Building and Accelerating a Declarative Platform for Machine Learning Model Serving

Yao Lu

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2018

Reading Committee:

Linda Shapiro, Chair

Magdalena Balazinska

Srikanth Kandula

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science and Engineering

University of Washington

Abstract

Building and Accelerating a Declarative Platform
for Machine Learning Model Serving

Yao Lu

Chair of the Supervisory Committee:
Professor Linda Shapiro
Computer Science and Engineering

Artificial intelligence has become the topic of the current and next decade. Numerous AI-related applications in computer vision, natural language processing and audio are deployed to change people's lives. Building effective algorithms and highly available systems to fulfill the exploding demand are the key challenges for both researchers and industry workers. To achieve these goals, we would expect technical advances in multiple areas including machine learning, databases and distributed systems.

This work focuses on providing better big-data platforms for AI applications. One core challenge here is to map various machine learning and domain-specific algorithms onto the big-data platform. Well-known systems including Spark or Hadoop apply user-defined functions (UDFs) and let the system engineer specify runtime details such as storage, degree of parallelism etc. Instead, we aim at applying UDFs upon a relational big-data platform; in such a manner, complex machine learning functionalities and legacy optimizations from the database domain can both come to bear. With this ground, we have developed Optasia [81], a dataflow system to efficiently process machine learning inference queries on video feeds from multiple cameras. Key gains of Optasia result from modularizing machine learning pipelines in way that relational query optimization can be applied. Specifically, Optasia can de-duplicate the work of common modules, auto-parallelize the query plans based on input size, number of inputs and operation complexity, and offer chunk-level parallelism

that allows multiple tasks to process the feed of a single camera. We show evaluation on complex vision inference queries with traffic videos from many cameras. Optasia produces high accuracy with many fold improvements in query completion time and resource usage relative to existing systems.

To this end, basic query optimization is explored in Optasia for better runtime parallelism. However, we identify that many other query optimization techniques, including predicate pushdown, are of limited use for machine learning inference queries. This is because the UDFs which extract relational columns from unstructured inputs are often very expensive; query predicates will remain stuck behind these UDFs if they happen to require relational columns that are generated by the UDFs. In our recent work [83], we show constructing and applying probabilistic predicates to filter data blobs that do not satisfy the query predicate; such filtering is parametrized to different target accuracies. To support complex predicates and to avoid per-query training, we augment a cost-based query optimizer to choose plans with appropriate combinations of simpler probabilistic predicates. Experiments with several machine learning workloads on a big-data cluster show that query processing improves by as much as 10×. Moreover, we showcase an interactive demonstration system in [84] with probabilistic predicates to accelerate machine learning inference queries. Users can query upon various document, image and video inputs, and inspect modified query plans as well as results.

These are the initial steps towards bringing declarative dataflow engines to bear for scalable machine learning model serving. Much work remains in AI + systems; advances for each of the subproblems will be applicable to end applications and will lead to better user experiences as well as less cost.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure Number Page

ACKNOWLEDGMENTS

I would like to thank my advisor Professor Linda Shapiro for unfailing support.

I would like to thank my mentors at Adobe and Microsoft Research who gave me generous mentorship during multiple internships (in alphabetical order): Xue Bai, Aakanksha Chowdhery, Jue Wang, Srikanth Kandula, Christian Konig and Matthai Philipose.

I also thank Professor Magdalena Balazinska who served as a reading committee member and gave very useful comments to my projects.

Finally, I would like to thank fellow students, staff and the faculty of the Paul G. Allen School of Computer Science and Engineering for creating a supportive and productive environment.

DEDICATION

to my wife, Jiali, and my son, Daniel

Chapter 1

INTRODUCTION

There has been a rapid growth in big-data processing on the cloud. Globally, the data stored in data centers will quintuple to reach 915 EB by 2020, up five-fold from 171 EB in 2015. Meanwhile, cloud workloads will more than triple from 2015 to 2020 [1]. Serving different machine learning applications like video analytics and recommendations is becoming an important customer for current big-data systems and poses significant burden to the underlying infrastructures.

To facilitate this, distributed or cloud systems have been developed to improve the performance for different aspects of machine learning. Recent systems such as TensorFlow [12] focus on fast training and exploration of deep neural networks (DNNs). Several functionalities are considered in these systems including constructing the computational graph and automatic differentiation. The training systems are usually optimized in an algorithm-specific way; several schemes have been developed to improve weight communication [74], supporting heterogeneous hardwares [12], and scheduling concurrent tasks [129]; such optimization can effect only a small family of machine learning classifiers or regressors.

In this work, we focus on machine learning model serving, not only for single-model classification/regression but also for the entire machine learning pipeline which may involve multiple classifiers and domain-specific modules such as feature extractors. Imagine to retrieve a "red SUV" among the video frames from a traffic surveillance camera, we have to run the input frames through an object detector to extract possible vehicles for each frame, different feature extractors and classifiers to recognize the color and type for each detected vehicle, and also potential video-specific algorithms such as background subtraction for efficiency purposes (details can be found in 2.1.2). Serving such comprehensive machine learning pipelines is pervasive to many big-data systems for different data analytics and machine learning applications. Existing systems such as MapReduce [42] and Spark [128]

succeed by implementing machine learning algorithms as user-defined functions (UDFs). The goal is to provide high throughput, low latency, or both, while ensuring correctness. An interpretive programming interface usually is used; that is, the runtime execution is defined exactly by the user code.[1] Therefore, unfortunately, optimizing machine learning pipelines in such big-data platforms is manual and ad-hoc: (1) for every query, end users have to specify runtime configurations, e.g., degree of parallelism and I/O; (2) reconfiguration is needed if any of the user code, input data, or runtime profile is changed, and (3) optimizing the machine learning queries requires expertise in both application and system domains.

We begin with building Optasia [81], a declarative platform for processing machine learning model serving queries on the cloud. Advantages of declarative query processing are twofold. First, the roles between machine learning and system engineering can be decoupled. Optasia maps machine learning pipelines to a relational, SQL-like programming language; different machine learning algorithms are encoded as *user-defined functions* (UDFs). Second, given this context, a cost-based query optimizer can be leveraged to automatically generate an optimal execution plan; query processing speed and resource usage can be improved without tedious manual tuning. More specifically, the query optimizer employed in Optasia is responsible for auto parallelization as well as multi-query deduplication - the optimized machine learning query plans differ with the input data size, while queries submitted by different users are merged into a big plan, so that duplicated processing can be saved.

To this end, we want to enforce auto-optimization to process machine learning queries and we have explored basic query optimization in Optasia for better runtime parallelism. However, we identify that many other query optimization techniques, including predicate pushdown, are of limited use for machine learning inference queries. This is because the UDFs, which extract relational columns from unstructured inputs, are often very expensive; query predicates will remain stuck behind these UDFs if they happen to require relational columns that are generated by the UDFs. Consider an example to retrieve images that contain oranges. When the selectivity is low, e.g., 1-in-100 images have oranges, much of the computation is wasted on non-orange images, because no matter if an image has oranges

[1]The cited webpage (http://yao.lu/Optasia/) shows an example query to extract image keypoints with Spark.

or not, it has to be sent to an object detector UDF, which can be expensive.

To solve this problem, in our recent work [83] we show constructing and applying probabilistic predicates (PPs) to filter data blobs that do not satisfy the query predicate. In this way, less data will go into the expensive machine learning UDF and therefore the query performance can be improved. In fact, such filters assemble the cascade filters that are well-studied in the machine learning literature. However, applying PPs for practical machine learning workloads faces new challenges. We can think of a PP as a binary classifier to distinguish inputs that check and do not check the query predicate; a simple linear SVM can solve this problem. However, given that inputs to actual machine learning workloads can be arbitrary (e.g., videos, images, text), not all of them are linearly separable. It is unknown that a single classification model can produce cheap, accurate and data reductive PPs for different inputs. We solve this problem by providing multiple classification backends and perform model selection to pick the right model that fits the input data.

The second challenge to apply PPs is that, machine learning queries can be diverse. Take the previous example again to retrieve a vehicle from traffic videos, each vehicle may be associated with multiple attributes (e.g., color, type, direction, speed, license plate number) that are generated by different machine learning pipelines. The combination of the predicate space is therefore huge. A PP build for red cars can not help queries that retrieve blue cars, and building a PP for every possible predicate is practically infeasible. Therefore, to support complex predicates and to avoid per-query training, we augment a cost-based query optimizer to choose plans with appropriate combinations of simpler probabilistic predicates. Our modified query optimizer will explore necessary conditions of a given query predicate, measure their performance, and pick the one that produces the best reduction rate given the accuracy threshold. We will also show experiments with several machine learning workloads on a big-data cluster in which query processing improves by as much as 10×.

We have built an initial version of Optasia on top of Microsoft's Cosmos system [30]. Optasia supports a few common vision and machine learning modules (see Table 2.1), and we describe some exemplar user queries (see Section 2.1.2). Optasia is evaluated on video feeds from tens of cameras from a highway monitoring company on a shared production cluster. We also use a variety of video feeds collected in and around the Microsoft campus.

The results show that the combination of vision modules and dataflow reduces resource requirements by about 3×. Further, we have implemented PPs upon Optasia. PPs are evaluated with different types of data that commonly occur in various machine learning serving applications, such as document classification on LSHTC, image labeling on COCO and ImageNet, and video activity recognition on UCF101. We also report on the performance of applying PPs on the traffic video feeds. The experimental results indicate that running online/batch machine learning queries with PPs achieves as much as 10× speedup with different predicate sets, compared with executing the queries as-is. To support complex predicates and to avoid per-query training, we augment a cost-based query optimizer to choose plans with appropriate combinations of simpler PPs.

To summary, the contributions of this thesis are as follows:

- A unified and customizable dataflow framework that computes optimal parallel query plans given any number of end-user queries for execution on a cluster.
- Fast and accurate implementation of several machine learning / computer vision modules that are needed in various model serving applications.
- A simple but broadly applicable design which incorporates a variety of PP construction techniques to accelerate online and batch machine learning inference queries.
- A query optimizer extension that matches complex predicates with available PPs and determines their parameters to meet the desired accuracy.
- Implementation and experiments on several real-world machine learning queries and datasets.

Finally, we showcase an interactive demonstration system in [84] with probabilistic predicates to accelerate machine learning inference queries. Users can query upon various document, image and video inputs, and inspect modified query plans and results.

These are the initial steps towards bringing declarative dataflow engines to bear for scalable machine learning model serving. Much work remains in AI + systems. A good big-data systems for AI spans a range of variety and presents the challenge of building large,

integrated systems that combine the advances from different areas. I want to participate in building an ecosystem in which a wide topics of AI/conventional applications can happen. On a low level, it would easily to deploy existing algorithms and models to a large cluster. On a high level, it would automatically optimize according to the tasks. Central to this ecosystem would be understanding the semantics of the tasks as well as the data which can be too complex for human articulation. Advances for each of the subproblems will be applicable to end applications and will lead to better user experiences as well as less cost.

The rest of this thesis is organized as follows. Chapter 2 describes the declarative platform Optasia, in which the design of the machine learning modules and the query processing system will be introduced. Chapter 2.3 demonstrates the evaluation and comparison of Optasia in terms of query execution time and resource usage. Chapter 3.1 summarizes the current machine learning serving systems and motivates the PPs. Chapter 3.2, 3.4, and 3.5 discuss the PPs in detail; parsing comprehensive query predicates into PPs will be introduced, while the effectiveness of applying PPs will be evaluated in Chapter 3.7. Chapter 4 further discusses an interactive demonstration system for PPs. Chapter 5 concludes the work and describes future work. Chapter 6 describes related works.

Chapter 2

OPTASIA: A DECLARATIVE PLATFORM FOR MACHINE LEARNING MODEL SERVING

Machine learning serving or inference pipelines in current big-data platforms are carefully hand-crafted with system engineers focusing on nitty gritty details such as how to parallelize, and in which order to execute the modules. Supporting ad-hoc queries or post facto big-data analytics remain key open problems. For the first part of this thesis, we study whether bringing together advances from two areas, machine learning and big data analytics systems, can lead to an efficient machine learning model serving system over big data.

A first challenge is to provide fundamental machine learning modules with high precision. To facilitate this, we focus on analyzing traffic surveillance videos, where several machine learning actions, such as classifying vehicles by color and type, re-identifying vehicles across cameras, tracking lane changes, and identifying license plates, are performed. Surveillance videos have low resolution, low frame rate and varying light and weather conditions. Processing surveillance videos accurately and efficiently contributes to a succeessful machine learning processing system. Further details are shown in Section 2.2.

Next, to address the challenge of scaling to a rich set of ad-hoc queries and data, we cast the problem as an application of a relational parallel dataflow system. Machine learning modules are wrapped inside some well-defined relational interfaces (processors, reducers and combiners [31]) that allow the query optimizer to reason about alternate plans. In this way, machine learning engineers can focus on individual modules, while end-users simply declare their queries over the modules in a modified form of SQL. The dataflow system translates user queries into appropriate parallel plans over the machine learning modules. Multiple standard query optimization improvements such as predicate push down (executing filters near input) and choosing appropriate join orders come to bear automatically [14]. We use a cost-based query optimizer that yields parallel plans [30] and is built per the Cascades [49]

framework. Further details are in Section 2.1.

The primary advantages of this combination are as follows:

1. Ease-of-use for end-users: we will show that complex queries such as amber alerts and traffic dashboards can be declared within a few lines.

2. Decoupling of roles between end-users and the machine learning engineers — machine learning engineers can ignore pipeline construction and need only focus on efficiency and accuracy of specific modules.

3. Automatic generation of optimal execution plans that among other things de-duplicate similar work across queries and parallelize appropriately: we will show examples where the resultant plans are much improved over those that are literally declared by the user query.

Note that our focus here is on machine learning model serving or inference over big data. This work is orthogonal to the commendable recent work in training deep neural networks (DNNs) on GPUs such as TensorFlow [12]. We review other related work in Section 6. The machine learning modules implemented in Optasia are simpler than DNNs and our dataflow system focuses on efficiently executing machine learning queries (that can use trained DNNs or other modules) on a cluster.

We have built an initial version of Optasia on top of Microsoft's Cosmos system [30]. Optasia supports several common machine learning modules, and we describe some exemplar user queries that perform traffic analysis (see Section 2.1.2). We evaluate Optasia by analyzing the video feeds from tens of cameras from a highway monitoring company on a shared production cluster. We also use a variety of video feeds collected in and around the Microsoft campus. Our results show that the combination of machine learning modules and declarative dataflow reduces resource requirements by about 3×; details are in Section 3.7.

To summarize, the novel contributions of Optasia are:

- Fast and accurate implementation of several machine learning modules that facilitate declarative big-data processing.

- A unified and customizable dataflow framework that computes optimal parallel query plans given any number of end-user queries for execution on a cluster.

- Implementation and initial results.

Much work remains; in particular, Optasia will benefit from more principled approaches to privacy (such as differential privacy or taint tracking) and improved video stores (compression, careful index generation). Nevertheless, we believe that Optasia targets a rich space of potential customers; customers that have a server farm or can upload data to a secure cloud provider can use Optasia today to benefit from fast, accurate, scalable, and customizable analysis of their data.

2.1 Declarative Dataflow for Machine Learning Model Serving

We build on top of the SCOPE [30] dataflow engine. Besides general SQL syntax, the dataflow engine offers some design patterns for user-defined operators: **Extractors**, **Processors**, **Reducers** and **Combiners**. We first describe how Optasia adopts these design patterns for different machine learning modules. Next, we describe our query optimization over machine learning model serving queries.

2.1.1 Dataflow for machine learning

Extractors ingest data from outside the system. Optasia supports ingesting data in different formats including image, videos and documents, as the user provides the ingestion logic. An extractor translates input data into a group of rows. An example to extract video frames follows.

```
... ← EXTRACT CameraID, FrameID, Blob
FROM video.avi
USING VideoExtractor();
```

Here `VideoExtractor()` after the **USING** syntax is a user defined function (UDF), for which the exact logic and parameters (if any) are defined by end users. Further UDFs afterwards will be introduced in Section 2.2. The columns extracted have both native

types (ints, floats, strings, etc.) and blobs (images, matrices, etc.). Optasia encodes image columns in the JPEG format to reduce data size and IO costs. The dataflow engine instantiates as many extractor tasks as needed given the size of input and the available degree of parallelism in the cluster. Extractor tasks run in parallel on different parts of the data input.

Processors are row manipulators. That is, they produce one or more output rows per input row. Several machine learning components are frame-local such as extracting various types of features (see Table 2.1), applying classifiers etc. A few examples follow. As with extractors, processors can be parallelized at a frame-level; Optasia chooses the degree-of-parallelism based on the amount of work done by the processor [14] and the available cluster resources. The following examples extract HOG features and license plate numbers from images.

```
... ← PROCESS ...
PRODUCE CameraID, FrameID, HOGFeatures
USING HOGFeatureGenerator();
```

```
... ← PROCESS ...
PRODUCE CameraID, FrameID, License, Confidence
USING LPRProcessor();
```

Reducers are operations over groups of rows that share some common aspects. Many machine learning components such as background subtraction (Section 2.2.4) and traffic flow (Section 2.2.2) use information across subsequent frames from the same camera. They are implemented using reducers; an example follows:

```
... ← REDUCE ...
PRODUCE CameraId, FrameId, VehicleCount
ON CameraId
USING TrafficFlowTrackingReducer();
```

Reducers translate to a partition-shuffle-aggregate. That is, the input is partitioned on the *group* and shuffled such that rows belonging to a group are on one machine. The number of reducers and partitions is picked, as before, per the amount of work to be done. Our

underlying dataflow engine supports both hash partitioning and range partitioning to avoid data skew [16].

Combiners implement custom join operations; they take as input *two groups* of rows that share some common aspects. Optasia uses combiners for correspondence algorithms, such as object re-identification (Section 2.2.4). Re-identification joins incoming data with a reference set and a kernel matrix that encodes the mapping between the two data streams. An example follows:

```
... ← COMBINE (SELECT * FROM X JOIN Z)
JOIN Kernel USING ReIDCombiner()
ON X.CamId = Kernel.Cam1, Z.CamId = Kernel.Cam2
PRODUCE Cam1, Cam2, FrameID1, FrameID2, Score;
```

The `ReIDCombiner()` function described above combines two video streams `X` and `Z` to compute the similarity between each pair of frames. A `Kernel` object is involved that is a pre-trained dictionary that stores the transformation between the two cameras. A combiner and other joins can be implemented in a few different ways. If one of the inputs is small, it can be broadcast in its entirety and joined in place with each portion of the other input; else, either side is partitioned and shuffled on the join keys and each pair of partitions are joined in parallel. The dataflow engine automatically reasons about the various join implementations.

2.1.2 Example machine learning model serving queries

To ground further discussion, we show three example scripts that mimic common queries to a video surveillance system. The complete data flow and user scripts can be found at http://yao.lu/Optasia.

User query 1: Amber alert

Problem: we consider the problem of an amber alert– retrieving a vehicle of a certain color, type, and license plate number. The dataflow is shown in Figure 2.1, and the user query is shown in Figure 2.2. Assume that machine learning engineers have written their modules as described in Section 2.2 using the dataflow explained in Section 2.1.1 and that the output

```
$v = EXTRACT vid, fid, frame
FROM @"/videos/*.avi" USING VideoExtractor();

$LPR = PROCESS $v USING LPRProcessor("LP.model,
"OCR.model")
PRODUCE vid, fid, LP, Confidence;

$featHOG = PROCESS $v PRODUCE vid, fid, feat
USING FeatureProcessor("HOG");

$featColor = PROCESS $v PRODUCE vid, fid, feat
USING FeatureProcessor("RGBHist");

$featType = COMBINE $featHOG WITH $featColor
ON $featHOG.{vid,fid} = $featColor.{vid,fid}
USING FeatureCombiner() PRODUCE vid, fid, feat;

$VehColor = PROCESS $featColor PRODUCE vid, fid,
color
USING ClassifierProcessor("color.model");

$VehType = PROCESS $featType PRODUCE vid, fid,
type
USING ClassifierProcessor("type.model");
```

Figure 2.1: Example dataflow to generate the `$LPR`, `$VehType`, `$VehColor` columns from multiple video streams. Multiple machine learning UDFs are utilized to extract features and classify them into semantically meaningful labels.

of these modules is available as *system tables*: $LPR, $VehType, $VehColor corresponding to license plates, vehicle types and vehicle colors. The user's query shown here is one select statement that joins three tables. Optasia only materializes the system tables when needed by user queries.

User query 2: Traffic violation

Problem: we consider the problem of detecting traffic law violations– vehicles that are overspeeding, weaving between lanes, or making illegal turns. The user query is shown in Figure 2.3. It is a single select statement.

User query 3: Re-identification

Problem: we consider the problem of retrieving a vehicle of the same type across two different cameras. The user query is shown in Figure 2.4.

Func: AmberAlert:
Input: search terms: vehicle type v_t, vehicle color v_c, license l
Output: matching {camera, timestamp}
State: Real-time tables for $`LPR`, $`VehType` and $`VehColor`

SELECT CameraID, FrameID, ($`LPR`.conf * $`VehType`.conf * $`VehColor`. conf)
AS Confidence

FROM $`LPR`, $`VehType`, $`VehColor`
ON $`LPR`.{CamId,FrameId}=$`VehType`.{CamId,FrameId},
$`LPR`.{CamId,FrameId}=$`VehColor`.{CamId,FrameId}
WHERE $LPR.licensePlate=$l \wedge$ $VehType.type=$v_t \wedge$ $VehColor.color=$v_c$

Figure 2.2: User query 1: Amber Alert.

Func: Traffic violation alert:
Input: Search terms: vehicle type v_t, vehicle speed v_s, illegal origin and destination boxes o, d
Output: Matching {Camera, Timestamp, VehicleImage}.
State: Real-time tables for traffic flow mapping `Traf`, `VehType`

SELECT CameraID, FrameID, VehImage
FROM `Traf`, `VehType`
ON `Traf`.{CameraID,FrameID}=`VehType`.{CameraID,FrameID}
WHERE VehType.vType=$v_t \wedge$ (Traf.vSpeed$\geq v_s \vee$ (Traf.vOri=$o \wedge$ Traf.vDes=d))

Figure 2.3: User query 2: Traffic Violation.

2.1.3 Optimizing machine learning model serving queries

Beyond the ease of specifying queries, we point out a few aspects of the above design. First, the end-user only needs to know the schema of the system tables that have been made available by the machine learning engineers. As long as they maintain the schema, machine learning engineers can change their pipelines transparent to users.

Second, Optasia substantially optimizes the execution of these queries. By recognizing that the filters are local to each input, they are pushed ahead of the join. That is, only rows matching the filters are joined rather than filtering after the join. This feature, called predicate push down [49], is standard in SQL query optimization. Other more novel aspects

Func: Re-ID: tracking a vehicle between two cameras:
Input: Search term: vehicle type vt
Output: Matching {camera1, timestamp1, camera2, timestamp2}.
State: Real-time tables for re-identification `ReID`, `VehType`$\{1, 2\}$

SELECT cameraId1, frameId1, cameraId2, frameId2
FROM `ReID`, `VehType1` as `VT1`, `VehType2` as `VT2`
ON `ReID`.{camId1,frameId1}={`VT1`, `VT2`}.{camId,frameId},
WHERE VT1.vType=vt $\wedge$ VT2.vType=vt;

Figure 2.4: User query 3: Re-identification.

of Optasia follow. (1) The system tables are materialized only on demand. That is, if no current query requires license plate recognition, the DAG of operations associated with that module does not execute. (2) Optasia exploits commonality between the various tables. For example, both `VehType` and `VehColor` require similar features from the raw data, and such features are computed only once. (3) When many queries run simultaneously, Optasia does even better. This is akin to multi query optimization [107] in the database literature. The filters coalesce across different queries. For example, amber alerts for `red SUV` and `green sedan` can be pushed down on to the `VehColor` table as the filter `red \/ green`. After the join, the individual amber alerts can separate out the frames that they desire (e.g., red frames). (4) Finally, a key aspect is that Optasia performs the most expensive operations exactly once (i.e., de-duplication) independent to the number of queries that may use such system tables.

To a reader familiar with relational operators [102], we note that `PROCESS` is a user-defined select and/or a project, `REDUCE` is a user-defined group-by and/or an aggregation and `COMBINE` is a user-defined join. Consequently, expressing machine learning queries with this vocabulary allows the query optimizer to reuse optimization rules from the corresponding relational operators. We believe that this lets the QO find generally good plans.

Method: Optasia achieves these advantages by treating all queries as if they were one large query for the purposes of optimization. However, during execution, the jobs corresponding to each query are only loosely coupled. As with other data-parallel frameworks [16], Optasia

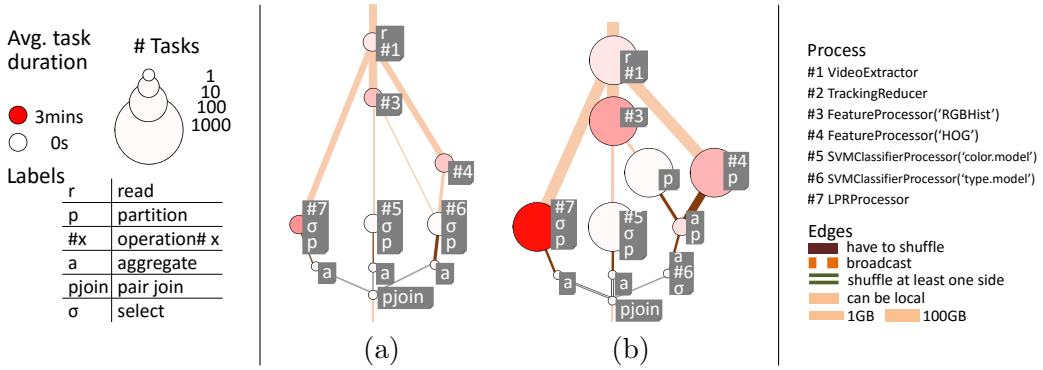


Figure 2.5: Dataflow and query plans of Amber Alert for (a) 1 GB input and (b) 100 GB video input. Note that the 100 GB input automatically parallelizes the tasks to minimize the query plan cost and the query latency. The workload is distributed nicely among the cluster nodes; partition and aggregation happen according to different input sizes.

stores the output of "tasks" in persistent storage; each task is a unit of execution that is idempotent and should finish within seconds. Optasia retries failing tasks. Faults in user-code will cause consistent failures and hence such queries will fail; queries with defect-free user code rarely fail in Optasia.

Query optimization details: Here, we sketch how the dataflow optimizations mentioned above are realized in Optasia. We do not claim contribution for these details, as they build upon a large body of work in relational query optimization [49, 50] and in adapting QO for parallel plans with user-defined operations [30]. We include them here for completeness. Optasia's contribution lies in reusing relational query optimization for machine learning queries.

The input is a collection of queries, each of which is a directed acyclic graph (DAG) of logical operations. The desired output is an execution plan that can be translated to a set of loosely coupled jobs. This plan should have the above-mentioned properties including appropriate parallelization and de-duplication of work.

Our QO can be explained with two main constructs. A memo data structure remembers for each sub-expression (i.e., an operator and its descendants) the best possible plan and the cost of that plan. A large collection of transformation rules offer alternatives for sub-expressions. Examples of rules include predicate push-down:

$$\mathcal{E}_1 \rightarrow \texttt{S} \rightarrow \texttt{Filter} \rightarrow \mathcal{E}_2 \iff \mathcal{E}_1 \rightarrow \texttt{Filter} \rightarrow \texttt{S} \rightarrow \mathcal{E}_2.$$

in which $\mathcal{E}_1, \mathcal{E}_2$ denote the proceeding and succeeding query components respectively. `S` is a project operator that extracts relation columns from the input. In this example, predicate pushdown transforms the query plan and pushes the `Filter` to before the projector `S` if the filter is upon data columns not related with `S`. For example, `S` generates a 'car color' column while `Filter` retrieves on the 'car type' column. As 'car color' and 'car type' can be thought of independent, filtering 'car type' can happen before generating the 'car color' column and hence saves the computational cost for `S`.

Transformations may or may not be useful; for example, which of the above choices is better depends on the relative costs of executing `Filter` and `S` and their selectivity on input. Hence, Optasia uses data statistics to determine the costs of various alternatives. The lowest cost plan is picked. Here, cost is measured in terms of the completion time of the queries given available cluster resources. The memo also allows de-duplication of common sub-expressions across queries. By applying these transformation rules till a fixed point is reached, Optasia searches for an efficient plan for all the queries.

To speed-up the search, we defer a few aspects such as the choice of appropriate degree-of-parallelism and avoiding re-partitions till after a good *logical* plan is discovered. Given a logical plan, the QO costs a variety of serial and parallel implementations of sub-expressions (e.g., 20 partitions on column `X`) and picks the best parallel plan.

Stepping back, we highlight with examples two aspects of the query optimization that we found useful for machine learning queries. First, Optasia adapts plans with varying input size. Simply changing the degree of parallelism (DOP) does not suffice. When plans transition from serial (DOP = 1) to parallel, corresponding partition-shuffle-aggregates have to be added and join implementations change (e.g. from broadcast join to pair-join). Figure 2.5 illustrates the plan for Amber Alerts (Figure 2.2) at two different input sizes. Next, Optasia automatically de-duplicates common machine learning portions of seemingly unrelated user queries. We illustrate this in Figure 2.12 when different user queries described

above run together. We defer further discussion to Section 2.3.2.

2.2 Example Applications in Optasia

To verify and evaluate Optasia, we develop several machine learning modules to support popular surveillance use-cases. In each case, we emphasize our innovations that (i) improve the accuracy and/or (ii) lower the computational cost on input video that is collected from deployments in the wild. We begin with a simple module.

2.2.1 Automatic license plate recognition (LPR)

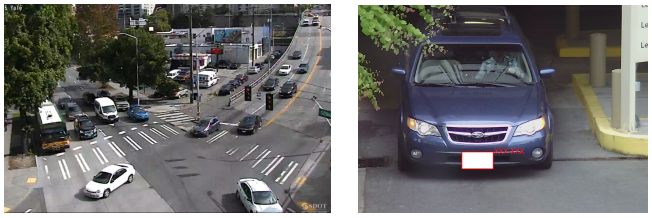
The license plate recognition module takes as input one or more images of vehicles passing through a *gateway* and outputs a set of possible license plates. The *gateway* can be a virtual line on a roadway or inside a garage.

Our goal here is to build a license plate recognition module over video that requires no additional hardware (such as magnetic coils, flashing lights or special-band lights). Furthermore, the video resolution is whatever is available from the wild. The goal is to extract for each frame the top few likely license plate numbers and the confidence associated with each number.

We use the following pipeline:

- *License plate localization* looks for a bounding box around the likely location of the license plate. We move a sliding window over the video frame and apply a linear SVM classifier [38] to estimate how likely each window is to have a license plate; the windows are sized in a camera-specific manner. The output is a set of potential bounding boxes per frame.

- *Binarization and character segmentation* converts each bounding box into binary pictures and cuts out individual characters of the license, if any. We use standard image processing techniques here such as edge detection [85], adaptive image thresholding [25], RANSAC baseline detection[47] and blob and character detection.



(a) Intersection. (b) Parking garage.

Figure 2.6: Example of traffic surveillance video feeds.

- *OCR:* a pre-trained random forest classifier [26] is applied to identify each character; the system searches for the characters 0–9, A–Z, and '-'. This yields, for each character in the image, several predicted values with soft probabilities for each value. The overall license plate is a combination of these predictions with confidence equal to their joint probability.

- *Post-processing:* Since license plates have some common formats (e.g., three numerals followed by three characters for plates in Washington state predating 2011), we use a pre-defined rule database to eliminate predictions that are unlikely to be valid license plates.

The LPR module requires a certain amount of resolution to be applicable. For example, we detect almost no license plates from the videos in Figure 2.6(a) but can find almost every license plate from the video in Figure 2.6(b). Qualitatively, we outperform existing LPR software due to the following reasons. (1) The exemplar SVM [92] is leveraged for license plate localization, while prior work [5] applies keypoint matching, which is less accurate. (2) A different OCR model is trained per state to account for the differences in characters across states; the baseline approach has a single OCR model and has found to be less accurate.

2.2.2 Real-time traffic flow mapping

On highways and at intersections, understanding the traffic flow has a variety of use-cases, including planning restricted-use lanes, speed limits, traffic signs and police deployment. Hence, there has been much interest in modeling vehicular traffic flow.

Figure 2.7: Step-by-step process of mapping traffic flow. Left: a vehicle entering the entry box. Right: a vehicle entering the exit box.

The most widely used method, however, is to deploy a set of cables ("pneumatic road tubes") across the roadway; this enables counting the number of vehicles that cross the coils and their velocity [93]. Such counts are typically not available in real-time. Further, the cables cannot capture information that is visible to the human eye (vehicle types, aggressive driving, vehicle origin-destination or how many right turn etc.).

Our goal here is to develop a module that extracts rich information about traffic flow from a video feed. Roadway surveillance cameras are typically mounted on towers or cross-beams; I use their fixed viewpoint to place labeled entrance and exit boxes on the roadway. An example of entrance and exit boxes is shown in Figure 2.7. Such annotation simplifies our traffic flow pipeline.

- Using a keypoint detection algorithm [109], we identify and track a vehicle that passes through the entrance box based on its keypoints [90].

- If (and when) the keypoints cross the exit box, we generate a *traffic flow record* stating the names of the entrance box, the exit box, the corresponding timestamps, and an estimate of vehicle velocity.

- These records are processed by our dataflow engine (Section 2.1) into real-time estimates of traffic flow or can be appended to a persistent store for later use.

Note that the above logic can simultaneously track the traffic flow between multiple entrance and exit boxes. In fact, we can compute a 3x3 matrix of traffic flow between each pair of entrance and exit boxes shown in Figure 2.7; the matrix denotes volume in each lane

and how often traffic changes lanes. Qualitatively, using keypoints to track objects is not new; I cite the following relevant prior work [109]. However, to the best of our knowledge applying these ideas in the context of real-time traffic flow is novel.

2.2.3 Vehicle type & color recognition

Building on the above pipeline, we do the following to identify the type and color of each vehicle.

- Once a vehicle is detected as above, we obtain an image patch for the vehicle by segmenting the image (see Section 2.2.4).

- Given the image patch of a vehicle, we extract various features including RGB histogram, and histogram of gradients (HOG) [39] and send them to a classifier.

- We use a linear SVM classifier trained with approximately 2K images belonging to each type and color. The output of the SVM is a class label (type or color) and the associated confidence. For vehicle type recognition the system classifies the vehicles into 'bike', 'sedan', 'van' , 'SUV', or 'truck'. For vehicle color recognition the system classifies the vehicles into 'white', 'black', 'silver', 'red', or 'others'. These labels were chosen based on their frequency of occurrence in the analyzed videos.

Our takeaway from this portion is that standard feature extraction and classifiers suffice to extract vehicle type and color from surveillance video; they do not suffice for more complex tasks such as detecting vehicle make and model. We chose mature and light-weight features and classifiers (see Table 2.1 for a list) and find that they yield reasonable results.

2.2.4 Object re-identification

Unlike object detection [121], the problem here is to identify an object that may be seen by different cameras. Potential applications include region-wise tracking of vehicles and humans.

Figure 2.8: Background subtraction. Left: a vehicle entering the camera view. Right: binary mask indicating moving objects.

At a high level, object reidentification involves (1) learning an effective image and object representation over features and (2) learning a feature transform matrix between each pair of cameras [76]. We do the following:

- The system learns a kernel matrix K for each camera pair by training on images of the same object that are captured at the two cameras. This matrix encodes how to "translate" an image from one camera's viewpoint to the viewpoint of the other camera.

- Then, the objects x seen at one camera are compared with objects z that appear at the other camera by computing a similarity score $d(x, z) = \phi(x) \cdot K \cdot \phi(z)^T$ where ϕ is a feature extraction function. Table 2.1 describes the features that are used for re-identification.

In practice, both x and z can contain multiple objects and hence the answer $d(x, z)$ could be interpreted as a pair-wise similarity matrix.

Background subtraction and segmentation

Background subtraction is a common practice; it reduces the redundancy in surveillance videos [44, 131]. We use the following method:

- Construct a model of the background (e.g., Mixture of Gaussians) based on pixels in the past frames.

- Use the model to remove the still pixels in each frame.

Relative to the other machine learning modules described thus far, background subtraction is lightweight and often executes first, as a pre-processor, in our analysis pipelines.

Take Figure 2.8 for an example, we segment the images into portions that are needed for further analyses as follows:

- Since the background subtractor removes still pixels, the remaining correspond to moving objects. We connect them using a connected-component algorithm [52] and return each component as a segment.

- The above approach does not work well with occlusions and dense frames; it can group cars in adjacent lanes as one object for example. Hence, we use heuristics based on the fixed viewpoint of surveillance cameras (e.g. typical size of objects of interest, lane annotations etc.) as well as an exemplar SVM [92] to further break the segments.

2.2.5 Conclusion on machine learning pipelines and modules

Table 2.1 describes a partial list of the techniques used in our machine learning modules. Our takeaway is that the described design lets us perform typical machine learning tasks with good accuracy and efficiency. We are unaware of a system that performs all of these tasks on surveillance videos. Further, Optasia improves upon point solutions (e.g. OpenALPR [5] for license plate recognition) because it (a) uses state-of-the-art machine learning techniques and (b) combines them with heuristics based on the fixed viewpoint of surveillance cameras. We note however that some of our video datasets have insufficient resolution for some tasks (e.g., inferring vehicle make/model). We next describe how to efficiently support user queries that use these machine learning modules at scale.

Module Name	Description	Involving Query
Feature Extraction - RGB Histogram	Extract RGB histogram feature.	Amber Alert, Re-ID
Feature Extraction - HOG	Extract Histogram of Gradient feature [39].	Amber Alert, Re-ID
Feature Extraction - Raw Pixels	Extract raw pixel feature.	Amber Alert
Feature Extraction - PyramidSILTPHist	Extract Pyramid SILTP histogram feature [76].	Re-ID
Feature Extraction - PyramidHSVHist	Extract Pyramid HSV histogram feature [76].	Object Re-ID
Classifier/regressor - Linear SVM	Apply linear SVM classifier/regressor [46] on feature vector.	Amber Alert, Re-ID
Classifier/regressor - Random Forest	Apply Random forest classifier/regressor [26].	Amber Alert
Classifier/regressor - XQDA	Object matching algorithm used in [76].	Object Re-ID
Keypoint Extraction - Shi-Tomasi	Extract the Shi-Tomasi keypoints in given image region [75, 109].	Traffic Violation
Keypoint Extraction - SIFT	Extract SIFT keypoints in given image region [79].	Amber Alert, Re-ID
Tracker - KLT	Tracking keypoints using KLT tracker [90].	Traffic Violation
Tracker - CamShift	Tracking objects using CamShift tracker [34].	Traffic Violation
Segmentation - MOG	Generate Mixture of Gaussian background subtraction [65].	All
Segmentation - Binarization	Binarize license plate images.	Amber Alert

Table 2.1: A partial list of machine learning modules provided by our framework.

Method	0 miss	$\leq$ 1 miss	$\leq$ 2 miss	rate (fps)
Optasia	0.57	0.75	0.82	4.8
OpenALPR	0.38	0.61	0.67	3.2

Table 2.2: LPR Evaluation.

2.3 *Experiment I: Evaluating* Optasia

2.3.1 Microbenchmarks of machine learning modules for traffic analysis

License plate recognition

Methodology: The dataset for this evaluation is a day-long video of the cars exiting a Microsoft campus garage. The video is pre-processed using background subtraction to prune frames that have no cars. We draw a random sample of 1000 images from the remaining frames and annotate the license plate area manually to train the localization module. Further, we annotate the license plate characters manually in 200 images to train the optical character recognition module. We use a test set of 200 *different* images to evaluate the License Plate Recognition module, end-to-end.

We benchmark our module against state-of-the-art OpenALPR [5], an open source Automatic License Plate Recognition library. Two metrics are used in the comparison: (i) *accuracy*, which measures the probability that the top N results contain the ground truth answer, and (ii) *maximum frame ingestion rate*, which is based on the processing time per frame. Both our module and OpenALPR run single threaded, and the average ingestion rate over a batch of video frames is reported.

	Seq1	Seq2	Seq3	Seq4	Avg	rate(fps)
Optasia	0.87	0.88	0.88	0.89	0.88	77
Baseline	0.46	0.40	0.31	0.58	0.44	42

Table 2.3: Vehicle counting accuracy and efficiency on four video sequences.

	Bike	Sedan	SUV	Truck	Van
Optasia	1.00	0.92	0.34	0.70	0.65
Baseline	0.01	0.67	0.17	0.05	0.10

Table 2.4: Car type classification accuracy. We compare with a simple guess according to the class distribution as baseline.

Results: Figure 2.9 shows that accuracy (probability that the recognized license plate is entirely correct) increases with N (the size of answers returned ordered by confidence); our method achieves reasonable results with only one answer. Table 2.2 demonstrates the detection ratios given different tolerance thresholds, i.e., we consider license plates with $\leq n$ wrong character(s) as correct. The table shows that our LPR module processes frames roughly 1.5× faster than the state-of-the-art license plate recognition software and also achieves better accuracy in terms of both absolute and relative correctness.

Real-time traffic flow mapping

Methodology: The dataset for this evaluation is 10 minute segments from WSDOT[8]; we picked cameras in the city of Seattle on both highways and surface roads. The goal is to count the vehicles in each lane.

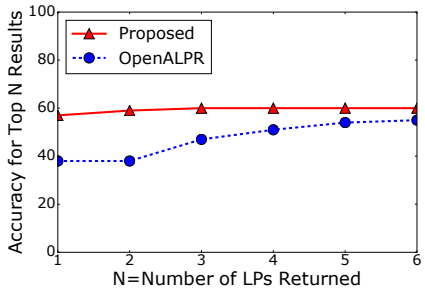


Figure 2.9: LPR Accuracy for Top N results.

Figure 2.10: Failure case for blob detection.

We compare against an open-source module [10], which does background subtraction and tracks blobs in the video. We measure the processing speed for each frame and the accuracy of the traffic volume in each lane.

Results: Table 2.3 shows that Optasia achieves an accuracy of 85–90% on four different video segments, while the accuracy of the car blob detection module is less than 60%. The baseline method detects blobs of moving objects and often fails when different vehicles occlude each other, as shown in Figure 2.10. Unlike this approach, our proposed method is based on keypoints and leverages per-camera annotation (entry and exit boxes in each lane) to protect against such shortcomings. We also see that our approach is less computationally complex leading to a 1.8× higher frame processing rate compared to the baseline.

Classification of vehicles

Methodology: The dataset for this evaluation is a one hour video of the intersection of Fairview avenue and Mercer street available from WSDOT [8]. We apply the above discussed traffic flow module to segment this video into per-vehicle patches. Our goal here is to classify these patches into types and colors; that is, assign to each image the labels listed in Section 2.2.3. We compare against a baseline that guesses the class for each image with probability equalling the likelihood of that class. [1]

Results: Table 2.4 shows that Optasia achieves different accuracy levels per class; across all classes Optasia is much better than random guesses. The relatively lower accuracy for the

[1] Uniformly random guesses for the class were less accurate.

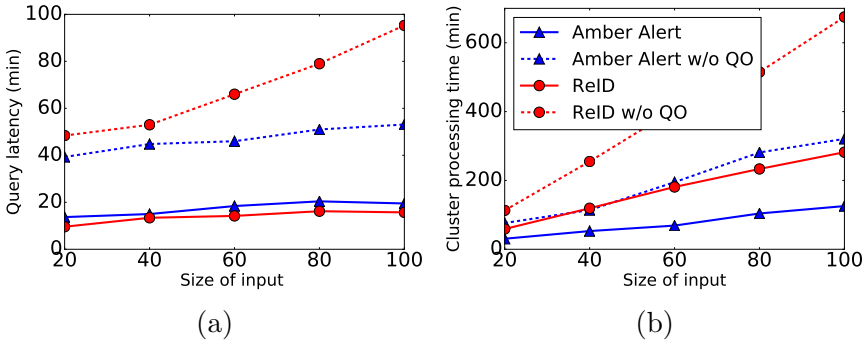


Figure 2.11: Query Optimization reduces the query completion time significantly for both Amber Alert and Re-ID (a) as the number of input videos increases for each query. Further, query optimization ensures the most efficient cluster resource utilization in terms of processing time (b).

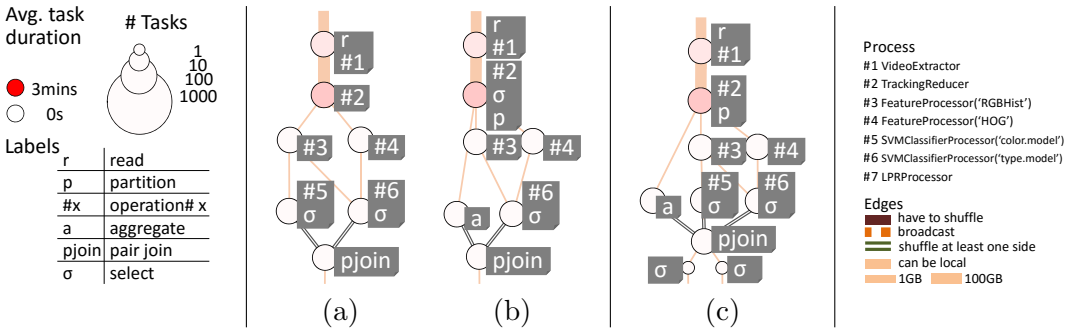


Figure 2.12: Query Plans of (a) Amber Alert query, (b) Traffic Violation query, and (c) Amber Alert+Traffic Violation query. Note that the combined query plan in (c) de-duplicates the common modules, thus minimizing the query plan cost and the query latency for both queries.

SUV class is because SUVs are routinely confused with sedans on the low-resolution videos in the dataset; the two classes have a similar size especially with "cross-overs". Overall, we believe that coarse granular categorization of vehicles is possible with the techniques built into Optasia.

2.3.2 Optimizing dataflow

Methodology: Over the video dataset from a Microsoft campus garage, we execute two end-to-end user queries: Amber Alert and Car Re-Identification across 10-100 sets of input. For Amber Alert, each input set contains a 90MB video from one camera, while for re-

identification, each input set contains video from two cameras. All the videos are 1 minute in length. We experiment by running each Amber Alert and Car Re-Id query independently as well as a group of (different) Amber Alert queries at one time on the input video set. Recall that an amber alert consists of a triple of (partial) license plate information, vehicle type and color. Further, for Car Re-Identification, we first filter by vehicle type, and then use re-identification over the set of matching frames.

Additionally, on a dataset of videos available from Seattle WSDOT website, we execute two end-to-end user queries: Amber Alert, and traffic violations across 50 sets of input. The Amber Alert query is similar to above, except it does not have license plate recognition; while for traffic violations, we measure the weaving of cars in the traffic flow from the leftmost lane to the rightmost lane.

We compare Optasia against a version of Optasia without query optimization. That is, the queries expressed by the end-user are run literally by the system. We measure the completion time of the query as well as the total resource usage across all queries (measured in terms of compute hours on the cluster). We repeat the experiment with different sizes of input to examine how Optasia scales. Besides, for Amber Alert, we vary the size of the query set (number of amber alert triples) from one to five to see how queries are affected by the optimizer.

Results: Figure 2.11 (a) plots the ratio of the completion time for Optasia with the version of Optasia that has no query optimization, for single queries on the garage feed. The results show that, with query optimization, Optasia is roughly 3× faster. Further, the completion time of Optasia remains constant as dataset sizes increase illustrating the fact that the QO sets the degree-of-parallelism correctly. The large gains arise from de-duplicating the work

	1 GB input	100 GB input
Average Task Duration	18.3 sec	38.6 sec
Cluster Computing Time	37.78 min	4101.75 min
Intermediate data size	1.95 GB	188.95 GB
Cross-rack Network IO	8.9%	8.9%

Table 2.5: Query optimization ensures efficient resource usage as the input video size scales from 1 GB to 100 GB for Amber alert with LPR query.

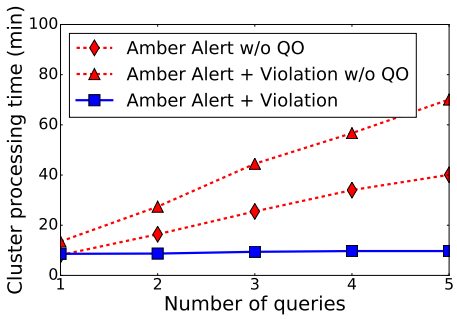


Figure 2.13: As the number of queries scale, query optimization ensures that the cluster processing time for both sets of queries stays constant by using auto-parallelization and de-duplication.

in the machine learning modules (e.g., generating HOG features etc.).

Further, Figure 2.11 (b) demonstrates the amount of cluster resources used by Optasia and the version of the Optasia that does not perform query optimization. We observe similar behavior to Figure 2.11 (a). The key difference is that the gap between the two lines in Figure 2.11 (b) measures the total-work-done by the query and is directly related to the size of the inputset; for small inputs the gap is lost in noise but at large inputs, the gap opens up quite a bit. On the other hand, the gap in Figure 2.11 (a) is query completion time; even a query that does more work can finish quickly, because our production cluster where these experiments were conducted is shared by jobs from many production groups and the cluster scheduler is work conserving; that is, it offers queries more than their share of resources if the cluster is otherwise idle.

Next, we evaluate how Optasia scales with different sizes of videos from the garage feed. Figure 2.5 shows the query plans for Amber Alert with LPR for two input sizes: 1 GB and 100 GB. In Figure 2.5 (b), the larger circle sizes and darker circles illustrate that the degree of parallelism is set correctly; hence, as Table 2.5 shows, the query completion time is almost similar even for larger input.

Figure 2.13 compares the improvement in completion time due to QO, while varying the number of queries on the WSDOT feed. We see that the improvements of Optasia increase when there are many similar queries; the value on the X axes here denotes the number of different queries of each type being executed simultaneously. Due to careful de-duplication

of work, the completion time of Optasia is roughly constant as the number of queries increase; the latency is only proportional to the amount of video examined. In contrast, the version of Optasia without QO is unable to de-duplicate the work, leading to substantially worse completion time as the number of queries increase. Figure 2.12 (a) and (b) show the query plans when the Amber Alert and Car Re-Identification queries are run individually, while Figure 2.12 (c) shows the query plan when the two queries are run simultaneously on the cluster. QO ensures that efficient de-duplication of the common modules in (c) thereby minimizing the query latency and resource usage on the cluster.

It is of course possible to carefully handcraft these machine learning pipelines to achieve a similar result. The key aspect of Optasia, however, is that such de-duplication (and query optimization, in general) occurs automatically even for quite complex queries. Thus, Optasia can offer these performance improvements along with substantial ease-of-use and with the ability to naturally extend to future user queries and machine learning modules.

Overall, we conclude that Optasia's dataflow engine not only allows end-users to specify queries in simple SQL-like syntax but by employing a powerful query optimization engine offers (a) the ability to run *similar queries* with nearly zero additional cost and (b) automatically scales the execution plan appropriately with growing volume of datasets.

Chapter 3

PROBABILISTIC PREDICATES

Relational data platforms are increasingly being used to analyze data blobs such as unstructured text, images or videos [18, 30, 82, 114]. In the previous sections, we introduced Optasia, a system that enables declarative machine learning on the cloud. However, several issues remain. Consider the following example, which finds all *red SUVs* from city-wide surveillance videos; the dataflow is shown in Figure 3.1.

SELECT `cameraID, frameID,`

$\mathcal{C}_1(\mathcal{F}_1($`vehBox`$))$ **AS** `vehType`, $\mathcal{C}_2(\mathcal{F}_2($`vehBox`$))$ **AS** `vehColor`

FROM (**PROCESS** `inputVideo`

PRODUCE `cameraID, frameID, vehBox`

USING $VehDetector$)

WHERE `vehType` $= SUV \wedge$ `vehColor` $= red$;

Here, $VehDetector$ extracts vehicle bounding boxes from each video frame. $\mathcal{F}_1$ and $\mathcal{F}_2$ extract relevant features from each bounding box, and finally $\mathcal{C}_1, \mathcal{C}_2$ are classifiers that identify the vehicle type and color using the extracted features.

How can we execute such machine learning inference queries efficiently? Clearly, traditional query optimization techniques such as predicate pushdown are not useful here, because they will not push predicates below the UDFs that generate the predicate columns. In the above example, `vehType` and `vehColor` are available only after $VehDetector$, $\mathcal{C}$ and $\mathcal{F}$ have been executed. Even when the predicate has low selectivity (perhaps 1-in-100 images have red SUVs), every video frame has to be processed by all the UDFs. Figure 3.1 shows a typical query plan for this query.

It is tempting to simplify the problem by separating the machine-learning components from the relational portion. For example, some component exogenous to the data platform

$$\texttt{Input} \rightarrow \underline{VehDetector \rightarrow \mathcal{F}_1, \mathcal{F}_2 \rightarrow \mathcal{C}_1, \mathcal{C}_2} \rightarrow \sigma_{SUV} \wedge \sigma_{red} \rightarrow \texttt{Result}$$

Figure 3.1: The query plan to retrieve red SUVs from traffic surveillance videos. Materializing the `vehType` and the `vehColor` columns (underlined) takes 99.8% of the query cost.

$$\texttt{Input} \rightarrow \texttt{PP}_{SUV}, \texttt{PP}_{red} \rightarrow \underline{VehDetector \rightarrow \mathcal{F}_1, \mathcal{F}_2 \rightarrow \mathcal{C}_1, \mathcal{C}_2} \rightarrow \sigma_{SUV} \wedge \sigma_{red} \rightarrow \texttt{result}$$

Figure 3.2: We construct and apply probabilistic predicates (`PPs`) to filter data blobs that do not satisfy the predicates.

may pre-process the blobs and materialize all the necessary columns; a traditional query optimizer is then applied on the remaining query. This approach may be feasible in certain cases but is, in general, infeasible. In many workloads, the queries are complex and use many different types of feature extractors and classifiers; pre-computing all possible options would be expensive. Moreover, pre-computing will be wasteful for ad-hoc queries since many of the columns with extracted features may never be used. In surveillance scenarios, for example, ad-hoc queries typically obtain retroactive video evidence for traffic incidents. While some videos and columns may be accessed by many queries, some may not be accessed at all. Finally, for online queries (e.g., queries on live newscasts or broadcast games), it could be faster to execute the queries and ML components directly on the live data.

In this work, our goal is to accelerate machine learning inference queries with expensive UDFs. Specifically, we propose the notion of probabilistic predicates (PPs). PPs are binary classifiers on the unstructured input that shortcut subsequent UDFs for those data blobs that will not pass the query predicate; the query cost is therefore reduced. As shown in Figure 3.2, if the query predicate has a small selectivity and the PP is able to discard half of the frames that do not have red SUVs, the query may speed up by 2×.

Furthermore, whereas conventional predicate pushdown produces deterministic filtering results, filtering with PPs is parametric over a precision-recall curve; different filtering rates (and hence speed-ups) are achievable based on the desired accuracy. Notice that we have departed from the strict boolean semantics of a predicate. However, machine learning queries are inherently tolerant to error, because even the unmodified queries have machine

learning UDFs with some false positives and false negatives. We show that injecting PPs does not change the false positive rate but can increase the false negative rate. We develop a mechanism to bound the query-wide accuracy loss by choosing which PPs to use and how to combine them. Our experiments show sizable speed-ups with negligibly small accuracy loss on a variety of queries and datasets.

We find that different techniques to construct PPs are appropriate for different inputs and predicates (e.g., based on input sparsity, the number of dimensions and whether subsets of the input that pass and fail the predicate are linearly separable). We use several PP construction techniques (e.g., linear SVMs, kernel density estimators, DNNs) and use model selection to pick an appropriate technique that has high execution efficiency, high data reduction rate and low false negatives.

We also propose new query optimization techniques to support complex predicates and ad-hoc queries. We show how to integrate PPs into queries that have selects, projects and foreign-key joins. These techniques reduce the number of PPs that have to be trained. Our system only trains PPs for simple predicates and relies on the query optimizer to choose, for a complex or ad-hoc predicate, appropriate combinations of available PPs based not just on the selectivity of the PPs but also on their accuracy.

We have prototyped probabilistic predicates in a large production data-parallel query processing cluster at Microsoft [30]. We demonstrate the usefulness of PPs on various commonly occurring machine learning inference tasks over different large-scale datasets such as document classification on LSHTC [96], image labeling on ImageNet [69], COCO [77] and SUNAttributes [97] and video activity recognition on UCF101 [112]. We also show how to run more complex queries on the traffic video feeds from tens of cameras. Our experiments indicate that running online/batch machine learning inference with PPs achieves as much as 10× speedup with different predicates compared with executing the queries as-is.

To summarize, our key contributions are:

- A simple but broadly applicable design which incorporates a variety of PP construction techniques to accelerate online and batch machine learning inference queries.
- A query optimizer extension that matches complex predicates with available PPs and

determines their parameters to meet the desired accuracy.

- Implementation and experiments on several real-world machine learning model serving queries and datasets.

System	Features	Classifiers/Regressors	Mat. Cost (sec)	Selectivity
Ads recommendations [101]	Bag-of-words	Collaborative Regressors	10^{-2} — 10^{-1}	1-in-hundreds
Video recommendations [41]	Browse history	Bayesian Regressors	10^{-1} — 10^{1}	1-in-thousands
Credit card fraud [113]	Physical loc. etc.	Neural Network	10^{-2} — 10^{-1}	1-in-thousands
Video tagging [60]	Keypoints	SVM w/ RBF kernel	10^{-1} — 10^{1}	1-in-thousands
Spam filtering [19]	Bag-of-word	Naive Bayes Classifier	10^{-2} — 10^{-1}	1-in-several
Image tagging [89, 126, 130]	Keypoints	Collaborative Regressors	10^{-1} — 10^{1}	1-in-thousands

Table 3.1: We examine queries from a few machine learning systems and list the features and classifiers that were used. Typical materialization costs are shown for each data item. We also list characteristics of typical predicates (number and type of clauses, selectivity).

3.1 Machine Learning Model Serving

We consider the problem of querying non-relational input such as videos, audios, images, unstructured text etc. This problem is crucial to many applications and services.

Consider for example the analysis of surveillance video [36, 99, 122]; recently, there have been city-wide deployments with thousands of cameras [4], body cameras worn by police [17] and security cameras deployed at homes. Some example inference queries include:

`Q1`: Find cars with speed ≥ 80 mph on a highway.

`Q2`: What is the average car volume on each lane?

`Q3`: Find a black SUV with license plate 'ABC123'.

`Q4`: Find cars seen in camera C_1 and then in C_2.

`Q5`: Send text to phone if any external door is opened.

`Q6`: Alert police control room if shots are fired.

To answer such queries, multiple machine learning UDFs such as feature extractors, classifiers etc. are applied on the input. The subsequent rowsets are filtered, sometimes implicitly (e.g., video frames without vehicles are dropped in `Q2`). Queries may also contain grouping, aggregation (e.g., `Q2`) and joins (e.g., `Q4`).

It is easy to see that the *materialization cost*, i.e., time and resources used to execute the machine learning UDFs, would dominate in processing these queries. It is also easy to see that materialization is query-specific; while there is some commonality, in general, different queries invoke different feature extractors, regressors, classifiers etc. Considering all the possible queries that may be supported by a system, the number of distinct UDFs on the input is vast. Hence, a priori application of all UDFs on the input has a high cost. Furthermore, the query predicates may be rather complex, and the queries can be both online and offline. Security alerts, such as `Q5` and `Q6`, are time-sensitive. Moreover, `Q2` may be executed online to update driving directions [57] or to vary the toll price of express lanes in realtime [29].

Beyond surveillance analytics, many applications share the above three aspects: large materialization cost, diverse body of machine learning UDFs, latency and/or cost sensitivity. We review a few such applications in Table 3.1. The materialization cost in these systems ranges from milliseconds to seconds per input data item, which can be significant when millions of data blobs are generated in a short period of time in, say, a video streaming system. Since queries use many different UDFs, offline systems would need large amounts of compute and storage resources to pre-materialize the outputs of all possible UDFs. Online systems which often require rapid responses can also become bottlenecked by the latency to pre-materialize UDFs.

Recently, many systems support triggers over live video streams (newscasts, sportscasts etc.) [3]; the user specifies a trigger such as "music concert" and the system finds matching video feeds by analyzing a large corpus of live video feeds (e.g., from youtube live or periscope). These systems also satisfy the three aspects above: the space of possible triggers that a user can specify is quite large, applying machine learning functions on live feeds dominates the query cost, and query selectivity is small if only a small number of feeds match the trigger and the query is latency-sensitive because users expect a quick answer.

3.2 Key Ideas and Challenges of PPs

To reduce the execution cost and latency of the machine learning queries, suppose we can apply a filter directly on the raw input which discards input data that will not pass the

original query predicate. Cost decreases because the UDFs following the filter only have to process inputs that pass the filter; a higher *data reduction rate* r of the filter leads to larger possible performance improvement. Let the cost of applying the filter and the UDF be c and u respectively; then the gains from early filtering will be $\frac{1}{1-r+(c/u)}\times$. Hence, the more efficient the early filter is relative to the UDFs (small c/u), the larger the gains will be. Moreover, the query performance can become worse (instead of improving) if $r \leq c/u$, i.e., the early filter has a smaller data reduction relative to its additional cost; hence, only filters that have a large data reduction rate will speedup the query.

Another important consideration is the *accuracy* of the early filter; since the original UDFs and query predicate will process input that is passed by the early filter, the false positive rate of the query is unaffected. However, the filter may drop input data that would pass the original query predicate, i.e., can increase false negatives. Unlike queries on relational data, machine learning applications have a built-in tolerance for error, since the original UDFs in the query also have some false positive and false negative rate. Hence, it is feasible, in our experience, to ask the users to specify a desired accuracy threshold a. Some queries, such as `Q1` and `Q2` in §3.1, tolerate a known amount of inaccuracy.

Challenges. To achieve sizable query speedup with desired accuracy, the following questions become important. First, how should these early filters be constructed? Since the raw input does not have the columns required by the original query predicate, constructing early filters is not akin to predicate pushdown [72] and is not the same as ordering predicates based on their cost and data reduction [43]. Instead, we propose to train binary classifiers that group the input blobs into those that *disagree* and those that *may agree* with the query predicate. The former are discarded, and the latter are passed to the original query plan. We call these classifiers *probabilistic predicates* (PPs), because each PP has associated values for the tuple ⟨data reduction rate, cost, accuracy⟩; it is possible to train PPs with different tuple values.

Next, how can probabilistic predicates be constructed that are *useful*, i.e., those that have a good trade-off between data reduction rate, cost and accuracy? Success in partitioning the data into two classes, a class that passes the original query predicate and the other that

does not, depends on the underlying data distributions. A predicate can be thought of as a decision boundary separating the two classes. Intuitively, any classifier that can identify inputs far away from this decision boundary can be a useful PP. However, the nature of the inputs and the decision boundary affects which classifiers are effective at separating the two classes. We use different techniques to build PPs– linear support vector machine (SVM) [117] for linearly separable cases, and kernel density estimator (KDE) [105] and deep neural networks [71] for non-linearly separable cases. We note that PPs can also be created using any other classifier technique (e.g., [27]). To handle data blobs with high dimensionality, we utilize sampling, principal component analysis (PCA) [64] and feature hashing [124]. We apply model selection to choose appropriate classification and dimensionality reduction techniques.

A third question is how complex predicates and ad-hoc queries can be supported? Since query predicates can be diverse, trivially constructing a PP for each query is unlikely to scale. Consider the example in Figure 3.2, a PP trained for $red \wedge SUV$ cannot be applied to $red \wedge car$ or $blue \wedge SUV$. Moreover, ad-hoc queries with previously unseen predicates cannot be supported. To generalize, we propose to only build PPs per simple clause and have the query optimizer, at query compilation time, assemble an appropriate combination of PPs that (1) has the lowest cost, (2) is within the accuracy target and (3) is semantically implied by the original query predicate; i.e., the PP combination has to be a *necessary* condition of the query predicate (since we use PPs to drop blobs that are unlikely to satisfy the predicate). We will show in Section 3.5 how we extend a standard cost-based predicate exploration procedure to generate various possible plans that use one or more of the available PPs and stay within the given accuracy threshold; our QO then picks the lowest cost plan from among these alternatives.

Scope, limitations, and connections. More precisely, we build probabilistic predicates for clauses of the form $f\left(g_i(b), \ldots\right) \phi\, v$, where f, g_i are functions, b is an input blob, ϕ is an operator that can be $=, \neq, <, \leq, >, \geq$ and v is a constant. As noted above, we build PPs using a diverse set of techniques and only for clauses that have useful ⟨data-reduction, accuracy, cost⟩. Using these PPs, our QO can support predicates that contain arbitrary

conjunctions, disjunctions or negations of the above clauses. Furthermore, we show in Section 3.5.4 how to inject PPs into queries that have selections, projections and foreign-key joins.

Some important limitations are worth noting. Predicates that do not decompose onto individual inputs are not supported; for example `SELECT * FROM` T_1`,`T_2 `WHERE` $\mathcal{F}$`(`T_1`.a,` T_2`.b) > 0` and $\mathcal{F}$ is not a separable function. UDFs that are not deterministic (e.g., those that have random components or adapt to the input) are also not supported because the mapping from the input to the predicate outcome, which the PPs learn and use, will also have to adapt along with the UDF.

The basic intuition behind probabilistic predicates is akin to that of cascaded classifiers in machine learning [119, 123]; a more efficient but inaccurate classifier can be used in front of an expensive classifier to lower the overall cost. Typical cascades, however, use classifiers that have equivalent functionality (e.g., all are object detectors). In contrast, PPs are not equivalent to the UDFs that they bypass; agnostic to the functionality of the UDFs that are bypassed, PPs are always binary predicate-specific classifiers. Without this specialization (reduction in functionality), it may be impossible to obtain a classifier that executes over raw input and still achieves good data reduction without losing accuracy. Furthermore, typical cascades accept and reject input anywhere in the pipeline; while this could work for selection queries whose output is simply a subset of the input, it will not easily extend to queries having projections, joins or aggregations. In general, our PPs apply directly on an input and reject irrelevant blobs; the rest of the input is passed to the actual query.

Our technical advances are in identifying and building useful PP classifiers (Section 3.4) and a deep integration with the QO (Section 3.5); the former involves careful model selection and the latter generalizes applicability to complex predicates and adhoc queries. A related system [63] identifies correlations between input columns and a user-defined predicate and then learns a probabilistic selection method which accepts or rejects inputs, based on the value of the identified correlated input columns, without evaluating the user-defined predicate. A contemporaneous system [67] uses a specialized DNN and video-specific filtering techniques such as background subtraction to speed up object detection on videos.

Probabilistic predicates have broader applicability and offer comparable or more gains as we show empirically in Section 3.7. Both the above systems accept blobs early and hence do not easily extend beyond selection queries. Furthermore, the probabilistic selection method used in [63] maintains state per distinct value of the correlated input columns; the proposed extension to handle multiple predicates and joins substantially increases the state needed (exponential in # of predicates and per distinct combined value of the correlated columns and join columns [62]). The above systems also pick a specialized pipeline per query, i.e., need training for each query. Since we train PPs for simple predicates and use the QO to generalize, our approach can help many more queries, even those that are previously unseen, at lower training and runtime costs. Empirical comparisons and some more details are in Section 3.7.

3.3 System Design

Language support for UDFs: Similar to recent query languages that support user defined functions (UDFs) [30, 31, 82], our query language offers some new templates for UDFs; a developer can implement a UDF by inheriting from the appropriate UDF template. The *processor* template, which we saw earlier encapsulates row manipulators; they produce one or more output rows per input row. Processors are typically used to ingest data and per-blob ML operations such as feature extraction. *Reducers* encapsulate operations over groups of related items. Context-based ML operations, such as object tracking which uses an ordered sequence of frames from a camera, are built as reducers. On the query plan, reducers may translate to a partition-shuffle-aggregate. *Combiners* encapsulate custom joins, i.e., operations over multiple groups of related items. Similar to a join, they can be implemented in a few different ways, e.g., broadcast join, hash join etc. More details can be found in the previous Optasia sections.

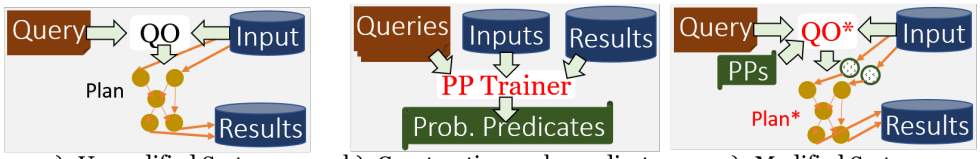
System inputs and outputs: With the above background, the inputs to our system are queries that may optionally have one or more user functions defined using the offered templates. The outputs are query results. As shown in Figure 3.3(a), the baseline system computes a query plan using a cascades-style cost based query optimizer. Our proposed

architecture, shown on the right, extends the baseline system in two ways: it trains and injects probabilistic predicates into query plans. The architecture has slight differences based on whether it is used in an online or batch context as we will describe next.

Constructing PPs: The basic task of constructing a probabilistic predicate uses binary labeled input data. The labels specify whether an input blob passes or fails the predicate. The output is a PP annotated with the predicate clause that it corresponds to, the cost of execution, and the predicted data reduction vs. accuracy curve. Further details are in Section 3.4.

The "outer loop" of deciding which clauses to train PPs for and how to acquire labeled input, shown in Figure 3.3(b), is as follows. In a batch system, we use historical queries to infer the simple clauses (defined in Section 3.2) that appear frequently in the queries. To train probabilistic predicates for these clauses, we find that labeled input data is sometimes already available because a similar corpus was used to build the original UDFs (e.g., training the classifiers). Alternatively, we can generate the labeled corpus by annotating the query plans; i.e., the first query to use a certain clause will output labeled input in addition to its query results. In an online system, the above process runs contemporaneously with the query execution. That is, at cold start when no PP is available, the query plans output labeled inputs for relevant clauses; periodically or when enough labeled input is available, the PPs are trained and subsequent runs of the query use query plans containing the trained PPs.

Applying PPs: Our modified query optimizer, shown in Figure 3.3(c) takes two additional inputs compared to the baseline QO: a list of trained probabilistic predicates and a desired accuracy threshold for the query. As described in Section 3.5, the modified query optimizer injects appropriate combinations of PPs for each query based on the accuracy threshold; the PPs, shown in the figure as green dotted circles, execute directly on the raw inputs and the remaining query plan is semantically equivalent to the original query plan.



a) Unmodified System b) Constructing prob. predicates c) Modified System

Figure 3.3: Comparing the unmodified system on the left with the proposed system on the right. Key changes are in the training and use of probabilistic predicates (PPs). See Section 3.3 for details.

3.4 Training Individual PPs

In this section, we describe the details of building a probabilistic predicate (PP). A PP for predicate clause p is uniquely characterized by the triple $\texttt{PP}_p = \{\mathcal{D}, m, r[a]\}$ where:

Training Set $\mathcal{D}$ is the portion of data blobs on which $\texttt{PP}_p$ is constructed. Each blob $\texttt{x} \in \mathcal{D}$ has an associated label $\ell(\texttt{x})$ which is $+1$ for blobs that agree with p, and -1 for those that disagree with p.

Approach m is the filtering strategy picked by our model selection scheme, indicating which classification $f(\cdot)$ and dimension reduction $\psi(\cdot)$ algorithms to use. The cost of the PP can be read from Table 3.2 for different approaches.

Data reduction rate $r[a]$ is the portion of data blobs filtered by $\texttt{PP}_p$ given the above settings. $a \in [0, 1]$ is the target accuracy, e.g., 1.0 or 0.95. We will train PPs that are parametrized with a target accuracy.

3.4.1 PP classifier 1: linear SVM

To identify data blobs that disagree with p, we consider linear support vector machines (SVMs) [117], which are well-known binary classifiers. A linear SVM classifier has the form of:

$$f_{\text{lsvm}}(\psi(\mathbf{x})) = \mathbf{w}^T \cdot \psi(\mathbf{x}) + b, \tag{3.1}$$

where $\psi(\mathbf{x})$ denotes a dimension reduction technique to project the input blob $\mathbf{x}$ onto fewer dimensions. We will discuss different dimension reduction techniques later in Section 3.4.4. $\mathbf{w}$ is a weight matrix and b is a bias term; the training fits $f(\cdot)$ to the labels $\ell(\cdot)$ of the blobs

in the training set $\mathcal{D}$ [61].

Constructing the PP: Equation 3.1 can be interpreted as a hyperplane that separates the labeled inputs into two classes as shown in Figure 3.4 (left). Perfect separation may not always be possible and hence we use the following decision function to predict the labels:

$$\texttt{PP}(\mathbf{x}) = \begin{cases} +1 & \text{if } f(\psi(\mathbf{x})) > th[a] \\ -1 & \text{otherwise} \end{cases} \tag{3.2}$$

where $th[a]$ is a decision threshold under the desired filtering accuracy a. It is easy to see that different values of $th[a]$ will produce different accuracy and reduction ratio. For example, with $th = -\infty$ all blobs will be predicted to pass the predicate ($\texttt{PP}(\mathbf{x}) = +1$), leading to zero reduction and perfect accuracy $a = 1$. We choose the parametric threshold $th[a]$ as follows:

$$th[a] = \max th \text{ s. t. } \frac{|\{x \in \mathcal{D} : f(\psi(\mathbf{x})) > th\}|}{|\{x \in \mathcal{D} : \ell(x) = +1\}|} \geq a. \tag{3.3}$$

It is useful to note that since the decision function is deterministic regardless of the $th[a]$ value, a PP parametrized for different accuracy thresholds can be built without retraining the SVM classifier. Figure 3.5 shows examples of choosing $th[a]$. Finally, the reduction ratio achieved by the PP can be computed as:

$$r[a] = 1 - \frac{|\{x \in \mathcal{D} : f(\psi(\mathbf{x})) > th[a]\}|}{|\mathcal{D}|} \tag{3.4}$$

Usage notes: Linear SVMs have pros and cons. They can be trained efficiently (see Table 3.2) and have small cost at test. However, linear SVMs yield a poor PP if (a) the input blobs are not linearly separable or (b) meeting the desired filtering accuracy results in a small data reduction. Using non-linear SVM kernels (e.g., RBF kernel [117]) is a potential fix; however, the computational complexity significantly increases for both training and inference, resulting in practically ineffective PPs. We introduce an alternate classification method below that is effective even when the problem is not linearly-separable.

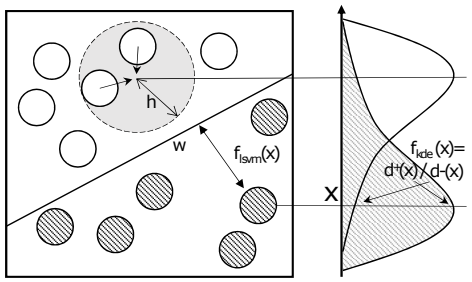


Figure 3.4: Demonstration of computing $f(x)$ by the PP classifiers. Left: SVM-based PP tries to find the decision boundary w. Right: 1-D visualization of the +1/-1 densities (dark circles for +1 and white circles for -1). KDE-based PP measures $f_{kde}(x) = d^+(x)/d^-(x)$ where $d^{\cdot}$ is estimated with a neighborhood of h.

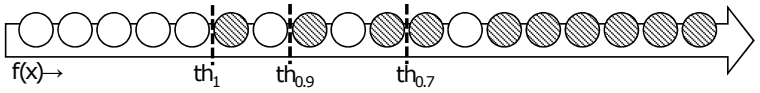


Figure 3.5: Data rows are ranked in ascending order according to their $f(x)$ values. Dark and white circles represent data blobs with +1 and -1 labels respectively. Threshold $th[a]$ is chosen to be the largest threshold value that correctly identifies an a portion of the +1 data points.

3.4.2 PP classifier 2: KDE

Machine learning blobs such as images and videos are high dimensional and not always linearly-separable. Here, we construct a non-parametric PP classifier that does not assume any underlying data distribution. Intuitively, a set of labeled blobs can be translated into a density function such that the density at any location x indicates the likelihood of its belonging to the set. Consider the density functions in Figure 3.4 (right). We propose to compute two density functions for the blobs in the training set according to their labels; let $d^+(\psi(\mathbf{x}))$ and $d^-(\psi(\mathbf{x}))$ be the density (likelihood) that $\psi(\mathbf{x})$ has a +1 or -1 label, respectively. As shown in the figure the density functions may overlap. As before, $\psi(\mathbf{x})$ denotes a dimension reduction technique. We then have the following kernel density estimator:

$$f_{\text{kde}}(\psi(\mathbf{x})) = \quad d^+(\psi(\mathbf{x}))/d^-(\psi(\mathbf{x})). \tag{3.5}$$

Intuitively, data points $\mathbf{x}$ with a true label of +1 should have a higher value on $d^+(\psi(\mathbf{x}))$ than $d^-(\psi(\mathbf{x}))$, leading to a high f_{kde} value; similarly if $\mathbf{x}$ has a true label of -1, f_{kde} should

be low.

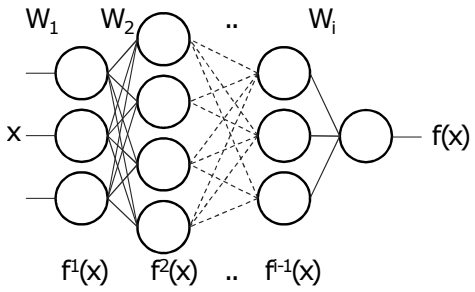
To build the density functions d^+ and d^-, we leverage kernel density estimation (KDE) [105]. $d^+(\psi(\mathbf{x}))$, the density of points with $+1$ labels, is defined as

$$d_h^+(\psi(\mathbf{x})) = \sum_{i=0,\ell_i=+1}^{n} K\Big(\frac{\psi(\mathbf{x}) - \psi(\mathbf{x_i})}{h}\Big) \tag{3.6}$$

where h is a fixed parameter indicating the size of $\psi(x)$'s neighborhood that we should look into. K is the kernel function to normalize $\psi(\mathbf{x})$'s neighborhood and we use a Gaussian kernel which yields smooth density estimations. $d^-(\psi(\mathbf{x}))$ is defined similarly over data blobs having -1 labels. We choose h using cross-validation; Silverman's rule of thumb [111] can also be used to pick an initial h.

Constructing the PP: To complete the construction of a probabilistic predicate using the KDE method, we note that Equations 3.2, 3.3, 3.4 can be applied by using f_{kde} in place of f_{lsvm}. In particular, as with the case of the linear SVM PP, we can parametrize the KDE PP without retraining the classifier.

Usage notes: Probabilistic predicates using the KDE method are effective even when the underlying data is not linearly separable; however this comes with some additional cost during test as noted in Table 3.2. In particular, applying the KDE PP at test time may require a pass through the entire training set, because the densities d^+ and d^- are computed based on the distance between the test point $\mathbf{x}$ and each of the training points. To avoid this, we use the k-d tree [24], a data structure that partitions the data by its dimensions. Similar data points are assigned to the same or nearby tree nodes. With a k-d tree, the density of an input blob $\mathbf{x}$ is approximately computed by applying Eq. 3.6 only on $\psi(\mathbf{x})$'s neighbors retrieved from the k-d tree (i.e., n' nodes as shown in Table 3.2 where $n' \ll n$, the number of training samples). The retrieval complexity is (on average) logarithmic in the feature length of the input blob.



$$f^i_{\text{fcn}}(\mathbf{x}) = g_i(W_i \cdot f^{i-1}_{\text{fcn}}(\mathbf{x}) + b_i),$$
$$g_i(x) = \texttt{ReLU}(x) = \max(0, x) \text{ or...}$$

Figure 3.6: Left: structure of a fully connected neural network. W_i are different fully connected layers. Right: formula at layer i. The input $f^0_{\text{fcn}}(\mathbf{x}) = \mathbf{x}$.

3.4.3 PP classifier 3: DNN

To demonstrate how the core classification methods can be extended, we consider the case of building a PP using a deep neural network (DNN) [71]. As shown in Figure 3.6, the classifier can have multiple fully connected layers interpreted as multiplying an input blob with different weight matrices sequentially. The function g_i, implemented as ReLU, sigmoid etc., is a non-linear activation applied after each fully connected layer, introducing non-linearity to the whole model.

Constructing the PPs: We argue that the PP design can incorporate any classifier that can be cast as a real-valued function with a threshold (i.e., as f in Eq. 3.2); the applicability of the classifier, of course, depends on the data distribution, predicates and classifier costs. In particular, DNNs also fit this requirement and we build DNN PPs by using f_{fcn} from Figure 3.6 in Eq. 3.2, 3.3 and 3.4.

Usage notes: DNNs have shown promising classification performance in various ML applications [69, 70]. However, the number of parameters to train is much larger (e.g., weight matrices) than the other classifiers we have discussed. Hence, training a DNN requires more data, and the training cost is significant. In practice, PPs built using DNNs are appropriate for queries and predicates that (a) have very expensive UDFs (e.g., a large DNN), (b) have a large training corpus or (c) repeat frequently to justify higher training costs.

3.4.4 Dimension reduction $\psi(\cdot)$

In practice, input blobs have many dimensions; for example, in videos, each pixel in a frame or an 8x8 patch of pixels can be construed as a dimension. In bag-of-words representations of natural language text, each distinct word is a dimension and the vector $\mathbf{x}$ for a document is the frequency of its words. When the dimensionality increases, the Euclidean distances used to compute $\mathbf{w} \cdot \mathbf{x}$ and $\mathbf{x} - \mathbf{x_i}$ lose discriminative power. Our overall approach to address this concern is to apply dimension reduction techniques before the classifier. However, this is optional, i.e., $\psi(\mathbf{x})$ can be $\mathbf{x}$.

Principal Component Analysis (PCA) [64] is a popular technique for dimension reduction. The input $\mathbf{x}$ is projected using $\psi(\mathbf{x}) = \mathbf{x}P$, where P is the linear basis extracted from the training data. We note two aspects. First, even when the underlying data is not linearly separable, applying PCAs does not prevent the subsequent classifier from identifying blobs that are away from the decision boundary. Second, computing the PCA basis using singular value decomposition is quadratic in either the number of blobs in the training set or in the number of dimensions $O(\min(n^2d, nd^2))$ [48]. To speed this up further, we compute PCA over a small sampled subset of the training data $\mathcal{D}$, trading off reduction rate for speed. Note the formulas in Table 3.2 where n can be either the full training set or the sampled subset.

Feature Hashing (FH). Feature hashing [124] is another popular dimension reduction technique which can be thought of as a simplified form of PCA that requires no training and is well-suited for sparse features. It uses two hash functions h and η as follows:

$$\forall i = 1 \ldots d_r, \quad \psi_i^{(h,\eta)}(\mathbf{x}) = \sum_{j=1}^{d} \mathbf{1}_{h(j)=i} \cdot \eta(j) \cdot \mathbf{x}_j, \tag{3.7}$$

where the first hash function $h(\cdot)$ projects each original dimension index ($j = 1 \ldots d$) into exactly one of d_r dimensions and the second hash function $\eta(\cdot)$ projects each original dimension index into ± 1, indicating the sign of that feature value. Thus the feature vector is reduced from d to d_r dimensions. It is easy to see that feature hashing is inexpensive and it has been shown to be unbiased [124]. However, if the input feature vector is dense, hash

Approach		Space complexity (per n input)		Computational complexity Training (per n input)		Testing (per input)	
Dim. reduction $\psi(\cdot)$	Classifier $f(\cdot)$	ψ cost	f cost	ψ cost	f cost	ψ cost	f cost
None, $\psi(\mathbf{x}) = \mathbf{x}$	Linear SVM	-	$O(d)$	-	$O(\max(n,d)\min(n,d)^2)$	-	$O(d)$
	KDE	-	$O(nd)$	-	$O(n \log d)$	-	$O(n' \log d)$
	DNN	-	$O(d_m)$	-	$O(bn(c_f + c_b + c_u))$	-	$O(c_f)$
PCA, $\psi(\mathbf{x}) = \mathbf{x}P$	Linear SVM	$O(dd_r)$	$O(d_r)$	$O(\min(n^2 d, nd^2))$	$O(nd_r^2)$	$O(d)$	$O(d_r)$
	KDE	$O(dd_r)$	$O(nd_r)$	$O(\min(n^2 d, nd^2))$	$O(n \log d_r)$	$O(d)$	$O(n' \log d_r)$
Feature Hashing, $\psi(\mathbf{x}) = \sum_j \eta(j) \cdot \mathbf{x}_j$	Linear SVM	-	$O(d_r)$	$O(nd)$	$O(nd_r^2)$	$O(d)$	$O(d_r)$
	KDE	-	$O(nd_r)$	$O(nd)$	$O(n \log d_r)$	$O(d)$	$O(n' \log d_r)$

Table 3.2: Complexity of different PP approaches for different dimension reduction ψ and classifier f techniques. n is the number of data items in the (sampled) training set; d (d_r) is the number of dimensions in vector $\mathbf{x}$ (that remain after dimensionality reduction); n' is the number of neighbor nodes in the k-d tree; d_m is the number of parameters in the DNN model; b is the number of epochs; $c_f/c_b/c_u$ are the forward/backward propagation/update costs. We assume $d_r \ll n$.

Approach		Applicability		
Dim. reduction $\psi(\cdot)$	Classifier $f(\cdot)$	Non-Linear	Non-Sparse	High Dim
None, $\psi(\mathbf{x}) = \mathbf{x}$	Linear SVM	X	✓	✓
	KDE	✓	✓	X
	DNN	✓	✓	✓
PCA, $\psi(\mathbf{x}) = \mathbf{x}P$	Linear SVM	✓	✓	
	KDE	✓	✓	✓
Feature Hashing, $\psi(\mathbf{x}) = \sum_j \eta(j) \cdot \mathbf{x}_j$	Linear SVM	X	X	✓
	KDE	✓	X	✓

Table 3.3: Applicability of different PP approaches.

collisions are frequent and classifier accuracy becomes worse.

3.4.5 Model Selection

Thus far, we have described three techniques to construct PPs and two dimension reduction techniques, all of which can be used with or without sampling the training data and several parameter choices (e.g., number of reduced dimensions d_r for FH); this leads to many possible techniques for PPs. As we describe next, we expect future systems to use a few other techniques. Hence, it is crucial to determine quickly which technique is the most appropriate for a given input dataset. We use the following model selection.

Given different PP methods $\mathcal{M}$, we select the best approach m by maximizing the

reduction rate r_m for that approach:

$$m = \arg\max_{m \in \mathcal{M}} r_m[a]. \tag{3.8}$$

Furthermore, these methods have different applicability constraints as summarized in Table 3.3. We first prune $\mathcal{M}$ using these applicability constraints. To compute $r_m[a]$ quickly, we use a sample of the training data, fix $a = 0.95$, randomly choose a few different simple clauses, train the classifiers described above and use the technique that performs *better*. Our experiments show that the input dataset has the strongest influence on technique choice; that is, given a certain type of input blobs, the same PP technique is appropriate for different predicates and accuracy thresholds etc.

3.4.6 Other PP details

Overfitting: To avoid overfitting on the training data, we randomly divide the input set of blobs $\mathcal{D}$ into training and validation portions. The classifiers are trained using the training portion $\mathcal{D}_{\text{train}}$ but the accuracy-data reduction curve $r[a]$ is calculated on the validation portion $\mathcal{D}_{\text{val}}$.

Classifiers built for a PP on predicate p can be reused for the PP on predicate $\neg p$: Given the classifier functions (e.g., $f_{\text{lsvm}}, f_{\text{kde}}$) built for a predicate p, note that multiplying these functions with -1 yields the corresponding classifier functions for predicate $\neg p$. Hence, the PP for predicate $\neg p$ can reuse the classifier and compute Eq. 3.3 and 3.4 with $-1 * f$ instead.

The input feature to PP is a simple representation of the data blob, e.g., raw pixels for images, concatenation of raw pixels over consecutive frames (of equal duration) for videos, and tokenized word vectors for documents.

3.5 Query optimization over PPs

In Section 3.4, we have seen how to construct PPs for simple clauses. Here, we describe interaction with the query optimizer which achieves the following goals.

First, for a query with a complex predicate or previously unseen predicate, which PPs may be useful? Recall that a query can use any available PP or combination of available PPs that is a necessary condition to the actual predicate. Given a complex query predicate $\mathcal{P}$, the QO generates zero, one or more logical expressions $\mathcal{E}$ that are equivalent or necessary conditions for $\mathcal{P}$ but only contain conjunctions or disjunctions over simple clauses. That is, $\mathcal{P} \Rightarrow \mathcal{E}$. The challenge, as we will show, is that there can be innumerably many choices of $\mathcal{E}$; so exploration of choices has to be quick and effective. Further details are in Section 3.5.1.

Next, how should the best implementation over the available expressions of PPs be picked, while meeting the query's accuracy threshold? For individual PPs, their training already yields a cost estimate and the accuracy v.s. data reduction curve. The challenge is to generate these estimates for logical expressions over PPs. Our QO extension explores different orderings of the PPs within an expression $\mathcal{E}$ and explores different assignments of accuracy to each PP which ensure that the overall expression meets the query-level accuracy threshold. Further details are in Section 3.5.2. The QO extension outputs a query plan with the chosen implementation.

Example: Consider a complex predicate of the form: $\mathcal{P} = (p \vee q) \wedge \neg r \wedge \mathcal{P}_{\texttt{rem}}$. Here p, q and r are simple clauses for which PPs have been trained and $\mathcal{P}_{\texttt{rem}}$ is the remainder of the predicate. Each PP is uniquely characterized in part by the simple clause that it mimics; we use $\texttt{PP}_{\texttt{p}}$ to denote the PP corresponding to the simple clause p. Figure 4.3 (right) shows the various possible expressions over PPs that may be used to support this complex predicate. We note a few points here. (1) Some parts of $\mathcal{P}$, such as $\mathcal{P}_{\texttt{rem}}$ in this example, that are attached by an '*and*' can be ignored since PPs corresponding to the rest part will be necessary conditions for $\mathcal{P}$. (2) When the predicate has a conjunction over simple clauses, PPs for one or more of these clauses can be used. This is shown in the first two rows of the table. (3) A disjunction of two PPs, e.g., $\texttt{PP}_p \vee \texttt{PP}_q$ is a valid PP for the disjunction $p \vee q$. The proof follows from observing Figure 3.7; only blobs that do not pass both the PPs will be discarded (shown using lines labeled with a '$-$'). As before, there will be no false positives since the actual predicate applies on the passed blobs but there

Complex predicate	Implied logical expr. over PPs
$(p \vee q) \wedge \neg r \wedge \mathcal{P}_{\texttt{rem}}$	$\Rightarrow p \vee q \Rightarrow \texttt{PP}_{p \vee q} \Rightarrow \texttt{PP}_p \vee \texttt{PP}_q$ $\Rightarrow \neg r \Rightarrow \texttt{PP}_{\neg r}$ $\Rightarrow \texttt{PP}_{(p \vee q) \wedge \neg r} \Rightarrow (\texttt{PP}_p \vee \texttt{PP}_q) \wedge \texttt{PP}_{\neg r}$ $\Rightarrow \texttt{PP}_{(p \wedge \neg r) \vee (q \wedge \neg r)} \Rightarrow \texttt{PP}_{p \wedge \neg r} \vee \texttt{PP}_{q \wedge \neg r} \Rightarrow$ $(\texttt{PP}_p \wedge \texttt{PP}_{\neg r}) \vee (\texttt{PP}_q \wedge \texttt{PP}_{\neg r})$

Table 3.4: An example rewriting of a complex predicate to expressions having conjunctions or disjunctions of probabilistic predicates.

can be some false negatives. A similar proof holds for a conjunction as well; an example is shown in Figure 3.8. Rows one and three of Figure 4.3 show the use of the disjunction and conjunction rewrite respectively. Such rewrites substantially expand the usefulness of PPs; because otherwise PPs would need to be trained not just for individual simple clauses but for all combinations of simple clauses. (4) The predicate can also be rewritten logically, leading to more possibilities for matching with PPs; for example, $(p \vee q) \wedge \neg r \Leftrightarrow (p \wedge \neg r) \vee (q \wedge \neg r)$ leads to the PP expression shown in the fourth and fifth row of the table. (5) The number of implied expressions over PPs that correspond to a complex predicate can be substantial; the table shows eight possibilities.

3.5.1 Complex predicate to expressions over PPs

The inputs are a complex predicate $\mathcal{P}$ and a set $\mathcal{S}$ of trained PPs, each of which corresponds to some simple clause, i.e., $\mathcal{S} = \{\texttt{PP}_p\}$. The goal is to obtain expressions $\mathcal{E}$ that are conjunctions or disjunctions of the PPs in $\mathcal{S}$ which are implied by $\mathcal{P}$, i.e., $\mathcal{P} \Rightarrow \mathcal{E}$.

If there are m PPs, i.e., $|\mathcal{S}| = m$, and n of the PPs directly match some clauses in a CNF representation of $\mathcal{P}$, then there are at least 2^n choices for $\mathcal{E}$. Since this problem has exponential-sized output, it will require exponential time.

We offer a greedy solution that is based on the intuition that expressions with many PPs will have higher execution cost; early filters that have a high cost must have a relatively larger data reduction in order to perform better than the baseline plan.

The input query predicate is sent to a wrangler which greedily improves matchability with available PPs. Examples of the wrangling rules include transforming a not-equal check

into disjunctions of equal checks (e.g., $t \neq 2 \Rightarrow t > 2 \vee t < 2$) or relaxing a comparison check (e.g., $t < 5 \Rightarrow t < 10$). We defer the details to Section 3.5.3.

Next, we convert predicates to expressions over PPs; examples of which are shown in Figure 4.3. For a predicate $\mathcal{P}$, let $\mathcal{P} \backslash p$ denote the remainder of the $\mathcal{P}$ after removing a simple clause p. With this notation, we use the rewrite rules below to generate expressions over PPs. All of the expressions in Figure 4.3 can be generated by repeated application of the first three rewrite rules.

$$\textbf{Rule R1: } p \wedge (\mathcal{P} \backslash p) \Rightarrow \texttt{PP}_p,$$

$$\textbf{Rule R2: } \texttt{PP}_{p \wedge q} \Rightarrow \texttt{PP}_p \wedge \texttt{PP}_q,$$

$$\textbf{Rule R3: } \texttt{PP}_{p \vee q} \Rightarrow \texttt{PP}_p \vee \texttt{PP}_q,$$

$$\textbf{Rule R4: } p \wedge (\mathcal{P} \backslash p) \Rightarrow \neg \texttt{PP}_{\neg p}.$$

The fourth rule above helps for predicates with high selectivity; however, it has narrower applicability. For simplicity, we defer discussion of this rule to later in this section. To construct implied logical expressions over PPs, we use the following greedy steps. (1) We limit the number of different PPs that are in any expression $\mathcal{E}$ to be at most a small configurable constant k. (2) We apply rules R2 and R3 only if the larger clause (e.g., $p \vee q$ or $p \wedge q$) does not have an available PP in $\mathcal{S}$ or if at least one of the simpler clauses has a PP that performs better (a smaller ratio of cost to data reduction $\frac{c}{r[1]}$ indicates better performance); intuitively, this prevents exploring possibilities that are unlikely to perform better.

For the example in Figure 4.3, suppose $k = 2$ and the set of available PPs, $\mathcal{S}$, in increasing order of $\frac{c}{r[1]}$ is $\{\texttt{PP}_{p \vee q}, \texttt{PP}_p, \texttt{PP}_{p \wedge \neg r}, \texttt{PP}_{q \wedge \neg r}, \texttt{PP}_q, \texttt{PP}_{\neg r}\}$. It is easy to see that our algorithm only outputs three possibilities; i.e., $\{\mathcal{E}\} = \{\texttt{PP}_{p \vee q}, \texttt{PP}_{\neg r}, \texttt{PP}_{p \wedge \neg r} \vee \texttt{PP}_{q \wedge \neg r}\}$. The other possibilities are pruned by our greedy checks.

3.5.2 Costing query plans with PP expressions

Given a set of expressions $\{\mathcal{E}\}$ that are conjunctions or disjunctions of PPs, the goal is to compute the lowest cost query plan which meets query's accuracy threshold. If some execution plan for $\mathcal{E}$ has a per-blob cost of c and reduction-vs-accuracy of $r[a]$, then the

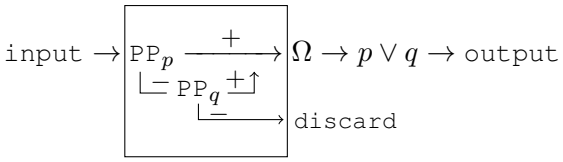


Figure 3.7: Injected query plan for the pattern $p \vee q \Rightarrow \texttt{PP}_p \vee \texttt{PP}_q$

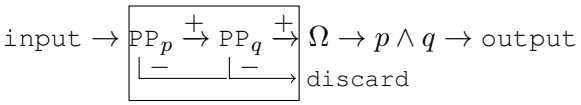


Figure 3.8: Injected query plan for the pattern $p \wedge q \Rightarrow \texttt{PP}_p \wedge \texttt{PP}_q$

query plan cost is $\propto c + (1 - r[a]) * u$, where u is the cost per blob of executing the original query. u and a are inputs to the algorithm but c and $r[a]$ have to be computed.

Since the order in which the PPs in $\mathcal{E}$ execute and how the accuracy budget is allocated among the individual PPs crucially affect the plan cost, we have three sub-problems. First, we have to explore different allocations of the query's accuracy budget to individual PPs. Next, we have to explore different orderings of PPs within a conjunction or disjunction; this process recurses for nested conjunctions or disjunctions. Finally, after fixing both the accuracy thresholds and the order of PPs, we have to compute the cost and reduction rate of the resulting plan. The first problem translates to a dynamic program which we omit for brevity. For the second part, recall that there are at most k PPs in any $\mathcal{E}$; if k is small, then all of the exponentially many orderings can be explored. When k is large, we use the following heuristic: consider ordering the PPs by the ratio of their intrinsic $\frac{c}{r[1]}$ and then consider all other orderings that are an edit-distance of at most 2 away from this greedy order. In practice, we found these to be the most useful orderings. The last part, computing cost and reduction rate given a fixed PP order and fixed accuracy thresholds, proceeds inductively as follows.

Base case: $\mathcal{E} = \texttt{PP}_p$. Here the cost and accuracy vs. the data reduction curve of $\mathcal{E}$ is the same as that of $\texttt{PP}_p$.

Conjunction: $\mathcal{E} = \mathcal{E}_1 \wedge \mathcal{E}_2$. Let the cost of the two logical expressions be c_1, c_2, and their

accuracy vs. data reduction curves be $r_1[a], r_2[a]$ respectively. Figure 3.8 shows an example conjunction. Suppose each PP has been given an accuracy threshold of a_1 and a_2. We make the simplifying assumption that the PPs are independent; a fix is described later in this section (PPs on dependent predicates). We now have:

$$\begin{aligned} a &= a_1 * a_2 \\ r[a] &= r_1[a_1] + r_2[a_2] - r_1[a_1] * r_2[a_2] \\ c[a] &= \min(c_1 + (1 - r_1[a_1]) * c_2, \quad c_2 + (1 - r_2[a_2]) * c_1) \end{aligned} \tag{3.9}$$

Disjunction: $\mathcal{E} = \mathcal{E}_1 \vee \mathcal{E}_2$. Figure 3.7 shows an example disjunction. With the same notation as in the case of conjunction and with similar assumptions, we have:

$$\begin{aligned} a &= a_1 + a_2 - a_1 * a_2 \\ r[a] &= r_1[a_1] * r_2[a_2] \\ c[a] &= \min(c_1 + r_1[a_1] * c_2, \quad c_2 + r_2[a_2] * c_1) \end{aligned} \tag{3.10}$$

Note the following intuitions for conjunction based on Eq. 3.9; analogous intuitions apply for disjunctions. (1) Accuracy reduces multiplicatively. (2) Data reduction ratio improves but the marginal improvement is less when many PPs are used and if the individual sub-expressions are already highly reductive. For example, if two expressions have a reduction rate of 0.1, the conjunction nearly doubles its data reduction to 0.19; however when each reduction rate is 0.8 the conjunction only increases to 0.96. (3) The cumulative cost is smaller when the sub-expression with the smaller $\frac{c}{r[a]}$ executes first. Our heuristic algorithm above is based on these intuitions.

3.5.3 Wrangling rules for complex predicates

Here we discuss how to wrangle predicate clauses so that they can be exactly matched onto PPs.

- Not-equals check ($f(C) \neq v$): If the range of $f(C)$ is finite and discrete, then $f(C) \neq v \Rightarrow \bigvee_{t \in Range(f(C)) \setminus v} f(C) = t$. For example, if vehicle `type` $\in \{SUV, truck, car\}$, then `type` $\neq SUV \Rightarrow$ `type` $= truck \vee$ `type` $= car$. This wrangling is useful if PPs exist only for the clauses on the left.

- Comparison: $f(C) > v \Rightarrow f(C) > t, \forall t \leq v$. The expression on the right relaxes the comparison and may be useful if a PP has been trained for some value t. Another rewrite is possible when $f(C)$ is finite and discrete, $f(C) > v \Rightarrow \bigvee_{t \in Range(f(C));\ t<v} f(C) = t$. Similar rewrites exist for $>, \leq, \geq$.

- Range-check ($v_1 \leq f(C) \leq v_2$) is a special case of comparison which is bounded on both sides and can be wrangled as above.

- *No-predicate.* If some columnset C in the query output has a finite and discrete range, even a query with no predicate can benefit from PPs because $\mathbf{1} \Leftrightarrow \bigvee_{t \in Range(C)} C = t$. For the above example of vehicle type, $\mathbf{1} \Leftrightarrow \texttt{type} = car \vee \texttt{type} = truck \vee \texttt{type} = SUV$.

3.5.4 PP seeding and pushdown rules

Table 3.5 describes our PP seeding and pushdown rules. We use a placeholder to seed a possible PP, denoted X_p, and attempt to push the placeholder down using these rules until it executes directly on the raw input; note that only predicates on a raw input can possibly be replaced with some combination of PPs. If not possible, the placeholder is simply omitted by the QO from the final plan. In the first rule, the expression on the right is less accurate, i.e., it has a given amount of false positives and false negatives. For each subsequent rule, the expressions have equivalent accuracy but the one on the right can be more performant. Some rules hold only under certain conditions. Pushdown below selection requires that the predicates p and q are independent. For the foreign-key join rule, let R and S be rowsets being equijoined on columnset $\mathcal{D}$ which is a primary key for S and a foreign key for R. This rule holds if the selection performed implicitly by the foreign-key join (recall: each row from R contributes at most one row to the join output) is independent of the predicate p. Finally, the pushdown rules for project change the columns in the predicate to invert the effect of the projection.

3.5.5 Negation rewrites and other details

Recall the fourth predicate rewrite rule mentioned in Section 3.5.1:

Seed PP for select	$\sigma_p(R) \tilde{\Leftrightarrow} \sigma_p(X_p(R))$
PP over select	$X_p(\sigma_q(R)) \overset{*}{\Leftrightarrow} \sigma_q(X_p(R))$ (additional conditions needed)
PP over foreign-key joins	$X_p(R \bowtie_{\mathcal{D}} S) \overset{*}{\Leftrightarrow} X_p(R) \bowtie_{\mathcal{D}} S$ if $p_c \subseteq R_c$ (additional conditions needed)
PP over col renaming project	$X_p(\pi_{\mathcal{C}_a \to \mathcal{C}_b}(R)) \overset{*}{\Leftrightarrow} \pi_{\mathcal{C}_a \to \mathcal{C}_b}(X_{p_{\mathcal{C}_a \to \mathcal{C}_b}}(R))$
PP over project creating new columns	$X_p(\pi_{f(\mathcal{D})=d}(R)) \overset{*}{\Leftrightarrow} \pi_{f(\mathcal{D})=d}(X_{p_{d \to f(\mathcal{D})}}(R))$

Table 3.5: Pushdown rules for probabilistic predicates. See Section 3.5.4.

Rule R4: $p \wedge (\mathcal{P} \backslash p) \Rightarrow \neg \texttt{PP}_{\neg p}$.

Figure 3.9 shows how such a PP can be used. This rule is quite powerful because predicates that have high selectivity will not yield useful PPs but their negations can achieve substantial data reductions. However, the rule has somewhat narrower applicability. As shown in Figure 3.9, blobs that fail the negative PP are output immediately; this requires that the schema of the query output match the schema of the query input; i.e., that the query be simply selecting a subset of blobs. Further, the rule composes in a complex way with the other rules because its application can lead to *false positives*.

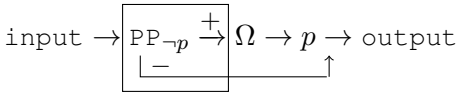


Figure 3.9: Injected query plan for the pattern $p \Rightarrow \neg \texttt{PP}_{\neg p}$

3.5.6 Hardness of the QO problem.

Optimal choice of PPs to train: Given a query set and a constraint on the overall training budget, consider the problem of choosing which PPs to train so as to obtain the best possible speed-up over that query set. Let $TrainCost_{\texttt{PP}_p}$ be the cost to train $\texttt{PP}_p$. Observe that the PP for predicate p will help any query q for which p is a necessary condition. Let $Queries_{\texttt{PP}_p}$ be the set of queries that will benefit if $\texttt{PP}_p$ were to be trained. For each query q in this set, let $r_p[a]^q$ denote the data reduction rate achieved from using $\texttt{PP}_p$ on query

q when ensuring accuracy is above a. We also know that a query can use more than one PPs. So, given a set of available PPs, $\mathcal{P}$, let $r_{\mathcal{P}}[a]^q$ be the best data reduction achieved by q through some combination of PPs in $\mathcal{P}$. Finally, let $\mathcal{Q}$ be the set of given queries, $\mathcal{S}$ be the set of all predicates in $\mathcal{Q}$ as well as all necessary conditions of those predicates and let T be the training budget. This problem becomes:

$$\max_{\mathcal{P}\subseteq\mathcal{S}} \left(\sum_{q\in\mathcal{Q}} r_{\mathcal{P}}[a]^q \right) s.t. \sum_{p\in\mathcal{P}} TrainCost_{\texttt{PP}_p} \leq T. \tag{3.11}$$

We show that this problem is NP-hard by reducing set cover to a simple version of the above problem.

Proof: Recall that given a set of elements $\{1, 2, \ldots, n\}$ (called the universe) and a collection S of m sets whose union equals the universe, the set cover problem is to identify the smallest sub-collection of S whose union equals the universe. The reduction proceeds by creating a query for each element in the universe and a predicate corresponding to each set in S with the understanding that training a PP for this predicate will help all the queries whose elements belong to that set. Hence, set the cost of training every PP to be the same and set the reduction rates to be unit; that is a query will receive the maximum benefit if it is covered by at least one PP. Note that the maximum achievable benefit to these queries will be obtained only when the union of the chosen sub-collection of sets equals the universe. To find the smallest possible sub-collection of S, we can vary the training budget from 1 to $|S| = m$ and find the smallest training budget at which the total benefit equals n.

Optimal use of available PPs: Given a set of available PPs, $\mathcal{P}$, consider the problem of finding a combination of PPs that offer the best data reduction for a query q given accuracy target a. Let c_p be the cost and $r_p[a]$ be the data reduction rate for a PP $p \in \mathcal{P}$. We can show that this problem is NP-hard by reducing the knapsack problem to a very simple version of the above problem.

Proof: For the purposes of this reduction, suppose that only conjunctions of the available PPs are allowed. Furthermore, the above problem has two parts: how to apportion the accuracy budget among the available PPs and how to order the chosen PPs. Let us ignore

the second portion (ordering PPs), and the reduction then proceeds as follows. Associate for each item a corresponding PP whose reduction rate is equal to the value of the item if the accuracy budget to this PP is at most log of the weight of the item and is zero otherwise. That is, the PP will offer reduction rate (value) only if given at least as much accuracy budget (weight). Set the log of the limit as the accuracy budget; sum of logs is product of individual accuracy budgets as per conjunction PP formula Eq. 3.9.

3.5.7 PPs on dependent predicates

In our experiments we have observed reasonable performance for queries with multiple PPs. However, if the PPs upon multiple predicate columns are dependent, the cost and reduction rate estimation and therefore the PP planning will be suboptimal. In such case, we apply a runtime fix. If we observe that the PP cost and reduction rate at runtime differ dramatically from their estimations, we flag such predicates as possibly dependent so that the QO will only use one PP (and not a combination of dependent PPs) in the future for that predicate. We also note that because practical accuracy targets are very close to 1, the independence assumption can be replaced with an upper bound that is fairly tight.

3.6 Case Studies

We discuss four case-studies used in our experimental evaluations: document analysis, image analysis, video activity recognition and comprehensive traffic surveillance. The input datasets have numbers of dimensions ranging from thousands (e.g., low-res images) to hundreds of thousands (e.g., bag-of-words representations of documents, which can be very sparse). Some predicates are correlated (e.g., hierarchical labels of document and activity types in videos). The selectivity of predicates also varies widely; some predicates have very low selectivity (e.g., 'has truck' in traffic video). We evaluate different machine learning queries on these datasets as described below.

Case1: Document analysis. We use the LSHTC [96] dataset which contains 2.4M documents from Wikipedia. Each document is represented as a bag of words with a frequency value for each of 244K words; this vector is sparse in practice. The LSHTC dataset classifies

the documents into 400K categories. The mapping between documents and categories is many-to-many; that is, a document can belong to many categories and vice versa. The dataset also offers a hierarchy over categories. We consider queries that retrieve documents having one or more categories.

Case2: Image labeling. The SUNAttribute [97] dataset contains 14K images of various scenes. The images are annotated with 802 binary attributes that describe the scene, such as 'is kitchen', 'is office', 'is clean', 'is empty' etc. We consider queries that retrieve images having one or more attributes. We also use the popular COCO [77] and ImageNet datasets [69] in a similar manner; i.e., queries retrieve images that contain one or more labels. COCO contains 120K images, each labeled with one or more of the 80 object classes. We use a subset of 110K images from ImageNet with the same 80 classes as in the COCO dataset to evaluate the cross-domain application of PPs, i.e., training PPs on COCO but testing on ImageNet.

Case3: Video activity recognition. We use the UCF101 video activity recognition dataset [112], which has 13K video clips with durations ranging from ten seconds to a few minutes. 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. Figure 3.10 demonstrates example video clips and labels within UCF101.

Case4: Comprehensive Traffic Surveillance Video Analytics. The queries thus far retrieve (different) portions of the inputs. Here, we consider the problem of answering comprehensive queries on traffic surveillance videos. Our datasets include hours of surveillance videos from the DETRAC [91] vehicle detection and tracking benchmark. We design a query set, TRAF20 (Section 3.7.2), upon these videos; the queries perform machine learning actions such as vehicle detection, color and type classification, traffic flow estimation (vehicle speed and flow) etc. While DETRAC already annotates vehicles by their types (sedan, SUV, truck, and van/bus), we manually annotate the vehicles in the video with their color (red, black, white, silver and other).

Figure 3.10: Example videos clips and labels in the UCF101 dataset.

3.7 Experiments II: Evaluating PPs

The experiments shown in this section have the following purposes:

Validating individual PPs. The first-order question we are interested in is how much speed-up can PPs offer to various machine learning inference queries over unstructured input blobs. We inject a PP into queries that have one simple predicate in Section 3.7.1. We also examine the suitability of PPs that are trained using different techniques. Our results will show that injecting PPs achieves speed-ups that are 3×–19× more than a state-of-the-art baseline [63] on different machine learning datasets.

Evaluation of our query processing system. Putting everything together, Section 3.7.2 evaluates using PPs on complex query predicates in Microsoft's Cosmos big-data cluster [30]. We demonstrate the costs to construct PPs on large datasets, how the QO chooses appropriate combinations of available PPs and the inference costs of applying PPs. These end-to-end experiments show that using probabilistic predicates can accelerate real-world machine learning inference by up to 12.5× under reasonable target accuracy and budget on training cost.

3.7.1 Micro-benchmarks on individual PPs

Dataset, predicates, UDFs and queries. To demonstrate that we can train PPs for a variety of datasets, we evaluate using PPs on queries that have one simple predicate. We use Cases 1-3 here; recall that the queries check for inputs that match a given category. To support these queries, we have built various feature extraction [39, 90] and classifier [15] UDFs. The classifier output, per category, is a binary column with value 1 if and only if the input blob matches that category, and the query predicates check the value of this column. For Case1, we randomly pick 140 categories, and use all categories for the other datasets. In all, this experiment has about a thousand queries and upwards of a thousand different UDFs.

Training PPs: For each query, we randomly take 60% of the entire dataset as the training set to construct the PP classifiers; the validation and testing set each takes 20% of the dataset. We also experiment with different training sizes.

Metrics used in our evaluations include the selectivity s_p of each predicate p, the accuracy a of the PP which is the fraction of output of the original query that is returned after using the PP, data reduction ratio $r_p[a]$ due to the PP at accuracy a. Note that accuracy is relative to the ground truth labels; the UDFs can often be imperfect. We also focus on the relative reduction ($= \frac{r_p[a]}{1-s_p}$), which is the actual number of input blobs that are dropped by the PP ($r_p[a]$) divided by the maximum possible number of input blobs that can be dropped by the PP ($1 - s_p$).

Can we train effective PPs? Building effective PPs depends on several factors; here, we consider the following key elements. (1) Do we have techniques that yield PPs with a good data reduction rate and high accuracy on a variety of datasets? (2) Are PPs trained on one dataset useful for other similar datasets?

Figure 3.11 shows whisker plots of the data reduction ratio ($r_p[a]$) from using PPs on different datasets. Each bar is a whisker plot; the lines are the min and max reduction across queries; the ends of the box are the 25th and 75th percentiles; the horizontal line in the box is the 50th percentile and x marks the average. A couple of points are worth noting. With a

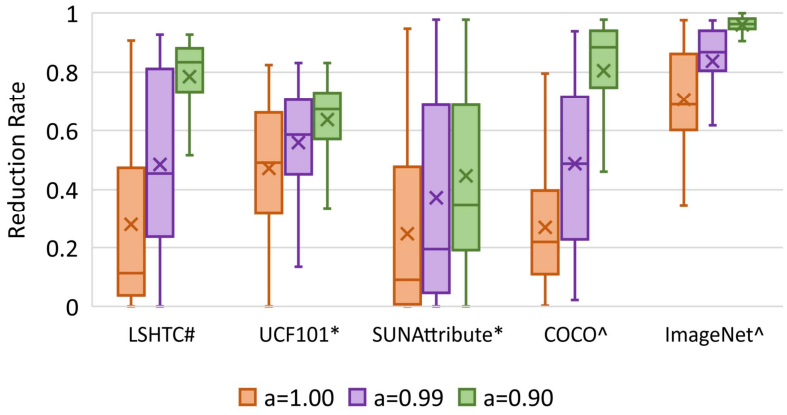


Figure 3.11: Whisker plots of the data reduction rates across various datasets. Each bar is a whisker plot; the lines are the min and max reduction across queries; the ends of the box are the 25th and 75th percentiles; the horizontal line in the box is the 50th percentile and x marks the average. Different PP techniques are used across datasets: # indicates PPs that use feature hashing + SVM, ∗ indicates PPs with PCA + KDE and ˆ indicates PPs with a DNN.

Dataset	PP	ts=30%	ts=40%	ts=50%
SUNattribute	PCA+KDE	.31/.92/6s	.32/.95/7s	.35/.96/8s
UCF101	PCA+KDE	.46/.92/10s	.51/.97/12s	.54/.98/14s
UCF101	RAW+SVM	.26/.87/1s	.39/.94/1s	.43/.96/2s
LSHTC	FH+SVM	.40/.95/1s	.45/.97/1s	.48/.98/1s
COCO	DNN*	-	-	.81/.99/110s

Table 3.6: For different PP methods on different datasets, with different training set sizes (ts=30%–50%) and an accuracy target of 0.99, the values shown in each table entry are the average data reduction rate/ the achieved accuracy/ and the training time per 1000 inputs. * denotes experiments using a GPU.

strict accuracy target $a = 1$, the PPs already achieve substantial data reduction. Half of the PPs on UCF101 filter more than 50% of the input. The data reduction varies across datasets due in part to the nature of the datasets, the queries and the predicates. Furthermore, a small trade-off in accuracy leads to much larger improvements in the reduction rates, e.g., a 1% decrease of a improves average data reduction by about 20% on COCO, ImageNet and LSHTC. Such small changes are often acceptable for aggregation queries (e.g., counting # of cars) or for queries where the desired object occurs in multiple frames (e.g., amber alert queries).

Figure 3.12: Demonstration of different PP outputs on COCO. The figure shows confidences f for 4 different PPs. See text for explanation.

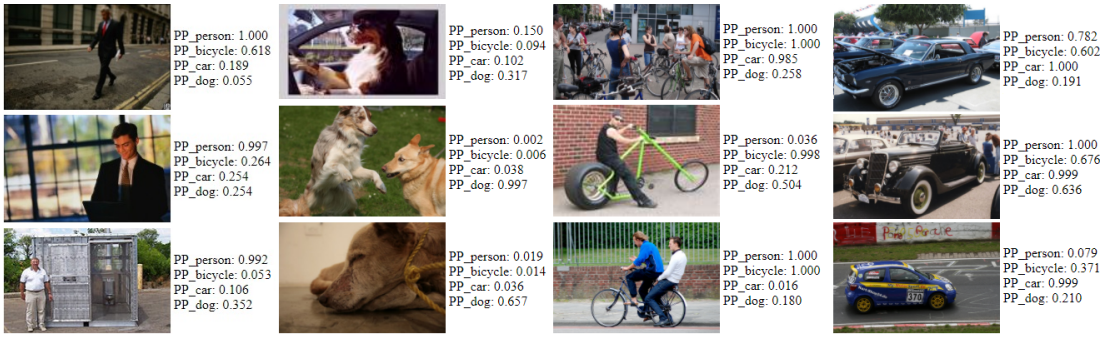


Figure 3.13: Demonstration of different PP outputs. The PPs are trained on COCO and applied on ImageNet. The figure shows confidences f for 4 different PPs.

How much training data is needed to construct PPs? Table 3.6 demonstrates the empirical data reduction rate and accuracy on the test sets with different training sizes; PCA, if used, is based on the same 1K rows. We note that more training data usually leads to better PP classifiers in terms of reduction rate and accuracy. The training cost grows sub-linearly with the training set size primarily because PCA (for dimension reduction) has a considerable fixed cost. However, since the PCA basis is specific to a dataset, it can be reused across PPs (we have not accounted for this above). We use feature hashing for the document analysis dataset (which is sparse); FH is extremely efficient, and combined with the linear SVM produces useful PPs. On the other hand, although with impressive data reduction rates, training cost for DNNs is relatively enormous.

Model selection. We also note that different PP training techniques, as noted in the caption of Figure 3.11, achieve the best data reduction on different datasets. The LSHTC document dataset is very sparse and the query categories are linearly separable over the features, so feature hashing + SVM leads to good PPs for Case1. Video activity recognition (UCF101) is not linearly separable but the different activities in this dataset are distinctive, so PCA + KDE suffices here. Table 3.7 shows that PPs using SVM achieve roughly 10% less data reduction. Image category labels are not linearly separable and the blobs are highly dimensional. For the relatively simple images in SUNAttribute, PCA + KDE leads to good PPs. However, for the more complex images in COCO and ImageNet (multiple objects in image etc., examples are in Figure 3.12) DNNs are needed to get useful PPs. Table 3.7 shows that SVM PPs on COCO and ImageNet achieve 20% to 40% lower data reduction. Compared with state-of-the-art DNNs (e.g., ResNet [54]), the DNN used for PPs here has 8 convolutional layers followed by a fully connected layer and is relatively very light-weight. Yet, the DNN PPs offer good data reduction. We believe that there is no silver bullet (i.e., best for all cases) PP approach. We use simple heuristics, e.g., do not use feature hashing for dense features, and use the least complex model that returns a good data reduction etc. Nevertheless, model selection is critical. Luckily, we also see that the behavior of PP approaches for a query and dataset can be estimated well by training on a small sampled subset of the corpus which reduces the cost of model selection. Another important aspect that reduces training and model selection cost is the ability to cross-train; that is, if we can use PPs trained on a dataset for other similar datasets. Table 3.7 shows in red the data reduction achieved when the DNN PPs trained on COCO are used on ImageNet. We see that cross-trained PPs are not as good as PPs trained on the same dataset, but they perform reasonably well especially at relaxed accuracy targets; we consider this to be a low-cost alternative to training DNN PPs on each dataset.

Demonstrating PPs. Figure 3.12 visually demonstrates how PPs work. We show for several example images, the confidence value computed for four different PPs: 'has person', 'has bicycle', 'has car' and 'has dog'. Recall that a PP would drop blobs (images in this case) whose confidence is below a threshold that is chosen based on desired accuracy; the

Dataset	Approach	Avg. data reduction $\bar{r}$ for accuracy a		
		$\bar{r}[1]$	$\bar{r}[0.99]$	$\bar{r}[0.9]$
UCF101	**PCA+KDE**	**0.47**	**0.56**	**0.64**
	PCA + SVM	0.35	0.45	0.54
	Raw + SVM	0.35	0.47	0.59
COCO	**DNN**	**0.28**	**0.50**	**0.83**
	SVM		0.31	
ImageNet	**DNN**	**0.71**	**0.84**	**0.96**
	SVM		0.39	
	DNN trained on COCO	0.25	0.49	0.82

Table 3.7: Comparing the data reduction achieved by PPs that use different techniques. The best technique appears to improve data reduction by 10% to 20% in absolute terms. Finally, cross-training, i.e., using PPs trained on a different albeit similar dataset appears promising.

Dataset	Approach	PP cost to ...		Optimality for a	
		Train (per 1K rows)	Test	$a = 1$	$a = 0.9$
UCF101	PCA+KDE	14s	3ms	0.55	0.77
LSHTC	FH + SVM	1s	1ms	0.29	0.87
COCO	DNN*	110s	10ms	0.28	0.83

Table 3.8: The latency to train and test PPs of different types as well as the optimality gap for different accuracy targets, a, which is $= \text{avg}_p\left(\frac{r_p[a]}{1-s_p}\right)$; i.e., the average over all predicates of the fraction of blobs that are discarded by a predicate which are discarded by the corresponding PP. * indicates w/ GPU.

more blobs that can be dropped the larger the data reduction. It is easy to see from these images that the gap between the confidence for appropriate labels and inappropriate labels is large. A PP trained for 'has person' with a confidence threshold of 0.9 will achieve a data reduction of 58% and an accuracy of 100%; this is the best possible data reduction because 5-out-of-12 pictures have a person in them. $\texttt{PP}_{has\_dog}$ with a 0.7 confidence threshold will achieve a data reduction of 83% and accuracy of 100%. These PPs use as input the raw pixels from images. Finally, Figure 3.13 shows details for PPs trained on COCO being applied on ImageNet.

Costs. Table 3.8 reports the time to train a PP per 1000 input blobs and the time to test on each input blob. As expected, we see that the KDE and SVM PPs can process several hundreds of blobs per second per thread. Using a GPU, the DNN PPs can process only about one hundred blobs per second. The training costs are also much larger for DNN PPs. All of these timing measurements were performed on a desktop running linux with an Intel

Target a	Method	LSHTC	SUNAttribute	UCF101
0.99	PP	0.51	0.43	0.56
	PCA + Joglekar et al. [63]	0.19	0.11	0.09
	Speed-up	2.7x	3.9x	6.2x
	Joglekar et al. [63]	0.16	0.05	0.03
	Speed-up	3.2x	8.6x	19x
Target a	Method	LSHTC	SUNAttribute	UCF101
0.90	PP	0.81	0.46	0.64
	PCA + Joglekar et al. [63]	0.36	0.15	0.14
	Speed-up	2.3x	3.1x	4.6x
	Joglekar et al. [63]	0.25	0.09	0.05
	Speed-up	3.2x	5.1x	12.8x

Table 3.9: Empirical reduction rates on three datasets with different target filtering accuracy.

i7-5930K processor, 16 GB of RAM and an Nvidia 1080Ti GPU.

Optimality. Table 3.8 also estimates an optimality gap of sorts; that is, what fraction of all the input blobs that can possibly be dropped by a PP, because the blobs will not satisfy the predicate, are actually dropped by that PP ($= \frac{r_p[a]}{1-s_p}$). The table shows values averaged over all predicates. By normalizing with predicate selectivity, this number tells us the room for improvement. We see that the PPs described in this work only achieve 28% to 55% of the optimal data reduction at $a = 1$, but at a relaxed accuracy target of $a = 0.9$ they are closer to optimal. Hence, we believe that more work on novel PP techniques is warranted especially at high accuracy targets, although it is apriori unclear that more data reduction can be achieved without also paying higher training and/or execution time costs.

Comparing with Joglekar et al. [63] We compare the PP classifiers with Joglekar et al [63], a system optimized for processing expensive predicates. This work leverages correlation between the input columns and the UDF outputs; consequently, they drop early based on the values of the input columns. We use their code and treat each dimension of our blobs as an input column. We compare with our PPs at different target accuracy settings ($a = .99/.90$) on 10 randomly picked queries from each of the three cases. Table 3.9 shows the comparison based on the same amount of training data. The baseline system can filter some of the sparse LSHTC inputs, since each dimension of a text input depicts a word, and intuitively correlations exist between words and the document label. However,

the baseline method does not work for dense machine learning blobs (e.g., images and videos). The baseline system improves marginally when it is offered the results of applying PCA over the raw data as input. The reason, we believe is that a dimension in such blobs hardly means anything, and the correlation is usually over some complex possibly non-linear combination of multiple dimensions. On the contrary, our PPs are more suited to handle machine learning blobs that have different data distributions.

3.7.2 Evaluating ML with PPs

TRAF-20 benchmark. The purpose of this section is to evaluate the end-to-end system speed-up from injecting probabilistic predicates. To the best of our knowledge, there is no off-the-shelf benchmark of queries with machine learning UDFs and complex predicates. Hence, we created a benchmark, TRAF-20, with 20 inference queries over datasets from Case 4 (described in Section3.6). Five predicate columns are generated by different machine learning UDFs, including vehicle color c and type t, speed s and direction (from i/to o). These UDFs are trained over annotated inputs. The queries mimic retrieval of vehicles that meet a specified predicate (e.g., an over-speed truck or an illegal turn). TRAF-20 has complex predicates including disjunctions and conjunctions of range, equality and inequality checks. Each query is equally likely to have between one and four predicate clauses. There are no nested predicates. Table 3.10 shows some example predicates from TRAF-20. Suppose the speed column is discretized to $0 - 80$ mph, there are roughly 100 different values that different UDF-generated columns can take. Hence, the space of potential query predicates is about 100^4. Training a filter for every possible predicate may not be feasible in practice.

Method and metrics. We mimic the use of PPs in an online setting. PPs are built upon the first 1GB of input data and UDF outputs; 80% of the blobs are used for training and 20% for validation. Overall, we have built 32 PPs, all of which are trained using SVMs, each corresponding to a single predicate clause. Our system executes query plans having some appropriate combination of these PPs.

We report the training costs as well as the overall system speed-up to process the subse-

#clauses	**Query ID**: Predicates (Type)
1	**Q1**: t=SUV (E), **Q2**: $s > 60$ (N), **Q4**: $c \neq white$ (I)
2	**Q7**: $s > 60$ & $s < 65$ (NR) , **Q8**: $t \in \{sedan, truck\}$ (ER)
3	**Q14**: i=$pt303$ & (o=$pt335$ — o=$pt306$) (ECD)
4	**Q20**: t=SUV & c=red & i=$pt335$ & o=$pt211$ (EC)

Table 3.10: TRAF-20 predicate examples. We use ptX to indicate traffic intersections in the dataset. E: equality check. I: inequality check. N: real numbers. R: range check. C: conjunction. D: disjunction.

quent data blobs. We measure query performance using two metrics: cluster processing time and query latency; these metrics are commonly used in recent data-parallel systems [18, 30]. Cluster processing time is the overall cluster resource usage and includes the cost of executing PPs, and query latency is the end-to-end user waiting time taking PP overhead into account. We also report the empirical reduction rate and the percentage of cluster processing time saved by applying PPs. Note that query latency is affected by a small number of outlier tasks and other scheduling artifacts; hence, it is much more variable than the cluster processing time of queries.

Comparisons. We compare our query processing system, end-to-end, with two baselines. (1) Optasia [82] is a relational data-parallel platform for large-scale vision/ machine learning which is built upon Microsoft Cosmos. It does not apply any early-filtering strategy. We refer to this baseline as NoP in our experiments. Our system uses a similar cost-based query optimizer to translate machine learning scripts into relational operators. (2) Deshpande et al. [43], building upon [21], optimally order multiple predicates such that cheap and data-reductive predicates execute earlier in the plan. They also output conditional query plans when predicate costs or selectivity vary (e.g., $temp. > 40°C$ has low selectivity at night). However, they still require predicate columns to be available on the inputs. We implement their scheme in our query processing system and refer to it as $SortP$.

End-to-end results. Figure 3.14 illustrates the speed-up in cluster processing time on 100 GBs of traffic surveillance videos relative to the baseline without PPs (NoP). The queries on the x-axes are ordered in increasing order of the speed-up. Table 3.11 reports the query execution latency for different schemes when processing different amounts of input. From Figure 3.14, we see that every scheme uses fewer resources than NoP [82];

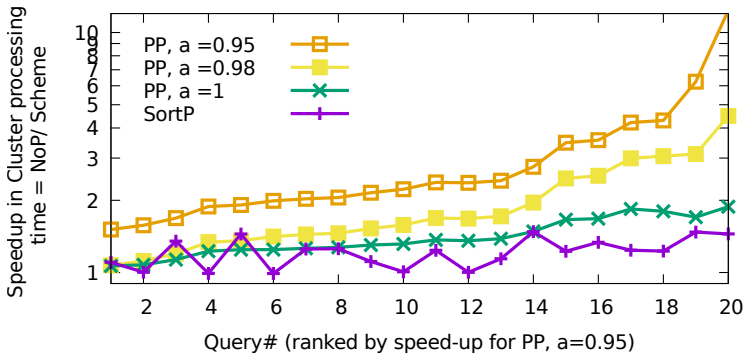


Figure 3.14: Evaluating TRAF20 query set on 100 GB online data. The figure shows the speed-up in cluster processing time relative to *NoP*, i.e., the total resources used to answer a query by *NoP* divided by that used by each scheme.

this is as expected, since in *NoP* all blobs go through all UDFs. *SortP* [43] has a small speed-up (average is 1.2×) because based on the ordering of predicates, when predicates have multiple clauses, blobs that do not pass predicates early in the plan can avoid being processed by the UDFs that generate columns for predicates that are later in the plan. Note however that while *SortP* lowers resource usage, it substantially increases the job latency because serializing the predicates (and UDFs) leads to longer critical paths. We see that our query processing system obtains large speed-ups in cluster processing time as well as query latency; especially when accuracy targets are relaxed. With an accuracy target of 1.0 (i.e., no false negatives), queries receive an average speed-up of 1.4×. For a relaxed accuracy target of 0.95, resource usage improvement ranges from 1.52× to 12.5× depending on the predicate and its selectivity, and the average query in TRAF-20 speeds up by 3.2×. Furthermore the average query latency is 60% of the latency in *NoP*. These improvements hold agnostic to data volume; i.e., larger input sizes receive larger reductions in latency as expected.

Details. Table 3.12 reports additional details on the costs of training and applying PPs on some typical queries in TRAF-20 as well as the average over all queries. We report here the time to train a PP on one thread. We note in practice that multiple threads can be used, model selection is done over sampled subsets, and PPs trained once are reused for other queries, all of which reduce the amortized training latency per query. We see that PP

	System	33 GB	67 GB	100 GB
Query latency	NoP [82]	0.37	0.69	**1**
	PP (a=0.95)	0.22	0.39	0.61

Table 3.11: Normalized average query latency (including PP training/inference overhead) on the TRAF-20 with different input sizes.

ID	PP cons.	#PPs	PP inf.	Sub.UDF	Selectivity	Reduction
4	27s	1	2ms	23ms	0.67	11%
8	68s	2	5ms	55ms	0.41	20%
20	155s	4	12ms	85ms	0.01	60%
Avg.	79s	2.5	6ms	52ms	0.20	59%

Table 3.12: Training and inference overhead for deploying PPs in online machine learning query processing. PP cons. is the PP construction time (normalized to single thread) on 15K rows. PP inf. is the PP inference time per row. Sub.UDF is the subsequent UDF cost per row. Selectivity, s_p, is the fraction of rows picked by query predicate. Reduction is $\frac{NoP-PP}{NoP}$ where each term is the cluster processing time to execute the query with the corresponding scheme. Avg. is average over all TRAF-20 queries.

PP Corpus	Query predicate	sel.	# plans	Est. r	Picked and Alter. plans. (Est. r)
32 PPs	$t \in \{SUV, van\}$	0.41	4	0.06–0.42	$PP_{SUV} \vee PP_{van}$ (0.42), $PP_{\neg sedan} \wedge PP_{\neg truck}$ (0.40)
	$s > 60 \wedge s < 65$	0.05	18	0.02–0.79	$PP_{s>60} \wedge PP_{s<65}$ (0.79), $PP_{s>60 \wedge s<70}$ (0.75), $PP_{s>60}$ (0.55)
full coverage	$s > 60 \wedge s < 65 \wedge$ $c = white \wedge t \in \{SUV, van\}$	0.01	216	0.08–0.77	$PP_{s>60} \wedge PP_{s<65} \wedge PP_{\neg sedan} \wedge PP_{\neg truck} \wedge PP_{white}$ (0.77) $PP_{s>50} \wedge PP_{s<70}$ (0.43), $PP_{s>60} \wedge PP_{s<65} \wedge PP_{\neg sedan}$ (0.52)
16 PPs half of above dropped at random%	$t \in \{SUV, van\}$	0.41	3	0.06–0.40	$PP_{\neg sedan} \wedge PP_{\neg truck}$ (0.40), $PP_{\neg sedan}$ (0.23)
	$s > 60 \wedge s < 65$	0.05	6	0.02–0.75	$PP_{s>60 \wedge s<70}$ (0.75), $PP_{s>60}$ (0.55)
	$s > 60 \wedge s < 65 \wedge$ $c = white \wedge t \in \{SUV, van\}$	0.01	88	0.08–0.76	$PP_{s>60} \wedge PP_{s<70} \wedge PP_{\neg sedan} \wedge PP_{\neg truck} \wedge PP_{white}$ (0.76)

Table 3.13: For some example queries, understanding the nature of feasible PP expressions.

training finishes in minutes. The overhead of applying PPs is generally small, compared with the subsequent machine learning UDFs. Our QO takes 80 to 100ms to translate the query predicates into PP expressions and to parametrize these expressions. Finally, the table also shows the selectivity of each query predicate and the achieved data reduction in cluster processing time (at $a = 0.95$). On average, we achieve a 59% reduction in cluster processing time, which is 74% of the theoretical maximum reduction of 80% (because the average query selectivity is 0.20). This is a sizable and promising speed-up for practical machine learning tasks. All of the algorithmic modules in our system are implemented in C/C++.

Query optimizer in action. To understand how the QO chooses PP combinations, we show more detail for a few queries in TRAF-20. Recall from the method description that there are five predicate columns and our QO uses a corpus of 32 PPs while the number of possible predicates is about 100^4. By construction this PP corpus completely covers the space of the predicates, i.e., any possible predicate will have at least one PP in the corpus that is a necessary condition. For example, the vehicle type column t can take four different values $SUV, van, truck, sedan$ and the corpus contains PPs for $t = SUV$, $t = van$, $t = truck$ and $t = sedan$. For numerical columns, we train PPs for $\leq$ and $\geq$ comparisons on value boundaries, e.g., PPs for speed are of the type $s \geq v_1 \in \{40, 50, 60\}$ or $s \leq v_2 \in \{65, 70\}$. For typical queries, Table 3.13 shows the query predicate, the number of available PP combinations that are feasible, the range of data reduction rates achievable by the feasible PPs, the combination of available PPs picked by the QO and the reduction rates for a few alternate plans. We see that for many queries, the QO has a meaningful choice to make, i.e., there are a lot of feasible PP combinations and picking one at random is unlikely to yield close to the best possible data reduction. The table also shows that the combination picked by the QO can have multiple PPs even when the predicate has only a single clause. Furthermore, the empirical observed reduction rates are close to the estimated reduction rate and so the QO choice is nearly optimal. A key point to emphasize is that because our QO prepares appropriate PP combinations, the training overhead is reduced from per-query (there are 100^4 possible predicates) to just 32 PPs, one per simple predicate clause. Table 3.13 also shows results for an even smaller PP corpus, wherein for each predicate column we have randomly dropped half of the PPs that are available on that column. We see that data reduction rates of the best possible PP combination decrease but not substantially. For example, for the predicate $t \in \{SUV, van\}$, data reduction rate drops from 0.42 to 0.40. While more investigation and empirical evidence is needed, our intuition is that a small corpus of PPs suffices to provide sizable data reductions even when the space of possible predicates is large (because a complex predicate will receive data reduction as long as some combination of PPs in the corpus is a necessary condition for the complex predicate).

Chapter 4

AN INTERACTIVE DEMONSTRATION SYSTEM FOR PROBABILISTIC PREDICATES

Demo system. The prototype used in Chapters 2 and 3 ran on a production cluster and extended a proprietary query processing platform [30]. To facilitate an interactive and small-scale demonstration, we will use a new prototype that extends Timely Dataflow [9]. The system supports queries written in Rust [7] 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 4.1 shows the implementation stack of this demo system; we have implemented the machine learning UDFs as listed in Table 2.1. To facilitate PPs with different classifiers as well as different applications, we have implemented tree-based kernel density estimator μKDE, as well as a light-weight C-based deep learning inference engine μDL, based on which we have trained deep neural networks for image labeling and object detection [103].

In our demonstration, we use the above system as a baseline. We will use several example queries and datasets (discussed later in Section4). The query plan for retrieving $(Red \vee Blue) \wedge SUV$ from traffic videos is shown in Figure 4.2. 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 3.3(a), the baseline query processing system employs a query plan that may be built using a cascades-style cost-based query optimizer. Figure 3.3(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; please refer to Section 3.7 for experiments that construct PPs online. Unlike the actual UDF classifiers, which take complex features as inputs (some example features are shown in Table 2.1), we use PPs that apply over raw input blobs, i.e., they take as input pixels from images and

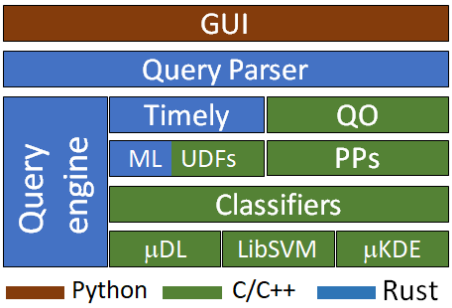


Figure 4.1: A cross-platform, data-parallel engine for DNN-enabled ML. User query: Standard SQL + UDF syntax. See [81]. Timely Dataflow: Rust implementation of Naiad [9]. Query engine: Join node, filter logic, columnar store for blobs etc. μDL: A light-weight DNN engine in C/C++. μKDE: A Kernel Density Estimator in C/C++.

videos and bag-of-words representations for documents. The modified platform, as shown in Figure 3.3(c), takes two additional inputs compared to the 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 demo 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.

- *Filtering accuracy and reduction rate:* Whereas conventional predicate pushdown produces deterministic results, how the classifiers used as probabilistic predicates will

function on previously unseen inputs is unknown. Filtering with PPs is parametrized over a filtering accuracy-reduction rate 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.

4.1 Demonstrations

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 filtering accuracy/reduction rate 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 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-resolution video clips) whereas others are sparsely distributed (e.g., Wikipedia documents in bag-of-words

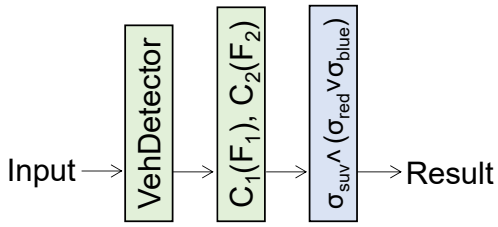


Figure 4.2: Query plan for retrieving $(Red \vee Blue) \wedge SUV$ from traffic videos, where `VehDetector` is a DNN-based vehicle detector, F_1, F_2 and C_1, C_2, are feature extractors and classifiers respectively, to extract the color and type for each detected vehicle.

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 [77] 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' $\wedge$ 'has dog' etc. We also generate similar queries over images and classes from the ImageNet [69] dataset. Figure 3.10 shows some example images from COCO.
- **Case2: Video activity recognition.** We use a popular video activity recognition dataset, UCF101 [112], 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.

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-

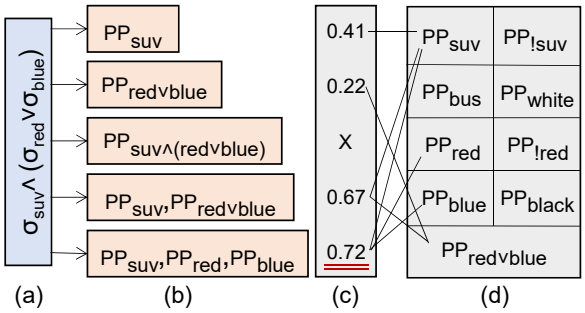


Figure 4.3: Demonstration of the query optimization process in our system. (a) Input complex predicate. (b) Candidate PP expressions that are implied by the original predicate. (c) Estimated data reduction rate for each PP expression while satisfying accuracy threshold: 'X' indicates no matching PP in corpus. We underline the expression that has the largest data filtering rate. (d) Corpus of available PPs.

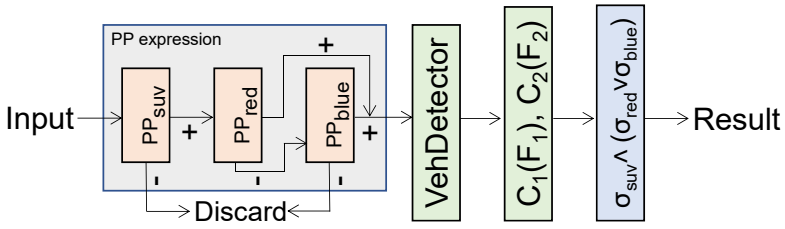


Figure 4.4: Modified query plan with PPs for retrieving $(Red \vee Blue) \wedge SUV$ from traffic videos.

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 4.3 (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 {accuracy, reduction rate} curves of individual PPs, which are generated during PP training, to compute the filtering accuracy and reduction rate 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 4.3 (b,c) illustrates this process 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; an example is shown in Figure 4.4.

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, and joins et cetera.

- **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 [91] 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 Section 3.7. 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.

The demonstration details (e.g., UI, example queries, results etc.) can be inspected at `http://yao.lu/PPdemo`. Here we show three example queries and visualizations. Figure 4.5 demonstrates the query to retrieve images with oranges and bananas. End-users need only to specify whether to use PPs and what is the target filtering accuracy. The queries remain unchanged if PPs are used. In this example, for the query with PPs, we set the target filtering accuracy as 100% and both PPs for orange and banana available. The speed chart demonstrates the efficiency of the two schemes with and without PPs. The query speed with PPs is about 3× faster than the version without PPs. Ideally, queries with/without

PPs should have the same number of hits, and the hits should be in the same places, which is shown in this case. Query plan with PPs can be inspected on the top right part of the UI. Figure 4.6 demonstrates the query to retrieve video clips with the "applying lipstick" action. In this example, the target accuracy is set to 100% and query with PPs is approximately 8x faster than query without PPs. Figure 4.7 shows the query to retrieve "red∧motorbike", and the query with PPs is again about 7x faster.

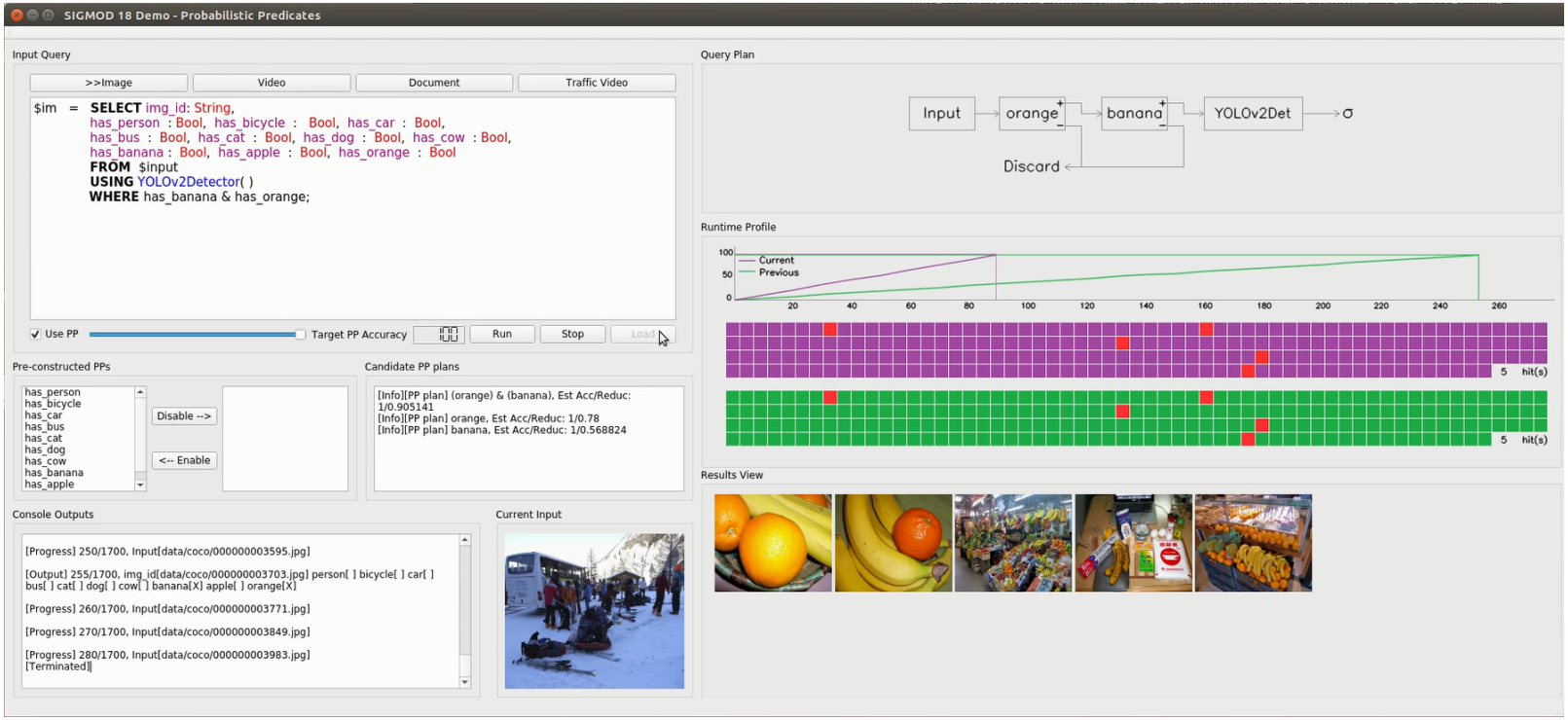


Figure 4.5: Demonstration of PPs. We show an example query to retrieve images with oranges and bananas. The visualization panel on the right part of the UI demonstrates (from top to bottom) query plan with PPs, runtime speed chart (the x-axis is seconds and the y-axis is percentage of completion), results chart (the input is compressed into a binary vector; a red cell indicates one or more hits the query predicate for that batch of input), and images retrieved. The current run with PPs (in purple) is compared with the previous run without PPs (in green).

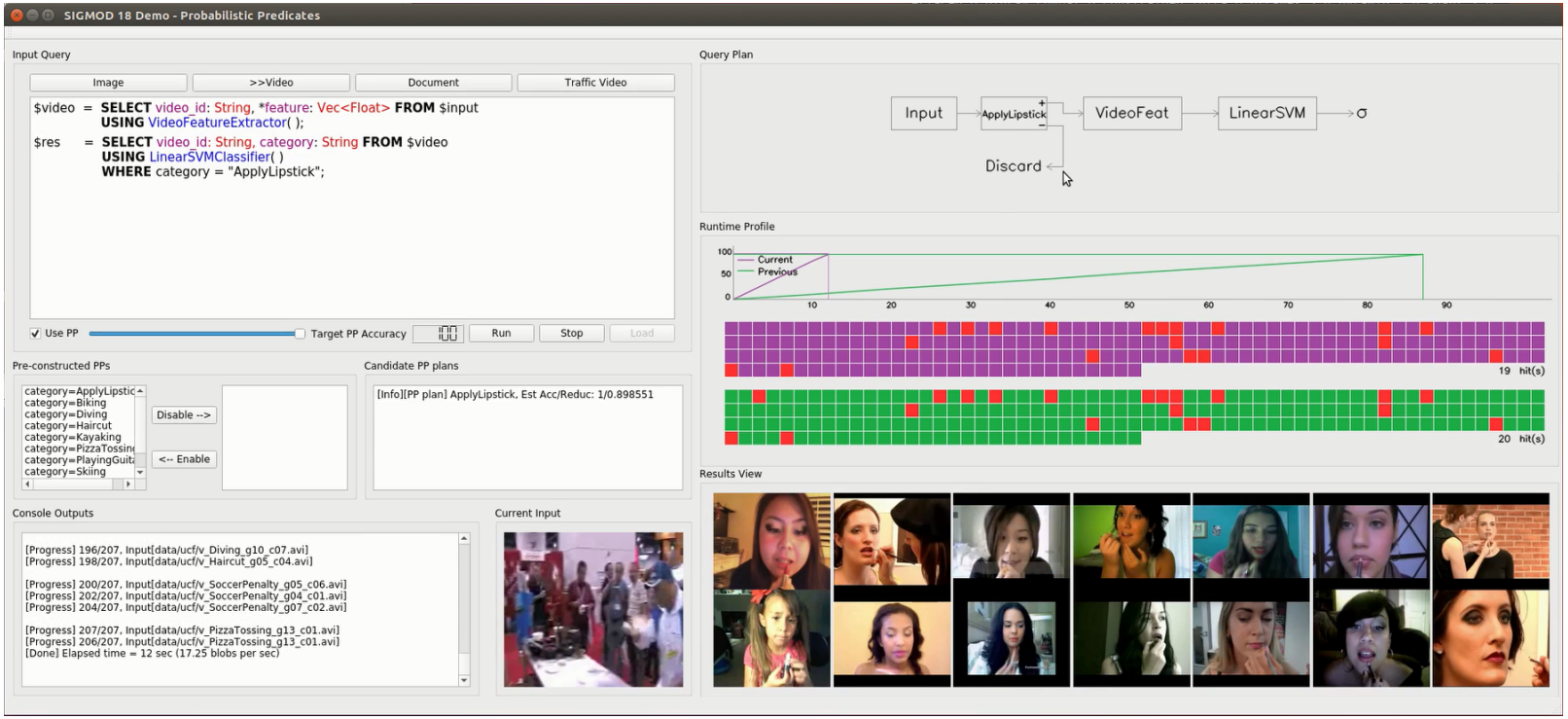


Figure 4.6: Demonstration of PPs. We show an example query to retrieve videos label as "applying lipstick". Please see Figure 4.5 for explanation.

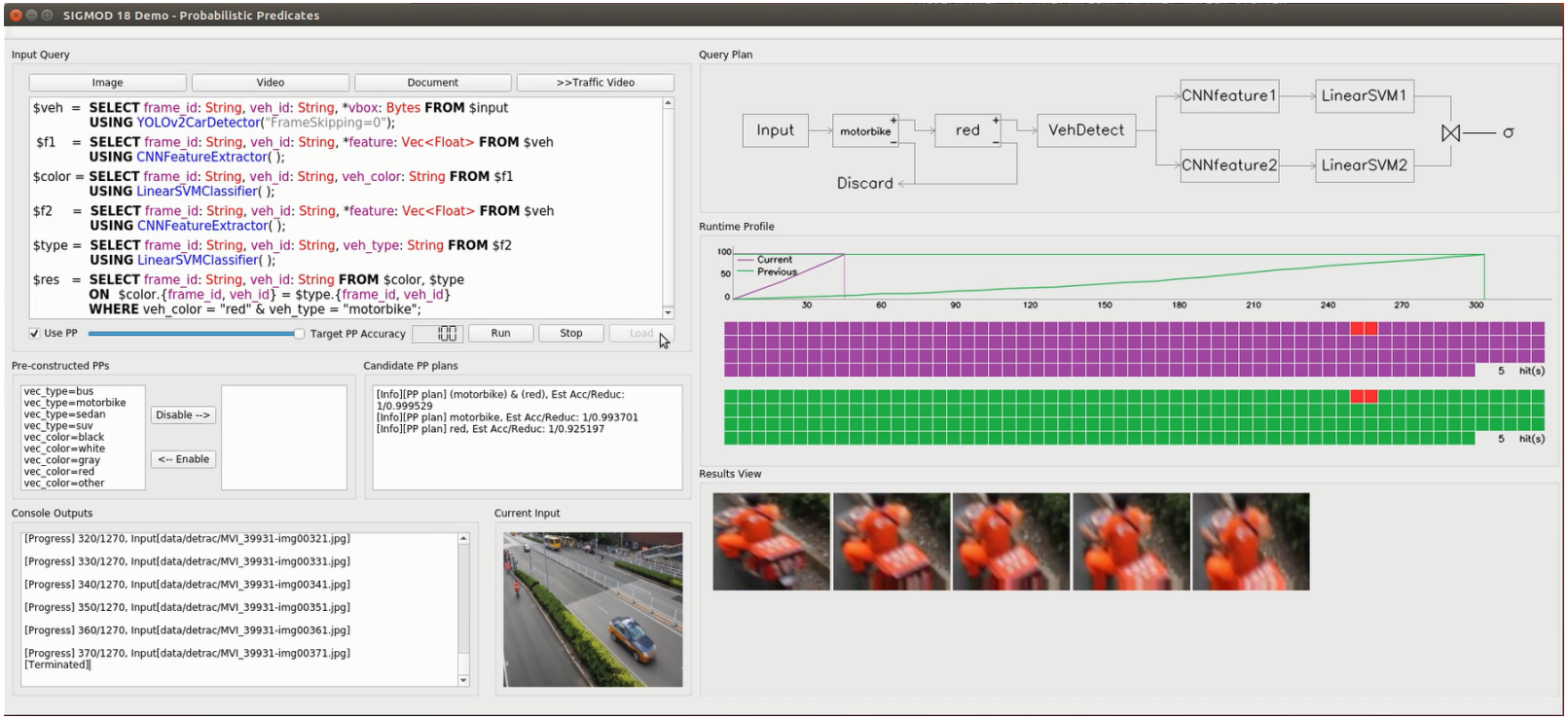


Figure 4.7: Demonstration of PPs. We show an example query to retrieve "red motorbike" from traffic videos. Please see Figure 4.5 for explanation.

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this report, we have shown big-data systems for machine learning model serving face several significant challenges; how we can execute machine learning algorithms efficiently on big-data clusters is one of the core problems. To address this, we have put forward Optasia, a dataflow system that maps machine learning operations as relational User-Defined Functions (UDFs); doing so enables us with the legacy query optimization tricks from the database literature such as auto-parallelism and query de-duplication. We have shown experiments in which resource usage for running traffic analytics queries on multiple traffic video cameras can be greatly reduced. Another problem we have identified is that, query predicates are stuck behind the UDFs, which prevents many in-depth optimizations such as predicate pushdown. We have built probabilistic predicates (PPs) that are binary, lightweight classifiers to drop input blobs that do not check the query predicate. To support complex machine learning inputs and query predicates, we have developed model selection schemes to choose optimal classification models and also a modified query optimizer to generalize binary PPs to complex query predicates at runtime.

Contributions. The contributions can be summarized as follows:

- A unified and customizable dataflow framework that computes optimal parallel query plans given any number of end-user queries for execution on a cluster.

- Fast and accurate implementation of several machine learning / computer vision modules that are needed in various model serving applications.

- A simple but broadly applicable design which incorporates a variety of PP construction techniques to accelerate online and batch machine learning inference queries.

- A query optimizer extension that matches complex predicates with available PPs and determines their parameters to meet the desired accuracy.

- Implementation and experiments on several real-world machine learning queries and datasets.

5.2 Future work

These are the initial steps towards bringing declarative dataflow engines to bear for scalable machine learning model serving. Much work remains in AI + systems; advances for each of the subproblems will be applicable to end applications and will lead to better user experiences as well as less cost.

A good big-data systems for AI spans a range of variety and presents the challenge of building large, integrated systems that combine the advances from different areas. I want to participate in building an ecosystem in which a wide topics of AI/conventional applications can happen. On a low level, it would easily to deploy existing algorithms and models to a large cluster. On a high level, it would automatically optimize according to the tasks. Central to this ecosystem would be understanding the semantics of the tasks as well as the data which can be too complex for human articulation. More specifically, the following aspects, are worth further investigation:

Big-data systems for declarative AI. As AI and machine learning applications become major customers for recent big-data ananlytics platforms, how to easily train, deploy, serve and scale-up the machine learning models with low cost is a critical area of research. A recent approach is to use a declarative front-end, where users declare queries with high-level semantics, and the query engine automatically figures out an optimal plan or model. There are plenty of open questions in this domain. Since machine learning algorithms can be complex, previous systems mostly treated them as black-boxes, but this prevents in-depth optimizations. To improve such a scheme and to think out of the box, we should find and solve questions using knowledge from both ML and systems domains. We could develop better systems to support machine learning model training in which the user only needs to specify the dataset. This is currently an active research field (AutoML), but most pioneering

projects have been focusing on the algorithmic end. For machine learning model inference or serving, we have taken an initial step with probabilistic predicates. More can be done in this direction; examples include extending probabilistic predicates to more query patterns, and associating query optimization with physical costs.

Runtime performance turning and optimization. Another interesting direction to explore is runtime system performance turning for AI/conventional tasks. For example, to apply probabilistic predicates in our previous work, we assume independence between the query predicates. This works well for most of the scenarios in our experiments but when strong correlation occurs, accuracy cannot be ensured; another example is on the system side when data skew happens, optimal parallelism cannot be guaranteed as well. Existing solutions include applying rule-based triggers to monitor data correlations or system hangups. Clearly, this can be done in a better way with recent advances in AI and machine learning. In fact, most of the performance loss comes from and can be addressed by the data itself. We could design algorithms and DNN models to learn the patterns for data skew and correlations, so that in runtime we could prevent these artifacts. Meanwhile, we could also develop reinforcement learning schemes for runtime performance optimization, which involves multiple discrete turning operations in sequence.

Chapter 6

RELATED WORK

6.1 Video analytics systems

To the best of our knowledge, Optasia uniquely shows how to execute sophisticated vision queries on top of a distributed dataflow system. Below, we review some relevant prior work.

Automatic analyses of videos, including that collected from highways and intersections, has a rich literature; the following are excellent surveys of the latest in this space [28, 78, 115, 127]. Key differences for Optasia are its use of simple camera-specific annotation and its use of state-of-the-art vision techniques such as exemplar SVMs.

6.2 Dataflow systems

There has been significant recent interest in distributed dataflow systems and programming models, e.g., Dryad [58], Map-Reduce [37, 42, 59], Hive [114], Pig [95], Sawzall [98] and Spark [18, 128]. At a high level, the recent work is characterized by a few key aspects: much larger scale as in clusters of tens of thousands of servers, higher degrees of parallelism, simpler fault-tolerance and consistency mechanisms, and stylistically different languages. The more recent frameworks adopt relational user-interfaces, i.e. SQL-like [18, 30, 114]. Most have a rule-based optimizer [18, 114]; except for SCOPE, which uses a Cascades-style [49] cost-based optimizer. The key distinction between the two is that the latter allows considering alternatives that need not be strictly better than the original plan; rather which alternative is better depends on properties of the code (e.g., the computational or memory cost of an operation) as well as data properties (e.g., the number of rows that pass through a filter).

Relative to these systems, Optasia offers a library of vision-specific modules built in a manner that lets users specify their queries in a SQL-like language. Further, Optasia tweaks the underlying query optimization logic in a few ways (e.g., incorporates costs and other

aspects of the vision modules) to achieve performant parallel execution plans for a vision query system.

6.3 Query optimization for expensive predicates

We reviewed some relevant works in Section3.2 and in Section3.7. Advanced indexing techniques [45] and data cubes [51] leverage the predictable nature of decision support queries and answer them directly from more compact representation. However, these approaches do not work well for machine learning inference on live streams such as audio and video, where the queries are not known a priori or are more complex.

There is a rich literature on optimizing queries with predicates: pushing predicates closer to input [116], optimal ordering of conjunctions [55], normalizing disjunctive and other complex predicates [68, 72] etc. When predicates rely on columns generated by user-defined operators, [94] shows that performance-optimal ordering of the UDFs and predicates is NP-hard. Our approach differs from these works because it uniquely adds new probabilistic predicates (PPs) rather than optimally ordering the existing predicates in the query. Approximate predicates [110] are applied to pre-filter unlikely inputs for expensive user-defined predicates; however they use the same relation as the query predicates and are not for blobs. One recent work observes that if existing column(s) in the data are correlated with user-defined predicates, then a function over those column(s) can be used to bypass the user-defined predicate [63]. While such functions over correlated columns are (simple) PPs, in our experience, such correlated columns rarely exist for ML queries. Instead, we train PPs using SVMs or kernel densities instead. For queries that apply predictive models on relational data, [33] derives implied predicates based on the details of the predictive model. Our approach differs in two ways. First, PPs are trained without any knowledge of the inference modules that are used in a query and hence PPs are more broadly usable whereas [33] applies only to decision trees and naive bayes classifiers and has a custom algorithm for each type of predictive model. Second, PPs also apply on non-relational datasets. NoScope [67] is a domain-specific model cascade for video data; it uses background subtractors, frame sampling and a simple DNN in front of the reference CNN and reports several orders of magnitude improvement on video processing rate. Our system differs from No-

Scope in supporting a wider range of queries (e.g., not just selections) and datasets (e.g., not only in the video domain). IDK cascade [123] is another model cascade to accelerate heavy classification models using cheaper ones. The key difference is that PPs are not functionally equivalent to the classifiers that they bypass and so efficient PPs are available for a broader class of queries and datasets.

BIBLIOGRAPHY

[1] Cisco global cloud index: Forecast and methodology. http://bit.ly/2iWNNeg.

[2] Facebook live. https://live.fb.com/.

[3] Free video trigger app. http://bit.ly/2ufJSSs.

[4] In more cities, a camera on every corner, park and sidewalk. http://n.pr/2tKQEg3.

[5] Open automatic license plate recognition library. https://github.com/openalpr/openalpr.

[6] Periscope. https://www.pscp.tv/.

[7] The rust programming language. http://www.rust-lang.org.

[8] Seattle department of transportation live traffic videos. http://web6.seattle.gov/travelers/.

[9] Timely dataflow. https://github.com/frankmcsherry/timely-dataflow.

[10] Vehicle counting based on blob detection. https://github.com/andrewssobral/simple_vehicle_counting.

[11] Youtube statistics 2017. http://bit.ly/2tFxBW5.

[12] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *OSDI*, 2016.

[13] Sameer Agarwal et al. Blinkdb: Queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[14] Sameer Agarwal, Srikanth Kandula, Nico Burno, Ming-Chuan Wu, Ion Stoica, and Jingren Zhou. Re-optimizing data parallel computing. In *NSDI*, 2012.

[15] Shun-ichi Amari and Si Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–789, 1999.

[16] G. Ananthanarayanan et al. Reining in the Outliers in MapReduce Clusters Using Mantri. In *OSDI*, 2010.

[17] Barak Ariel, William Farrar, and Alex Sutherland. The effect of police body-worn cameras on use of force and citizens complaints against the police: A randomized controlled trial. *J. of quantitative criminology*, 31(3):509–535, 2015.

[18] Michael Armbrust et al. Spark SQL: Relational Data Processing in Spark. In *SIGMOD*, 2015.

[19] Josh Attenberg, Kilian Weinberger, Anirban Dasgupta, Alex Smola, and Martin Zinkevich. Collaborative email-spam filtering with the hashing trick. In *6th Conf. on Email and Anti-Spam*, 2009.

[20] Brian Babcock, Surajit Chaudhuri, and Gautam Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, SIGMOD '03, 2003.

[21] Shivnath Babu, Rajeev Motwani, Kamesh Munagala, Itaru Nishizawa, and Jennifer Widom. Adaptive ordering of pipelined stream filters. In *ACM SIGMOD*, 2004.

[22] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[23] Yael Ben-Haim and Elad Tom-Tov. A streaming parallel decision tree algorithm. *J. Mach. Learn. Res.*, 11:849–872, 2010.

[24] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Comm. of the ACM*, 18(9):509–517, 1975.

[25] Derek Bradley and Gerhard Roth. Adaptive thresholding using the integral image. *Journal of graphics, gpu, and game tools*, 12(2):13–21, 2007.

[26] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[27] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.

[28] N. Buch, S.A. Velastin, and J. Orwell. A review of computer vision techniques for the analysis of urban traffic. *IEEE Transactions on Intelligent Transportation Systems*, 12(3):920–939, 2011.

[29] Mark W Burris. Application of variable tolls on congested toll road. *Journal of transportation engineering*, 129(4):354–361, 2003.

[30] Ronnie Chaiken et al. SCOPE: Easy and Efficient Parallel Processing of Massive Datasets. In *VLDB*, 2008.

[31] Craig Chambers et al. Flumejava: easy, efficient data-parallel pipelines. In *PLDI*, 2010.

[32] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. Optimized stratified sampling for approximate query processing. *ACM Trans. Database Syst.*, 32(2), June 2007.

[33] Surajit Chaudhuri, Vivek R. Narasayya, and Sunita Sarawagi. Efficient evaluation of queries with mining predicates. In *ICDE*, 2002.

[34] Yizong Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790–799, 1995.

[35] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In *NIPS*, 2006.

[36] Robert T Collins et al. A system for video surveillance and monitoring. *VSAM final report*, pages 1–68, 2000.

[37] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. Mapreduce online. In *NSDI*, pages 21–21, 2010.

[38] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines: And Other Kernel-based Learning Methods*. Cambridge University Press, New York, NY, USA, 2000.

[39] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[40] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40(2):5, 2008.

[41] James Davidson et al. The youtube video recommendation system. In *ACM conference on Recommender systems*, 2010.

[42] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[43] Amol Deshpande, Carlos Guestrin, Sam Madden, and Wei Hong. Exploiting correlated attributes in acquisitional query processing. In *ICDE*, 2005.

[44] Ahmed Elgammal, David Harwood, and Larry Davis. Non-parametric model for background subtraction. In *ECCV. 2000*, pages 751–767. Springer, 2000.

[45] Christos Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[46] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *JMLR*, 9:1871–1874, 2008.

[47] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[48] Gene H Golub and Charles F Van Loan. *Matrix computations*. 2012.

[49] Goetz Graefe. The cascades framework for query optimization. *IEEE Data Eng. Bull.*, 1995.

[50] Goetz Graefe and William J McKenna. The volcano optimizer generator: Extensibility and efficient search. In *Data Engineering, 1993. Proceedings. Ninth International Conference on*, pages 209–218. IEEE, 1993.

[51] Jim Gray et al. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *ICDE*, 1996.

[52] Robert M Haralick and Linda G Shapiro. Image segmentation techniques. *Computer vision, graphics, and image processing*, 29(1):100–132, 1985.

[53] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[55] Joseph M Hellerstein and Michael Stonebraker. Predicate migration: Optimizing queries with expensive predicates. *ACM SIGMOD*, 1993.

[56] Larry Huston, Rahul Sukthankar, Rajiv Wickremesinghe, M. Satyanarayanan, Gregory R. Ganger, Erik Riedel, and Anastassia Ailamaki. Diamond: A storage architecture for early discard in interactive search. In *FAST*, 2004.

[57] Nacim Ihaddadene and Chabane Djeraba. Real-time crowd motion analysis. In *ICPR*, 2008.

[58] Michael Isard et al. Dryad: Distributed Data-Parallel Programs From Sequential Building Blocks. In *EuroSys*, 2007.

[59] Dawei Jiang, Beng Ooi, Lei Shi, and Sai Wu. The performance of mapreduce: An in-depth study. *Proc. VLDB Endow.*, 3(1), 2010.

[60] Yu-Gang Jiang, Chong-Wah Ngo, and Jun Yang. Towards optimal bag-of-features for object categorization and semantic video retrieval. In *ACM Conf. on Image and video retrieval*, 2007.

[61] Thorsten Joachims. Training linear svms in linear time. In *SIGKDD*, 2006.

[62] Manas Joglekar, Hector Garcia-Molina, Aditya Parameswaran, and Christopher Re. Exploiting correlations for expensive predicate evaluation. *arXiv preprint arXiv:1411.3374*, 2014.

[63] Manas Joglekar, Hector Garcia-Molina, Aditya Parameswaran, and Christopher Re. Exploiting correlations for expensive predicate evaluation. In *SIGMOD*, 2015.

[64] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[65] Pakorn KaewTraKulPong and Richard Bowden. An improved adaptive background mixture model for real-time tracking with shadow detection. In *Video-based surveillance systems*, pages 135–144. Springer, 2002.

[66] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. Quickr: Lazily approximating complex ad-hoc queries in big-data clusters. In *SIGMOD*, 2016.

[67] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. NoScope: Optimizing neural network queries over video at scale. *VLDB*, 2017.

[68] A Kemper, G Moerkotte, K Peithner, and M Steinbrunn. Optimizing disjunctive queries with expensive predicates. In *SIGMOD*, 1994.

[69] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[70] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[71] Yann LeCun et al. Handwritten digit recognition with a back-propagation network. In *NIPS*, 1990.

[72] Alon Levy, Inderpal Mumick, and Yehoshua Sagiv. Query optimization by predicate move-around. In *VLDB*, 1994.

[73] Kang Li and Zhenyu Zhong. Fast statistical spam filter by approximate classifications. In *Joint Inter. Conf. on Measurement and Modeling of Computer Systems*, 2006.

[74] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *OSDI*, 2014.

[75] Shu Liang, Linda G Shapiro, and Ira Kemelmacher-Shlizerman. Head reconstruction from internet photos. In *ECCV*, 2016.

[76] Shengcai Liao, Yang Hu, Xiangyu Zhu, and Stan Z Li. Person re-identification by local maximal occurrence representation and metric learning. In *CVPR*, pages 2197–2206, 2015.

[77] Tsung-Yi Lin et al. Microsoft COCO: Common objects in context. In *ECCV*, 2014.

[78] Siyuan Liu, Jiansu Pu, Qiong Luo, Huamin Qu, L.M. Ni, and R Krishnan. Vait: A visual analytics system for metropolitan transportation. *IEEE T. on Intelligent Transportation Systems*, 2013.

[79] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[80] Yao Lu, Xue Bai, Linda Shapiro, and Jue Wang. Coherent parametric contours for interactive video object segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[81] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. Optasia: A relational platform for efficient large-scale video analytics. In *ACM Symposium on Cloud Computing (SoCC)*, 2016.

[82] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. Optasia: A relational platform for efficient large-scale video analytics. In *ACM SoCC*, 2016.

[83] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. Accelerating machine learning inference with probabilistic predicates. In *ACM International Conference on Management of Data (SIGMOD)*, 2018.

[84] Yao Lu, Srikanth Kandula, and Surajit Chaudhuri. Interactive demonstration of probabilistic predicates. In *ACM International Conference on Management of Data (SIGMOD) Demo*, 2018.

[85] Yao Lu and Linda Shapiro. Closing the loop for object proposals and edge detection. In *AAAI*, 2017.

[86] Yao Lu and Linda G Shapiro. Closing the loop for edge detection and object proposals. In *AAAI*, 2017.

[87] Yao Lu, Wei Zhang, Cheng Jin, and Xiangyang Xue. Learning attention map from images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[88] Yao Lu, Wei Zhang, Hong Lu, and Xiangyang Xue. Salient object detection using concavity context. In *IEEE International Conference on Computer Vision (ICCV)*, 2011.

[89] Yao Lu, Wei Zhang, Ke Zhang, and Xiangyang Xue. Semantic context learning with large-scale weakly-labeled image set. In *ACM international conference on Information and knowledge management (CIKM)*, 2012.

[90] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, 1981.

[91] Siwei Lyu, Ming-Ching Chang, Pierluigi Carcagni, Dmitriy Anisimov, Erik Bochinski, Fabio Galasso, Filiz Bunyak, Guang Han, Hao Ye, Hong Wang, et al. Ua-detrac 2017: Report of avss2017 & iwt4s challenge on advanced traffic monitoring. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–7. IEEE, 2017.

[92] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A Efros. Ensemble of exemplar-svms for object detection and beyond. In *ICCV*, pages 89–96, 2011.

[93] Luz Elena Y Mimbela and Lawrence A Klein. Summary of vehicle detection and surveillance technologies used in intelligent transportation systems. 2000.

[94] Thomas Neumann, Sven Helmer, and Guido Moerkotte. On the optimal ordering of maps and selections under factorization. In *ICDE*, 2005.

[95] C. Olston et al. Pig Latin: A Not-So-Foreign Language for Data Processing. In *SIGMOD*, 2008.

[96] Ioannis Partalas et al. LSHTC: A benchmark for large-scale text classification. *arXiv preprint arXiv:1503.08581*, 2015.

[97] Genevieve Patterson, Chen Xu, Hang Su, and James Hays. The sun attribute database: Beyond categories for deeper scene understanding. *IJCV*, 2014.

[98] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming*, 2003.

[99] Haonan Qiu, Yingbin Zheng, Hao Ye, Yao Lu, Feng Wang, and Liang He. Precise temporal action localization by evolving temporal proposals. *IEEE International Conference on Multimedia Retrieval (ICMR)*, 2018.

[100] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.

[101] Anand Rajaraman, Jeffrey D Ullman, Jeffrey David Ullman, and Jeffrey David Ullman. *Mining of massive datasets*. 2012.

[102] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, Inc., New York, NY, USA, 3 edition, 2003.

[103] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *CVPR*, 2016.

[104] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[105] Murray Rosenblatt et al. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.

[106] Sam Roweis. Em algorithms for pca and spca. *NIPS*, 1998.

[107] Prasan Roy, Sridhar Seshadri, S Sudarshan, and Siddhesh Bhobe. Efficient and extensible algorithms for multi query optimization. In *ACM SIGMOD Record*, 2000.

[108] Gerard Salton and Michael J McGill. Introduction to modern information retrieval. 1986.

[109] Jianbo Shi and Carlo Tomasi. Good features to track. In *CVPR*, pages 593–600, 1994.

[110] Narayanan Shivakumar, Hector Garcia-Molina, and Chandra Chekuri. Filtering with approximate predicates. In *VLDB*, 1998.

[111] Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[112] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *Preprint arXiv:1212.0402*, 2012.

[113] Abhinav Srivastava, Amlan Kundu, Shamik Sural, and Arun K Majumdar. Credit card fraud detection using hidden markov model. *IEEE Trans. on Dependable and Secure Computing*, 2008.

[114] Ashish Thusoo et al. Hive: A Warehousing Solution Over A Map-Reduce Framework. *Proc. VLDB Endow.*, 2009.

[115] Bin Tian, B.T. Morris, Ming Tang, Yuqiang Liu, Yanjie Yao, Chao Gou, Dayong Shen, and Shaohu Tang. Hierarchical and networked vehicle surveillance in its: A survey. *IEEE T. on Intelligent Transportation Systems*, 16(2):557–580, April 2015.

[116] Jeffrey Ullman. Principles of database and knowledge-base systems, 1989.

[117] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[118] H. Vceraraghavan, O. Masoud, and N. Papanikolopoulos. Vision-based monitoring of intersections. In *IEEE International Conference on Intelligent Transportation Systems*, pages 7–12, 2002.

[119] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.

[120] Paul Viola and Michael J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, May 2004.

[121] Li Wang, Yao Lu, Hong Wang, Yingbin Zheng, Hao Ye, and Xiangyang Xue. Evolving boxes for fast vehicle detection. In *IEEE International Conference on Multimedia and Expo (ICME)*, 2017.

[122] Li Wang, Weiyuan Shao, Yao Lu, Hao Ye, Jian Pu, and Yingbin Zheng. Crowd counting with density adaption networks. *arXiv preprint arXiv:1806.10040*, 2018.

[123] Xin Wang et al. IDK Cascades: Fast Deep Learning by Learning not to Overthink. *Preprint arXiv:1706.00885*, 2017.

[124] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *ICML*, 2009.

[125] Xindong Wu et al. Top 10 algorithms in data mining. *Knowledge and info. sys.*, 14(1):1–37, 2008.

[126] Xiangyang Xue, Wei Zhang, Jie Zhang, Bin Wu, Jianping Fan, and Yao Lu. Correlative multi-label multi-instance image annotation. In *ICCV*, 2011.

[127] Gu Yuan, Xin Zhang, Qingming Yao, and Kunfeng Wang. Hierarchical and modular surveillance systems in its. *IEEE Transactions on Intelligent Systems*.

[128] M. Zaharia et al. Spark: Cluster computing with working sets. Technical Report UCB/EECS-2010-53, 2010.

[129] Haoyu Zhang, Logan Stafman, Andrew Or, and Michael J Freedman. Slaq: quality-driven scheduling for distributed machine learning. In *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 2017.

[130] Wei Zhang, Yao Lu, Xiangyang Xue, and Jianping Fan. Automatic image annotation with weakly labeled dataset. In *ACM international conference on Multimedia*, 2011.

[131] Zoran Zivkovic. Improved adaptive gaussian mixture model for background subtraction. In *ICPR*, volume 2, pages 28–31, 2004.

[132] Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286, 2006.