**QMiner – data analytics platform for processing streams of structured and unstructured data**

Blaz Fortuna, Jan Rupnik, Carolina Fortuna, Marko Grobelnik, Viktor Jovanoski, Mario Karlovcec, Blaz Kazic, Klemen Kenda, Gregor Leban, Jost Novljan, Miha Papler, Luis Rei, Blaz Sovdat, Luka Stopar, Andrej Muhic

Jožef Stefan Institute, Jamova cesta 39, Ljubljana, Slovenia

**1. Introduction**

QMiner grew from projects in the areas of text, web and stream mining that Artificial Intelligence group at Jožef Stefan Institute was involved in over the last couple of years. We wanted developed solutions to be interactive and operate in real-time on data sets of 100s of GBs while still using the computational resources we have available. These goals and constraints resulted in a unique set of features that we implemented in QMiner.

QMiner applications are implemented in JavaScript, making it easy for novice users to get started. Using the JavaScript API it is easy to compose complete data processing pipelines and integrate with other systems via RESTful web services. The backend is implemented in C++ and can be included as a library into custom C++ projects, thus providing them with stream processing and data analytics capabilities.

QMiner is available as open source project in GitHub under AGPL licence. The repository contains source code, introduction guide and complete documentations of QMiner JavaScript APIs.

**2. Features**

**Connecting storage, indexing and analytics** – QMiner stores and indexes the data in a way that makes the implemented machine learning methods more scalable. Computing feature vectors from stored records tries to reduce any duplication of data in the process and relies heavily on the integrated indexing and its features (e.g. probabilistic joins) to make the transformations faster. Data schemas provide types for directly storing vectors and sparse vectors as part of records.

**Multimodal data support** – QMiner provides native support for handling and learning from unstructured data such as text or graphs. For example, full text search, aggregation over text fields, creating and storing bag-of-words vectors directly in the data stores, and full integration of Stanford graph analysis library SNAP in the C++ and JavaScript APIs

**Processing streaming data** – QMiner provides building blocks for processing and learning from streams of text documents, website logs or numeric streams. For example, pipeline-able stream aggregates for maintaining aggregate statistics over the stream, and machine learning algorithms for learning from stream.

**Probabilistic joins between tables** – For some operations computing a complete join between tables is not necessary. For example, to compute the gender distribution of visitors of a particular web page, we do not really need a complete list of visitors for that particular web page. What suffices is a statistically representative sample. QMiner supports several sampling techniques for achieving this, and integrates the support for probabilistic joins in query language and feature extractors.

**3. Examples**

QMiner let’s you get from data to working models exposed through web service API in less then an hour. Most applications can be scripted fully in the JavaScript layer, taking advantage of components implemented in C++ and libraries such as Intel MKL.

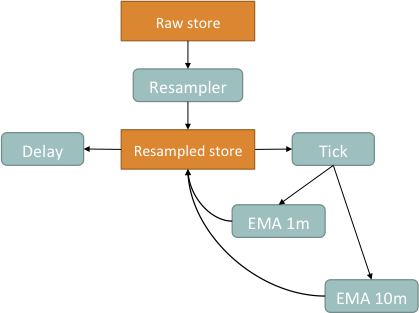
**3.1 Text processing, classification and regression.**

This example shows how to extract features from the movie dataset and use them to build classification and regression models to predict movie genre and rating.

* // Import analytics module
* var analytics = require("analytics.js");
* // Loading in the dataset.
* qm.load.jsonFile(Movies, "./sandbox/movies/movies.json");
* // Declare the features we will use to build genre classification models
* var genreFeatures = [
* { type: "text", source: "Movies", field: "Title" },
* { type: "text", source: "Movies", field: "Plot" },
* { type: "join", source: { store: "Movies", join: "Actor" } },
* { type: "join", source: { store: "Movies", join: "Director"} }
* ];
* // Create a model for the Genres field, using all the movies as training set.
* var genreModel = analytics.newBatchModel(Movies.recs,
* genreFeatures, Movies.field("Genres"));
* // Predict genres of a new movie
* var newMovie = qm.store("Movies").newRec({...});
* var result = genreModel.predict(newMovie);

**3.2 Time Series processing**

This example shows how resampling, enrichment and prediction work on time series data.



* // Initialize resamper from Raw to Resampled store. This results in
* // in an equaly spaced time series with 10 second interval.
* Raw.addStreamAggr({ name: "Resample10second", type: "resampler",
* outStore: "Resampled", timestamp: "Time",
* fields: [ { name: "Value", interpolator: "previous" } ],
* createStore: false, interval: 10 \* 1000
* });
* // Initialize stream aggregates on Resampled store for computing
* // 1 minute and 10 minute exponential moving averages.
* Resampled.addStreamAggr({ name: "tick", type: "timeSeriesTick",
* timestamp: "Time", value: "Value" });
* Resampled.addStreamAggr({ name: "ema1m", type: "ema",
* inAggr: "tick", emaType: "previous", interval: 60000, initWindow: 10000 });
* Resampled.addStreamAggr({ name: "ema10m", type: "ema",
* inAggr: "tick", emaType: "previous", interval: 600000, initWindow: 10000
* });
* // Buffer for keeping track of the record from 1 minute ago
* Resampled.addStreamAggr({ name: "delay", type: "recordBuffer", size: 6});
* // Declare features from the resampled timeseries
* var ftrSpace = analytics.newFeatureSpace([
* { type: "numeric", source: "Resampled", field: "Value" },
* { type: "numeric", source: "Resampled", field: "Ema1" },
* { type: "numeric", source: "Resampled", field: "Ema2" },
* { type: "multinomial", source: "Resampled", field: "Time", datetime: true }
* ]);
* // Initialize linear regression model.
* var linreg = analytics.newRecLinReg({ dim: ftrSpace.dim, forgetFact: 0.9999 });
* // We register a trigger to Resampled store
* Resampled.addTrigger({ onAdd: function (val) {
* // Get the latest value for EMAs
* val.Ema1 = Resampled.getStreamAggr("ema1m").EMA;
* val.Ema2 = Resampled.getStreamAggr("ema10m").EMA;
* // Get the id of the record from a minute ago.
* var trainRecId = Resampled.getStreamAggr("delay").last;
* // Update the model, once we have at leats 1 minute worth of data
* linreg.learn(ftrSpace.ftrVec(Resampled[trainRecId]), val.Value);
* }
* });

**4.3 Twitter**

This example shows how to perform text mining (feature spaces, active learning, classification) and record set filtering. It also applies a record set aggregate that builds communication graphs based on collections of Twitter messages (Twitter specific QMiner aggregate).

**4.4 SNAP**

Example from Mario?

**5. Conclusions**

**Acknowledgment**