Incrementally Maintaining Classification using an RDBMS

Presented by: Noah Golmant

October 3, 2016

HAZY

“An end-to-end system for imprecision management”

Goals:

- Integrate classification models into run-time operation with RDBMS
- Incorporate new training examples in a real-time, dynamic environment

How?

- Model-based Views
- Incremental Maintenance

Goals:

- **Integrate classification models into run-time operation with RDBMS**
- Incorporate new training examples in a real-time, dynamic environment

How?

- **Model-based Views**
- Incremental Maintenance

Model-based Views

- Expose statistical computations through relational views
- Standard SQL semantics:
  - Queries for updates, inserts, and deletes
  - Triggers to propagate updates

$$
\begin{array}{l}
V(\mathrm{In}, \mathrm{T}) \\
\mathrm{In}(id, f) \\
\mathrm{T}(id, l) \mid l \in \{-1, +1\} \\
\text{Model: } (\vec{w}, b)
\end{array}
\quad \Longrightarrow \quad
V = \{(id, c) \mid (id, f) \in \mathrm{In} \text{ and } c = \mathrm{sign}(\vec{w} \cdot f - b)\}
$$

Model-based Views

```
CREATE CLASSIFICATION VIEW
  Labeled_Papers KEY id -- (id, class)
  ENTITIES FROM Papers KEY id -- (id, title, ...)
  LABELS   FROM Paper_Area LABEL l  -- (label)
  EXAMPLES FROM Example_Papers KEY id LABEL l  -- (id, label)
  FEATURE FUNCTION tf_bag_of_words
```

T(id, l)

Model-based Views

Eager Approach: maintain *V* as a materialized view where updates to class labels occur immediately after a model update.

Lazy approach: in response to read of an input *id*, read feature vector and label using current model.

Incremental model maintenance should improve both of these approaches for:

1. *Single Entity* read
2. *All Members* read
3. *Update*

Goals:

- Integrate classification models into run-time operation with RDBMS
- **Incorporate new training examples in a real-time, dynamic environment**

How?

- Model-based Views
- **Incremental Maintenance**

Incremental Maintenance

At round *i*, we receive new examples to update a materialized view $V^{(i)}$.

HAZY divides this into two problems:

1. How to perform an *incremental step* to update the old model to $V^{(i+1)}$ with (preferably low) cost $c^{(i)}$.
2. Decide when to *reorganize* to obtain a model by training on the whole dataset with a fixed cost S.

Incremental Maintenance

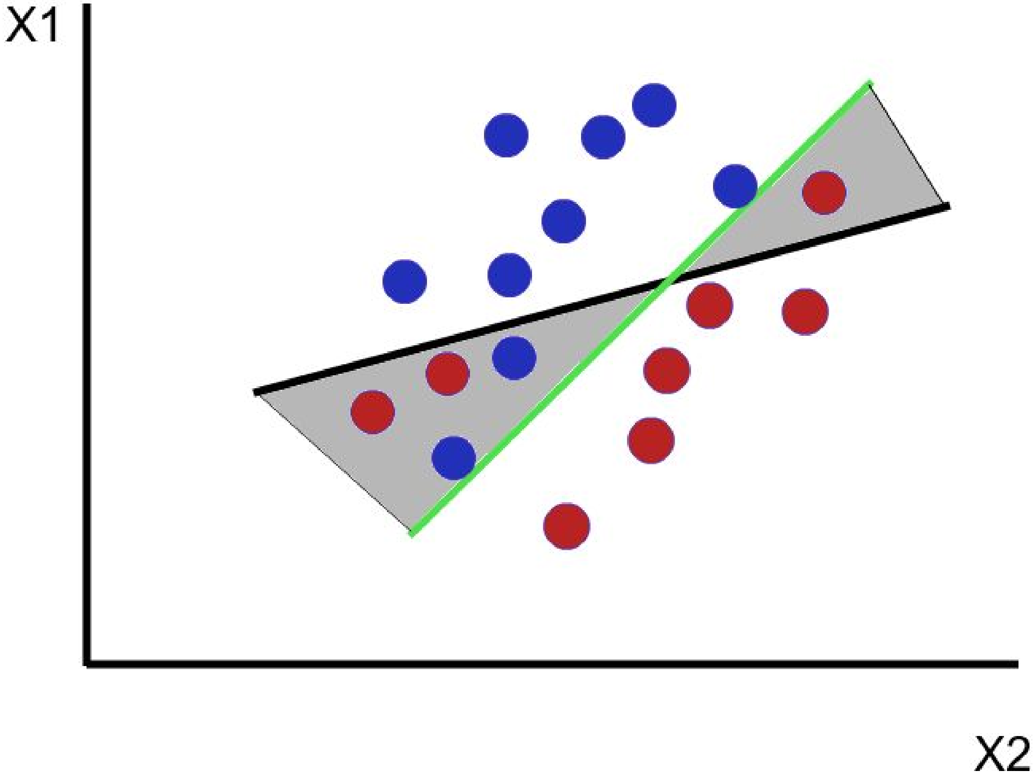
At round *i*, maintain a materialized view:

$$V^{(i)}(\text{id, class, eps})$$

Where:

$$\text{eps} = \epsilon = \vec{w}^{(i)} \cdot f - b^{(i)}$$

Let *s* be the last round at which HAZY reorganized the model.


X1
X2

Incremental Maintenance

$$\vec{w}^{(i+1)}, b^{(i+1)}$$

$$p, q \geq 0 \mid p^{-1} + q^{-1} = 1$$

$$M = max_{t \in In} ||t.f||_q$$

Let $j \geq s$ be a round after the “anchor” *s*.

$$\epsilon_{high}^{(s,j)} := M||w^{(j)} - w^{(s)}||_p + b^{(j)} - b^{(s)}$$

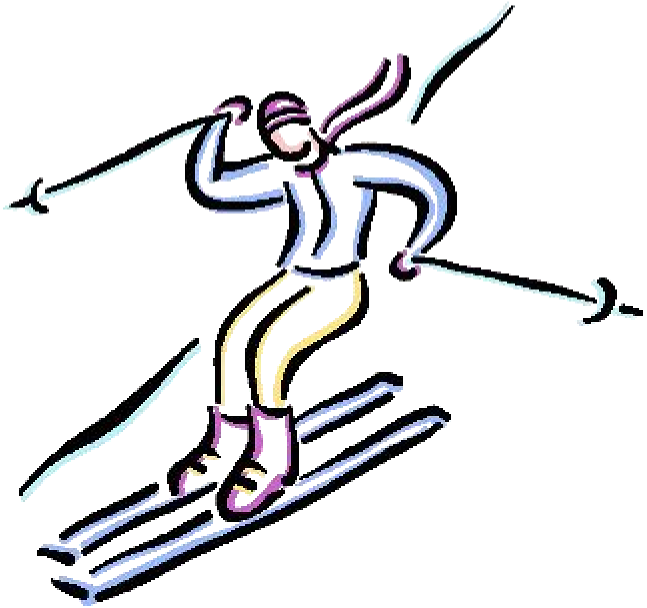
$$\epsilon_{low}^{(s,j)} := -M||w^{(j)} - w^{(s)}||_p + b^{(j)} - b^{(s)}$$

$$lw^{(s,j)} := \min_{l=s,\dots,j} \epsilon_{low}^{(s,l)} \qquad hw^{(s,j)} := \max_{l=s,\dots,j} \epsilon_{high}^{(s,l)}$$

Only need to reclassify tuples satisfying:

$$t.eps \in [lw^{(s,j)}, hw^{(s,j)}]$$

Reorganization - The Skiing Strategy



Reorganization - The Skiing Strategy

You are going skiing for an unknown number of days d. Every day you choose between renting a pair of skis for \$1 and buying the pair for \$10.

When should you purchase the skis?

Minimize the ratio between what you would pay using some decision strategy and what you would pay if you knew d.

Reorganization - The Skiing Strategy

- Choose some $\alpha \in (0, 1]$.
- At each round *i*, accumulate a total cost: $a^{(i+1)} = a^{(i)} + c^{(i)}$
- Reorganize when: $a^{(i)} \geq \alpha S$
- Then reset the accumulated cost to 0.

This is a 2-approximation of the optimal strategy, and is optimal among all online, deterministic strategies**.

<u>Reorganization</u>: update the model, re-cluster on t.eps, and rebuild indices.

**Assuming reorganizing more recently does not raise the cost.

Architectural Optimizations

- In-memory architecture:
  - Maintain classification view in memory, discard when memory needs to be revoked.
  - Cluster the data (on t.eps) in memory.
  - We only need to persist the entities and training examples since everything else can be re-computed.
- Hybrid architecture:
  - Maintain buffer for entities.
  - Cache t.eps if we can’t store all the entities in memory.

Questions, Issues

- How does this fit with the semantics of streaming systems?
- Not “black box” enough:
  - Featurization still exposed (vs. LASER source nodes).
  - Lazy and eager approaches have different semantics.
- Scalability - balancing throughput with feature length, dataset size.
- Drift, dataset size and the monotonicity assumption.

Questions, Issues

- Pushing incremental maintenance through the model training process:
  - SGD is orders of magnitude slower on larger datasets when performed in HAZY system vs. a hand-coded C file. Can this be improved without bulk-loading?
  - Can we take advantage of epsilon clustering during training?