

MC3: A System for Minimization of Classifier Construction Cost

Shay Gershtein
Tel Aviv University
shayg1@mail.tau.ac.il

Tova Milo
Tel Aviv University
milo@post.tau.ac.il

Gefen Morami
Tel Aviv University
gefenkeinan@mail.tau.ac.il

Slava Novgorodov
eBay Research
snovgorodov@ebay.com

ABSTRACT

Search mechanisms over massive sets of items are the cornerstone of many modern applications, particularly in e-commerce websites. Consumers express in search queries a set of properties, and expect the system to retrieve qualifying items. A common difficulty, however, is that the information on whether or not an item satisfies the search criteria is sometimes not explicitly recorded in the repository. Instead, it may be considered as general knowledge or “hidden” in a picture/description, thereby leading to incomplete search results. To overcome these problems companies invest in building dedicated *classifiers* that determine whether an item satisfies the given search criteria. However, building classifiers typically incurs non-trivial costs due to the required volumes of high-quality labeled training data.

In this demo, we introduce *MC3*, a real-time system that helps data analysts decide which classifiers to construct to minimize the costs of answering a set of search queries. *MC3* is interactive and facilitates real-time analysis, by providing detailed classifiers impact information. We demonstrate the effectiveness of *MC3* on real-world data and scenarios taken from a large e-commerce system, by interacting with the SIGMOD’20 audience members who act as analysts.

ACM Reference Format:

Shay Gershtein, Tova Milo, Gefen Morami, and Slava Novgorodov. 2020. MC3: A System for Minimization of Classifier Construction Cost. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD’20)*, June 14–19, 2020, Portland, OR, USA. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3318464.3384690>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGMOD’20, June 14–19, 2020, Portland, OR, USA

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-6735-6/20/06...\$15.00

<https://doi.org/10.1145/3318464.3384690>

1 INTRODUCTION

Search over massive sets of items is the cornerstone of many modern applications, particularly in e-commerce websites. Consumers express in search queries a set of properties, and expect the system to retrieve qualifying items. A common difficulty, however, is that the information on whether or not a product satisfies the search criteria is sometimes not explicitly recorded in the repository. Instead, it may be considered as general knowledge, or “hidden” in the picture/textual description associated with the product, thereby leading to incomplete search results. This occurs since sellers tend to provide only the most important properties, such as product name and cost, with additional information given through natural language description and/or a picture. For example, the fact that a given clothing item is a “red cotton shirt”, may be derivable only from the picture (for the color) and the textual description (for the cloth material).

To overcome this problem, there is a growing trend of companies leveraging Machine Learning (ML) algorithms to complete the missing information. In particular, much effort is put into building dedicated *classifiers*¹ that allow determining whether an item satisfies the given search criteria or not [10]. However, building accurate classifiers incurs non-trivial cost due to the volumes of high-quality labeled training data that is needed: labeling is often performed by humans for each data item separately, leading to bottleneck concerns, particularly when expertise is required, as there are fewer experts and their time is more valuable. Thus optimizing the classifiers construction process is an important goal.

We focus in this work on a common class of conjunctive search queries that test the satisfaction of a set of properties. Note that one can build a dedicated classifier for each query separately, or a classifier for each of the individual properties and then conjunct their output to evaluate the queries. Midway solutions are also possible.

The choice of classifiers has great monetary significance. Our system, *MC3*, allows selecting a set of classifiers of overall low-cost to cover a given query load. We first illustrate through a simple example how carefully selecting which classifiers to construct can lead to more cost-efficient solutions.

¹We consider here binary classifiers. These are often preferred in practice, as the increased granularity is essential for accurate performance [9].

pr_id	pr_title	description	price	team	color	brand
Ju18W1	Juventus White 18/19	Juventus shirt from 2018/19 season, ...	\$74.99			Adidas
Ch17B2	Chelsea Blue #18	Chelsea Olivier Giroud #18 shirt, ...	\$64.90	Chelsea	White	
CS19Re	CSKA Moscow #14	CSKA Moscow shirt, Nababkin #14	\$69.90	CSKA		Umbro

Figure 1: ‘Shirts’ relation example.

Example 1.1. Consider an e-commerce website where sellers upload soccer shirts. The website has a products database (depicted in Figure 1) which includes general information such as product title, price, and image. In addition, each shirt has a set of attributes (these have a grey background in the figure), e.g., brand, either provided by the seller, or derived automatically from the title, description, and image. Consider the following two search queries: “white adidas shirt” and “adidas chelsea shirt”. These are translated via NLP-based methods into the following SQL expressions:

```
SELECT * FROM Shirts WHERE `color` = 'White'
AND `brand` = 'Adidas';
```

```
SELECT * FROM Shirts WHERE `team` = 'Chelsea'
AND `brand` = 'Adidas';
```

Accurately answering the queries depends on correctly classifying each item w.r.t. whether or not they satisfy the color, brand and team requirements. To answer the first query one may train two binary classifiers (which we denote by the initials of the properties): a *W* classifier that tests whether the shirt’s color is white, and an *A* classifier that tests whether the brand is Adidas. One can also train a classifier for combinations of properties, i.e., an *AW* classifier. Similarly, for the second query, one could train the classifiers: *C* (Chelsea), *A* or *AC*. The choice of which classifiers to build depends on their construction costs. Note that while both queries need to know whether or not the shirt’s brand is “Adidas”, building this single-property classifier may not be the best choice. Indeed, while it may seem counter-intuitive, in some cases the cost of training a multi-property classifier (e.g., “X and Y classifier”) may be cheaper than the sum of the corresponding single-property ones (e.g. “X classifier” and “Y classifier”). For instance, detecting an Adidas shirt may be non-trivial and require many labeled samples (due to the variety of the Adidas shirt types that are used by different teams). Similarly, general detection of Chelsea shirts may be challenging (as the team switched multiple designs and sponsors in the last decade). In this example classification for the “Adidas Chelsea” conjunction is an easier task, since these shirts have only a few variants.

Assume that the classifier construction costs (for a given accuracy level) are as follows, for some cost unit *N*:

C: 5*N*, A: 3*N*, W: 1*N*, AC: 3*N*, AW: 5*N*

The optimal set of classifiers to build, given these queries and costs, is $\{AC, A, W\}$, whose cost is 7*N*. Interestingly, while already having the *A* classifier, it is better to train the *AC* classifier (instead of *C*), due to its lower cost.

The problem of selecting a minimal-cost set of classifiers to cover a given query load was initially introduced by [3] in a vision paper with several simplifying assumptions (all classifiers had the same construction costs, and queries contained at most two properties). In *MC3* we remove these assumptions, and provide a solution for the general case where queries and classifiers may be of arbitrary lengths and costs. The system we demonstrate in this work relies on our theoretical results, presented in [4]. *MC3* allows analysts to compute a provably low-cost set of classifiers that cover the given query load, and to add external constraints to adjust the requirements until a satisfying trade-off is achieved.

Demonstration Overview. We demonstrate the operation of *MC3* over real-world e-commerce data. Our demonstration illustrates a real-life scenario where an e-commerce analyst needs to select which new models to train given their costs and a weekly search-query load. The audience will play the role of the e-commerce analysts and experience the full cycle of processing, from input selection to solution generation and tuning, and finally actual classifiers generation.

2 TECHNICAL BACKGROUND

We first formally define the problem that we study in this work, which we call the Minimization of Classifier Construction Cost problem, and briefly discuss our key theoretical results and corresponding algorithms. Detailed discussion of our results along with formal proofs can be found in [4].

2.1 Formal Problem Definition

Recall that in the motivating setting, as described in the introduction, we have a set of distinct queries, which we denote by *Q*, each specifying a conjunction of properties. Let *P* denote a universe of properties. A query $q \subseteq P$ is a set of properties, and $C_q = 2^q \setminus \emptyset$ is the set of all possible binary classifiers that are relevant for *q*, each corresponding to a different subset of its properties. Given any item in the search space, a binary classifier, which pertains to some specific set of properties, returns *true* if the item satisfies all these properties, and *false* otherwise. Given a set of queries *Q*, let $C_Q = \cup_{q \in Q} C_q$ denote the set of all relevant classifiers. The input for the problem is a query set *Q*, and a weighting function *W* which assigns a cost for every classifier in C_Q .

A query *q* is said to be covered by a set of classifiers if the union of the properties in these classifiers is exactly the set *q*. The weight of a classifier set is defined as the sum of its weights. The objective is to output a set of classifiers of minimal weight such that it covers all queries in *Q*.

To simplify notation, we use x, y and z to denote properties, thereby denoting a query $\{x, y, z\}$ as xyz , whereas a classifier $\{x, y, z\}$, that tests for the conjunctive satisfaction of the properties (x, y, z) , is denoted by XYZ . Observe that C_Q does not include all possible classifiers corresponding to all subsets of P . For instance, if $P = \{x, y, z, u\}$ and $Q = \{xy, zu\}$, then $C_Q = \{X, Y, Z, U, XY, ZU\}$. Classifiers such as XZ are not included in C_Q , since they are not relevant to the solution of the problem. Concretely, since no query includes both x and z , the classifier XZ cannot be used to cover any query.

For a given query set Q , let $k = k_Q$ denote the maximal length of a query in it. This is an important parameter of the problem for two reasons. First, we study separately the variant of the problem where $k = 2$. Second, for the general problem, our approximation lower and upper bounds are functions of k . In our analysis, unless stated otherwise, k is assumed to be a constant, and in practice it rarely even exceeds 5. Furthermore, we denote by $I = I_Q$ the *incidence* of Q , which is defined as the maximal number of queries in Q a (single) property appears in, over all properties.

2.2 Model characteristics

In our model we assume that the classifiers are constructed in parallel with independent construction costs. While in practice there may be some overlap in terms of data labeling or crowd-worker time, it is arguably not trivial to quantify such overlap in advance. Hence, in our model the cost of each classifier is independent, and the total cost of a set of classifiers is the sum of all the individual costs. We note that in practice queries also often contain properties for which there already exists knowledge sufficient to accurately determine in real-time whether they hold for any given item. To ensure consistency with our model, we assign a cost of zero for any classifier testing a property (or a conjunction of such properties) for which a classifier construction is not necessary. Observe that this is not equivalent to stripping such properties from Q . For example, consider a query xy for which the property x does not require a classifier. In this case we set $W(X) = 0$, however we still consider for this query the classifier XY (in addition to Y), whose cost may be non-zero (and may be lower than that of Y).

Another characteristic of our problem definition is the requirement to cover all the queries in Q . Alternatively, one could consider a variant of the problem where queries have different weights (e.g. based on their importance) and there is a bound on the how much can be spent on classifiers construction. Then the goal would be to choose a set of classifiers whose cost is within the given budget, such that the sum of weights of all the queries it covers is maximized. We can show that this problem is much harder to approximate, and thus it requires different solution techniques than the ones we employ in our system, and leave it for future work.

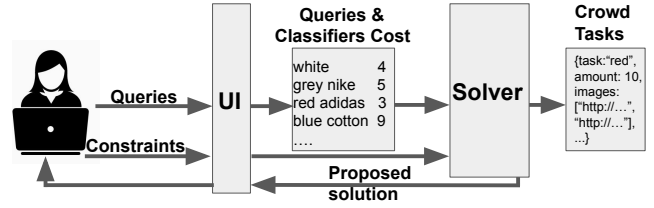


Figure 2: System Architecture

2.3 Theoretical Bounds and Algorithms

We proved that the special case where all queries are of size at most 2 can be solved optimally using maximum-flow-based techniques in almost linear time, unlike the general problem which is NP-hard. In practice this length bound holds for queries in many search categories [6]. Moreover, we showed empirically, that even when only 90% of the queries are of size at most 2, combining our algorithm for this special case with our general algorithm improves the quality of the solution.

As for the general problem, we proved that it is NP-hard, and cannot be approximated below a $\min\{(k-2), \ln I\}$ factor. We devised a novel algorithm, achieving an approximation factor of $\min\{\ln I + \ln(k-1) + 1, 2^{k-1}\}$. Concretely, we first perform a preprocessing procedure, that partitions the input into independent sub-instances to enable parallelization, and further reduces the input size by discarding classifiers which one can prove are not included in an optimal solution. The system then runs our algorithm in parallel with an algorithm for queries of size two, followed by our general algorithm over the residual problem. The solution of the best quality is then produced. The preprocessing step and the parallelization facilitate the real-time performance of our system. We performed an extensive set of experiments, demonstrating the efficiency of our solution, as well as its effectiveness, as it often greatly exceeds the worst-case guarantees [4].

3 SYSTEM AND DEMONSTRATION

We implemented *MC3* using Python and Flask. The system architecture is depicted in Figure 2. The analyst interacts with the system via the UI, and provides the list of queries that should be covered by the classifiers and the classifiers costs as an input. Both the queries and the classifier cost estimations are taken from a real-world dataset provided to us by a large e-commerce company. The Solver module generates the solution and presents it over a UI which allows to explore impact information of any selected classifier. The system is interactive, and allows the analyst to adjust the solution by adding/removing constraints. The process continues until the analyst is satisfied with the proposed solution. The system then generates dedicated labeling tasks that are automatically deployed and executed to a crowdsourcing platform.


 MC³ Ministration of Classifier Construction Cost	Queries:		
	Query	Cost	Must have?
	Adidas Red	4	<input type="checkbox"/>
	Red V-Neck	5	<input type="checkbox"/>
	Black Cotton	3	<input type="checkbox"/>
	Adidas Blue Cotton	2	<input type="checkbox"/>
	Adidas Black	5	<input type="checkbox"/>
	Puma White	4	<input checked="" type="checkbox"/>
	Nike Red	7	<input type="checkbox"/>
	White	2	<input checked="" type="checkbox"/>

Figure 3: Costs and Constraints definition

Demonstration Scenario We demonstrate the operation of *MC3*, an interactive system that identifies a provably low-cost set of classifiers to address a given query load. We reenact a real-life scenario where an e-commerce analyst, played by the audience, needs to select which models to train given a weekly search-query load and the estimated classifier costs. First, we will present a UI which captures the input, as depicted in Figure 3. Then, our algorithm will produce a solution, and we will show how it is recomputed given various user constraints, as shown in Figure 4. Finally, we will show how actual classifiers specifications are constructed to be uploaded to a crowdsourcing platform.

Input: To start the demonstration the analyst initializes the process by loading the queries from the e-commerce categories of her choice. Our UI visualizes the input as a list of queries alongside a graph whose nodes are the classifiers. A directed edge connects node A to node B , if the properties A tests are a subset of the ones B tests. The UI is equipped with zoom-in/out capabilities, enabling to explore different subcategories of queries and their corresponding classifiers. Moreover, clicking on a classifier prompts the system to highlight the queries that it addresses. Clicking on classifiers also allows to add constraints - a set of “must-have” classifiers to include in the final result. This is usually driven by the business logic of the company, e.g., upcoming seasonal changes and the expected queries they bring about.

Analysis: After exploring the input data, the analyst will run our algorithm to produce the initial solution. Together with the audience we will use the UI to examine the proposed solution, and explore which of the selected classifiers are used for which queries. Here again, the audience will be able to add different constraints, based on the produced solution, omitting less important (based on the accompanying statistics we will provide the audience with) or costly queries, and possibly specifying which classifiers must be selected or removed from consideration. MC3 recomputes the refined solutions on-the-fly. The new output will be presented with the changes, relative to the previous result, highlighted.

Generation: Once the system produces the final solution, approved by the analysts, it automatically generates annotation jobs, that can be loaded to the various crowd platforms, such as Figure-Eight². To conclude the demonstration we will show examples of these generated tasks.

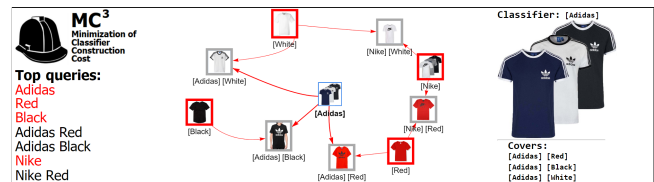


Figure 4: Solution graph representation

Related Work Supervised machine learning is a key solution for many real world problems. As in modern hybrid systems [2, 7, 10], the human component is the most costly and error prone, many works studied minimization of interactions with human workers [11]. Our current work falls under latter category. Previous work [3] considered a simplified variant of our model where all classifiers had the same cost and queries included at most two properties.

A related line of work is materialized view selection (MVS) in data warehouses [5, 8], where one materializes a set of queries that enable to answer an expected query load. The materialized objects are relations, rather than binary classifiers, the query language is richer, and a key objective is fast query load execution time. Partial query answers are often tolerable [1], in contrast to our setting. The approximation guarantees achieved by our PTIME algorithms, as mentioned in Section 2.3, do not follow, to our knowledge, from any previous MVS results for models that generalize ours.

Acknowledgements This work has been partially funded by the Israel Innovation Authority, the Israel Science Foundation, the Binational US-Israel Science foundation.

REFERENCES

- [1] J. R. Bernardino, P. S. Furtado, and H. C. Madeira. Approximate query answering using data warehouse striping. *JJIS*, 19(2):145–167, 2002.
- [2] J. Cheng and M. S. Bernstein. Flock: Hybrid crowd-machine learning classifiers. In *CSCW*, pages 600–611. ACM, 2015.
- [3] E. Dushkin, S. Gershtein, T. Milo, and S. Novgorodov. Query driven data labeling with experts: Why pay twice? In *EDBT*, 2019.
- [4] S. Gershtein, T. Milo, G. Morami, and S. Novgorodov. Minimization of classifier construction cost for search queries. In *SIGMOD*, 2020.
- [5] H. Gupta and I. S. Mumick. Selection of views to materialize in a data warehouse. *TKDE*, 17(1):24–43, 2005.
- [6] I. Guy. Searching by talking: Analysis of voice queries on mobile web search. In *SIGIR 2016*, pages 35–44, 2016.
- [7] A. Jarovsky, T. Milo, S. Novgorodov, and W.-C. Tan. Rule sharing for fraud detection via adaptation. *ICDE*, 2018.
- [8] Y. Kotidis and N. Roussopoulos. A case for dynamic view management. *TODS*, 26(4):388–423, 2001.
- [9] M. S. Sorower. A literature survey on algorithms for multi-label learning. *Oregon State University, Corvallis*, 18, 2010.
- [10] C. Sun, N. Rampalli, F. Yang, and A. Doan. Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *PVLDB*, 7(13):1529–1540, 2014.
- [11] M.-C. Yuen, I. King, and K.-S. Leung. A survey of crowdsourcing systems. In *SocialCom/PASSAT*, pages 766–773, 2011.

²<https://www.figure-eight.com>