Hivemall

APACHECON

EUROPE

Apache Hivemall Meets PySpark

Scalable Machine Learning with Hive, Spark, and Python

arm TREASURE DATA

Takuya Kitazawa **@takuti**

Apache Hivemall PPMC

Machine Learning in **Query Language**

Q. Solve ML problem on massive data stored in data warehouse

Theory / math

Scalability

Tool / Data model

Q. Solve ML problem on massive data stored in data warehouse

Practical experience in science and engineering

Done by **~10 lines of queries**

```sql
select
  feature,
  avg(weight) as weight
from (
  select
    logress(features, label) as (feature, weight)
  from (
    select features, label
    from train_oversampling
    CLUSTER BY rand(1) -- random shuffling
  ) t1
) t2
group by feature;
```

Machine Learning ***for everyone***

Open source query-based machine learning solution

- Incubating since Sept 13, 2016
- **@ApacheHivemall**
- GitHub: **apache/incubator-hivemall**
- Team: 6 PPMCs + 3 committers
- Latest release: **v0.5.2** (Dec 3, 2018)
- Toward graduation:
  - ✓ Community growth
  - ✓ 1+ Apache releases
  - ✓ Documentation improvements

Hivemall

APACHE INCUBATOR

Introduction to Apache **Hivemall**

How Hivemall Works with **PySpark**

Hivemall <3 Python

Introduction to Apache **Hivemall**

How Hivemall Works with **PySpark**

Hivemall <3 Python

Apache **Hive**

- **Data warehousing solution** built on top of Apache **Hadoop**
- Efficiently access and analyze large-scale data via SQL-like interface, **HiveQL**
  - `create table`
  - `select`
  - `join`
  - `group by`
    - `count()`
    - `sum()`
    - …
  - `order by`
  - `cluster by`
  - …

Apache **Hivemall**

- OSS project under **Apache Software Foundation** APACHE INCUBATOR
- Scalable ML library implemented as **Hive user-defined functions** (UDFs)

UDF

column 1
aaa
bbb
ccc

column 1’
xxx
yyy
zzz

- `l1_normalize()`

UDAF (aggregation)

column 1
aaa
bbb
ccc

column 2
scalar

- `rmse()`

UDTF (tabular)

column 1
aaa
bbb
ccc

column 2	column 3
xxx	*111*
yyy	222

- `train_regressor()`

Apache **Hivemall**

Easy-to-use

ML in SQL

Scalable

Runs in parallel on Hadoop ecosystem

Versatile

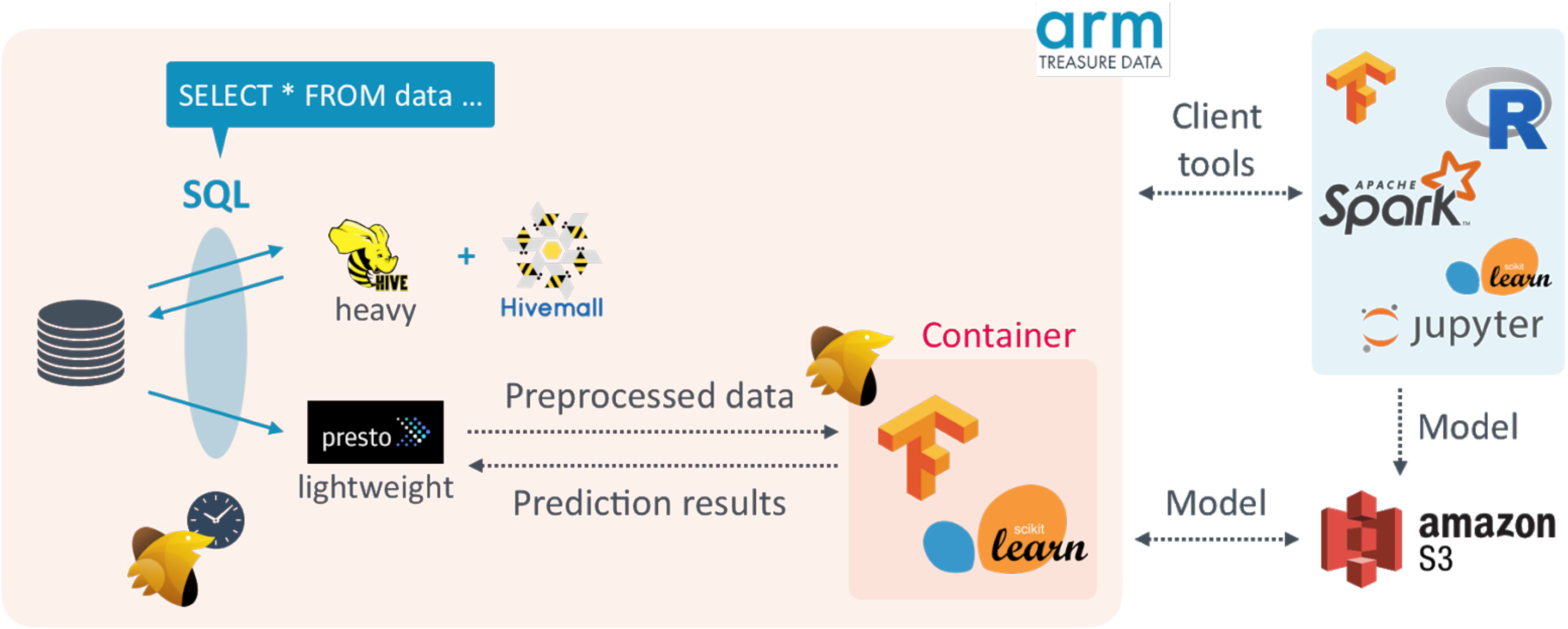
Efficient, generic functions

Multi-platform

Hive, Spark, Pig

Use case #1: Enterprise Big Data analytics platform

Hivemall makes ML more simple, handy on arm TREASURE DATA



Use case #2: Large-scale recommender systems

Demo paper @ ACM RecSys 2018

Query-Based Simple and Scalable Recommender Systems with Apache Hivemall

Takuya Kitazawa
Treasure Data, Inc.
kitazawa@treasure-data.com

Makoto Yui
Treasure Data, Inc.
myui@treasure-data.com

ABSTRACT

This study demonstrates a way to build large-scale recommender systems by just writing a series of SQL-like queries. In order to efficiently run recommendation logics on a cluster of computers, we implemented a variety of recommendation algorithms and common recommendation functions (e.g., efficient similarity computation, top-k retrieval, and evaluation measures) as Hive user-defined functions (UDFs) in Apache Hivemall. We demonstrate that how Apache Hivemall can easily be used for building a scalable recommendation system with satisfying business requirements such as scalability, latency, and stability.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; *Query languages*; MapReduce languages; Top-k retrieval in databases;

KEYWORDS

Hadoop; Hive; Spark; Factorization methods; Top-k item recommendation

ACM Reference Format:

Takuya Kitazawa and Makoto Yui. 2018. Query-Based Simple and Scalable Recommender Systems with Apache Hivemall. In *Twelfth ACM Confer-*

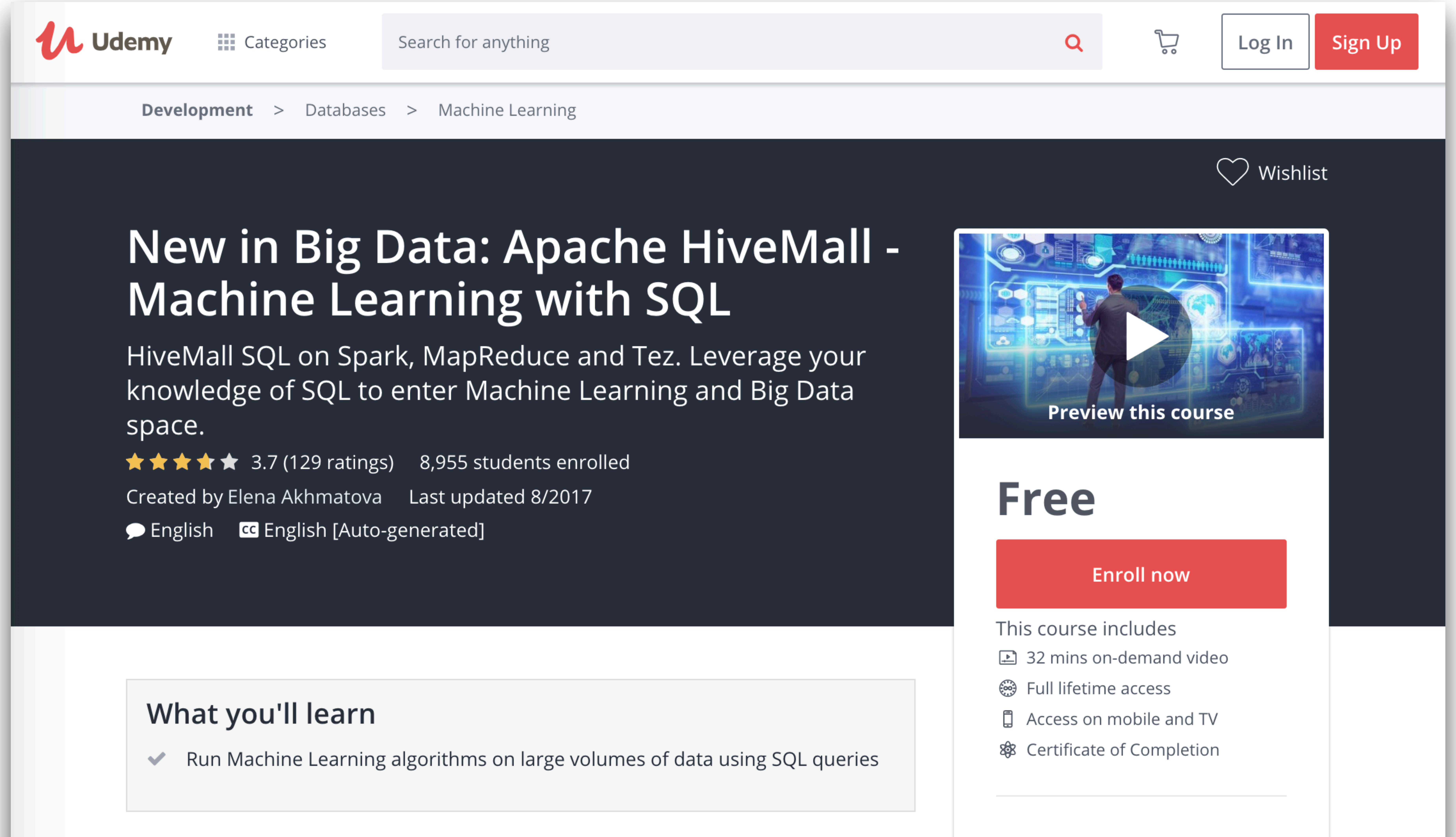
This paper provides an alternative way for building a scalable recommender system using **Apache Hivemall**, a scalable machine learning and recommendation library that runs on Apache Hive and Spark [4]. We demonstrate a unique query-based recommender system which is particularly suitable for large-scale user-item interactions aggregated as tables on Apache Hive, a data warehouse environment built on the top of Apache Hadoop. Apache Hivemall enables us to easily implement various recommendation logics from well-studied techniques to up-to-date algorithms in a scalable manner just by issuing series of HiveQL queries.

2 APACHE HIVEMALL

In addition to recommendation algorithms, Apache Hivemall has a variety of functionalities including regression, classification, anomaly detection, TF-IDF computation, top-k data processing, and natural language processing; some of them are particularly useful for building a recommendation engine. As illustrated in Figure 1, Hivemall is built on the Hadoop ecosystem so that it can leverage the computational efficiency of Hadoop for large-scale data processing. Notice that Hivemall is flexible at the choice of each layer; it can use Apache Tez, Apache Spark, or the plain old MapReduce runtime for parallel data processing.

Use case #3: E-learning

"New in Big Data" Machine Learning with SQL @ Udemy



Easy-to-use

ML in SQL

Scalable

Runs in parallel on Hadoop ecosystem

Versatile

Efficient, generic functions

Multi-platform

Hive, Spark, Pig

Example: Scalable Logistic Regression written in ~10 lines of queries

mahout

```java
/**
 * Train a logistic regression for the examples from Chapter 13 of Mahout in Action
 */
public final class TrainLogistic {

  private static String inputFile;
  private static String outputFile;
  private static LogisticModelParameters lmp;
  private static int passes;
  private static boolean scores;
  private static OnlineLogisticRegression model;

  private TrainLogistic() {
  }

  public static void main(String[] args) throws Exception {
    mainToOutput(args, new PrintWriter(new OutputStreamWriter(System.out, Charsets.UTF_8), true));
  }

  static void mainToOutput(String[] args, PrintWriter output) throws Exception {
    if (parseArgs(args)) {
      double logPEstimate = 0;
      int samples = 0;

      CsvRecordFactory csv = lmp.getCsvRecordFactory();
      OnlineLogisticRegression lr = lmp.createRegression();
      for (int pass = 0; pass < passes; pass++) {
        try (BufferedReader in = open(inputFile)) {
          // read variable names
          csv.firstLine(in.readLine());

          String line = in.readLine();
          while (line != null) {
            // for each new line, get target and predictors
            Vector input = new RandomAccessSparseVector(lmp.getNumFeatures());
            int targetValue = csv.processLine(line, input);

            // check performance while this is still news
            double logP = lr.logLikelihood(targetValue, input);
            if (!Double.isInfinite(logP)) {
              if (samples < 20) {
                logPEstimate = (samples * logPEstimate + logP) / (samples + 1);
              } else {
                logPEstimate = 0.95 * logPEstimate + 0.05 * logP;
              }
              samples++;
            }
            double p = lr.classifyScalar(input);
            if (scores) {
              output.printf(Locale.ENGLISH, "%10d %2d %10.2f %2.4f %10.4f %10.4f%n",
```

Hivemall

```sql
select
  feature,
  avg(weight) as weight
from (
  select
    logress(features, label) as (feature, weight)
  from (
    select features, label
    from train_oversampling
    CLUSTER BY rand(1) -- random shuffling
  ) t1
) t2
group by feature;
```

Automatically runs in **parallel** on Hadoop

Easy-to-use

ML in SQL

Scalable

Runs in parallel on Hadoop ecosystem

Versatile

Efficient, generic functions

Multi-platform

Hive, Spark, Pig

Feature engineering

- Feature hashing
- Feature scaling (normalization, z-score)
- Feature binning
- TF-IDF vectorizer
- Polynomial expansion
- Amplifier

Evaluation metrics

- AUC, nDCG, log loss, precision, recall, …

Array, vector, map

- Concatenation
- Intersection
- Remove
- Sort
- Average
- Sum
- …

From/To JSON conversion

Bit, compress, character encoding

Efficient top-k query processing

Efficient top-k retrieval

Internally hold bounded priority queue

List top-3 items per user:

item	user	score
1	B	70
2	A	80
3	A	90
4	B	60
5	A	70
…	…	…

```
SELECT
  item, user, score, rank
FROM (
  SELECT
    item, user, score,
    rank() over (PARTITION BY user ORDER BY score DESC)
      as rank
  FROM
    table
) t
WHERE rank <= 2
```

Not finish in 24 hrs. for 20M users and ~1k items in each

```
SELECT
  each_top_k(
    2, user, score,
    user, item -- output columns
  ) as (rank, score, user, item)
FROM (
  SELECT * FROM table
  CLUSTER BY user
) t
```

Finish in 2 hrs.

Recommendation with Hivemall

k-nearest-neighbor

- MinHash and b-Bit MinHash (LSH)
- Similarities
  - Euclid
  - Cosine
  - Jaccard
  - Angular

Matrix completion

- Matrix Factorization
- Factorization Machines

Efficient item-based collaborative filtering

- Sparse Linear Method (SLIM)
- Approximated all-pair similarities (DIMSUM)

Natural Language Processing — English, Japanese and Chinese tokenizer, word N-grams, …

- ```
  select tokenize('Hello, world!')
  ```
  ["Hello", "world"]

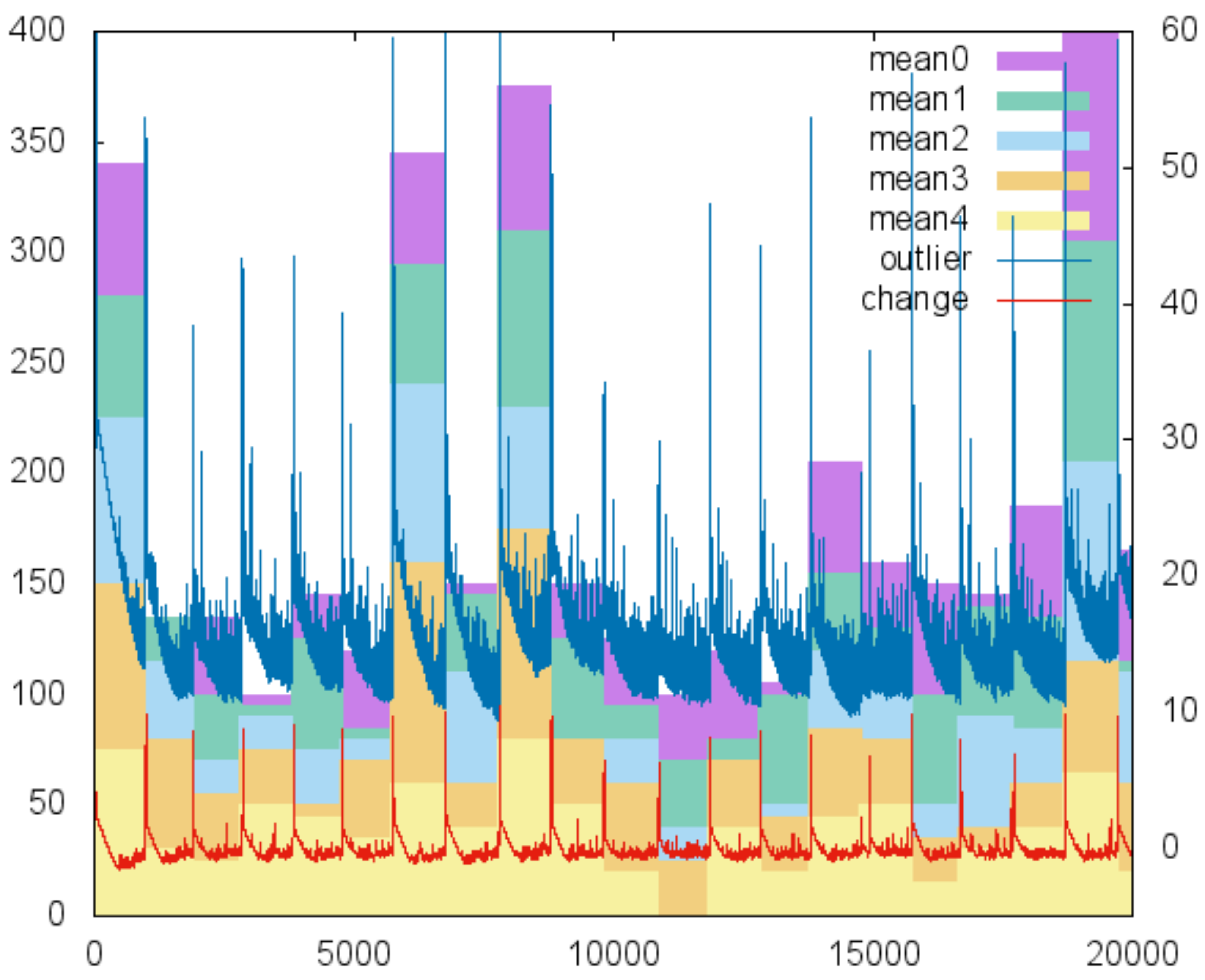
- ```
  select singularize('apples')
  ```
  apple

Geospatial functions

```
SELECT
   map_url(lat, lon, zoom) as osm_url,
   map_url(lat, lon, zoom,'-type googlemaps') as gmap_url
FROM (
  SELECT 51.51202 as lat, 0.02435 as lon, 17 as zoom
  UNION ALL
  SELECT 51.51202 as lat, 0.02435 as lon, 4 as zoom
) t
```

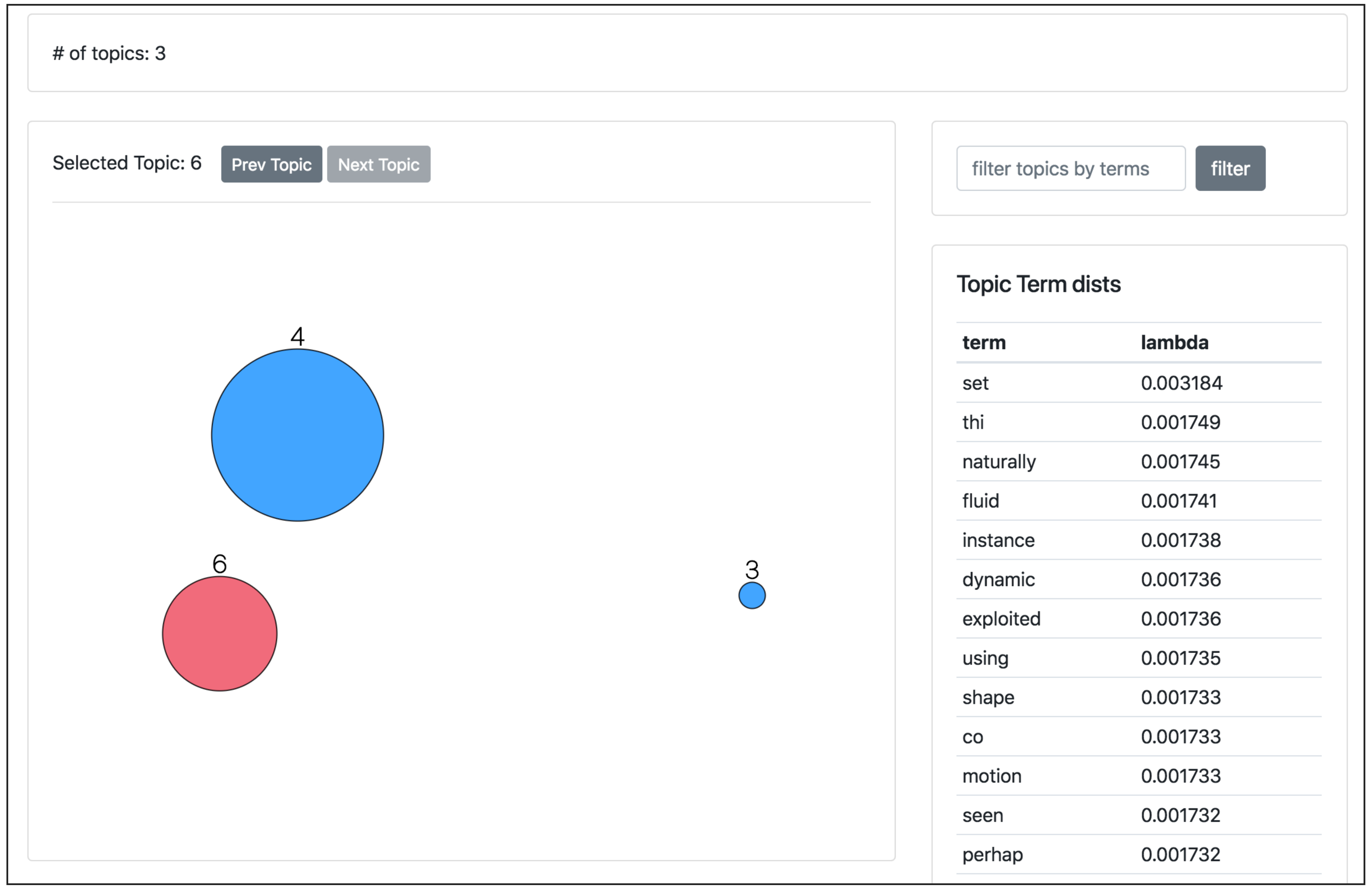
Anomaly / Change-point detection

- Local outlier factor (k-NN-based technique)
- ChangeFinder
- Singular Spectrum Transformation



Clustering / Topic modeling

- Latent Dirichlet Allocation
- Probabilistic Latent Semantic Analysis



Sketching

- Approximated distinct count:

```
SELECT count(distinct user_id) FROM t
```

```
SELECT approx_count_distinct(user_id) FROM t
```

- Bloom filtering:

```
WITH high_rated_items as (
  SELECT bloom(itemid) as items
  FROM (
    SELECT itemid
    FROM ratings
    GROUP BY itemid
    HAVING avg(rating) >= 4.0
  ) t
)
SELECT
  l.rating,
  count(distinct l.userid) as cnt
FROM
  ratings l
  CROSS JOIN high_rated_items r
WHERE
  bloom_contains(r.items, l.itemid)
GROUP BY
  l.rating;
```

Build Bloom Filter (i.e., probabilistic set of) high-rated items

Check if item is in Bloom Filter, and see their actual ratings:

	.# rating	# cnt
1	1.0	1296
2	2.0	2770
3	3.0	5008
4	4.0	5824
5	5.0	5925

Easy-to-use

ML in SQL

Scalable

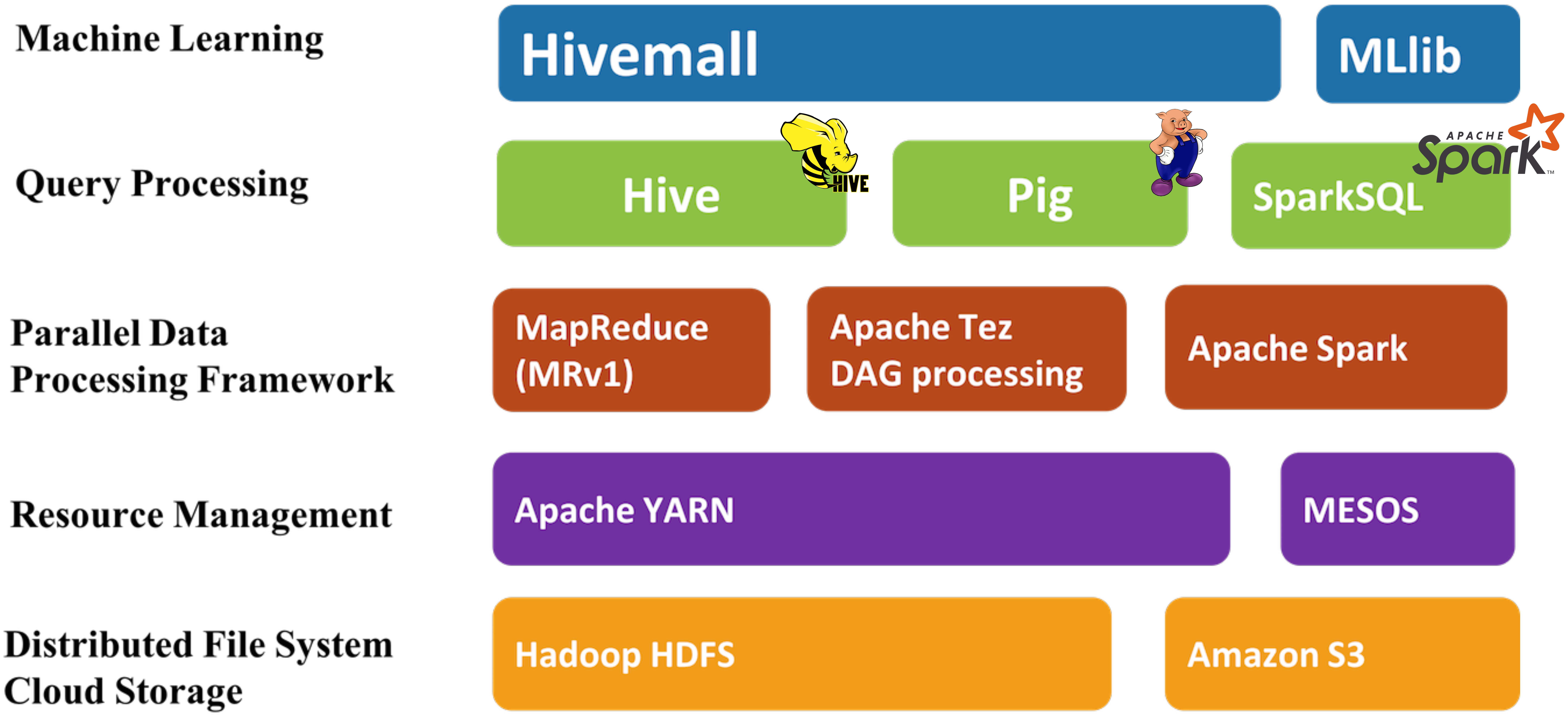
Runs in parallel on Hadoop ecosystem

Versatile

Efficient, generic functions

Multi-platform

Hive, Spark, Pig


Machine Learning
Hivemall
MLlib
Query Processing
Hive
HIVE
Pig
SparkSQL
APACHE
Spark
Parallel Data
Processing Framework
MapReduce
(MRv1)
Apache Tez
DAG processing
Apache Spark
Resource Management
Apache YARN
MESOS
Distributed File System
Cloud Storage
Hadoop HDFS
Amazon S3

Apache Hive

```
CREATE TABLE lr_model
AS
SELECT
 feature,
 avg(weight) as weight
FROM (
  SELECT
    logress(features, label, "-total_steps ${total_steps}") as (feature, weight)
  FROM
    training
) t
GROUP BY feature;
```

Apache Pig

```
a = load 'a9a.train'
       as (rowid:int, label:float, features:{(featurepair:chararray)});

b = foreach a generate flatten(
       logress(features, label, '-total_steps ${total_steps}')
    ) as (feature, weight);

c = group b by feature;

d = foreach c generate group, AVG(b.weight);
store d into 'a9a_model';
```

Apache Spark

Query in **`HiveContext`**

```
context = HiveContext(sc)

context.sql("
    SELECT
     feature,
     avg(weight) as weight
    FROM (
      SELECT
        train_logregr(features, label) as (feature, weight)
      FROM
        training
    ) t
    GROUP BY feature
  ")
```

Introduction to Apache **Hivemall**

How Hivemall Works with **PySpark**

Hivemall <3 Python

Installation and creating SparkSession

```
$ wget -q http://mirror.reverse.net/pub/apache/incubator/hivemall/0.5.2-incubating/
hivemall-spark2.x-0.5.2-incubating-with-dependencies.jar
```

```python
from pyspark.sql import SparkSession

spark = SparkSession \
          .builder \
          .master('local[*]') \
          .config('spark.jars',
                  'hivemall-spark2.x-0.5.2-incubating-with-dependencies.jar') \
          .enableHiveSupport() \
          .getOrCreate()
```

Register Hive(mall) UDF to SparkSession

```
spark.sql("""
CREATE TEMPORARY FUNCTION hivemall_version AS 'hivemall.HivemallVersionUDF'
""")

spark.sql("SELECT hivemall_version()").show()
```

```
+------------------+
|hivemall_version()|
+------------------+
|  0.5.2-incubating|
+------------------+
```

See resources/ddl/define-all.spark in Hivemall repository for list of all UDFs

Preprocessing

Training

Prediction

Evaluation

Example: Binary classification for churn prediction

```
import re
import pandas as pd

df = spark.createDataFrame(
    pd.read_csv('churn.txt').rename(lambda c: re.sub(r'[^a-zA-Z0-9 ]', '',
str(c)).lower().replace(' ', '_'), axis='columns'))
```

OR

```
df = spark.read.option('header', True).schema(schema).csv('churn.txt')
```

Ab phone
399-5763
340-9449
363-1123
361-2170
406-6304

...

Ab state	# area_code	.# day_mins	# day_calls	# account_length	.# intl_charge	.# eve_charge	# night_calls	Ab churn
IA	415	211.3	87	45	3.59	14.08	72	False.
VT	408	219.4	112	100	3.24	19.18	95	False.
NY	415	190.4	91	94	3.67	7.82	108	False.
LA	415	147.7	94	128	1.86	24.08	124	False.
SC	408	229.9	130	181	3.83	12.27	110	False.

Preprocessing

Training

Prediction

Evaluation

```python
>>> df.createOrReplaceTempView('churn')

>>> df_preprocessed = spark.sql("""
SELECT
  phone,
  array_concat( -- Concatenate features as a feature vector
    categorical_features( -- Create categorical features
      array('intl_plan', 'state', 'area_code', 'vmail_plan'),
      intl_plan, state, area_code, vmail_plan
    ),
    quantitative_features( -- Create quantitative features
      array(
        'night_charge', 'day_charge', 'custserv_calls',
        'intl_charge', 'eve_charge', 'vmail_message'
      ),
      night_charge, day_charge, custserv_calls,
      intl_charge, eve_charge, vmail_message
    )
  ) as features,
  if(churn = 'True.', 1, 0) as label
FROM
  churn
""")
```

Feature vector = array of string

Array of quantitative features **index** : **value**

```
select quantitative_features(array("price", "size"), 600, 2.5)
```

["price:600.0", "size:2.5"]

Array of categorical features **index** # **value**

```
select categorical_features(array("gender", "category"), "male", "book")
```

[“gender#male", "category#book"]

* NULL is automatically omitted

Hivemall internally does one-hot encoding (e.g., book → 1, 0, 0, ...)

```sql
SELECT
  phone,
  array_concat( -- Concatenate features as a feature vector
    categorical_features( -- Create categorical features
      array('intl_plan', 'state', 'area_code', 'vmail_plan'),
      intl_plan, state, area_code, vmail_plan
    ),
    quantitative_features( -- Create quantitative features
      array(
        'night_charge', 'day_charge', 'custserv_calls',
        'intl_charge', 'eve_charge', 'vmail_message'
      ),
      night_charge, day_charge, custserv_calls,
      intl_charge, eve_charge, vmail_message
    )
  ) as features,
  if(churn = 'True.', 1, 0) as label
FROM
  churn
```

```
['intl_plan#no',
 'state#KS',
 'area_code#415',
 'vmail_plan#yes',
 'night_charge:11.01',
 'day_charge:45.07',
 'custserv_calls:1.0',
 'intl_charge:2.7',
 'eve_charge:16.78',
 'vmail_message:25.0']
```

```
>>>  df_train, df_test = df_preprocessed.randomSplit([0.8, 0.2], seed=31)

>>>  df_train.count(), df_test.count()  # => 2658, 675
```

Preprocessing

Training

Prediction

Evaluation

```
>>> df_train.createOrReplaceTempView('train')

>>> df_model = spark.sql("""
    SELECT
      feature,
      avg(weight) as weight
    FROM (
      SELECT
        train_classifier(
          features,
          label,
          '-loss logloss -opt SGD -reg l1 -lambda 0.03 -eta0 0.01'
        ) as (feature, weight)
      FROM
        train
    ) t
    GROUP BY 1
    """)
```

Aggregate multiple workers' results

Run in parallel on Spark workers

Supervised learning by unified function

```
SELECT
  train_classifier( -- train_regressor(
    features,
    label,
    '-loss logloss -opt SGD -reg no -eta simple -total_steps ${total_steps}'
  ) as (feature, weight)
FROM
  train
```

Classification

- HingeLoss
- LogLoss (a.k.a. *logistic loss*)
- SquaredHingeLoss
- ModifiedHuberLoss

Regression

- SquaredLoss
- QuantileLoss
- EpsilonInsensitiveLoss
- SquaredEpsilonInsensitiveLoss
- HuberLoss

Supervised learning by unified function

```
SELECT
  train_classifier( -- train_regressor(
    features,
    label,
    '-loss logloss -opt SGD -reg no -eta simple -total_steps ${total_steps}'
  ) as (feature, weight)
FROM
  train
```

Optimizer

- SGD
- AdaGrad
- AdaDelta
- ADAM

Regularization

- L1
- L2
- ElasticNet
- RDA

- Iteration with learning rate control
- Mini-batch training
- Early stopping

Model = table

	feature	weight
0	state#TX	0.088104
1	state#MN	0.024230
2	state#LA	-0.024339
3	area_code#408	-0.006044
4	night_charge	-0.194483
5	state#ND	-0.066669
6	state#VT	-0.063177
7	state#MA	0.000016
8	state#OH	0.031865
9	state#MD	0.038399

$$\hat{y}^{\mathrm{LR}}(\mathbf{x}) := w_0 + \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

Preprocessing

Training

Prediction

Evaluation

```
>>>  df_test.createOrReplaceTempView('test')
>>>  df_model.createOrReplaceTempView('model')

>>>  df_prediction = spark.sql("""
     SELECT
       phone,
       label as expected,
       sigmoid(sum(weight * value)) as prob
     FROM (
       SELECT
         phone,
         label,
         extract_feature(fv) AS feature,
         extract_weight(fv) AS value
       FROM
         test
       LATERAL VIEW explode(features) t2 AS fv
     ) t
     LEFT OUTER JOIN model m
       ON t.feature = m.feature
     GROUP BY 1, 2
     """)
```

	phone	expected	prob
0	375-3003	0	0.043165
1	344-4022	0	0.032754
2	356-2992	0	0.000035
3	420-3028	0	0.009459
4	354-9062	0	0.128704
5	360-6024	0	0.000029
6	376-4705	1	0.019202
7	373-1448	1	0.007880
8	349-2157	0	0.009182
9	337-9569	0	0.052731

Preprocessing

Training

Prediction

Evaluation

```
>>>  df_prediction.createOrReplaceTempView('prediction')

>>>  spark.sql("""
     SELECT
       auc(prob, expected) AS auc,
       logloss(prob, expected) AS logloss
     FROM (
       SELECT prob, expected
       FROM prediction
       ORDER BY prob DESC
     """).show()
```

```
+------------------+------------------+
|               auc|           logloss|
+------------------+------------------+
|0.6360049076499652|0.6456736373258979|
+------------------+------------------+
```

Preprocessing

Training — More options

Prediction

Evaluation

Classification and regression with variety of algorithms

Classification

- Generic classifier
- Perceptron
- Passive Aggressive (PA, PA1, PA2)
- Confidence Weighted (CW)
- Adaptive Regularization of Weight Vectors (AROW)
- Soft Confidence Weighted (SCW)
- (Field-Aware) **Factorization Machines**
- **RandomForest**

Regression

- Generic regressor
- PA Regression
- AROW Regression
- (Field-Aware) **Factorization Machines**
- **RandomForest**

Factorization Machines

$$\hat{y}^{\mathrm{FM}}(\mathbf{x}) := \underbrace{w_0}_{\textbf{global bias}} + \underbrace{\mathbf{w}^{\mathrm{T}}\mathbf{x}}_{\textbf{linear}} + \sum_{i=1}^{d}\sum_{j=i}^{d} \underbrace{\mathbf{v}_i^{\mathrm{T}}\mathbf{v}_j}_{\textbf{interaction}} x_i x_j$$

```
SELECT
  train_fm(
    features,
    label,
    '-classification -factor 30 -eta 0.001'
  ) as (feature, Wi, Vij)
FROM
  train
```

S. Rendle. **Factorization Machines with libFM**. ACM Transactions on Intelligent Systems and Technology, 3(3), May 2012.

Factorization Machines

$$\hat{y}^{\mathrm{FM}}(\mathbf{x}) := \underbrace{w_0}_{\text{global bias}} + \underbrace{\mathbf{w}^{\mathrm{T}}\mathbf{x}}_{\text{linear}} + \sum_{i=1}^{d}\sum_{j=i}^{d} \underbrace{\mathbf{v}_i^{\mathrm{T}}\mathbf{v}_j}_{\text{interaction}} x_i x_j$$

	feature	Wi	Vij
0	0	-0.122241	None
1	state#SC	0.002093	[-0.0649222806096077, 0.08943693339824677, 0.1...
2	area_code#408	-0.029035	[-0.019552724435925484, -0.03247314691543579, ...
3	vmail_message	-0.100170	[-0.019371509552001953, 0.01219850592315197, -...
4	state#WV	-0.031847	[0.18128858506679535, 0.08544187247753143, -0....
5	state#NC	0.002695	[-0.03858436271548271, 0.0519322007894516, 0.1...
6	state#KY	0.005622	[-0.05067330598831177, 0.05898619070649147, -0....
7	intl_charge	-0.042708	[0.00014178249693941325, 0.0028802729211747646...
8	state#CO	0.022583	[-0.13534343242645264, -0.1448756605386734, -0...
9	state#VA	-0.020664	[0.2548579275608063, -0.028995800763368607, 0....

RandomForest

Training

```
SELECT
  train_randomforest_classifier(
    feature_hashing(features),
    label,
    '-trees 50 -seed 71' -- hyperparameters
  ) as (model_id, model_weight, model, var_importance, oob_errors, oob_tests)
FROM
  train
```

Simplify name of quantitative feature **index** and categorical feature **index** # **value**

```
select feature_hashing(array("price:600", "category#book"))
```

→ ["14142887:600", "10413006"]

RandomForest

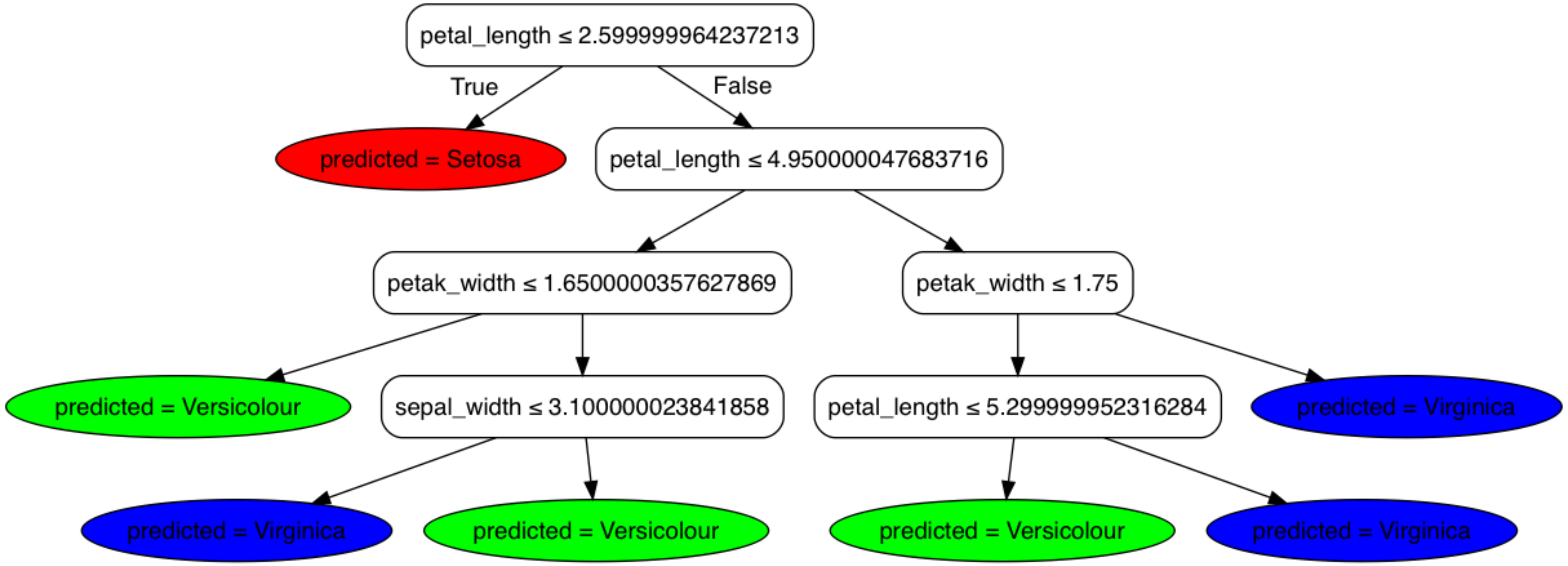
Model table

	model_id	model_weight	model	var_importance
0	d51b7642-79f7-4750-8e38-fb3e865d1eef	0.837838	I?XPI/%dsLd{LdDfaZTPM:)_@^i12w<Ba)_j40icEw3jYE...	{15900165: 1.8062073885289325}
1	b13adae2-6f38-442a-8b89-5324932acc13	0.816498	I?{Pt:IYsLd{3Ze;DYt:3%}g`]F:N}gYMLrRqViFPHu._X...	{15900165: 2.7499708167499106}
2	8a318077-cc66-479a-9505-a4f9b9ec9084	0.851724	I?{PI/%dsLd{2Yw+<LzSF`agXnUPzzou^7YI[Pws[y~yzD...	{15900165: 2.4154606004599684}
3	97b68bc3-dd08-4d16-9600-238df97c34c7	0.807692	I?{P)`IYwL$Iyz6!AGW.%p~UV=-B)WoH=;0c8uOOj+{BiM...	{15900165: 3.886217162213024}
4	b21b9d2a-ee23-48e0-9728-f4497be4d688	0.824503	I?hROIIYsL$I+Z>ZwzRhE`><L`+-vZ>D]1UbIfV48c=CI4...	{15900165: 3.422917848419591}

RandomForest

Export decision trees for visualization

```
SELECT
  tree_export(model, "-type javascript", ...) as js,
  tree_export(model, "-type graphvis", ...) as dot
FROM
  rf_model
```



RandomForest

Prediction

```
SELECT
  phone,
  rf_ensemble(predicted.value, predicted.posteriori, model_weight) as predicted
FROM (
  SELECT
    t.phone,
    m.model_weight,
    tree_predict(m.model_id, m.model, feature_hashing(t.features), true) as predicted
  FROM
    test t
  CROSS JOIN
    rf_model m
) t1
GROUP BY phone
```

Introduction to Apache **Hivemall**

How Hivemall Works with **PySpark**

Hivemall <3 Python

APACHECON

Keep **Scalable**, Make *More* **Programmable**

Preprocessing

Training

Prediction

Evaluation

```python
from pyspark.ml.feature import MinMaxScaler
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(
    inputCols=['account_length'],
    outputCol="account_length_vect"
)
scaler = MinMaxScaler(
    inputCol="account_length_vect",
    outputCol="account_length_scaled"
)

pipeline = Pipeline(stages=[assembler, scaler])
pipeline.fit(df) \
        .transform(df) \
        .select([
          'account_length', 'account_length_vect',
          'account_length_scaled'
        ]).show()
```

```
+--------------+-------------------+---------------------+
|account_length|account_length_vect|account_length_scaled|
+--------------+-------------------+---------------------+
|           128|            [128.0]| [0.5247933884297521]|
|           107|            [107.0]| [0.4380165289256198]|
|           137|            [137.0]| [0.5619834710743802]|
|            84|             [84.0]| [0.34297520661157...|
|            75|             [75.0]| [0.30578512396694...|
```

Preprocessing

Training

Prediction

Evaluation

```
q = """
SELECT
  feature,
  avg(weight) as weight
FROM (
  SELECT
    train_classifier(
      features,
      label,
      '-loss logloss -opt SGD -reg l1 -lambda {0} -eta0 {1}'
    ) as (feature, weight)
  FROM
    train
) t
GROUP BY 1
"""

hyperparams = [
    (0.01, 0.01),
    (0.03, 0.01),
    (0.03, 0.03),
    ( 0.1, 0.03)
    # ...
]

for reg_lambda, eta0 in hyperparams:
  sql.spark(q.format(reg_lambda, eta0))
```

Preprocessing

Training

Prediction

Evaluation

```
from pyspark.mllib.evaluation import BinaryClassificationMetrics

metrics = BinaryClassificationMetrics(
    df_prediction.select(
        df_prediction.prob,
        df_prediction.expected.cast('float')
    ).rdd.map(tuple)
)

metrics.areaUnderPR, metrics.areaUnderROC
# => (0.25783248058994873, 0.6360049076499648)
```

Preprocessing

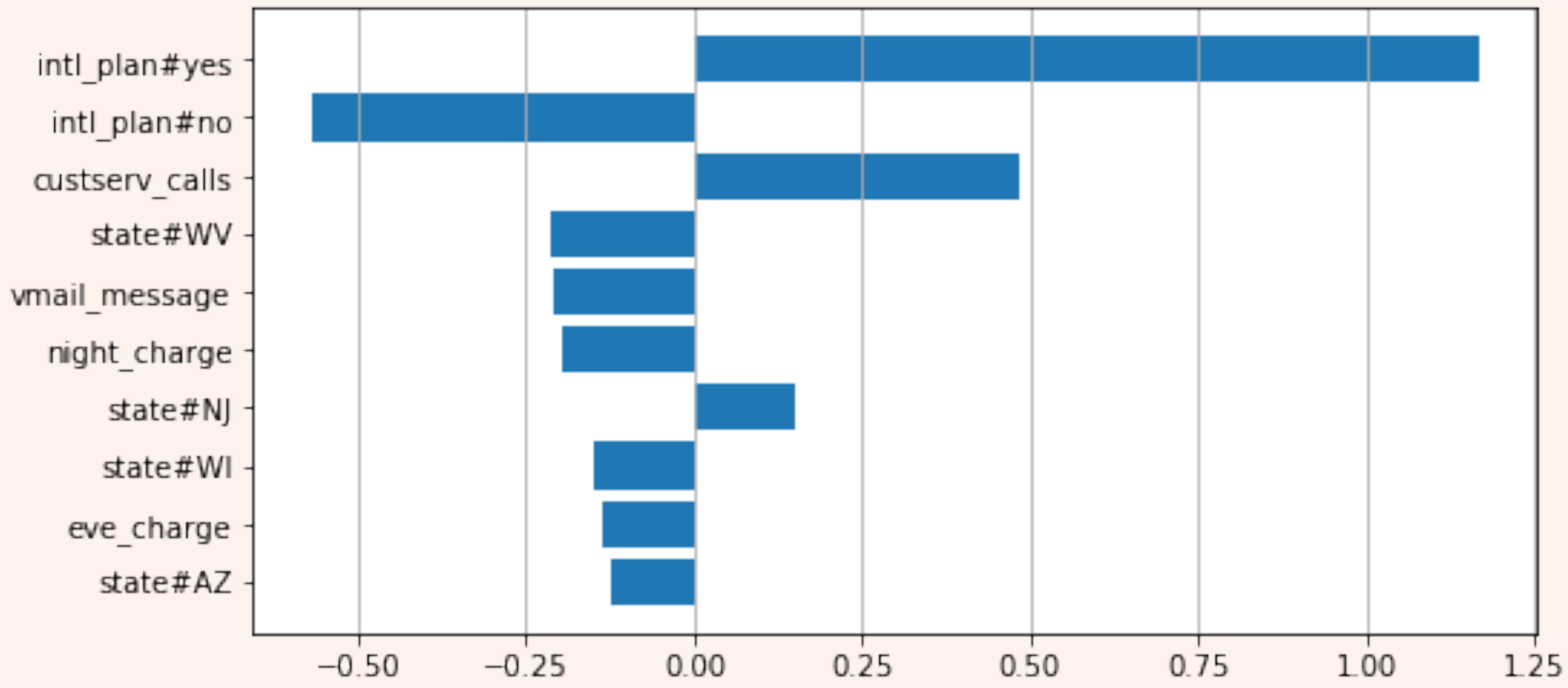
Training

Prediction

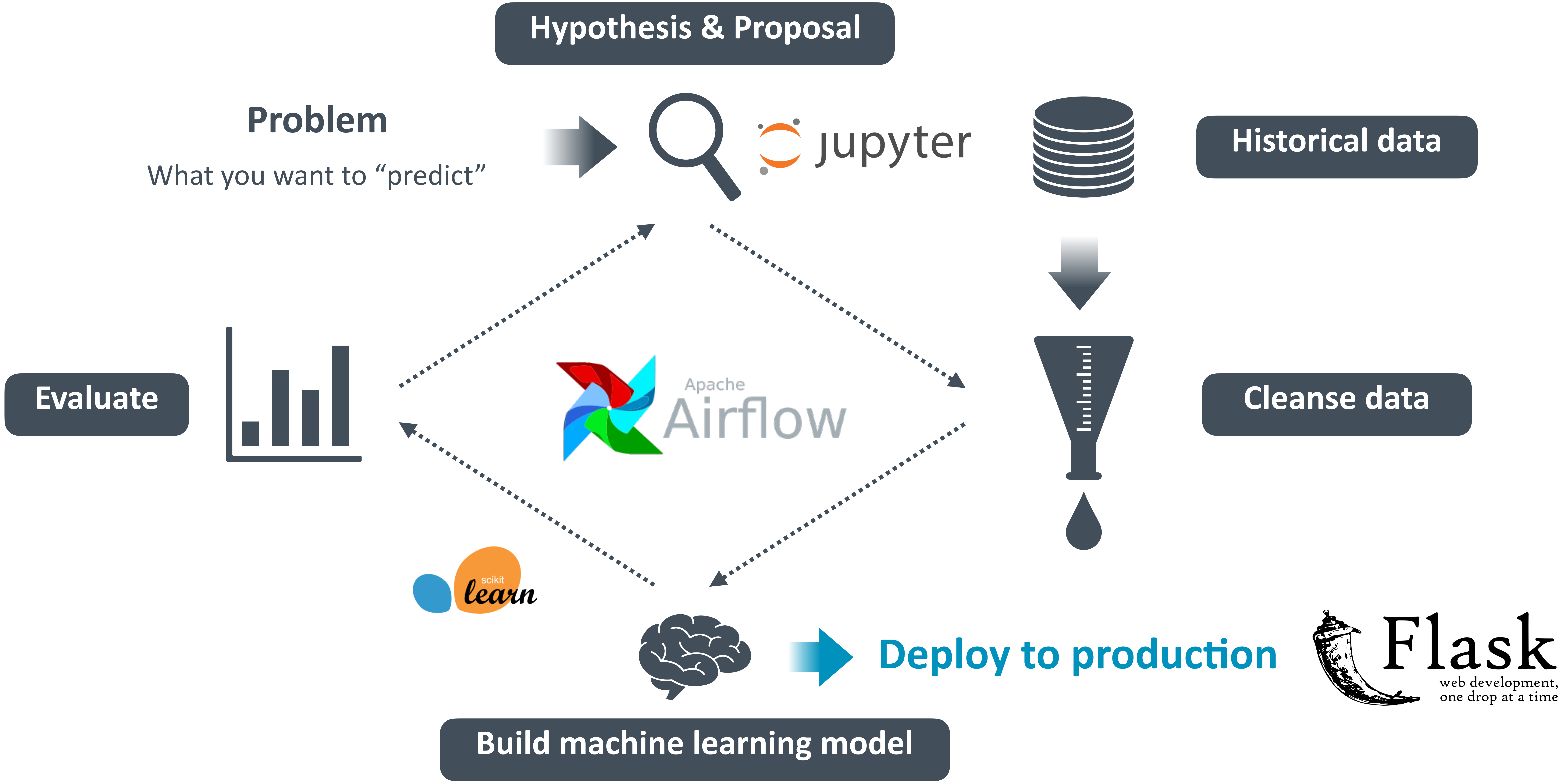
Evaluation

```python
import pyspark.sql.functions as F
df_model_top10 = df_model \
                    .orderBy(F.abs(df_model.weight).desc()) \
                    .limit(10) \
                    .toPandas()

import matplotlib.pyplot as plt
# ...
```



From EDA to production, Python adds flexibility to Hivemall



Hivemall | APACHECON EUROPE

Apache Hivemall Meets PySpark

Scalable Machine Learning with Hive, Spark, and Python

github.com/**apache/incubator-hivemall**

bit.ly/**2o8BQJW**

Takuya Kitazawa: **takuti@apache.org** / **@takuti**