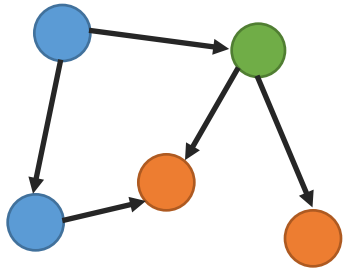
Qminer – efficient datamining in node.js

Jan Rupnik (@janrupnik),
AI Lab, Jožef Stefan Institute; Bloomberg LP

Typical Data Mining Workflow

Raw data

Load



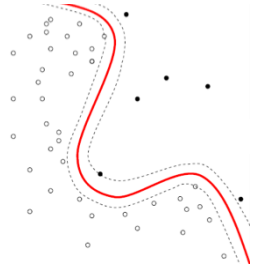
Storage and index

Transform

0.83									
		0.92							
0.49		0.85	0.59	0.58			0.71		
0.12					0.66				
		0.06				0.61	0.04	0.05	0.49
			0.58		0.38			0.41	
	0.99	0.73					0.01		
0.25	0.79					0.13	0.23		
0.50			0.24						0.70
							0.16	0.69	
			0.10		0.68				0.29
			0.87		0.96		0.77	0.03	0.94
					0.93	0.37		0.87	0.97
0.50									

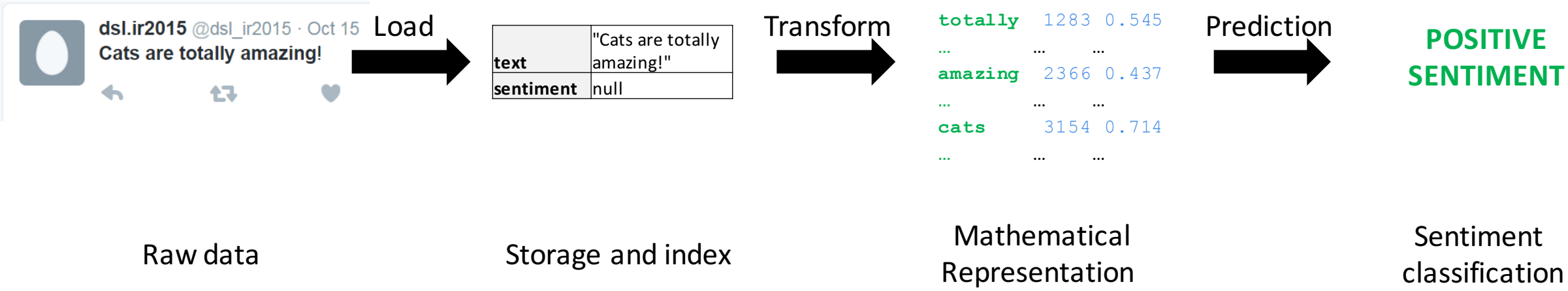
Mathematical Representation

Machine learning



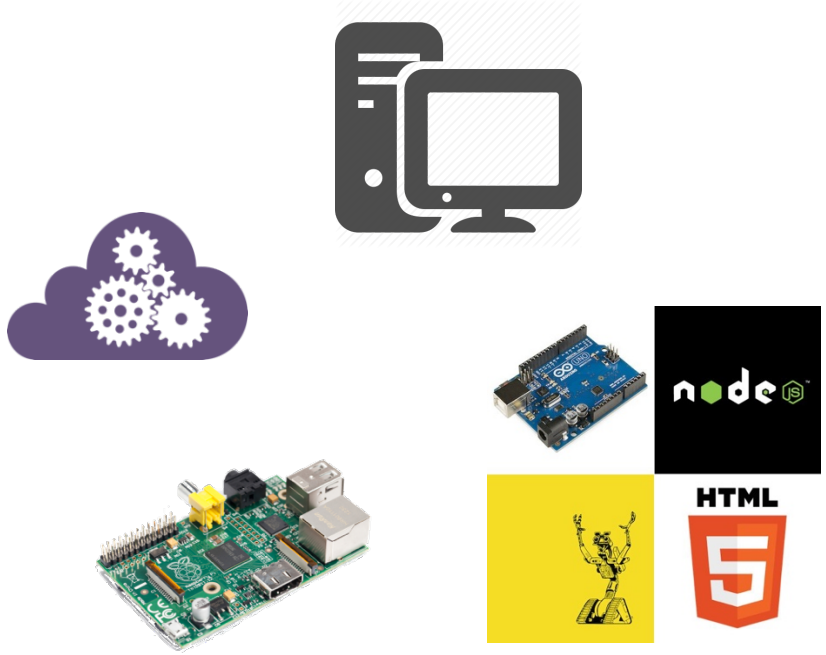
Model

Typical Data Mining Workflow



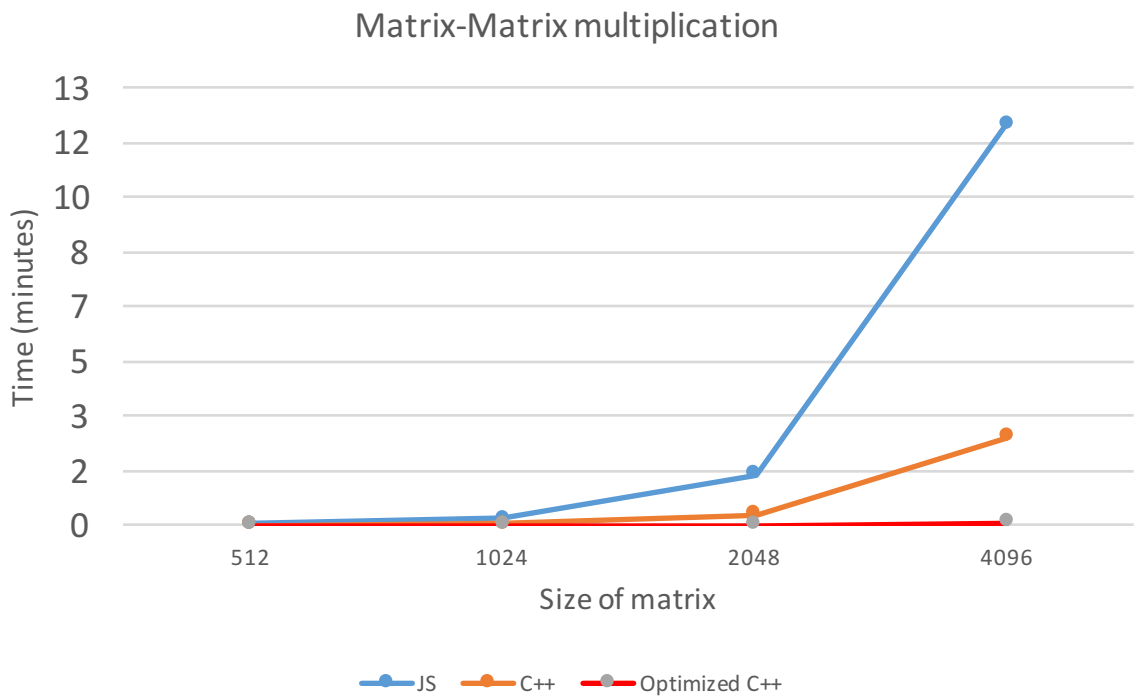
Using Node.js for Analytics: Pros

- Easy to build complete verticals from data acquisition to the API/GUI
- Rich ecosystem of available components
  - I/O (web APIs, sensors), Application frameworks
- Runs on many platforms
  - Standard desktops (Linux, Windows, Mac)
  - Cloud (Apache Storm, Amazon Lambda)
  - Data-friendly platforms: Raspberry Pi, Robotics, IoT



Using Node.js for Analytics: Cons

- Pure JavaScript not suitable for large data structures
  - Large Array can break V8 engine
  - Native structures difficult to scale



QMiner (https://github.com/qminer)

Native add-on for fast development of integrated data analytics applications which can scale to mid-size data (TBs)

- **Scalable data structures** (vectors, matrices, hash tables, graphs)
- **Data transformation** (text, social networks, streams)
- **Numerical computation** (linear algebra, machine learning)
- **Open source** (BSD license)

Use cases

Twitter sentiment classification

User profiling and recommendation

Twitter sentiment

- Raw data 1.6M tweets
  - 0.8M tweets with ☺
  - 0.8M tweets with ☹
  - Text stripped from emoticons


```
{
  "text":"@_constantstatic why arent you
answering me on skype ",
  "target":-1
}

{
  "text":" I just ran out of my favorite
cologne...Who wants to buy me some more??",
  "target":-1
}
```


http://help.sentiment140.com/for-students/

Twitter sentiment

Let's print the first tweet in the training set and its sentiment.

```
tweets[0].text;
```

```
"@_constantstatic why arent you answering me on skype "
```

```
tweets[0].target > 0 ? 'positive' : 'negative';
```

```
"negative"
```

Build feature space (mapping from records to linear algebra vectors). Here we use a simple tfidf weighted bag-of-words feature extractor.

```
var featureSpace = new qm.FeatureSpace(base, {
      type: 'text', source: 'tweets', field: 'text'});
featureSpace.updateRecords(tweets);
'built feature space with ' + featureSpace.dim + ' dimensions'
```

```
"built feature space with 5402 dimensions"
```

Build a sentiment classifier model. We use a Support Vector Classifier with default parameters (C = 1).

```
var SVC = new qm.analytics.SVC({ maxTime: 5 });
SVC.fit(featureSpace.extractSparseMatrix(tweets),
        tweets.getVector('target'));
'trained model'
```

```
"trained model"
```

https://runkit.com/rupnikj/sentiment-classifier

Import libraries: the main library qminer and a helper package for the example dataset.

```
var qm = require('qminer');
var loader = require('qminer-data-loader');
'loaded'
```

```
"loaded"
```

Define the storage schema. We define one store called 'tweets' where each record has two fields: text and target (+1 is for positive sentiment and -1 for negative sentiment).

```
var base = new qm.Base({
    mode: 'createClean',
    schema: [{ name: 'tweets',
               fields: [{ name: 'text', type: 'string' },
                        { name: 'target', type: 'float' }]
            }]
});
'defined schema'
```

```
"defined schema"
```

Import data and select all records.

```
12 loader.loadSentimentDataset(base.store('tweets'));
13 var tweets = base.store('tweets').allRecords;
14 'got ' + tweets.length + ' tweets';
```

```
"got 2000 tweets"
```

Let's print the first tweet in the training set and its sentiment.

```
15 tweets[0].text;
```

```
"@_constantstatic why arent you answering me on skype "
```

```
16 tweets[0].target > 0 ? 'positive' : 'negative';
```

```
"negative"
```

Build feature space (mapping from records to linear algebra vectors). Here we use a simple tfidf weighted bag-of-words feature extractor.

```
var featureSpace = new qm.FeatureSpace(base, {
      type: 'text', source: 'tweets', field: 'text'});
featureSpace.updateRecords(tweets);
'built feature space with ' + featureSpace.dim + ' dimensions'
```

```
"built feature space with 5402 dimensions"
```

Build a sentiment classifier model. We use a Support Vector Classifier with default parameters (C = 1).

```
var SVC = new qm.analytics.SVC({ maxTime: 5 });
SVC.fit(featureSpace.extractSparseMatrix(tweets),
        tweets.getVector('target'));
'trained model'
```

```
"trained model"
```

Predict sentiment of a new example. What is the sentiment of the sentence "Cats are stupid" ?

```
var y1 = SVC.predict(featureSpace.extractSparseVector({
    text: "Cats are stupid."
}));
'predicted sentiment: ' + (y1 > 0 ? 'positive' : 'negative');
```

```
"predicted sentiment: negative"
```

What about the sentence "Cats are totally amazing!" ?

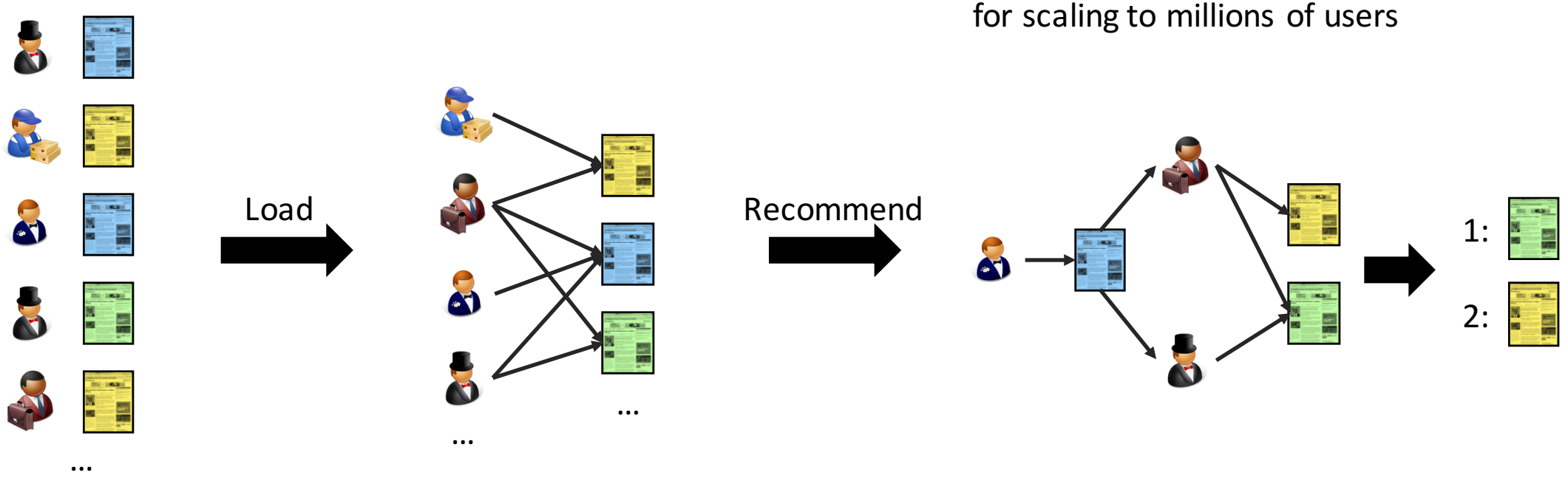
```
var y2 = SVC.predict(featureSpace.extractSparseVector({
    text: "Cats are totally amazing!"
}));
'predicted sentiment: ' + (y2 > 0 ? 'positive' : 'negative');
```

```
"predicted sentiment: positive"
```

```

```

Recommendation



Recommendation

- Raw data 4M page views
  - http://videolectures.net/
  - 750k visitors
  - 13k lectures
- Online demo on 10k visitors

```
{
  "ID":"808459829515990 8",
  "lectures":[
    {"Title":"Information Theory"},
    {"Title":"Online Learning of Music …"},
    {"Title":"Gaussian Process Basics"},
    {"Title":"Preference Learning in …"},
    …
  ]
}
```

Recommendation

Recommendation of items using "visitors that view this also viewed..." approach. That is, visitors are recommended items that are most often viewed by other visitors that watched some of the same items. In this example we will use visitor-item data from online lectures website VideoLectures.NET.

Import libraries: the main library qminer and a helper package for the example dataset.

```
var qm = require('qminer');
var loader = require('qminer-data-loader');
'libraries loaded'
```

```
"libraries loaded"
```

Define the storage schema. We have two stores: one for visitors descrbied by a unique ID and one for lectures described by their title. Visitors and lectures are connected by a join.

```
var base = new qm.Base({
  mode: 'createClean',
  schema: [{
    name: 'Visitors',
    fields: [{ name: 'ID', type: 'string', primary: true }],
    joins: [{ name: 'lectures', type: 'index', store: 'Lectures', inverse :
'visitors' }]
  }, {
    name: 'Lectures',
    fields: [{ name: 'Title', type: 'string', primary: true }],
    joins: [{ name: 'visitors', type: 'index', store: 'Visitors', inverse:
'lectures'}]
  }]
```

https://runkit.com/rupnikj/qminer-recommendation

Import libraries: the main library qminer and a helper package for the example dataset.

```
var qm = require('qminer');
var loader = require('qminer-data-loader');
'libraries loaded'
```

```
"libraries loaded"
```

Define the storage schema. We have two stores: one for visitors descrbied by a unique ID and one for lectures described by their title. Visitors and lectures are connected by a join.

```
var base = new qm.Base({
  mode: 'createClean',
  schema: [{
    name: 'Visitors',
    fields: [{ name: 'ID', type: 'string', primary: true }],
    joins: [{ name: 'lectures', type: 'index', store: 'Lectures', inverse : 'visitors' }]
  }, {
    name: 'Lectures',
    fields: [{ name: 'Title', type: 'string', primary: true }],
    joins: [{ name: 'visitors', type: 'index', store: 'Visitors', inverse: 'lectures'}]
  }]
});
'defined schema'
```

```
"defined schema"
```

Import visit data.

```
17 loader.loadRecommendationDataset(base.store('Visitors'));
18 'Loaded ' + base.store('Visitors').length + ' visitors'
```

```
"Loaded 10000 visitors"
```

```
19 'Loaded ' + base.store('Lectures').length + ' lectures'
```

```
"Loaded 9380 lectures"
```

Check what we know for one visitor. Let us call him John. We see which lectures did John see by following 'lectures' join.

```
20 var john = base.store('Visitors')[126];
21 'Visitor ' + john.ID + ' viewed ' + john.lectures.length + ' lectures'
```

```
"Visitor 6055868891858356 viewed 1 lectures"
```

```
22 john.lectures[0].Title
```

```
"Fuzzy Logic"
```

Now lets generate a list of lectures to recommend to john. First, let us see who else viewed the same lectures by following the 'visitors' join.

```
var coVisitors = john.lectures.join('visitors');
coVisitors.length + ' visitors watched the same lecture'
```

```
"162 visitors watched the same lecture"
```

Then, lets see what else did the co-visitors viewed. In order to speed up recommendation process we use sampling to select only 100 visitors and check what they viewed. Sampling takes into account the weights, meaning that visitors that share more lectures with John are more likely to be selected.

```
var coLectures = coVisitors.join('lectures', 100);
'Co-visitors watched ' + coLectures.length + ' lectures'
```

```
"Co-visitors watched 675 lectures"
```

Finally, removed the lectures already seen by John, sort the final result by frequency and keep top 5. In this scenario, frequencies correspond to number of co-visitors that viewed the lecture.

```
var result = coLectures.setDiff(john.lectures);
result.sortByFq(-1).trunc(5);
'Recommendations generated'
```

```
"Recommendations generated"
```

Let's list the recommendations!

```
30 result[0].Title
```

```
"Introduction to Machine Learning"
```

```
31 result[1].Title
```

```
"Gaussian Process Basics"
```

```
32 result[2].Title
```

```
"PhD Thesis Defense: Dynamics of large networks"
```

```
33 result[3].Title
```

```
"Statistical Learning Theory"
```

```
34 result[4].Title
```

```
"Introduction To Bayesian Inference"
```

Production tested

- Real-time news recommendation with 10s of Ms of users
- Global media monitoring (http://eventregistry.org)
- Real-time road traffic modeling and prediction
- Root-cause for network and device fault analysis

```
$ npm install qminer
```

https://github.com/qminer