Recommender System operators

In this example, the MovieLens dataset is used. The column separator was changed to tab.

The file `u_ordered.data` is ordered by timestamp (this is not necessary but allows implementations that take advantage of temporal effects, e. g., concept drift). The file `rfr.csv` has a sample of the the users in `u_ordered.data`. Both files are attached.

```
#DEFINE rating_data_input_file ${PROJECTPATH}/ml-100k/u_ordered.data
#DEFINE rfr_data_input_file ${PROJECTPATH}/ml-100k/rfr.csv


#PARSER PQL

#RUNQUERY
// A data stream of ratings.
rating_data := ACCESS
({source='rating_data', wrapper='GenericPull', transport='File', protocol='CSV', datahandler='Tuple',
        options=[
                ['Delimiter', '\t'],
                ['filename', '${rating_data_input_file}']],
                schema=[
                        ['user','Integer'], // some learners need Long instead of Integer
                        ['item','Integer'], // some learners need Long instead of Integer
                        ['rating','Double'],
                        ['timestamp','StartTimeStamp']
        ]
})

#RUNQUERY
// A data stream of request for recommendations of users.
rfr := TIMEWINDOW({size = 1}, ACCESS
({source='rfr', wrapper='GenericPull', transport='File', protocol='CSV', datahandler='Tuple',
        options=[
                ['Delimiter', '\t'],
                ['filename', '${rfr_data_input_file}']],
                schema=[
                        ['user','Integer'], // some learners need Long instead of Integer
                        ['timestamp','StartTimeStamp']
                ]
}))


#QNAME RecommenderSystem
#QUERY
/// split learning and test data
splitted_rating_data = EXTRACT_TEST_DATA({strategy = 'ITTT'}, rating_data)

/// continuous learning
windowed_learning_data = TIMEWINDOW({size = [30, 'days']}, 0:splitted_rating_data)
models = TRAIN_RECSYS_MODEL({learner = 'BRISMF.MOA'}, windowed_learning_data)

/// recommending
recomm_candidates = RECOMMENDATION_CANDIDATES(JOIN(rfr, 1:models))
predicted_candidates = PREDICT_RATING(JOIN(models, recomm_candidates))
recommendations = RECOMMEND({top_n = 8, min_rating = 3.5}, predicted_candidates)

/// evaluation
predicted_test_data = PREDICT_RATING(JOIN(models, 1:splitted_rating_data))
model_errors = TEST_PREDICTION({aggregation_window_size = [24, 'hours']}, predicted_test_data)
```