Stacking, Boosting and Online Learning in distributed mode with Apache Ignite

Yuriy Babak

Yuriy Babak

Head of ML/DL framework development at GridGain

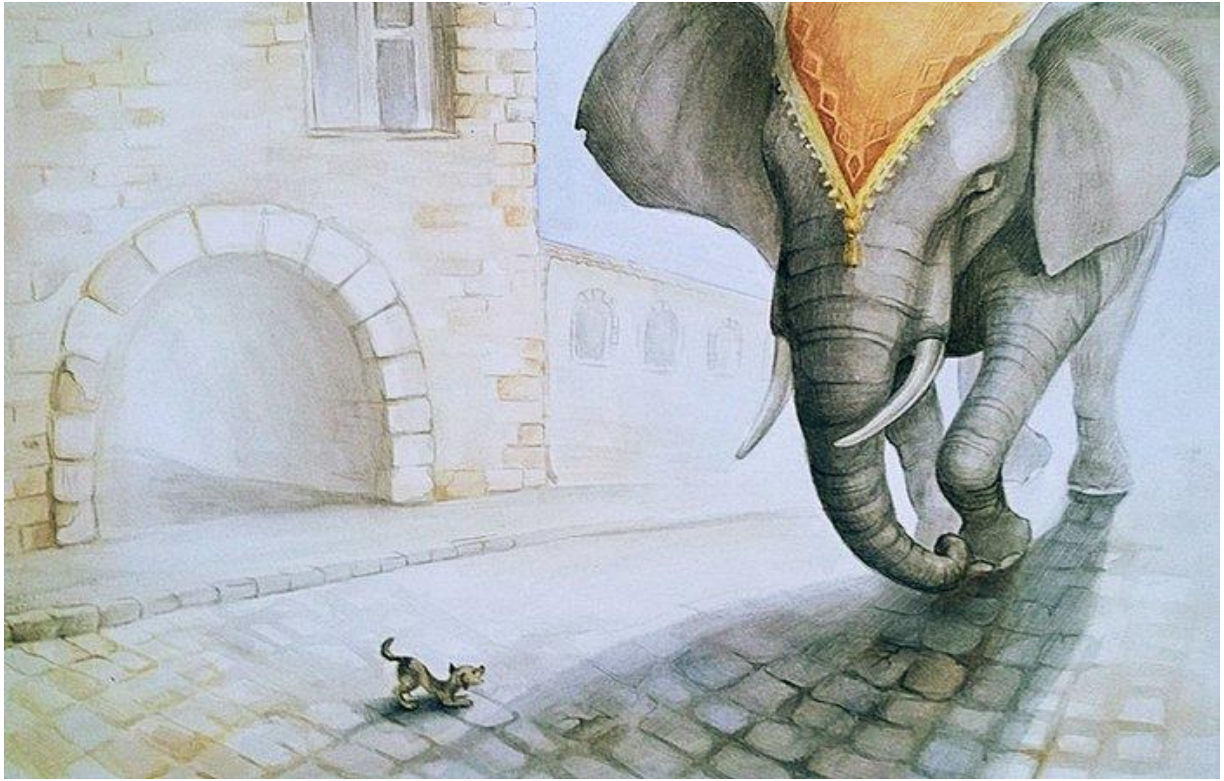
Apache Ignite committer

Agenda

- Overview of distributed ML/DL
- Data preprocessing in distributed environment
- Model training in distributed environment
- Building pipelines
- Stacking, Boosting and online learning
- Some extra features

Distributed Machine learning

Training on PBs with scikit-learn



Distributed ML with Apache Spark

- It supports classic ML algorithms

Distributed ML with Apache Spark

- It supports classic ML algorithms
- Algorithms are distributed by nature

Distributed ML with Apache Spark

- It supports classic ML algorithms
- Algorithms are distributed by nature
- Wide support of different data sources and sinks

Distributed ML with Apache Spark

- It supports classic ML algorithms
- Algorithms are distributed by nature
- Wide support of different data sources and sinks
- Easy building of Pipelines

Spark MLlib

Distributed ML with Apache Spark

- It supports classic ML algorithms
- Algorithms are distributed by nature
- Wide support of different data sources and sinks
- Easy building of Pipelines
- Model evaluation and hyper-parameter tuning support

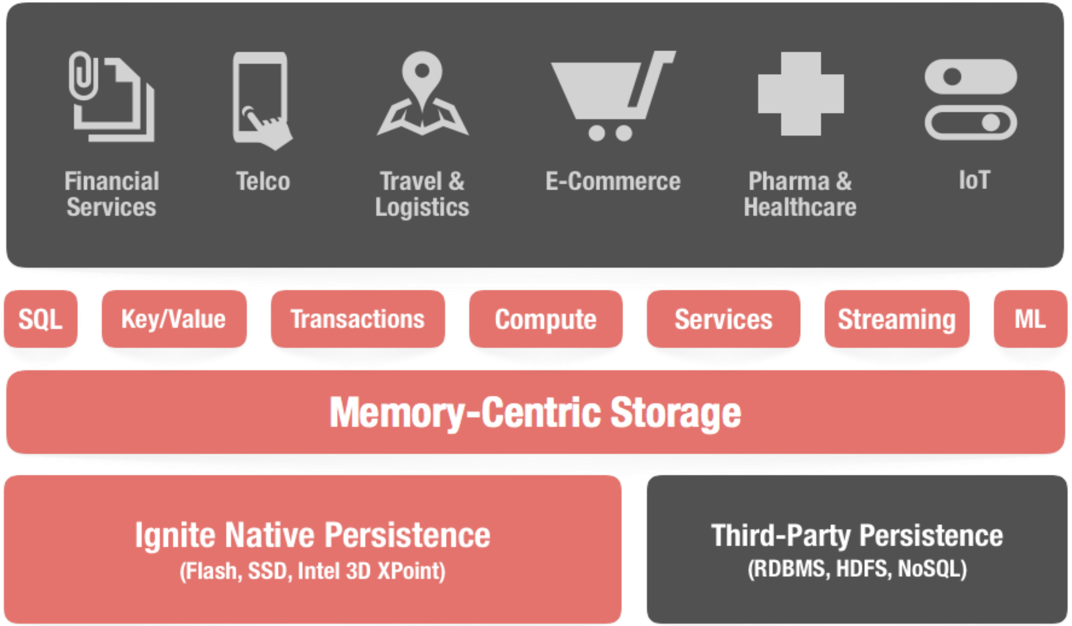
Distributed ML platforms

apache Ignite™

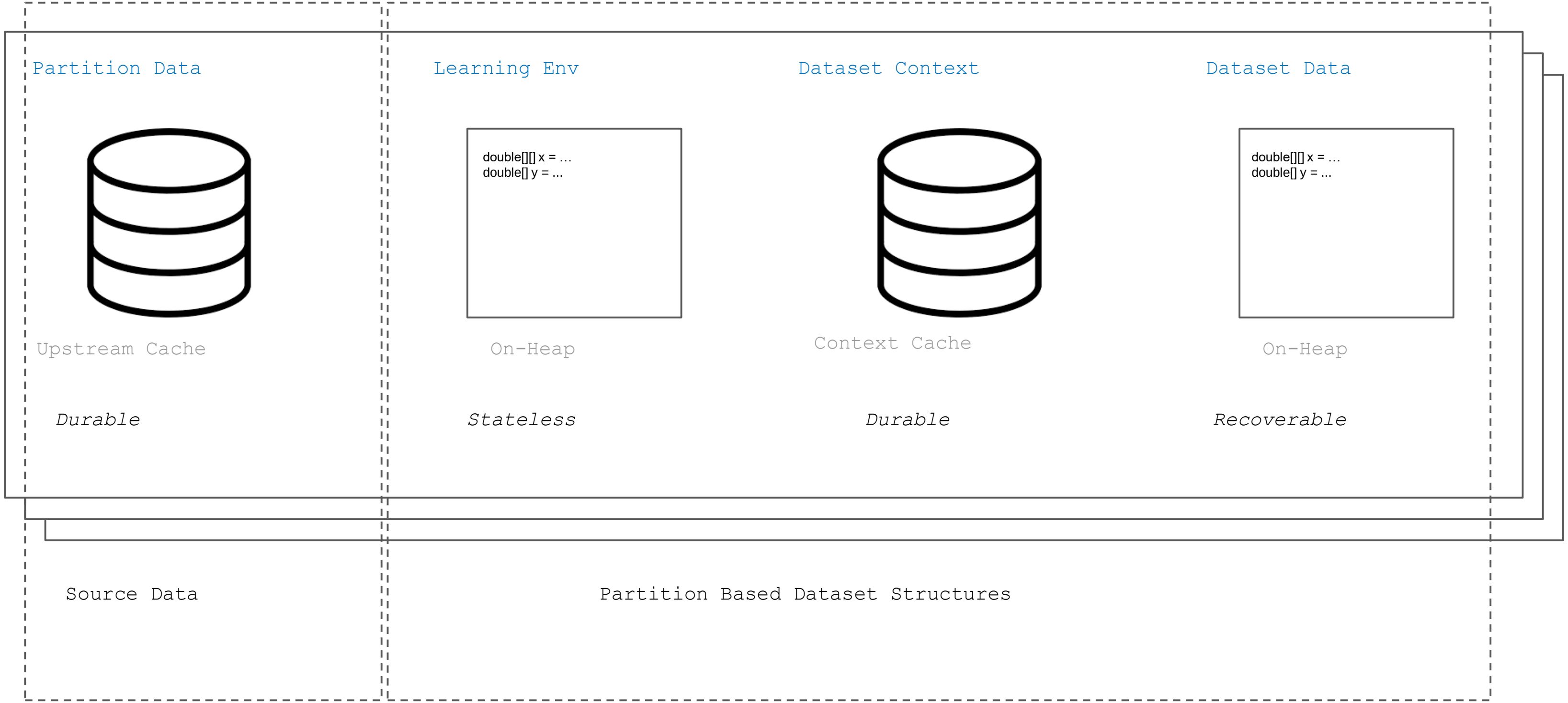
Spark MLlib

Distributed ML with Apache Ignite

What is Apache Ignite?

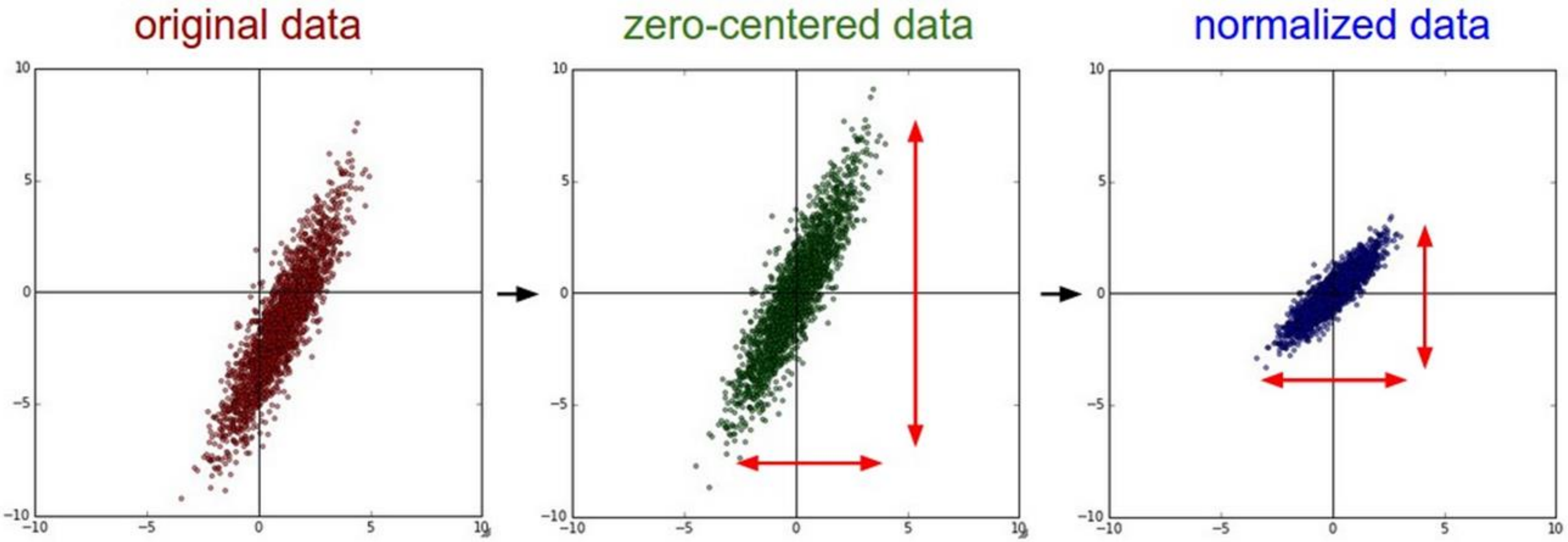


Distributed ML with Apache Ignite



Data preprocessing

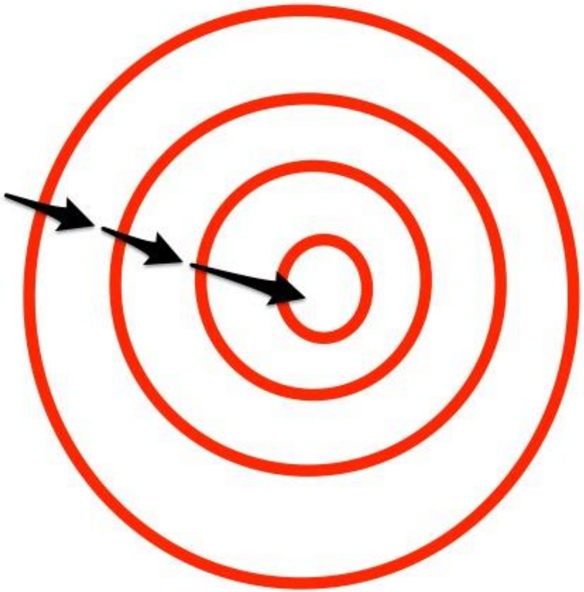
Data preprocessing: Normalization



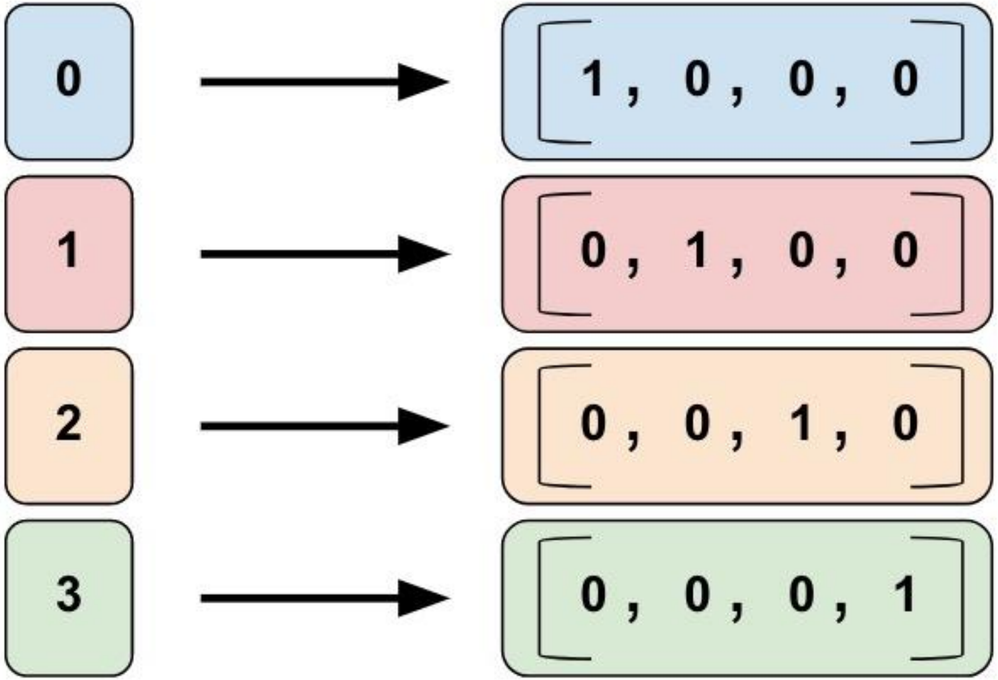
Data preprocessing: Scaling

Without feature scaling

With feature scaling



Data preprocessing: One-Hot Encoder



Data preprocessing: API

```
Preprocessor imputingPr = new ImputerTrainer().fit(ignite, dataCache, vectorizer);
```

Data preprocessing: API

```
Preprocessor imputingPr = new ImputerTrainer().fit(ignite, dataCache, vectorizer);

Preprocessor minMaxScalerPr = new MinMaxScalerTrainer()
                                  .fit(ignite, dataCache, imputingPr);
```

Data preprocessing: API

```
Preprocessor imputingPr = new ImputerTrainer().fit(ignite, dataCache, vectorizer);

Preprocessor minMaxScalerPr = new MinMaxScalerTrainer()
                                      .fit(ignite, dataCache, imputingPr);

Preprocessor normalizationPr = new NormalizationTrainer()
                                      .withP(1)
                                      .fit(ignite, dataCache, minMaxScalerPr);
```

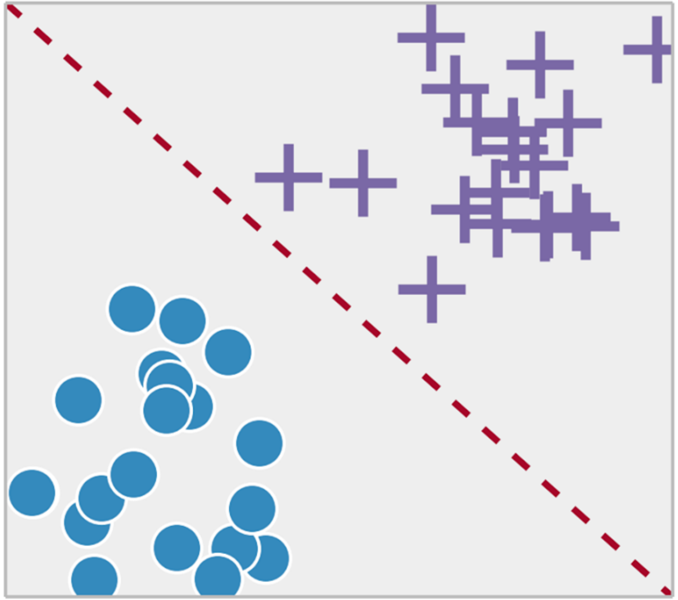
Data preprocessing: API

```
Preprocessor imputingPr = new ImputerTrainer().fit(ignite, dataCache, vectorizer);

Preprocessor minMaxScalerPr = new MinMaxScalerTrainer()
                                          .fit(ignite, dataCache, imputingPr);

Preprocessor normalizationPr = new NormalizationTrainer()
                                          .withP(1)
                                          .fit(ignite, dataCache, minMaxScalerPr);

DecisionTreeClassificationTrainer trainer = new DecisionTreeClassificationTrainer(5, 0);

DecisionTreeNode mdl = trainer.fit(ignite, dataCache, normalizationPr);

double accuracy = Evaluator.evaluate(dataCache, mdl, normalizationPr, new
Accuracy<>());
```

Model training

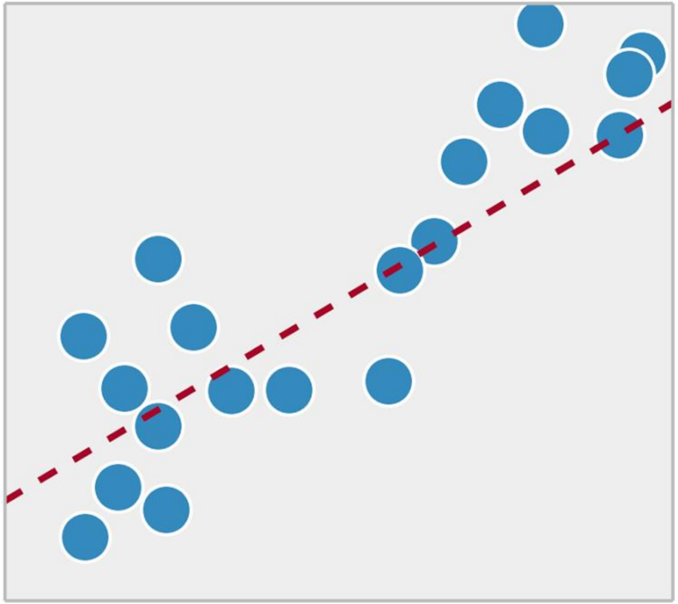
Algorithms: Classification

- Logistic Regression
- SVM
- KNN
- ANN
- Decision trees
- Random Forest
- Naive Bayes



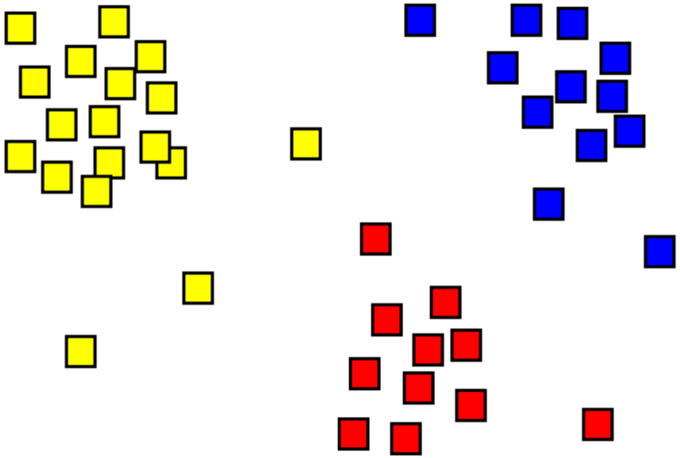
Algorithms: Regression

- KNN Regression
- Linear Regression
- Decision tree regression
- Random forest regression
- Gradient-boosted tree regression

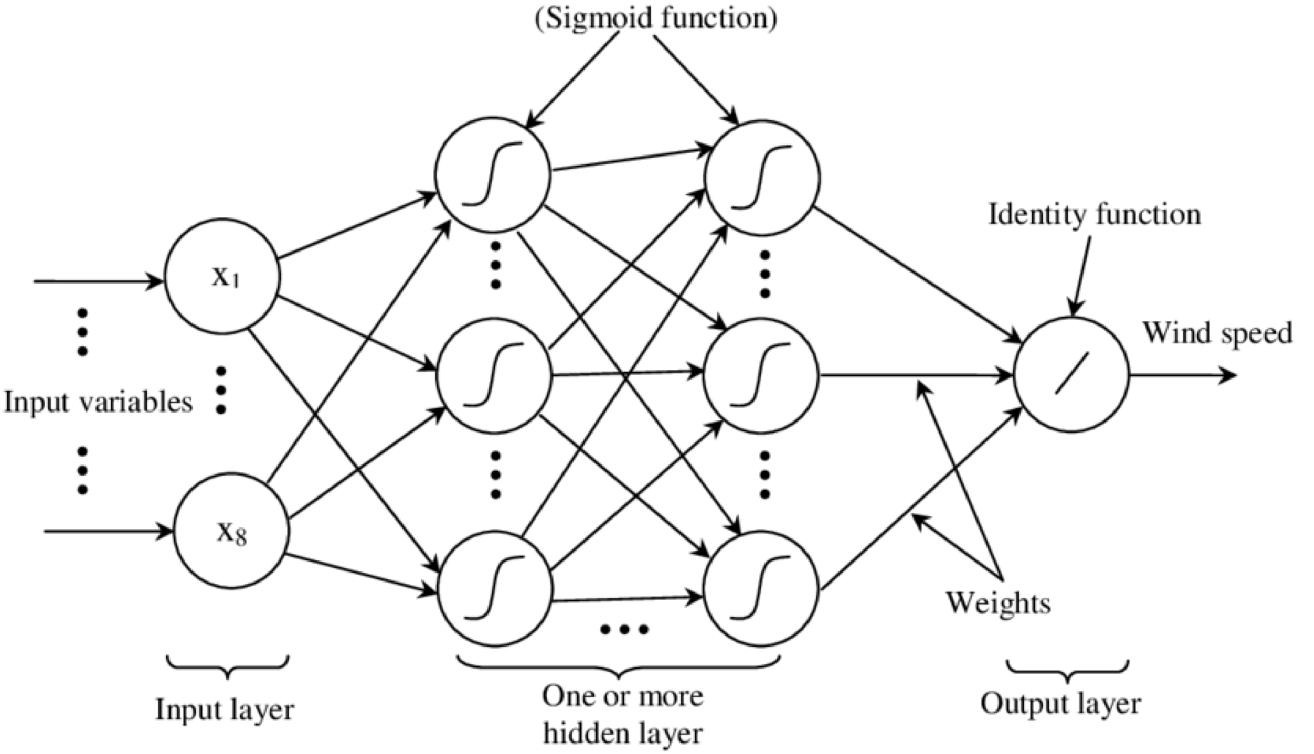


Algorithms: Clusterization

- K-means
- GMM



Multilayer Perceptron Neural Network



Fill the cache

```
IgniteCache<Integer, Vector> dataCache = TitanicUtils.readPassengers (ignite);
```

Build Labeled Vectors

```
IgniteCache<Integer, Vector> dataCache = TitanicUtils.readPassengers (ignite);


Vectorizer vectorizer = new DummyVectorizer(0, 5, 6).labeled(1);
```


GridGain

Define the trainer

```
IgniteCache<Integer, Vector> dataCache = TitanicUtils.readPassengers (ignite);

Vectorizer vectorizer = new DummyVectorizer(0, 5, 6).labeled(1);

DecisionTreeClassificationTrainer trainer = new DecisionTreeClassificationTrainer(5, 0);
```

Train the model

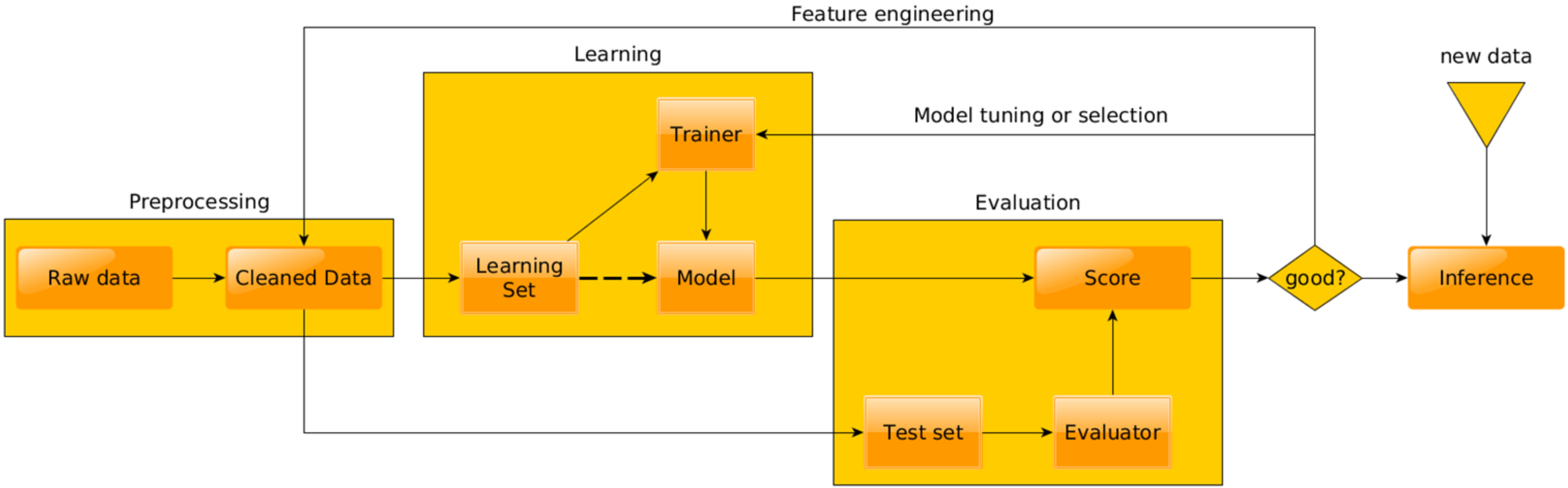
```
IgniteCache<Integer, Vector> dataCache = TitanicUtils.readPassengers (ignite);

Vectorizer vectorizer = new DummyVectorizer(0, 5, 6).labeled(1);

DecisionTreeClassificationTrainer trainer = new DecisionTreeClassificationTrainer(5, 0);

DecisionTreeNode mdl = trainer.fit(ignite, dataCache, vectorizer);
```

Evaluate the model

```java
IgniteCache<Integer, Vector> dataCache = TitanicUtils.readPassengers (ignite);

Vectorizer vectorizer = new DummyVectorizer(0, 5, 6).labeled(1);

DecisionTreeClassificationTrainer trainer = new DecisionTreeClassificationTrainer(5, 0);

DecisionTreeNode mdl = trainer.fit(ignite, dataCache, vectorizer);

double accuracy = Evaluator.evaluate(dataCache, mdl, vectorizer, new Accuracy<>());
```

Pipelines

ML Pipeline schema



ML Pipelines with Apache Ignite

```
IgniteCache<Integer, Vector> dataCache = TitanicUtils.readPassengers(ignite);

// Extracts "pclass", "sibsp", "parch", "sex", "embarked", "age", "fare".
Vectorizer<Integer, Vector, Integer, Double> vectorizer
  = new DummyVectorizer<Integer>(0, 3, 4, 5, 6, 8, 10).labeled(1);

PipelineMdl<Integer, Vector> mdl =
   new Pipeline<Integer, Vector, Integer, Double>()
  .addVectorizer(vectorizer)
  .addPreprocessingTrainer(new EncoderTrainer<Integer, Vector>()
     .withEncoderType(EncoderType.STRING_ENCODER)
     .withEncodedFeature(1)
     .withEncodedFeature(6))
  .addPreprocessingTrainer(new ImputerTrainer<Integer, Vector>())
  .addPreprocessingTrainer(new MinMaxScalerTrainer<Integer, Vector>())
  .addPreprocessingTrainer(new NormalizationTrainer<Integer, Vector>()
     .withP(1))
  .addTrainer(new DecisionTreeClassificationTrainer(5, 0))
  .fit(ignite, dataCache);
```

Beyond the limits of Apache Spark

Spark limits

- It doesn’t support model ensembles as stacking, boosting, bagging

Bagging, Boosting and Stacking

```
DatasetTrainer<LogisticRegressionModel, Double> trainer =
   new LogisticRegressionSGDTrainer(...)...;

BaggedTrainer<Double> baggedTrainer = TrainerTransformers.makeBagged(trainer,
// ensemble size, subsample ration, feature vector size, features subspace dim
  7, 0.7, 2, 2,
  new onMajorityPredictionsAggregator());
```

Spark limits

- It doesn’t support model ensembles as stacking, boosting, bagging
- It doesn’t support online-learning for all algorithms

Online learning

```
SVMLinearClassificationTrainer trainer = new SVMLinearClassificationTrainer();

SVMLinearClassificationModel mdl1 = trainer.fit(ignite, dataCache1, vectorizer);

SVMLinearClassificationModel mdl2 = trainer.update(mdl1, ignite, dataCache2,
vectorizer);
```

Spark limits

- It doesn’t support model ensembles as stacking, boosting, bagging
- It doesn’t support online-learning for all algorithms
- The hard integration with TensorFlow

TensorFlow on Apache Ignite

- Ignite Dataset
- IGFS Plugin
- Distributed Training
- More info here

```
>>> import tensorflow as tf
>>> from tensorflow.contrib.ignite import IgniteDataset
>>>
>>> dataset = IgniteDataset(cache_name="SQL_PUBLIC_KITTEN_CACHE")
>>> iterator = dataset.make_one_shot_iterator()
>>> next_obj = iterator.get_next()
>>>
>>> with tf.Session() as sess:
>>>   for _ in range(3):
>>>     print(sess.run(next_obj))

{'key': 1, 'val': {'NAME': b'WARM KITTY'}}
{'key': 2, 'val': {'NAME': b'SOFT KITTY'}}
{'key': 3, 'val': {'NAME': b'LITTLE BALL OF FUR'}}
```

Spark limits

- It doesn’t support model ensembles as stacking, boosting, bagging
- It doesn’t support online-learning for all algorithms
- The hard integration with TensorFlow
- A lot of data transformation/overhead from data source to ML types
- A part of algorithms use sparse matrix
- ML algorithms internally use Mllib on RDD

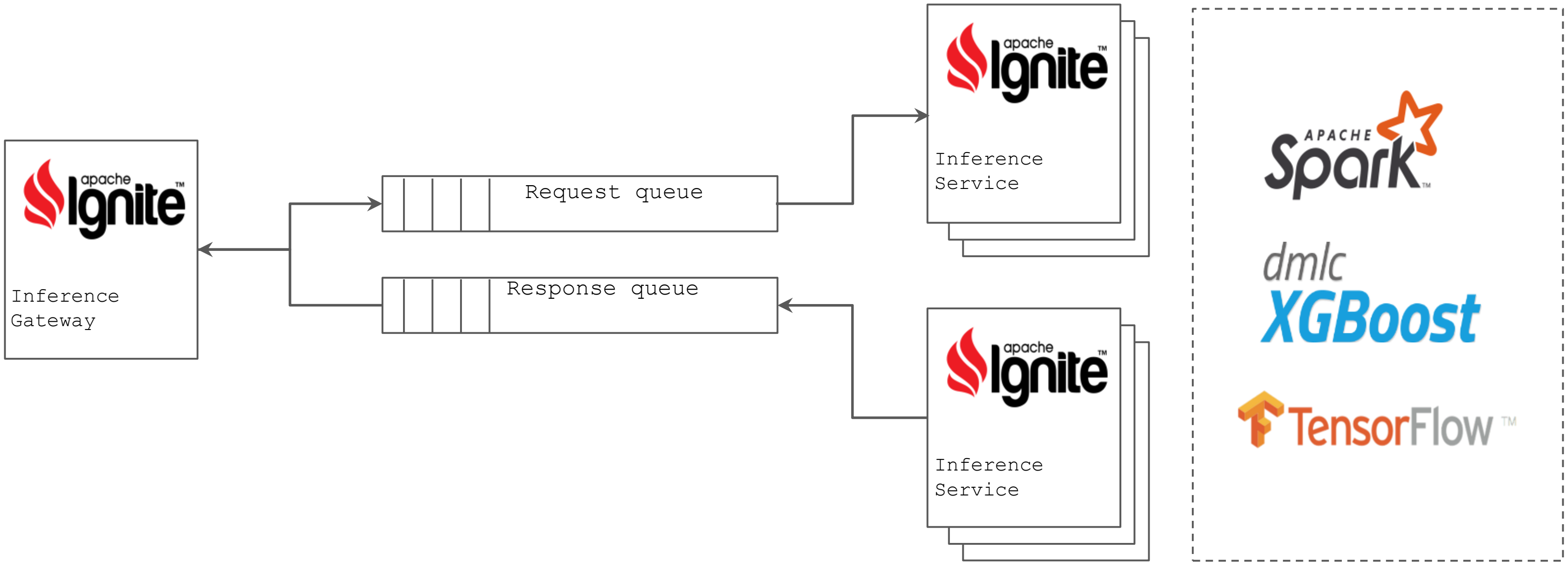
Friendship is optimal

IMDB with built-in ML

```
IgniteModelStorageUtil.saveModel(ignite, model, “titanik_model_tree”);

QueryCursor<List<?>> cursor = cache.query(new SqlFieldsQuery("select " +
    "survived as truth, " +
    "predict('titanik_model_tree', pclass, age, sibsp, parch, fare, case
sex when 'male' then 1 else 0 end) as prediction " +
    "from titanik_train"))
```

Model import

Inference in Ignite ML



Apache Ignite with GridGain ML Python API

GridGain ML client library provides user applications the ability to work with GridGain ML functionality using Py4J as an integration mechanism.

If you want to use *ggml* in your project, you may install it from PyPI:

```
$ pip install ggml
```

Apache Ignite with GridGain ML Python API

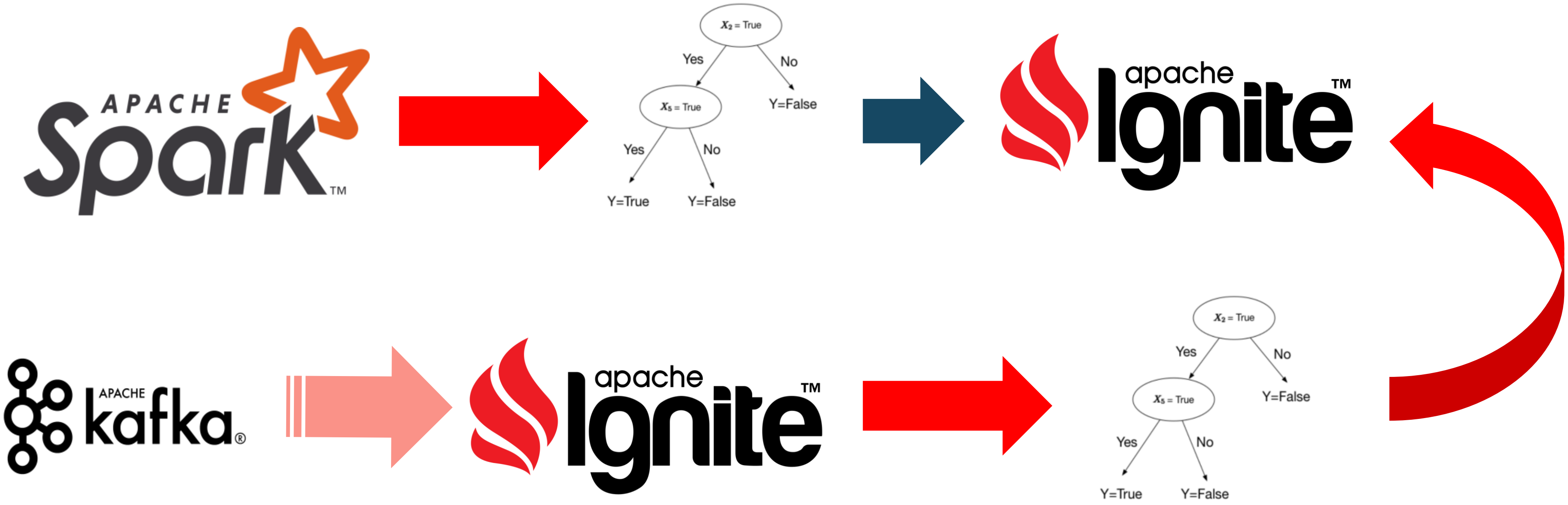
GridGain ML client library provides user applications the ability to work with GridGain ML functionality using Py4J as an integration mechanism.

If you want to use *ggml* in your project, you may install it from PyPI:

```
$ pip install ggml
```

NB: available only for Apache Ignite master and for GG 8.7.6 (17 Jul)

It could be your application



Conclusions

Conclusion

- Apache Ignite ready for building ML/DL systems

Conclusion

- Apache Ignite ready for building ML/DL systems
- You could use other systems for any part in your architecture

Conclusion

- Apache Ignite ready for building ML/DL systems
- You could use other systems for any part of your architecture
- You could use other systems with Apache Ignite and achieve extra abilities

Apache Ignite ML Tutorial

https://github.com/apache/ignite/

`org.apache.ignite.examples.ml.tutorial`

Distributed Machine and Deep Learning at Scale with Apache Ignite

Links:

- http://ignite.apache.org/
- https://medium.com/tensorflow/tensorflow-on-apache-ignite-99f1fc60efeb
- https://github.com/gridgain/ml-python-api

Email:

- user@ignite.apache.org
- dev@ignite.apache.org
- ybabak@gridgain.com

GridGain