21.03.19 12:55

Accelerating Machine Learning Inference with Probabilistic Predicates

SFU NatLangLab

Jetic Gū

CMPT 843

Overview

- Focus: Query Optimisation
- Architecture: Relational Data Platforms with UDFs
- Core Ideas:
  1. Observation: expensive UDFs can seriously influence performance
  2. Propose: probabilistic predicates using fast ML methods to filter data before feeding into the UDF layer

Queries with UDF

- UDF: user defined functions
    - Example: ML function for object detection
    - Input: Video data/Image data in the database
    - Output: Object type (cheese), Object Attributes (colour=Yellow)
- Using UDF in queries are more frequent nowadays

Review

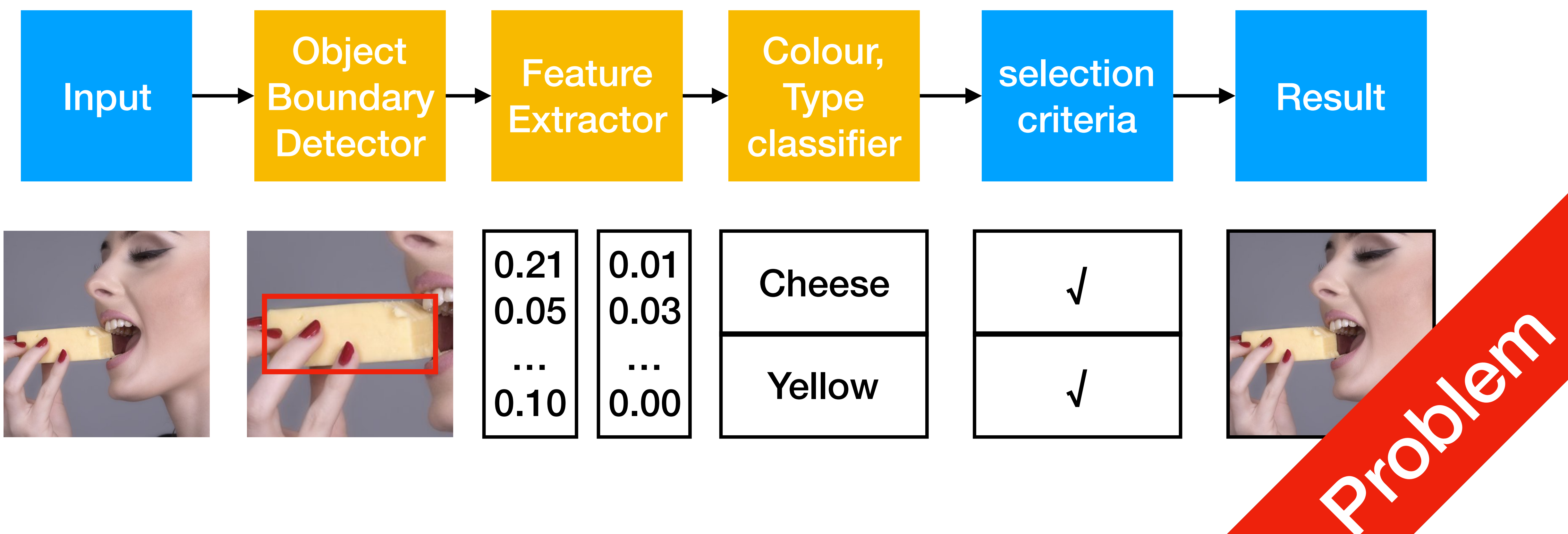
Queries with UDF

- Objective: get something within the DB
- Pipeline:

Example

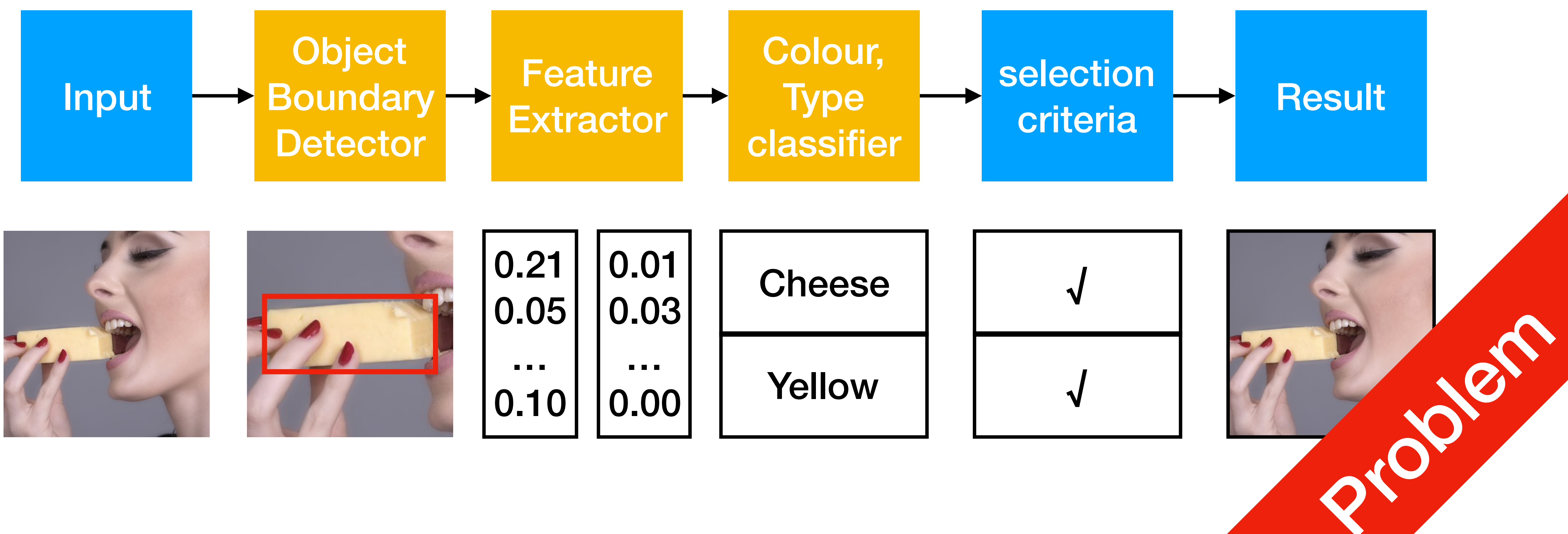
Queries with UDF

- UDFs are more general. Multiple feature extraction steps and inference stages will be needed in practice
- Objective: return pictures with Yellow Cheese
- Pipeline:



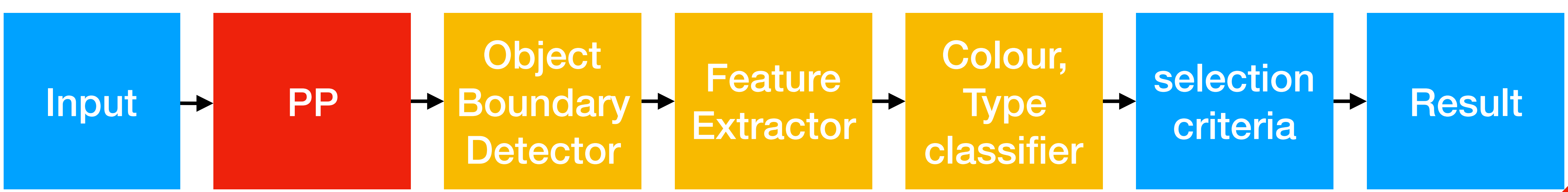
Queries with UDF

- Sloooooooooooow execution of UDFs
- Why? Because this entire chain of UDFs are run on ALL entries
- Traditional query optimisation techniques such as predicate pushdown are NOT useful here



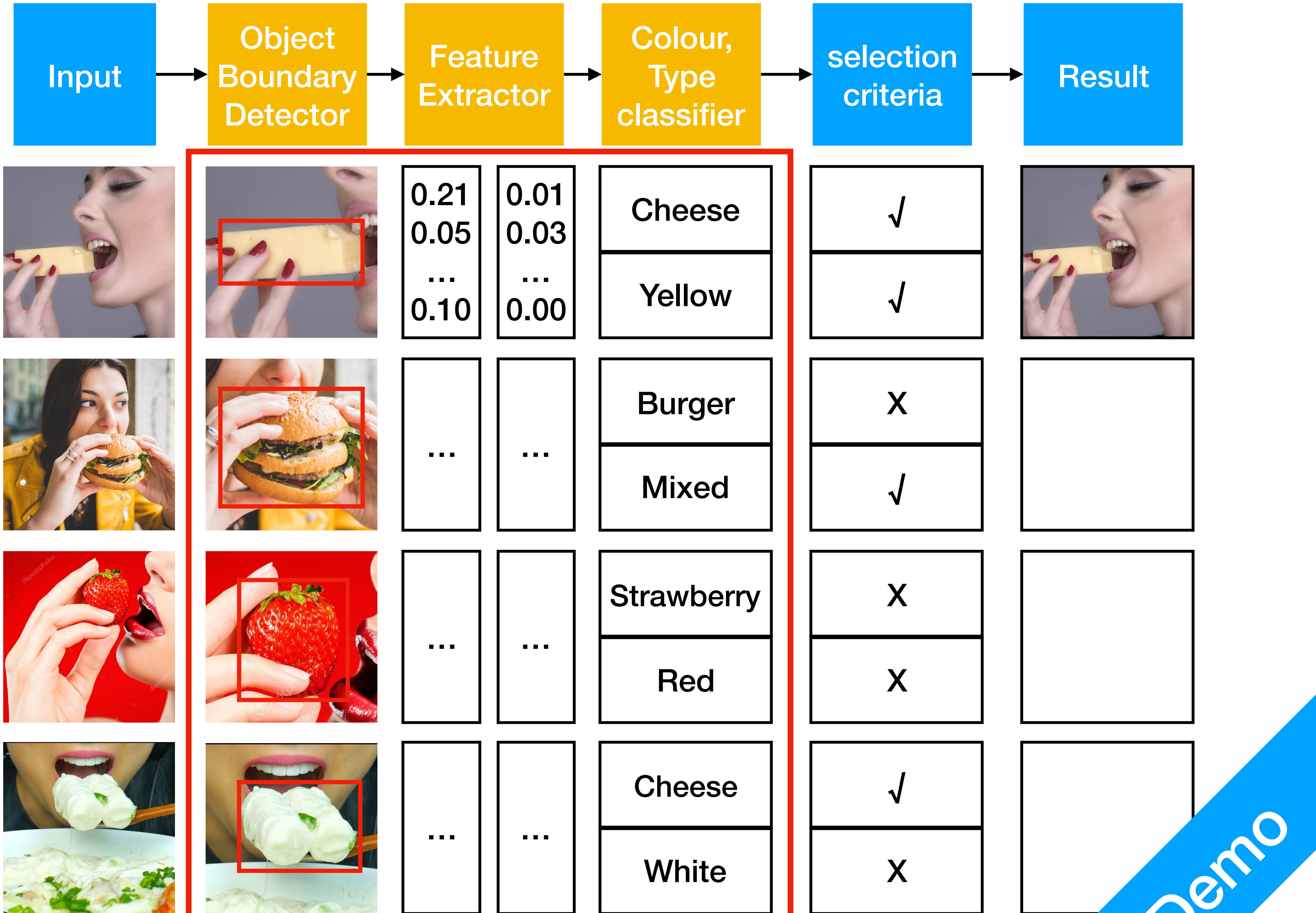
Probabilistic Predicates

- PPs are binary classifiers
- Get rid of highly unlikely entries before running through the UDFs.
- PPs are much more lightweight and error tolerant: high precision on negatives, low recall is OK.



Concept

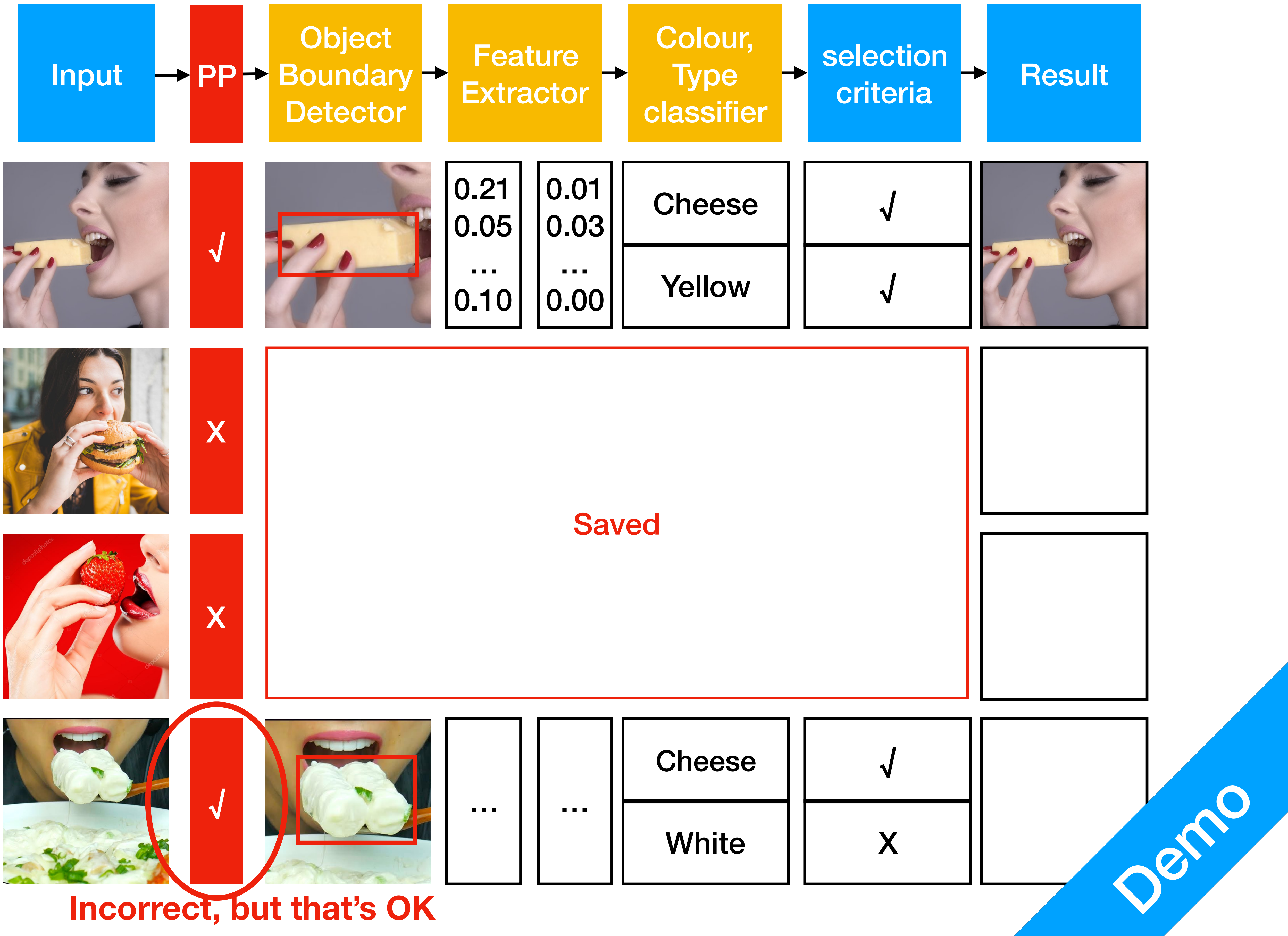
No Probabilistic Predicates



Query Optimiser cannot help, super slow execution chain

Demo

Probabilistic Predicates



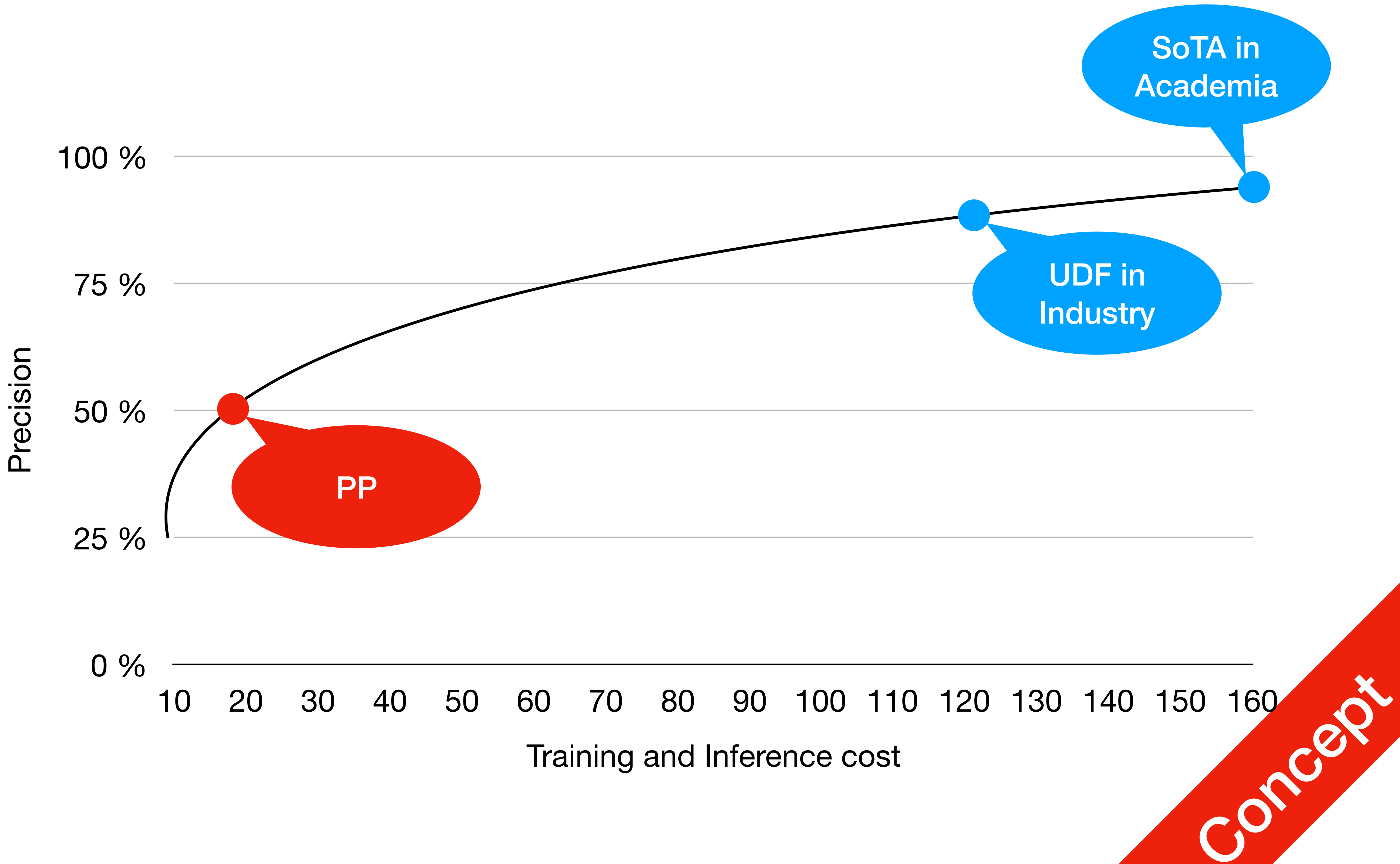
Comparison

- UDF
  - High execution cost
  - Long chain of execution
  - High precision
    - Give highly precise results
  - Training is costly

- PP
  - Very low execution cost
  - One time fast execution
  - High negative precision
    - Quickly eliminate highly unlikely entries
  - Cheap Training

Concept

Comparison


SoTA in
Academia
100 %
75 %
50 %
25 %
0 %
Precision
UDF in
Industry
PP
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
Training and Inference cost
Concept

Comparison

	System	Features	Classifiers/Regressors	Materialization Cost (sec)	Query predicate	Selectivity
Online	Ads recommendations [42]	Bag-of-words	Collaborative Regressor	$10^{-2} - 10^{-1}$	1 binary	1-in-hundreds
	Video recommendations [16]	Browse history	Bayesian Regressor	$10^{-1} - 10^{1}$	1 binary	1-in-thousands
	Credit card fraud [47]	Physical loc. etc.	Neural Network	$10^{-2} - 10^{-1}$	1 binary	1-in-thousands
Offline	Video tagging [24]	Keypoints	SVM w/ RBF kernel	$10^{-1} - 10^{1}$	n categorical	1-in-thousands
	Spam filtering [6]	Bag-of-word	Naive Bayes Classifier	$10^{-2} - 10^{-1}$	1 binary	1-in-several
	Image tagging [37, 55]	Keypoints	Collaborative Regressor	$10^{-1} - 10^{1}$	n categorical	1-in-thousands

- PP runs 10 times faster than UDF chain, filtering out 60% of entries
- That's 2x speed up!

Concept

Let's get technical

Technical

Different PPs

- Computational Models, use model selection techniques at inference time
    - linear support vector machine
    - kernel density estimator
    - deep neural networks
- Define and train one PP per simple clause $PP_p = D, m, r[a]$
    - Each PP has a reduction rate $r[a]$, where target accuracy is a.
        - Pre-sampling different a values, achieved using training data D. m indicates the computational model type.
    - Dimension reduction techniques are used to speed up

Technical

Reduction Rate and Target Accuracy

- The output of a PP is a probability distribution:

$$P(\textbf{rejected} \mid e, p) \in [0,1]$$

- Essentially, how likely the PP thinks this entry e will not satisfy the predicate p

- Activation threshold t: if $P(\textbf{reject}) \geq t$, filter it out

- By adjusting the activation threshold (how much we want to trust the PP to throw things out), we can get $r[a]$

Technical

Choosing Simple Clauses to Define PP

- Inferred from historical queries
- Define and train one PP per simple clause
    - Use query optimiser to assemble PPs

Technical

Assembly (Query Optimiser)

- Convert complex query expression to logical expressions using PPs (not strict it's just filtering here)

Complex predicate	Implied logical expr. over PPs
$(p \vee q) \wedge \neg r \wedge \boxed{\mathcal{P}_{\texttt{rem}}}$ **omitted, and that's OK**	$\Rightarrow p \vee q \Rightarrow \text{PP}_{p \vee q} \Rightarrow \text{PP}_p \vee \text{PP}_q$ $\Rightarrow \neg r \Rightarrow \text{PP}_{\neg r}$ $\Rightarrow \text{PP}_{(p \vee q) \wedge \neg r} \Rightarrow (\text{PP}_p \vee \text{PP}_q) \wedge \text{PP}_{\neg r}$ $\Rightarrow \text{PP}_{(p \wedge \neg r) \vee (q \wedge \neg r)} \Rightarrow \text{PP}_{p \wedge \neg r} \vee \text{PP}_{q \wedge \neg r} \Rightarrow$ $(PP_p \wedge PP_{\neg r}) \vee (PP_q \wedge PP_{\neg r})$

- Use query optimiser to
  - Determine for each PP, which CM m to use, within what target accuracy range a to reduce overall $r[a]$
  - Determine the execution order

Technical

Assembly (Query Optimiser)

$$(PP_p \wedge PP_{\neg r}) \vee (PP_q \wedge PP_{\neg r})$$

- 3 subproblems.

    1. different allocations of the query's accuracy budget to individual PPs

$$PP_p(r) = \begin{bmatrix} (a = 100\,\%\,, r = 0, m = \textbf{None}) \\ (a = 95\,\%\,, r = 15\,\%\,, m = \textbf{rule}_p) \\ (a = 90\,\%\,, r = 20\,\%\,, m = \textbf{SVM}_p) \\ (a = 85\,\%\,, r = 40\,\%\,, m = \textbf{NN}_p) \end{bmatrix} \qquad PP_{\neg r}(r) = \begin{bmatrix} (a = 100\,\%\,, r = 0, m = \textbf{None}) \\ (a = 95\,\%\,, r = 10\,\%\,, m = \textbf{rule}_p) \\ (a = 90\,\%\,, r = 15\,\%\,, m = \textbf{SVM}_p) \\ (a = 85\,\%\,, r = 50\,\%\,, m = \textbf{NN}_p) \end{bmatrix}$$

Given accuracy target, use dynamic programming to maximise reduction rate.

$$PP_{(PP_p \wedge PP_{\neg r}) \vee (PP_q \wedge PP_{\neg r})}(r)[a \geq A] = \,?$$

Demo

Assembly (Query Optimiser)

$$(PP_p \wedge PP_{\neg r}) \vee (PP_q \wedge PP_{\neg r})$$

- 3 subproblems.

    1. different allocations of the query's accuracy budget to individual PPs

    2. different orderings of PPs within a conjunction or disjunction

        to reduce the execution cost here

Demo

Assembly (Query Optimiser)

$$(PP_p \wedge PP_{\neg r}) \vee (PP_q \wedge PP_{\neg r})$$

- 3 subproblems.

1. different allocations of the query's accuracy budget to individual PPs
2. different orderings of PPs within a conjunction or disjunction
3. compute the cost and reduction rate of the resulting plan for query optimiser to work on

Demo

PP Experiments

Dataset	Approach	Avg. data reduction $\overline{r}$ for accuracy a		
		$\overline{r[1]}$	$\overline{r[0.99]}$	$\overline{r[0.9]}$
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

- Individual PP evaluation (recognising objects on images)

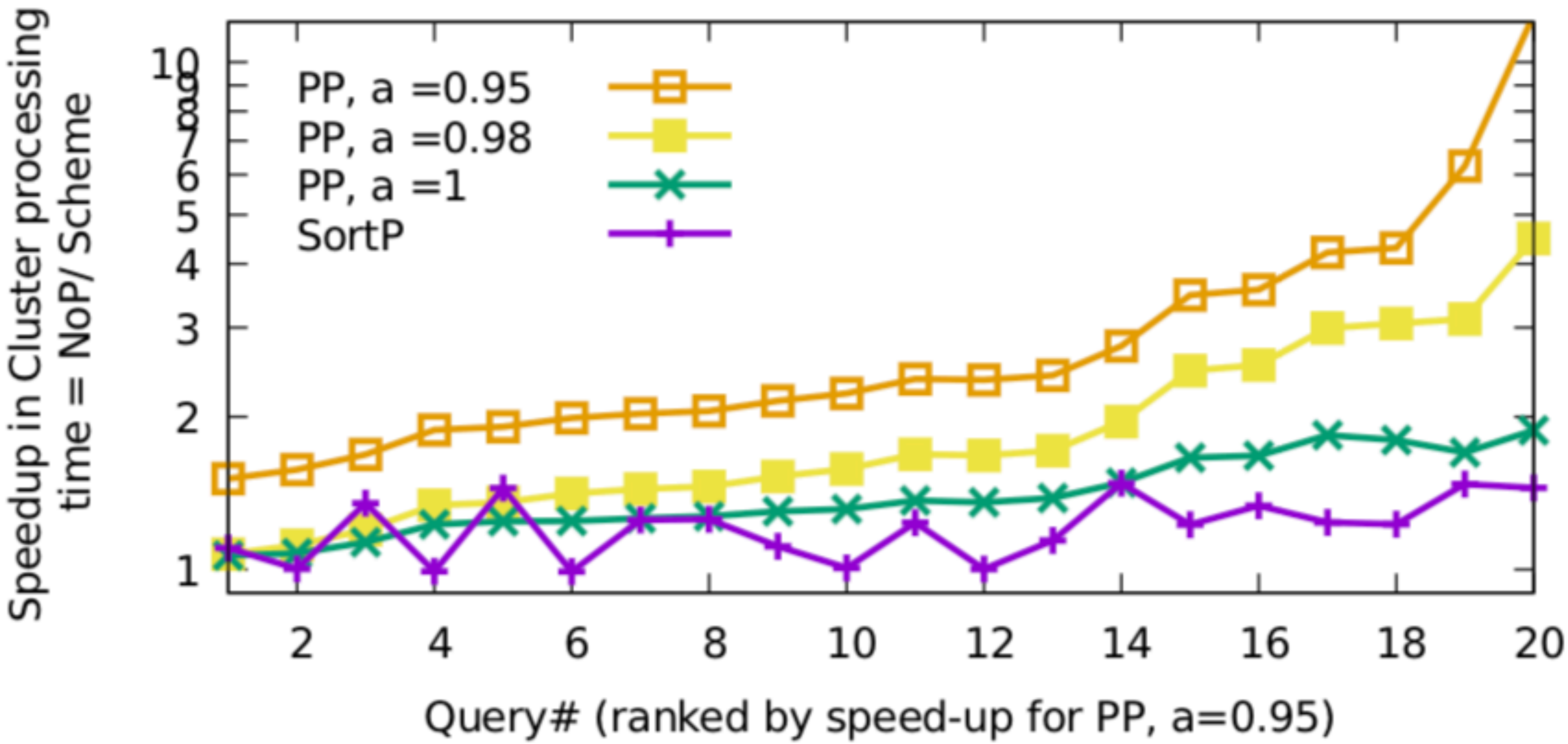
PP Experiments

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

- Individual PP evaluation (recognising objects on images)
- The latency to train and test PPs of different types

Experiments

End2End Experiments



- Evaluating TRAF20 query set on 100 GB online data

Experiments

End2End Experiments

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%

- Evaluating TRAF20 query set on 100 GB online data
- Training and inference overhead for deploying PPs in online machine learning query processing.
- ID indicates the individual query they used for this analysis

- Most query optimisation: optimally ordering the existing predicates in the query
- When predicates rely on columns generated by user- defined operators, shows that performance-optimal ordering of the UDFs and predicates is NP-hard.
- Approximate predicates use the same relation as the query predicates and are not for blobs
- PP
    - trained without any knowledge of the inference modules that are used in a query
    - can be applied on non-relational datasets

Review

Thank you.