Data Science with Linear Programming

Nantia Makrynioti, Nikolaos Vasiloglou, Emir Pasalic and Vasilis Vassalos
LogicBlox, Athens University of Economics and Business

DeLBP 2017, Melbourne, Australia

Problem Motivation

- ➢ Most of the data are still stored in relational databases.
- ➢ Typical data science loop:
  - ○ Prepare features inside database
  - ○ Export data as a denormalized data frame
  - ○ Apply machine learning algorithms
- ➢ Tedious process of exporting / importing data
- ➢ Loss of domain knowledge embedded in the relational representation

Our approach

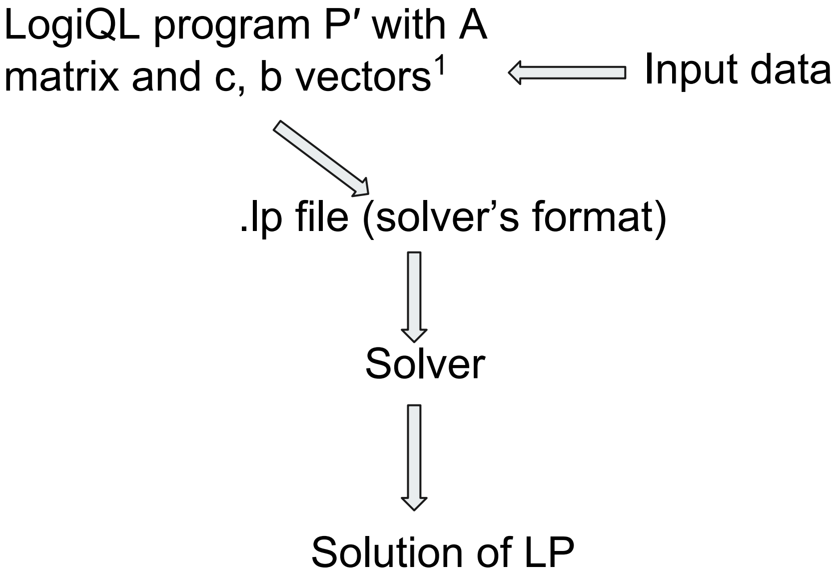
- Use of linear programming to model machine learning algorithms inside a relational database:
  - Define machine learning algorithms as Linear Programs (LP) in a declarative language
  - Automatic computation of the solution by the system
  - Seamless integration of constraints to express domain knowledge
  - Unification of data processing and machine learning tasks
- Implementation on the LogicBlox database

LogiQL and SolverBlox

- LogiQL: a declarative language derived from Datalog used in the LogicBlox database
- SolverBlox: a framework for expressing Linear and Mixed Integer Programs in LogiQL
  - Objective function and constraints expressed in LogiQL
  - Transformation of the LP in LogiQL to a matrix format consumed by an external solver, e.g. Gurobi
  - Solution of the LP stored back to the database and accessed via the typical LogicBlox commands / queries.

SolverBlox and Grounding

- Highly benefited by the LogiQL evaluation engine
- Incremental Maintenance when updating data inside database



[1] $$\begin{aligned} &maximize\ c^T x \\ &subject\ to\ Ax \leq b \end{aligned}$$

Machine Learning in SolverBlox

Linear Regression

- Objective function: Mean Absolute Error
- Retail domain: implementation of Linear Regression on the stock keeping unit (SKU) demand problem
  - Historical sales of a number of SKUs
  - Predict future demand for each SKU, at each store, on each day of the forecast horizon

```
observables(sku,str,day) -> sku(sku), store(str), day(day).

prediction[sku, str, day] = v -> sku(sku), store(str), day(day),
float(v).
```

EDB predicate: values imported to the database

IDB predicate: values defined by rules

```
lang:solver:variable(`sku_coeff).
lang:solver:variable(`brand_coeff).

sku_coeffF[sku]=v <- unique_skus(sku), sku_coeff[sku]=v.
brand_coeffF[sku]=v <- brand_coeff[br]=v, unique_skus(sku), brand[sku]=br.

sum_of_sku_features[sku]=v <- unique_skus(sku), sku_coeffF[sku]=v1, brand_coeffF[sku]=v2,
v=v1+v2.

prediction[sku, str, day] = v <- observables(sku,str,day), sum_of_sku_features[sku]=v.

//IDB predicate of error between prediction and actual value
error[sku, str, day] += prediction[sku, str, day] - total_sales[sku, str, day].

totalError[] += abserror[sku, str, day] <- observables(sku, str, day).
lang:solver:minimal(`totalError).

observables(sku, str, day), abserror[sku, str, day]=v1, error[sku, str, day]=v2 -> v1>=v2.
observables(sku, str, day),  abserror[sku, str, day]=v1, error[sku, str, day]=v2, w=0.0f-v2 ->
v1>=w.
```

```
lang:solver:variable(`sku_coeff).
lang:solver:variable(`brand_coeff).

sku_coeffF[sku]=v <- unique_skus(sku), sku_coeff[sku]=v.
brand_coeffF[sku]=v <- brand_coeff[br]=v, unique_skus(sku),
brand[sku]=br.

sum_of_sku_features[sku]=v <- unique_skus(sku),
sku_coeffF[sku]=v1, brand_coeffF[sku]=v2, v=v1+v2.

prediction[sku, str, day] = v <- observables(sku,str,day),
sum_of_sku_features[sku]=v.
```

LP variables

```
//IDB predicate of error between prediction and actual value
error[sku, str, day] += prediction[sku, str, day] -
total_sales[sku, str, day].

totalError[] += abserror[sku, str, day] <- observables(sku, str,
day).
lang:solver:minimal(`totalError).

observables(sku, str, day), abserror[sku, str, day]=v1, error[sku,
str, day]=v2 -> v1>=v2.
observables(sku, str, day),  abserror[sku, str, day]=v1,
error[sku, str, day]=v2, w=0.0f-v2 -> v1>=w.
```

Linear objective function

```
//IDB predicate of error between prediction and actual value
error[sku, str, day] += prediction[sku, str, day] -
total_sales[sku, str, day].

totalError[] += abserror[sku, str, day] <- observables(sku, str,
day).
lang:solver:minimal(`totalError).

observables(sku, str, day), abserror[sku, str, day]=v1, error[sku,
str, day]=v2 -> v1>=v2.
observables(sku, str, day),  abserror[sku, str, day]=v1,
error[sku, str, day]=v2, w=0.0f-v2 -> v1>=w.
```

Linear constraints

Factorization Machines

- Original algorithm:

$$y = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{m} < v_i, v_j > x_i x_j$$

$$< v_i, v_j >= \sum_{f=1}^{k} v_i, f \cdot v_j, f$$

$$w_0 \in R, w \in R^n, V \in R^{n \times k}$$

- Linear approximation:
  - Each interaction is placed to a bucket
  - Find coefficients for buckets

$$y = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{l=1}^{k} \sum_{i=1}^{n} \sum_{j=i+1}^{m} bc_l[i, j] \cdot x_i x_j$$

FM in SolverBlox

- Back to our retail problem:
  - Adding interactions between SKUs and months
  - Useful interaction for seasonal products

Determines buckets using a hash function

```
sku_monthOfYear_bucket[sku, moy] =  v <- observables(sku,_,day), monthOfYear[day]=moy,
sku_id[sku]=n1, month_id[moy]=n2, n=n1+n2, string:hash[n]=z, int:mod[z, 100]=v.

sku_monthOfYear_interaction[sku, day]=v <- observables(sku,_,day), monthOfYear[day]=moy,
sku_monthOfYear_bucket[sku, moy]=z3, bucket_coeff[z3]=v.
```

Interactive Data Science

Defining Models Step by Step

- Machine learning algorithms as LPs:
  - Gradually improving our models by adding constraints
- Integrating LPs to the database:
  - Easy filtering of training and test data by applying database processing operators

Defining a Forecasting Model - Step 1

➢ Starting by defining a Linear Regression model
  - Training and testing on 5 SKUs
  - Weighted Average Percent Error (WAPE):

$$WAPE = \frac{\sum |actual - forecast|}{\sum actual} \cdot 100$$

  - Bias:

$$Bias = \frac{\sum actual - forecast}{\sum actual} \cdot 100$$

Defining a Forecasting Model - Step 1

SKU id	3	6	8	9	26
WAPE on training	99.99	99.97	99.99	93.43	99.99
Bias on training	-99.99	-99.97	-99.99	-93.43	-99.99
WAPE on test	99.62	97.86	99.99	88.16	99.99
Bias on test	-80.68	-84.45	-99.99	-88.16	-99.99

Defining a Forecasting Model - Step 2

- Adding L1 regularization and a constraint forcing bias per SKU to zero:

SKU id	3	6	8	9	26
WAPE on training	67.73	62.77	95.7	36.26	102.6
Bias on training	0	0	0	0	0
WAPE on test	111.26	106.9	67.07	49.78	86.46
Bias on test	90.33	68.68	-36.54	-1.6	3.47

Defining a Forecasting Model - Step 3

- Turning bias constraint to a soft constraint and adding a domain specific constraint:
  - Sales predictions must be >=0

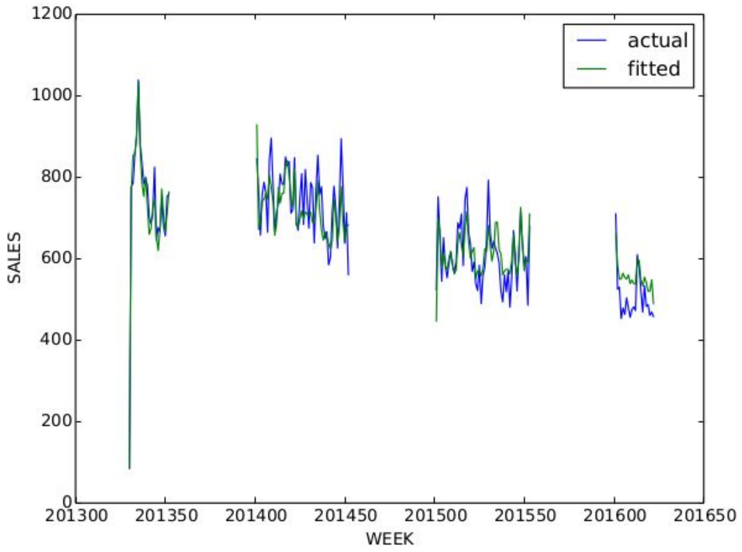
	WAPE on training		Bias on training		WAPE on test		Bias on test	
SKU id	Step 2	Step 3	Step 2	Step 3	Step 2	Step 3	Step 2	Step 3
3	67.73	63.74	0	0	111.26	75.33	90.33	5.99
6	62.77	59.99	0	0	106.9	73.29	68.68	0.97
9	36.26	34.44	0	0	49.78	50.89	-1.6	-0.5

Aggregated Forecasting

- So far we generated predictions at SKU, store, day level:
  - prediction[sku, str, day] = v <- observables(sku,str,day), sum_of_sku_features[sku]=v.
- By modeling ML algorithms as Linear Programs it’s very easy to predict sales at higher levels, e.g. at SKU, day level:
  - prediction_aggregated[sku, day]=v <- observables(sku,_,day), sum_of_sku_features[sku]=v.
- An effective technique when dealing with large datasets

Aggregated Forecasting - Data Fitting

- ➢ Factorization Machines model on 2033354 observations
- ➢ Generated 60346 predictions at subfamily - store - day level



Discussion

- ➢ Blending Machine Learning and relational databases accelerates and improves data science tasks
- ➢ As future work:
  - ○ Explore techniques to speed up grounding by harnessing functional dependencies and compressing the LP matrix
  - ○ Extension of SolverBlox to support more classes of convex optimization problems, such as Quadratic Programming

Thank you!
Questions?