Interactive Demonstration of Probabilistic Predicates

Yao Lu[1,2], Srikanth Kandula[2], Surajit Chaudhuri[2]
[1]UW, [2]Microsoft

ABSTRACT

We will demonstrate a prototype query processing engine that uses probabilistic predicates (PPs) to speed up machine learning inference jobs. In current analytic engines, machine learning functions are modeled as user-defined functions (UDFs) which are both time and resource intensive. These UDFs prevent predicate pushdown; predicates that use the outputs of these UDFs cannot be pushed to before the UDFs. Hence, considerable time and resources are wasted in applying the UDFs on inputs that will be rejected by the subsequent predicate. We uses PPs that are lightweight classifiers applied directly on the raw input and filter data blobs that disagree with the query predicate. By reducing the input to be processed by the UDFs, PPs substantially improve query processing. We will show that PPs are broadly applicable by constructing PPs for many inference tasks including image recognition, document classification and video analyses. We will also demonstrate query optimization methods that extend PPs to complex query predicates and support different accuracy requirements.

ACM Reference Format:
Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, Surajit Chaudhuri. 2018. Interactive Demonstration of Probabilistic Predicates. In *SIGMOD'18 Demo: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* https://doi.org/10.1145/3183713.3183751

1 INTRODUCTION

Relational data engines are increasingly being used to analyze data blobs such as unstructured text, images and videos [4, 5, 12, 17]. Such queries begin by applying user-defined functions to extract relational columns from blobs. Consider the following example which finds *red* or *blue SUVs* from city-wide surveillance video:

```
SELECT cameraID, frameID,
C1(F1(vehicleBox)) AS vehType,
C2(F2(vehicleBox)) AS vehColor
FROM (PROCESS inputVideo
        PRODUCE cameraID, frameNum, vehicleBox
        USING VehDetector)
WHERE vehType = "SUV" ∧ (vehColor=" red"|"blue");
```

Here, VehDetector extracts one or more bounding boxes that contain a vehicle from each video frame, $\mathcal{F}_1$ and $\mathcal{F}_2$ extract relevant features from each bounding box and finally C_1, C_2 are classifiers that identify the vehicle type and color using these features. The query plan is shown in Figure 1.

SIGMOD'18 Demo, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3183751

How to execute such machine learning queries efficiently? Clearly, traditional query optimization techniques such as predicate pushdown are not useful here because they will not push predicates below the user-defined functions (UDFs), which generate the columns used by the predicate. In the above example, vehType and vehColor are available only after VehDetector, the feature extractors and classifiers have been executed. Although the predicate has limited selectivity (perhaps 1-in-100 images have red or blue SUVs), every video frame has to be processed by all of the UDFs.

$$\texttt{Input} \rightarrow \underline{\texttt{VehDetector} \rightarrow \mathcal{F}_1, \mathcal{F}_2 \rightarrow C_1, C_2} \rightarrow \sigma_{\texttt{SUV}} \wedge (\sigma_{\texttt{red}} \vee \sigma_{\texttt{blue}}) \rightarrow \texttt{Result}$$

Figure 1: **Query plans to retrieve red or blue SUVs from traffic surveillance videos. Materializing the vehType and vehColor columns (underlined) accounts for almost all of the query cost, leaving little room for traditional optimization.**

$$\texttt{Input} \rightarrow \underline{\texttt{PP}_{\texttt{SUV}}, \texttt{PP}_{\texttt{red}}, \texttt{PP}_{\texttt{blue}}} \rightarrow \texttt{VehDetector} \rightarrow \mathcal{F}_1, \mathcal{F}_2 \rightarrow C_1, C_2 \rightarrow \sigma_{\texttt{SUV}} \wedge (\sigma_{\texttt{red}} \vee \sigma_{\texttt{blue}}) \rightarrow \texttt{Result}$$

Figure 2: **We construct and apply probabilistic predicates (PPs, dash-underlined) to filter data blobs that do not verify the predicates.**

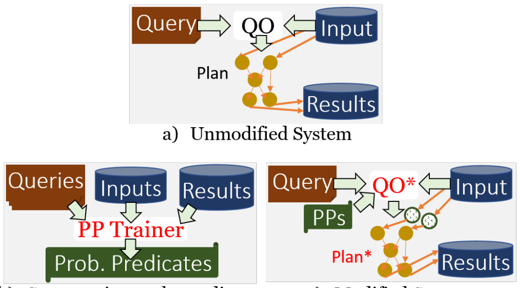
In our prior work [13], we proposed the notion of probabilistic predicates (PPs), which are binary classifiers that apply on the unstructured input and reduce the query cost by avoiding processing data blobs that will not pass the query predicate. Figure 2 shows a query plan with PPs; if the query predicate has a small selectivity and the PP is able to discard half of the frames that do not have the target objects, the query may speed up by 2×.

This demonstration will showcase our query processing engine to accelerate machine learning inference jobs by using probabilistic predicates. Our demo will use a variety of queries on images, documents and videos to show that PPs are applicable for a broad set of machine learning inference tasks. Users will be able to interact with our system in various ways including submitting new queries and comparing performance with and without PPs. Performance boosts of as much as 10× can be observed in this interaction.

Since query predicates can be diverse, trivially constructing a PP for each query predicate is unlikely to scale. Hence, we only constructs PPs for simple predicates and assembles, at query compilation time, an appropriate combination of available PPs that has the lowest cost, is within the accuracy target and is a necessary condition for (i.e., semantically implies) the original query predicate. We will demonstrate this functionality which can be embedded into a standard cost-based query optimizer; users will be able to see the various available plan choices, pick accuracy targets and examine job results.

2 THE DEMO SYSTEM

Demo system. The prototype used in our prior work [13] ran on a production cluster and extended a proprietary query processing



a) Unmodified System

b) Constructing prob. predicates c) Modified System

Figure 3: Comparing the unmodified query processing system on the top with the proposed system on the bottom. Key changes are in the training and use of probabilistic predicates (PPs). See [13] for details.

```
def query1(input, output){
    //define the ML pipeline; second parameter is schema
    let veh = input.apply_UDF(VehDetector, "frameID:int, boxID:int, vehBox:bin");
    let tfeat = veh.apply_UDF(C1, "frameID:int, boxID:int, feature:array");
    let cfeat = veh.apply_UDF(C2, "frameID:int, boxID:int, feature:array");
    let vehType = tfeat.apply_UDF(F1, "frameID:int, boxID:int, vehType:string");
    let vehColor = cfeat.apply_UDF(F2, "frameID:int, boxID:int, vehColor:string");
    output = vehType.join(vehColor, "frameID, boxID")    //join two streams
        .filter("vehType:SUV", "vehColor:red|blue")      //apply a three-clause predicate
        .distinct("frameID");                            //pick frames satisfying the filter
}
SystemP::execute(videos, std::out,       //define system input/output
        query1,                          //ML query handle
        use_PPs = True,                  //switch on/off PPs
        target_accuracy = 0.99)          //specify target filtering accuracy
```

Figure 4: An example query in our system.

platform [5]. To facilitate an interactive and small-scale demonstration, we will use a new prototype that extends Timely Dataflow [3]. The system supports queries written in Rust [2] and benefits from a variety of features including support for iterative and streaming queries. Our demo will run interactively on laptop-class hardware (with GPU support).

Figure 4 shows an example query used in our system which extracts semantic labels for image inputs. The feature extractors and classifiers are implemented as UDFs. Table 1 lists several UDFs that we have implemented in our system. In addition, we have also implemented a light-weight C-based deep learning inference engine, based on which we have trained deep neural networks for image labeling and object detection [15].

In our demonstration, we use the above system as a baseline. We will use several example queries and datasets (discussed later in §3). The query plan for Figure 4 is shown in Figure 5. Without PPs, the machine learning UDFs are executed as-is and the performance can be sub-optimal.

Queries with and without PPs. As shown in Figure 3(a), the baseline query processing system employs a query plan that may be built using a cascades-style cost based query optimizer. Figure 3(b) illustrates construction of PPs which can be either offline or online. While online construction of PPs is supported in our system as well, we will use offline constructed PPs in the demo due to interactive/latency constraints; we refer the readers to [13]

Module Name	Description
Feature Extraction - RGB Histogram	Extract RGB histogram feature.
Feature Extraction - HOG	Extract Histogram of Gradient feature.
Feature Extraction - Raw Pixels	Extract raw pixel feature.
Feature Extraction - PyramidHSVHist	Extract Pyramid HSV histogram feature.
Classifier/regressor - Linear SVM	Apply linear SVM on feature vector.
Classifier/regressor - Random Forest	Apply Random forest on feature vector.
Keypoint Extraction - SIFT	Extract SIFT keypoints in given image region.
Tracker - KLT	Tracking keypoints using KLT tracker.
Tracker - CamShift	Tracking objects using CamShift tracker.
Segmentation - MOG	Mixture of Gaussian background subtraction.
DNN Forward Propagation	A light-weight DNN inference engine.

Table 1: A partial list of ML modules provided in our system.

for experiments that construct PPs online. Unlike the actual UDF classifiers which take complex features as inputs (some example features are shown in Table 1), we use PPs that apply over raw input blobs, i.e., they take as input pixels from images and videos and bag-of-words representations for documents. The modified platform, as shown in Figure 3(c), takes two additional inputs compared to baseline systems: (1) the list of available probabilistic predicates and (2) a desired accuracy threshold for the query. The modified query optimizer injects an appropriate combination of PPs for the query based on the accuracy threshold; the PPs, shown in the figure as green dotted circles, execute directly on raw inputs and the remaining query plan is semantically equivalent to the baseline query plan.

Key technical challenges: The prototype system has been built to answer the following technical questions.

- *Filtering rate and efficiency:* PPs have to apply on the raw input which can be highly dimensional and arbitrarily distributed. If PPs are not efficient and/or do not lead to sizable data reduction, then query performance can worsen instead of improving. Hence, we uses a variety of classifiers to construct PPs including SVMs, kernel density functions and deep NNs; different techniques are appropriate for different queries and input types.
- *Query precision and recall:* Whereas conventional predicate pushdown produces deterministic results, how the classifiers used as probabilistic predicates will function on previously unseen inputs is unknown. Hence, filtering with PPs is parametrized over a precision-recall curve; different filtering rates (and hence speed-ups) are achievable based on the desired accuracy.
- *Handling complex predicates:* Since query predicates can be diverse, trivially constructing a PP for each query predicate is unlikely to scale. To generalize, we construct PPs for only simple clauses and extend the query optimizer to assemble, at query compilation time, an appropriate combination of PPs that has the lowest cost, is within the accuracy target and is a necessary condition for (i.e., semantically implies) the original query predicate.

Scope and limitations. We builds probabilistic predicates for simple clauses of the form $f(g_i(b), \ldots)\ \phi\ c$, where f and g_i are functions, b is an input blob, ϕ is an operator that can be $=, \neq, <, \leq, >, \geq$ and c is a constant. We builds PPs using diverse techniques and uses only PPs that are useful, i.e., high data-reduction, accuracy and throughput. With these PPs, the query optimizer in our system

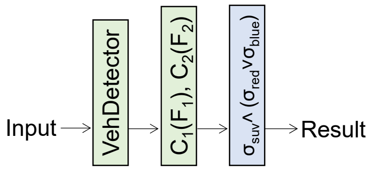


Figure 5: Query plan for the inference job described in Figure 4.

supports predicates containing arbitrary conjunctions, disjunctions or negations of the above clauses. On the other hand, predicates that do not decompose onto individual inputs are not supported, e.g., $SELECT\ *\ FROM\ T_1,\ T_2\ WHERE\ \mathcal{F}(T_1.a, T_2.b) > 0$ and $\mathcal{F}$ is not separable. UDFs that are not deterministic (e.g., those that adapt to input data or have randomized components) are also not supported.

3 DEMO SCENARIOS

The goals of our demonstration are listed below:

- *Broad applicability of PPs.* By using queries over documents, images and videos and by training PPs using a variety of techniques, we make a case that probabilistic predicates are broadly useful.
- *Query optimization with PPs.* Given complex query predicates, the query optimizer in our system picks which PPs to apply and determines their parameters for different precision/recall settings. We will show modified query plans for different target accuracy thresholds.
- *End-to-end system demonstration of query processing with PPs.* Our system offers the users an interface to submit different machine learning queries (such as the example in Figure 4) and will show the behavior with and without the use of PPs.

To demonstrate broad applicability of PPs, we consider different inference queries on different inputs. Some of the input blobs are highly dimensional (e.g., high-res video clips) whereas others are sparsely distributed (e.g., Wikipedia documents in bag-of-words format). Furthermore, several of the considered inference queries use non-linear classifiers (e.g., object recognition) as well as neural networks (e.g., image labeling). In more detail:

- **Case1: Image labeling.** The COCO dataset [11] has 120K images, each labeled with one or more of 80 object classes. Queries in this scenario retrieve images that contain objects satisfying the predicate which is a conjunction, disjunction, negation of one or more clauses over class labels such as 'has person' ∧ 'has dog' etc. We also generate similar queries over images and classes from the ImageNet [1] dataset. Figure 6 shows some example images from COCO.
- **Case2: Video activity recognition.** We use a popular video activity recognition dataset, UCF101 [16], which has 13K video clips ranging from ten seconds to several tens of seconds. Each video clip is annotated with one of 101 action categories such as 'applying lipstick', 'rowing', etc. We consider the problem of retrieving clips that illustrate an activity; some examples are in Figure 7.

Figure 6: Example images from COCO. A query may be retrieving images that satisfy the predicate 'has person' ∧ 'has dog'.

Figure 7: Example video clips and labels from UCF101. A query may be retrieving videos that satisfy the predicate 'applying lipstick' ∨ 'shaving beard'.

Our demo will show that, in general, queries having non-linear inference tasks require non-linear PPs (e.g., kernel-density-based PPs or PPs that are based on shallow NNs). However, linear SVM based PPs are inexpensive to execute and suffice for a large class of queries and datasets. Moreover, we will also see that dimensionality reduction techniques such as sampling, principal component analysis and feature hashing are needed in some cases; PPs can be too expensive to execute otherwise. We will also show that domain-specific data skipping tricks are highly relevant; for example, queries on videos benefit greatly from frame skipping, successive frame differencing, background subtraction etc.

To stress the ability to handle complex predicates, we will demonstrate performance on predicates which are conjunctions, disjunctions and negations of multiple clauses. We will train a small number of PPs, for simple clauses, and show that a large class of queries with complex predicates can receive performance improvements using these PPs.

Since PPs are only trained for simple clauses, complex query predicates will not have an exactly matching PP. We uses logical expressions over available PPs which are necessary conditions for the actual complex predicate. Optimal rewriting (fully exploring all possible necessary conditions) is an NP-hard problem. We use a heuristic algorithm, and Figure 8 (a,b) illustrates an example to parse a complex predicate into different alternate logical PP plans. Given an input query with complex predicates, our demo will show the parsing results.

We use the {precision, recall} curves of individual PPs, which are generated during PP training, to compute the precision and recall for conjunction/ disjunction expressions over multiple PPs. Key challenges, here, are to decide in which order the PPs should execute as well as the accuracy threshold to use per PP. For PP expressions identified in the above paragraph, we will show their cost and accuracy estimates. Figure 8 (b,c) illustrates this process

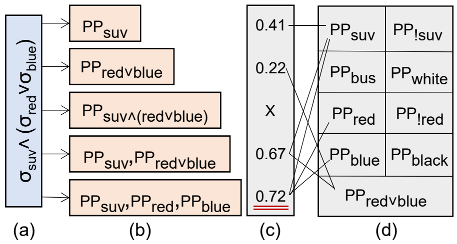


Figure 8: **Demonstration of the query optimization process in our system. (a) Input complex predicate. (b) Candidate PP expressions that are implied by the original predicate. (c) Estimated data reduction rate for each PP expression while satisfying accuracy threshold: 'X' indicates no matching PP in corpus. We underline the expression that has the largest data filtering rate. (d) Corpus of available PPs.**

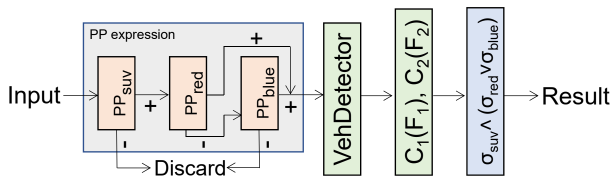


Figure 9: **Modified query plan with PPs for the machine learning inference job described in Figure 4.**

wherein a dynamic programming method is used to parametrize each PP. After obtaining the PP expression with the lowest cost that meets the accuracy threshold, the query optimizer in our system injects the PP expression into the query plan [13]; an example is shown in Figure 9.

Queries over the above inputs retrieve (different) portions of the inputs, i.e., they are selection queries. Beyond selection queries, we consider a more comprehensive query set that has group-by, aggregations, joins etc.

- **Case3: Comprehensive Traffic Surveillance.** Here, we consider the problem of answering comprehensive queries on traffic surveillance videos. Our datasets include surveillance videos of considerable size from the DETRAC [19] vehicle detection and tracking benchmark. We will use a query set with complex predicates including those checking on vehicle color, type, and traffic flow (vehicle speed and flow) etc.; details are in [13]. We manually annotate vehicle color (red, black, white, silver and other) and use the annotation of vehicle type (sedan, SUV, truck, and van/bus) provided by the benchmark.

4 RELATED WORK

There is a rich literature on optimizing queries with predicates: pushing predicates closer to input [18], optimal ordering of conjunctions [6], normalizing disjunctive and other complex predicates [9, 10] etc. When predicates rely on columns generated by user-defined operators, [14] shows that optimal ordering of the UDFs and predicates is NP-hard. Our system differs from these works because it uniquely injects additional probabilistic predicates (PPs) to the plan rather than optimally ordering existing predicates and UDFs. One prior work observes that if input column(s) are correlated with a user-defined predicate, then some function over those column(s) can be constructed and used to bypass the user-defined predicate [7]. While such functions over correlated columns are (simple) PPs, in our experience, correlation between input columns and predicates is harder to capture when inputs are images, videos or documents. Hence, we train PPs using SVMs, kernel densities, or shallow CNNs. Another prior work, NoScope [8], is a domain-specific model cascade customized for selection queries on videos. NoScope inserts a query-specific shallow DNN before a complex DNN and accepts/rejects frames early using the shallow DNN. We demonstrate gains across a wider range of datasets including images and documents, across a wider range of inference queries (accepting frames early only works for selection queries) and does not require per-query training (i.e., uses a small corpus of PPs for a large set of query predicates).

5 CONCLUSION

We focus on speeding-up machine learning inference queries, where classic static or post-facto optimization techniques, such as building indices or predicate push-down, are not feasible. Our key idea is to use probabilistic predicates (PPs) which execute over the raw input, without needing the predicate columns, and can successfully mirror the original query predicates. While introducing only a configurable amount of error, we show that PPs boost the performance of machine learning queries by as much as 10× on various large-scale datasets. This work is only a first step towards the larger goal of optimizing the execution of large-scale machine learning queries on relational big-data engines; open problems remain especially in dealing with correlated predicate clauses and handling queries with complex relational operations.

REFERENCES

[1] Imagenet. http://www.image-net.org.
[2] The rust programming language. http://www.rust-lang.org.
[3] Timely dataflow. https://github.com/frankmcsherry/timely-dataflow.
[4] Michael Armbrust et al. Spark SQL: Relational Data Processing in Spark. In *SIGMOD*, 2015.
[5] Ronnie Chaiken et al. SCOPE: Easy and Efficient Parallel Processing of Massive Datasets. In *VLDB*, 2008.
[6] Joseph M Hellerstein and Michael Stonebraker. Predicate migration: Optimizing queries with expensive predicates. *ACM SIGMOD*, 1993.
[7] Manas Joglekar, Hector Garcia-Molina, Aditya Parameswaran, and Christopher Re. Exploiting correlations for expensive predicate evaluation. In *SIGMOD*, 2015.
[8] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. NoScope: Optimizing neural network queries over video at scale. *VLDB*, 2017.
[9] A Kemper, G Moerkotte, K Peithner, and M Steinbrunn. Optimizing disjunctive queries with expensive predicates. In *SIGMOD*, 1994.
[10] Alon Levy, Inderpal Mumick, and Yehoshua Sagiv. Query optimization by predicate move-around. In *VLDB*, 1994.
[11] Tsung-Yi Lin et al. Microsoft COCO: Common objects in context. In *ECCV*, 2014.
[12] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. Optasia: A relational platform for efficient large-scale video analytics. In *ACM SoCC*, 2016.
[13] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. Accelerating machine learning inferences with probabilistic predicates. In *SIGMOD*, 2018.
[14] Thomas Neumann, Sven Helmer, and Guido Moerkotte. On the optimal ordering of maps and selections under factorization. In *ICDE*, 2005.
[15] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *CVPR*, 2016.
[16] Khurram Soomro et al. UCF101: A dataset of 101 human actions classes from videos in the wild. *Preprint arXiv:1212.0402*, 2012.
[17] Ashish Thusoo et al. Hive: A Warehousing Solution Over A Map-Reduce Framework. *Proc. VLDB Endow.*, 2009.
[18] Jeffrey Ullman. Principles of database and knowledge-base systems, 1989.
[19] Longyin Wen et al. Detrac: A new benchmark and protocol for multi-object tracking. *Preprint arXiv:1511.04136*, 2015.