocoa? Before we delve into the detail, we first describe the setup of our experimental evaluation.

5.1 Data and Experimental Setup

We tested our approach on top of several open databases. The Dresden webtable corpus (DWTC)[1] contains more than 145M tables and 870M unique columns. The Canada, US, and Uk Open Data corpus used in prior work [26] contains more than 745,000 columns and 14,000 tables. It is noteworthy that the Open Data

[1] https://wwwdb.inf.tu-dresden.de/misc/dwtc/

Table 2: Experimental datasets.

Dataset	*Query Column*	*Target*	*Instances*
Kaggle Open Food Facts	Product Name	Energy per 100g	356,000
Kaggle NYC Airbnb	Place Name	Price	48,000
Cities	City Name	Population	40,000
Kaggle Video Game Sales	Game	Global Sales	16,599
PageView	Name	Visit	11,000
Kaggle IMDB	Movie Title	IMDB Score	5,043
Presidential	County	Votes	3,000
Kaggle Craft Beers	Name	Alcoholic content	2,400
Kaggle Human Freedom Index	Country	Homicide Index	1,450
Kaggle World Happiness Report	Country/Region	Score	157
University	Name	World Ranking	100

corpus only contains numerical data. We ran our experiments on a server with 28 CPU cores and 250GBs of main memory. We implemented our solution in *Python 3.7* and used Vertica *v9.1.1-0* [16] as the storage. Implementations and datasets are available in our GitHub repository[2]. All compared approaches use the same table retrieval module. They only differ with regard to the join, filtering, and correlation calculation steps.

Table 2 shows the datasets that we chose from different domains as query datasets. For the Open Data corpus, we use the benchmark query columns provided in the *Josie* paper [26].

Baselines. For these experiments, we compare Cocoa with three baselines and two combined versions of Cocoa:

(1) Sort-based enrichment (Sbe). Sbe calculates the Scc and Rbc without using the provided order index. It has to sort online and create one-hot encoding of categorical columns. The comparison with this baseline allows us to evaluate the efficiency of the *Order index* in the enrichment pipeline.

(2) TR. It is a rule-based method that skips joins with non-informative tables before any further calculation [15].

(3) TR+Cocoa. TR is a complementary approach to Cocoa. TR drops the non-informative tables before joining. Then, Cocoa extracts the correlating columns from the tables.

(4) RF. Here, we apply the random forest feature selection (RF) on top of the overlap-based enrichment method [26]. In this method, we join k_t related tables and RF picks the most informative features with respect to the ML task.

(5) Cocoa+RF. Here, Cocoa enriches the input data with the top 100 correlated columns and delivers to RF.

5.2 Results

We start with the DWTC experiments. Figure 6(a) shows the average runtime of Cocoa, Sbe, and TR applied on all query datasets. The average runtime is shown for varied k_t values. In this experiment, we drop the break-down based on k_c because it has only a negligible improvement on the runtime of Cocoa and no impact on the other approaches. The depicted runtime includes the time for joining tables and calculating the Scc/Rbc. In the case of TR, runtime represents joining and rule validation time. Note that the runtime is depicted in logarithmic scale.

Cocoa outperforms Sbe on all datasets and for all different k_t values and ultimately, we see better performance on average. Cocoa can be up to 520X faster than Sbe using the introduced *Order index*. Cocoa, on average, is slower than *TR*. *TR* applies a rule-based table filter before joining the tables. It computes the cardinality of the tables and decides whether the join is safe to skip or not. Rules in TR - unlike Cocoa- are applied per external table and not per columns. Therefore, the rule verification in TR is computationally less expensive. The coarse-granular filtering comes at the cost of effectiveness [6]. According to Figure 6(a), the runtime of all approaches increases with k_t because more external tables have to be processed for the Scc calculation.

[2]https://github.com/BigDaMa/COCOA

Figure 6(b) shows the runtime experiment on Open Data. Here, k_t has a lower impact compared to the DWTC corpus, because of the small number of tables. So, we consider all tables that have overlap with q as candidate join tables. As expected, similar to the DWTC, Cocoa is faster than Sbe due to the fast correlation calculation through our introduced index structure. However, TR is surprisingly slower than the other two systems. The reason is that in comparison to DWTC, the tables in the Open Data benchmark, are not sparse and yield higher overlap between queries and the tables. Therefore, the TR rule is passed and most of the tables are joint. In this case not only does TR not provide any additional runtime benefits but also it introduces cardinality calculation overhead with almost zero pruned external table.

Figure 6(c) compares Cocoa to three hybrid system on DTWC with $k_t = 1000$. We build these hybrid systems to evaluate the impact of the feature pre-filtering on the runtime of the systems and determine the fastest system combination that considers feature-target association for the ML task. The combination *TR*+Cocoa results in better performance than every other strategy. TR drops the non-informative tables and reduces the search space for Cocoa to find the correlating features in a faster way. This experiment shows that although *TR* does not perform well with regard to effectiveness [6], it can be a complementary heuristic for Cocoa to reduce the search space and enrich the input dataset much faster than any other solutions at hand.

RF is the slowest approach on all datasets except the *University* dataset. This is due to the tremendous number of features that are delivered to *RF*. Note that experiments that are not finished within 8 hours are underestimated with 8 hours in the runtime. RF fails to terminate for 4 datasets: *Cities*, *IMDB*, *Food*, and *NYC Airbnb*. Pruning the search space using Cocoa improves runtime for RF. Cocoa +RF only fails to finish on the *Food* dataset.

We notice that *TR*+Cocoa performs even better than TR on larger datasets, such as *Food*. The reason is that TR materializes all joins but Cocoa uses the light-weight virtual joins that are much faster because only the top-k_c columns are materialized.

Scalability. To better evaluate the scalability of Sbe- and Cocoa-based solutions with regard to the dataset size, we measure the average runtime for a single correlation calculation in both approaches. Figure 7 shows this experiment on the *City* dataset scaling the number of rows. The average calculation time increases much faster for Sbe than for Cocoa. On a dataset with $1M$ rows, the runtime difference between the two approaches for one correlation calculation is about 118 seconds. This difference is crucial in the scale of thousands of external tables. In our experiments, for the *City* dataset the number of correlation calculations easily exceeds 52, 000 calculations to enrich the dataset, thus this scalability issue will lead to serious runtime problems. In this case, Sbe would need 71 days to complete the task while Cocoa would require only 9.5 minutes. Notably, the variances in the graph are negligible and at most 0.0001 milliseconds.

Index Size. We only store the row ids and bits instead of actual values, therefore, indexing the DWTC corpus requires less than 12GB disk storage compared to almost 300GB database size.

6 RELATED WORK

Data enrichment It is referred to the line of research that expands an input dataset using external sources such as webtables [11, 22, 24], Open data [17, 26], or knowledge bases [5].

Most pieces of work from the database community focus on finding joinable tables. For this purpose, some heuristics, such as

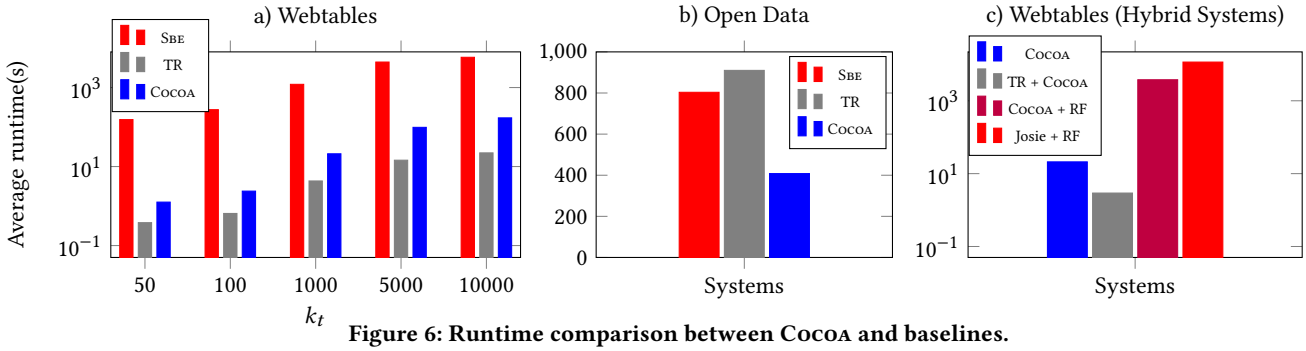


Figure 6: Runtime comparison between COCOA and baselines.

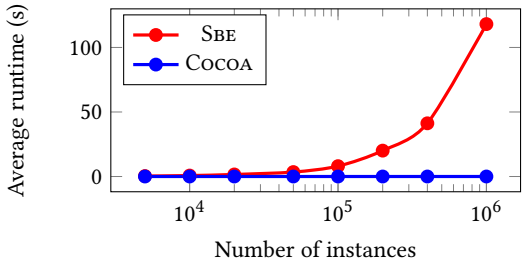


Figure 7: Scaling SCC and RBC on the *City* dataset.

overlap similarity [21, 25, 26] or a combination of the coverage and value consistency [8, 11] is used to find candidate tables for enrichment. These works do not consider the downstream ML task for the retrieval process and only rely on the joinability of the extracted tables. Furthermore, they assume that the external tables can fit in the main memory. Kumar et al. address the problem of data enrichment by defining a set of decision rules to prevent the joins that will not contribute to higher accuracy in the downstream ML models [15]. Their approach skips the joins to achieve the minimum information loss. Therefore, the dropping rate is quite low and in the scale of a large corpus of tables, the final enriched dataset will still contain a large number of columns, that have to be handled in the time-consuming feature selection process [23]. In COCOA, we propose an ML-aware data enrichment solution. It leverages the SCC/RBC to find the most promising columns for the downstream ML task. ARDA [6] is an enrichment system that leverages sampling techniques to find the informative joins and then uses an ensemble feature selection method to select the best features. Its feature selection algorithm RF focuses on accuracy and is not efficient when it comes to a large number of external tables.

Feature Selection Feature selection algorithms [6, 10, 12, 20] are designed to find the best feature set for a specific ML task after the enrichment process. In this paper, we aim to blend the feature selection with the extraction phase that can speed up the traditional *Extract-Then-Select* pipeline. We also discussed a combination of COCOA with the most promising feature selection method [6] in our experiments.

7 CONCLUSION

We presented COCOA, a new data enrichment system. It enables the efficient calculation of non-linear correlation coefficients to select the most correlating features for a user-defined ML task. In particular, we introduced an index structure that allows to calculate non-linear correlation coefficients in linear time complexity. COCOA is designed to be general and hence it can be complemented with other table-based filters or used for any analytic task that depends on value rankings and rank-based scores.

Acknowledgements. This project has been supported by the German Research Foundation (DFG) under grant agreement 387872445 and the German Ministry for Education and Research as BIFOLD — "Berlin Institute for the Foundations of Learning and Data" (01IS18025A and 01IS18037A).

REFERENCES

[1] Ziawasch Abedjan, John Morcos, Michael N Gubanov, Ihab F Ilyas, Michael Stonebraker, Paolo Papotti, and Mourad Ouzzani. 2015. Dataxformer: Leveraging the Web for Semantic Transformations.. In *CIDR*.
[2] Bortik Bandyopadhyay, Xiang Deng, Goonmeet Bajaj, Huan Sun, and Srinivasan Parthasarathy. 2019. Automatic Table completion using Knowledge Base. *arXiv preprint arXiv:1909.09565* (2019).
[3] Douglas G Bonett. 2019. Point-biserial correlation: Interval estimation, hypothesis testing, meta-analysis, and sample size determination. *Brit. J. Math. Statist. Psych.* (2019).
[4] Girish Chandrashekar and Ferat Sahin. 2014. A survey on feature selection methods. *Computers & Electrical Engineering* 40, 1 (2014), 16–28.
[5] Weiwei Cheng, Gjergji Kasneci, Thore Graepel, David Stern, and Ralf Herbrich. 2011. Automated feature generation from structured knowledge. In *CIKM*. 1395–1404.
[6] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *PVLDB* 13, 9 (2020).
[7] Christophe Croux and Catherine Dehon. 2010. Influence functions of the Spearman and Kendall correlation measures. *Stat. meth. & app.* 19, 4 (2010), 497–515.
[8] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding related tables. In *SIGMOD*. 817–828.
[9] Hakan Demirtas and Donald Hedeker. 2016. Computing the point-biserial correlation under any underlying continuous distribution. *Communications in Statistics-Simulation and Computation* 45, 8 (2016), 2744–2751.
[10] Włodzisław Duch, Tomasz Winiarski, Jacek Biesiada, and Adam Kachel. 2003. Feature selection and ranking filters. In *ICANN/ICONIP*, Vol. 251. 254.
[11] Julian Eberius, Maik Thiele, Katrin Braunschweig, and Wolfgang Lehner. 2015. Top-k entity augmentation using consistent set covering. In *SSDBM*. 8.
[12] Isabelle Guyon and André Elisseeff. 2003. An introduction to variable and feature selection. *JMLR* 3, Mar (2003), 1157–1182.
[13] Maurice G Kendall. 1938. A new measure of rank correlation. *Biometrika* 30, 1/2 (1938), 81–93.
[14] Yunmi Kim, Tae-Hwan Kim, and Tolga Ergün. 2015. The instability of the Pearson correlation coefficient in the presence of coincidental outliers. *Finance Research Letters* 13 (2015), 243–257.
[15] Arun Kumar, Jeffrey Naughton, Jignesh M. Patel, and Xiaojin Zhu. 2016. To join or not to join?: Thinking twice about joins before feature selection. In *SIGMOD*. 19–34.
[16] Andrew Lamb, Matt Fuller, Ramakrishna Varadarajan, Nga Tran, Ben Vandier, Lyric Doshi, and Chuck Bear. 2012. The vertica analytic database: C-store 7 years later. *PVLDB* 5, 12 (2012), 1790–1801.
[17] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. 2018. Table union search on open data. *PVLDB* 11, 7 (2018), 813–825.
[18] Robert F Tate. 1954. Correlation between a discrete and a continuous variable. Point-biserial correlation. *The Annals of mathematical statistics* 25, 3 (1954), 603–607.
[19] Patrick Valduriez. 1987. Join indices. *TODS* 12, 2 (1987), 218–246.
[20] Jason Weston, Sayan Mukherjee, Olivier Chapelle, Massimiliano Pontil, Tomaso Poggio, and Vladimir Vapnik. 2001. Feature selection for SVMs. In *NeurIPS*. 668–674.
[21] Chuan Xiao, Wei Wang, Xuemin Lin, and Haichuan Shang. 2009. Top-k set similarity joins. In *ICDE*. IEEE, 916–927.
[22] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD*. 97–108.
[23] Lei Yu and Huan Liu. 2003. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *ICML-03*. 856–863.
[24] Meihui Zhang and Kaushik Chakrabarti. 2013. InfoGather+ semantic matching and annotation of numeric and time-varying attributes in web tables. In *SIGMOD*. 145–156.
[25] Yi Zhang and Zachary G Ives. 2020. Finding Related Tables in Data Lakes for Interactive Data Science. In *SIGMOD*. 1951–1966.
[26] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *SIGMOD*. 847–864.