Implementing SVM in an RDBMS: Improved Scalability and Usability

Joseph Yarmus, Boriana Milenova, Marcos M. Campos
Data Mining Technologies
Oracle
Overview

- Oracle RDBMS resources leveraged by data mining
- Oracle Data Mining (ODM) infrastructure
- Oracle SVM scalability solutions
- Oracle SVM ease of use solutions
Data Mining in RDBMS

• Integration of data storage and analytic processing
  – No data movement
  – Data integrity and security

• Leverage of decades of RDBMS technology
  – Query scalability
  – Transparent distributed processing
  – Resource management benefits
Database Infrastructure

- Model build implemented as a row source
  - Processing node that takes in flows of rows (records) and produces a flow of rows
- Model is a first class (native) database object
- Model score implemented as a SQL operator
- Extensibility framework
  - user-defined data types, operators, index types, optimizer functions, aggregate functions, pipelined parallel table functions
Oracle Database Architecture

Client

Oracle [pga]

SGA
  - Shared Pool
  - Buffer Cache
  - Large Pool

P000 [pga]

P001 [pga]

Pxxx [pga]

Database data files, temp space
SVM in the Database

• Oracle Data Mining (ODM)
  – Commercial SVM implementation in the database supporting classification, regression, and one-class
  – Product targets application developers and data mining practitioners
  – Focuses on ease of use and efficiency

• Challenges
  – Good scalability
    • large quantities of data, low memory requirements, fast response time
  – Good out-of-the-box accuracy
SVM and the ODM Infrastructure

- Data sources: views, tables, multi-relational data (e.g., star schema)
- Complex data types: sparse data, XML, text, images
- Data preparation
- Model build/test
- Model deployment
- Tools for measuring accuracy
- Model description
SVM Data Preparation Support

- Automatic data preparation
- Outlier treatment
- Missing value treatment
- Scaling
- Categorical to numeric attribute recoding
SVM Scalability Issues

- Build scalability
- Scoring scalability
  - Large model sizes (non-linear kernels) make online scoring infeasible and batch scoring impractical
 Implemented Scalability Techniques

- Popular build scalability techniques
  - Decomposition
  - Working set selection
  - Kernel caching
  - Shrinking
  - Sparse data encoding
  - Specialized linear model representation

However, these established techniques were not sufficient for our needs...
Additional Implemented Scalability Techniques

- Improved working set selection
- Stratified sampling mechanisms
- Small model generation
Working Set Selection

- Smooth (non-oscillating) iterative process
  - overlap across working sets
  - retain non-bounded support vectors
- Computationally efficient choice of violators
  - add large violators without sorting
Who to Retain?

/* Examine previous working set */
if (non-bounded sv < 50% of WS size) {
    retain all non-bounded sv;
    add other randomly selected up to 50% of WS size;
}
else
    randomly select non-bounded sv;
Who to Add?

create violator list;
if (violators < 50% of WS size)
  add all violators
else {
  /* Scan I - pick large violators */
  while (added violators < 50% of WS size)
    if (violation > avg_violation)
      add to WS;
  /* Scan II - pick other violators */
  while (added violators < 50% of WS size)
    add randomly selected violators to WS;
}
Stratified Sampling

• Classification and regression
  - Prevent rare target values (classification) or ranges (regression) from being excluded

• Single scan of the data
  - Target cardinality and distribution unknown at outset
  - Dynamically adjust the sampling rates based on the observed target statistics
Small Model Generation

- Linear kernel models
  - Represented as coefficients
- Non-linear kernel models (active learning)
  1. Construct a small initial model
  2. Select additional influential training records
  3. Retrain on the augmented training sample
  4. Exit when the maximum allowed model size is reached
Build Scalability Results

Graph showing the comparison between Linear and Gaussian models for time (seconds) vs. number of records. The graph compares LIBSVM and Oracle performance.
Scoring Scalability Results

Linear

- LIBSVM
- Oracle

Gaussian

Number of records

Time (sec.)

1M 2M 3M 4M

0 1800 3600 5400 7200 9000 10800

0 1800 3600 5400 7200 9000 10800

Number of records

Time (sec.)
SVM Scoring as a SQL Operator

- Easy integration
  - DML statements (select, insert, delete, update), subqueries, functional indexes, triggers
- Parallelism
- Small memory footprint
  - Model cached in shared memory
- Pipelined operation

```
SELECT id, PREDICTION(svm_model_1 USING *)
FROM user_data
WHERE PREDICTION_PROBABILITY(svm_model_2, 'target_val' USING *) > 0.5
```
CREATE INDEX anninc_idx on customers
  (PREDICTION (svmR_model USING *))

- ETL missing value imputation (data enrichment)

- Order-by elimination
  SELECT * from customers
  ORDER BY PREDICTION (svmR_model USING *);

- Index-driven filtering
  SELECT cust_name
  FROM customers
  WHERE PREDICTION (svmR_model USING *) > 150000;
Ease of Use Issues

- Complex methodology for novice users
  - Data preparation
  - Parameter (model) selection
- High computational cost for finding appropriate parameters
  - Accuracy
  - Fast build
On-the-Fly SVM Parameter Estimation

- Data-driven
- Low computational cost
- Ensure good generalization
# Classification Accuracy

<table>
<thead>
<tr>
<th>Category</th>
<th>Default Params</th>
<th>Default Params + Scaling</th>
<th>Grid Search + Xval</th>
<th>Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astroparticle</td>
<td>0.67</td>
<td>0.96</td>
<td>0.97</td>
<td>0.97</td>
</tr>
<tr>
<td>Bioinformatics</td>
<td>0.57</td>
<td>0.79</td>
<td>0.85</td>
<td>0.84</td>
</tr>
<tr>
<td>Vehicle</td>
<td>0.02</td>
<td>0.12</td>
<td>0.88</td>
<td>0.71</td>
</tr>
</tbody>
</table>

# Regression Accuracy

<table>
<thead>
<tr>
<th></th>
<th>Grid search RMSE</th>
<th>Oracle RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boston housing</td>
<td>6.26</td>
<td>6.57</td>
</tr>
<tr>
<td>Computer activity</td>
<td>0.33</td>
<td>0.35</td>
</tr>
<tr>
<td>Pumadyn</td>
<td>0.02</td>
<td>0.02</td>
</tr>
</tbody>
</table>
Classification Capacity Estimate

Goal: Allocate sufficient capacity to separate typical examples

1. Pick $m$ random examples per class
2. Compute $f_i$ assuming $\alpha = C$
   \[
   f_i = \sum_{j=1}^{2m} C y_j K(x_j, x_i)
   \]
3. Exclude noise (incorrect sign)
4. Scale $C$, $f_i = \pm 1$ (non bounded sv)
   \[
   C^{(i)} = \text{sign}(f_i) / \sum_{j=1}^{2m} y_j K(x_j, x_i)
   \]
5. Order ascending
6. Select 90th percentile
Classification Standard Deviation Estimate

Goal: Estimate distance between classes

1. Pick random pairs from opposite classes
2. Measure distances
3. Order descending
4. Select 90\(^{th}\) percentile
Goal: estimate noise by fitting preliminary models

1. Pick small training and held-aside sets
2. Train SVM model with $\varepsilon = 0.01 \cdot \mu_y$
3. Compute residuals on held-aside data
4. Update $\varepsilon^{\text{new}} = \left( \varepsilon^{\text{old}} + \mu_r \right) / 2$
5. Retrain
Conclusions

• Oracle’s SVM implementation
  – Handles large quantities of data
  – Has a small memory footprint
  – Has fast response time
  – Allows database users with little data mining expertise to achieve reasonable out-of-the-box results

Corroborated by independent evaluations by the University of Rhode Island and the University of Genoa
Final Note

- SVM is available in Oracle 10g database
  - Implementation details described here refer to Oracle 10g Release 2
  - JAVA (J2EE) and PL/SQL APIs
  - Oracle Data Miner GUI

- Oracle’s SVM has been integrated by ISVs
  - SPSS (Clementine)
  - InforSense KDE Oracle Edition