COCOA: COrrelation COefficient-Aware Data Augmentation

Mahdi Esmailoghi  
Leibniz Universität Hannover  
Hannover, Germany  
esmailoghi@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].

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 lightweight 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.

Table 1: Correlation in the existence of outlier.

<table>
<thead>
<tr>
<th>Area (Million sq. miles)</th>
<th>Calling Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.29</td>
<td>56</td>
</tr>
<tr>
<td>0.3</td>
<td>90</td>
</tr>
<tr>
<td>3.8</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>51</td>
</tr>
<tr>
<td>600</td>
<td>9800</td>
</tr>
</tbody>
</table>

Pearson = 1.0  Spearman’s = 0.1
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 \( \text{Scc} \) or \( \text{Rnc} \), respectively. Equation 1 shows the formula of \( \text{Scc} \) 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 \([8, 1, 4, 9]\) because the value 8 is second in the sorted list \([4, 8, 9, 11]\). \( R(x) \) and \( R(y) \) represent the average of ranks in the first and the second column respectively.

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

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

To compute the correlation between categorical columns and \( t \), the \( \text{Rnc} \) calculates the \( \text{Scc} \) between each one-hot feature of the categorical column and \( t \) \([3, 9, 18]\):

\[
\text{Rnc} = \frac{M_1 M_0}{m \cdot \text{std}} \cdot \sqrt{\frac{n_1 n_0}{m^2}}
\]

\( M_1 \) 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 \( \text{std} \) 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 \( \text{Rnc} \) between each one-hot feature and \( t \).

Based on Equation 2, we can calculate the \( \text{Rnc} \) 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 compute both \( \text{Scc} \) and \( \text{Rnc} \) 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-\( k_c \) joinable tables.

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: \( v_i \mapsto \{ (T_j, C_{ij}, P_{ij}), (T_k, C_{kj}, P_{kj}), ... \} \), where \( T_j, C_{ij}, P_{ij} \) 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-\( k_c \) 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 \((T_i)\) that contains three columns: Country, Area, and Calling Code. In our example, the index entry for the values “Germany” in Country would be: Germany \( \mapsto \{ (T_1, \text{Country}, 5) \} \).

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 \( \text{Scc} \) and \( \text{Rnc} \). 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 \( D’ \).
The main goal of the Scc values enabling the system to compute second item (order list), contains a list of values where minimum is the first value in the alphabetically sorted list. The categorical columns. To maintain the order information in a in one-hot features in the right order to calculate the Rbc relative orders enables us to walk through all non-zero values.

### 4.1 Order Index Structure

For categorical columns, 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.

#### 4.2 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 T₁ 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 3rd row of the table.
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 \( \text{Order index} \) 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 features 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 \( S - \frac{M_m}{n_2} \) and \( m - m_1 \) respectively:

\[
\text{Rbc} = \frac{m \cdot s_1 - n_1 \cdot S}{\text{std} \cdot m \cdot \sqrt{n_1(m - n_1)}}
\]

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 \( m, S, \text{ and } \text{std} \) 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.

Cocoa uses the \( \text{Order index} \) 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 \( \text{Order index} \) 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 \( 4^{\text{th}} \) 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 \( \text{Order index} \) 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/
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 repository2. 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 Rsc 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 k1 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 number of rows for one correlation calculation. 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.

Figure 6(b) shows the runtime experiment on Open Data. Here, k1 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 non-pruned external table.

Figure 6(c) compares COCOA to three hybrid system on DTWC with k2 = 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-k1 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
We presented a combination of the coverage and value consistency [8, 11] and 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/Rnc 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