

Machine learning in finance using Spark ML pipeline

梁堰波@明略数据

@DataScientist

October 2015

Who am I ?



Outline

Spark and ML/MLlib background

Spark ML pipeline

Hyperparameter tuning

Spark ML/MLlib feature transformers & algorithms

Financial user cases

Credit scoring case

Spark background

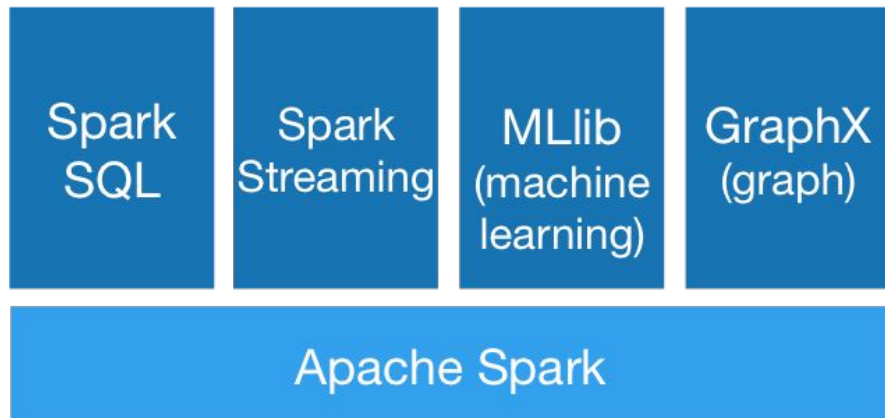
Distributed computing engine

Apache open source

Built for speed, ease of use, and sophisticated analytics

Resilient Distributed Dataset (RDD)

Expressive APIs in Python, Java, Scala and R



Machine learning in Spark

Spark is first general purpose big data processing engine build for ML from day one

The initial design in Spark was driven by ML optimization

- Caching - For running on data multiple times

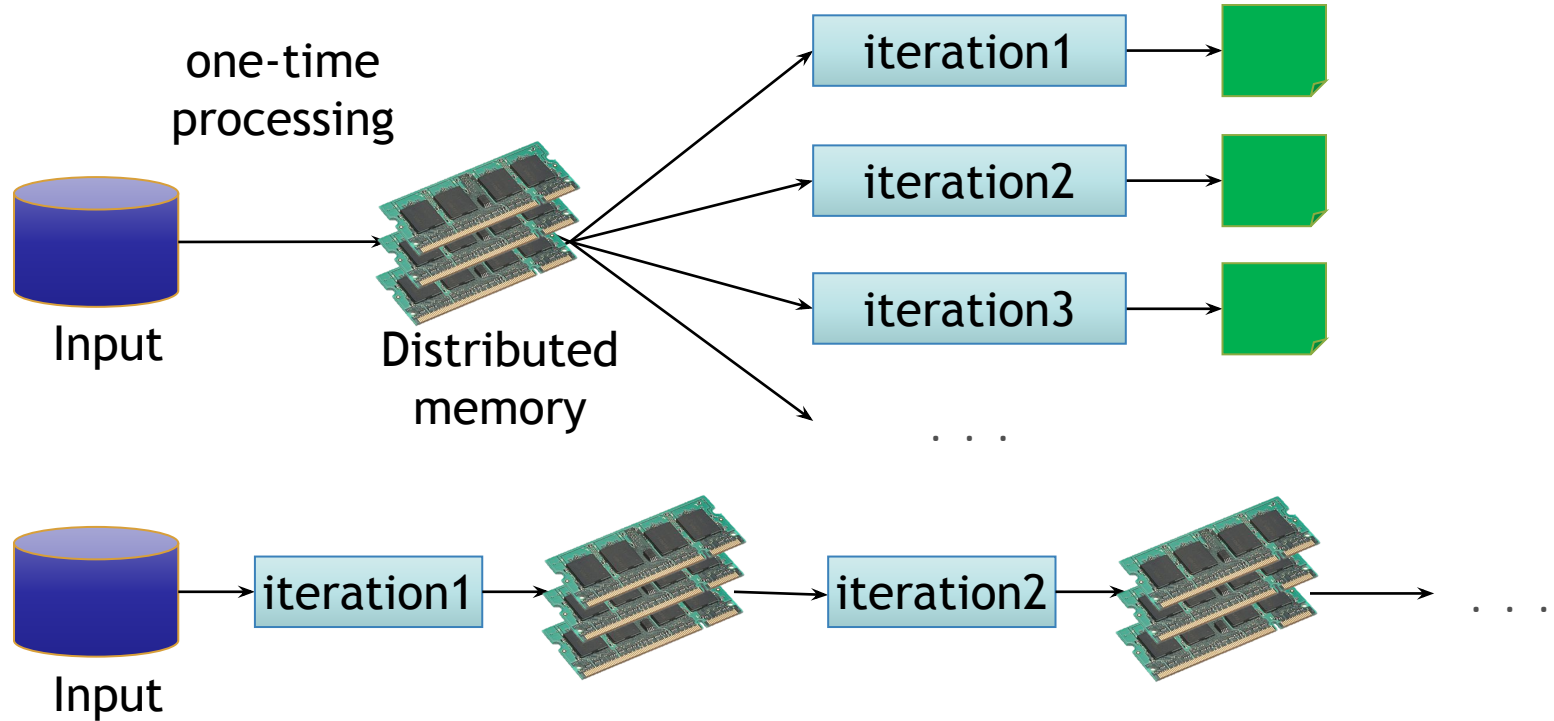
- Accumulator - To keep state across multiple iterations in memory

- Good support for CPU intensive tasks with laziness

- Aggregate & TreeAggregate

One of the examples in Spark first version was of ML

Key: Keep Working Set in RAM



Spark for Data Science

DataFrames

- Intuitive manipulation of distributed structured data

- Familiar API based on R & Python Pandas

- Distributed, optimized implementation

Machine Learning Pipelines

- Integration with DataFrames

- Familiar API based on scikit-learn

- Simple parameter tuning

ML Workflows are complex

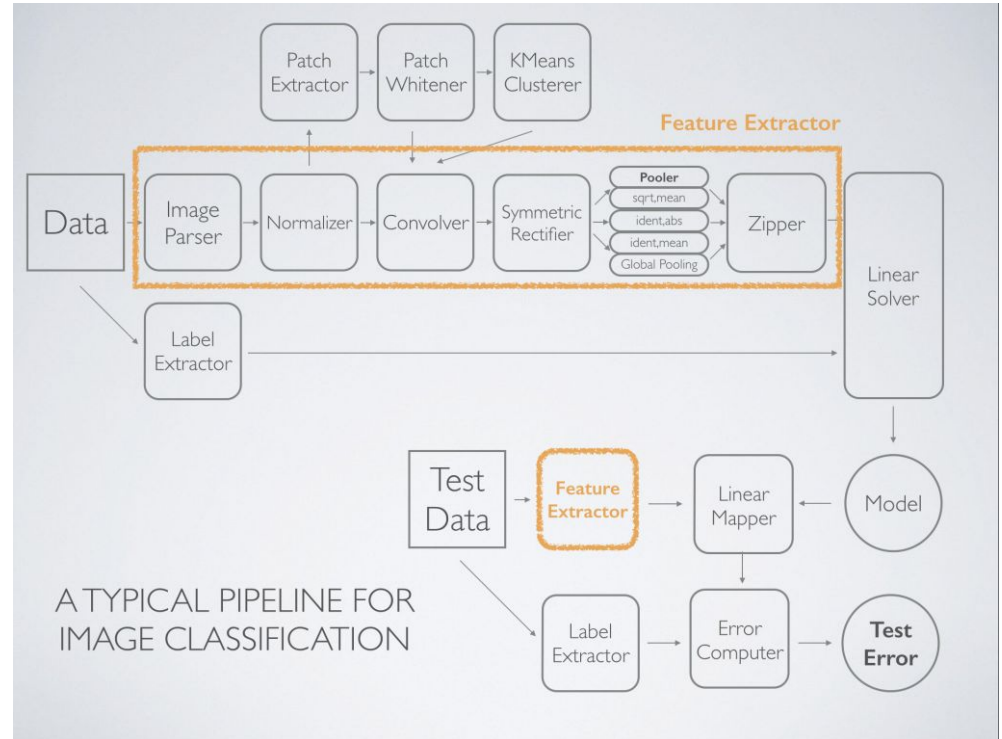
Image classification pipeline

Specify pipeline

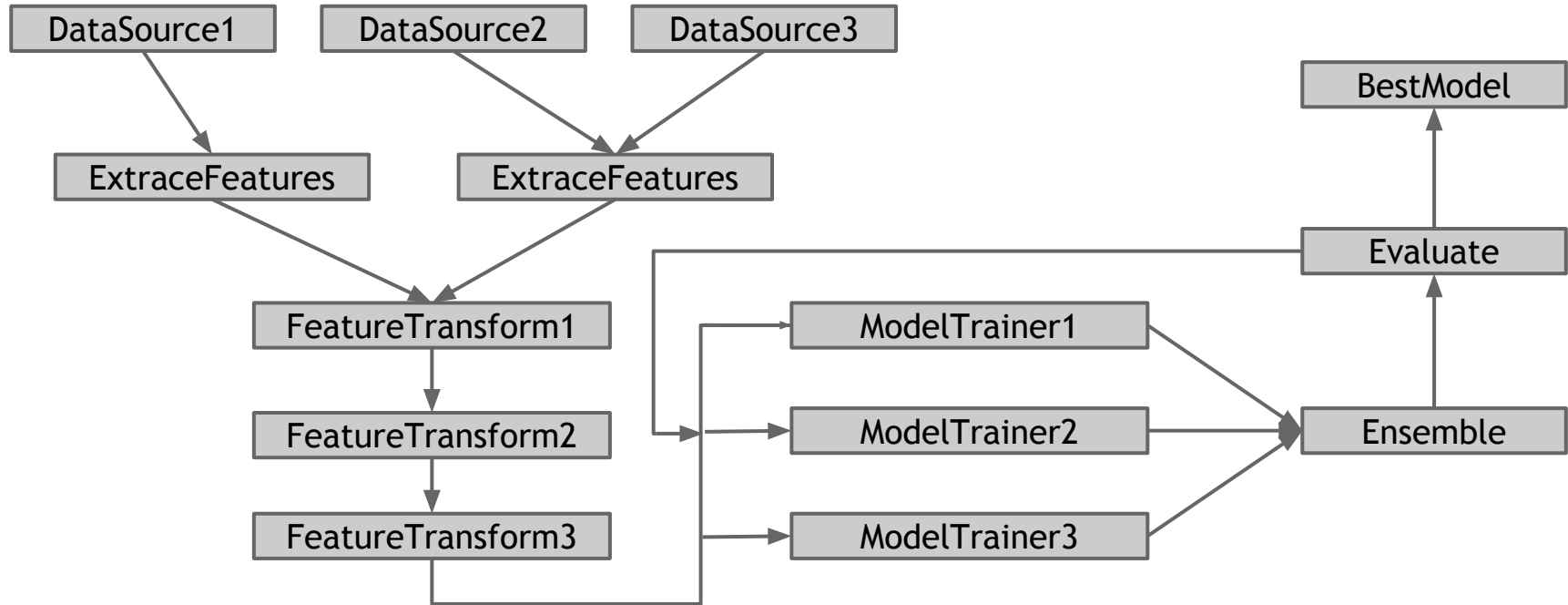
Inspect & debug

Re-run on new data

Tune parameters



ML Workflow are complex



Key abstraction of Spark ML pipeline

Transformer

Feature transformers (e.g., `OneHotEncoder`) and trained ML models (e.g., `LogisticRegressionModel`).

Estimator

ML algorithms for training models (e.g., `LogisticRegression`)

Evaluator

These evaluate predictions and compute metrics, useful for tuning algorithm parameters (e.g., `BinaryClassificationEvaluator`).

Example

PipelineModel
(Transformer)



PipelineModel
.transform()



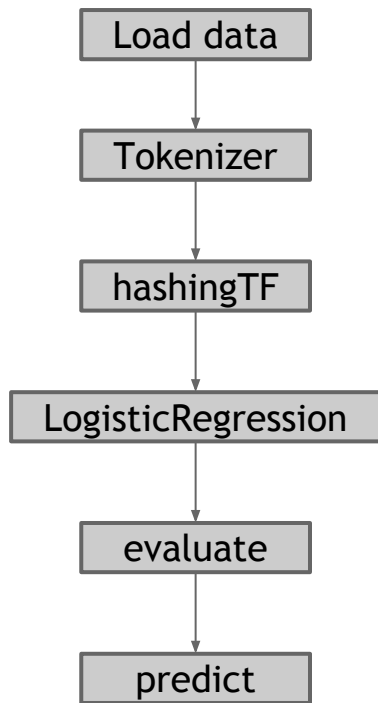
Raw
text

Words

Feature
vectors

Predictions

Load data



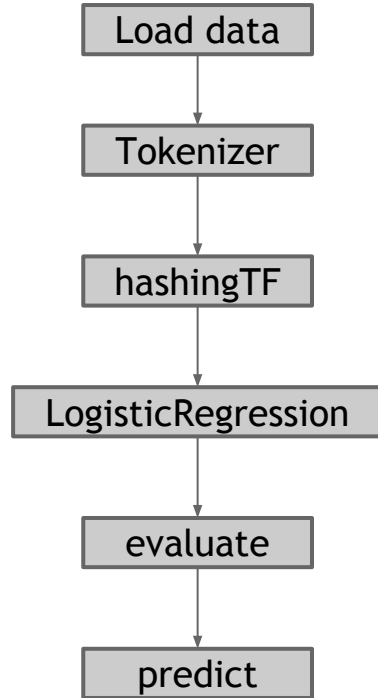
Data sources for DataFrames



LibSVMRelation

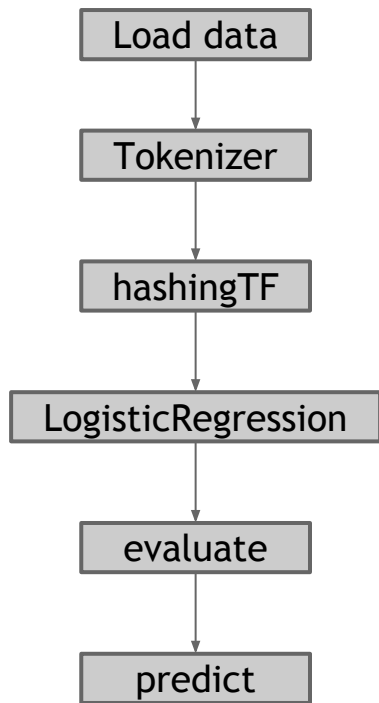
```
val df = sqlContext.read.format("libsvm").load(path)
```

Load data



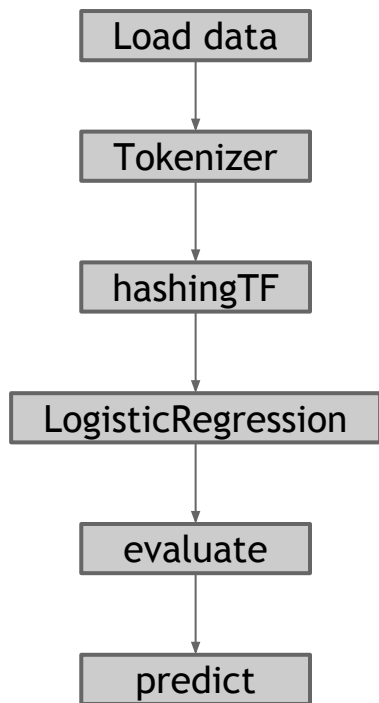
label	Int
text	String

Feature transform



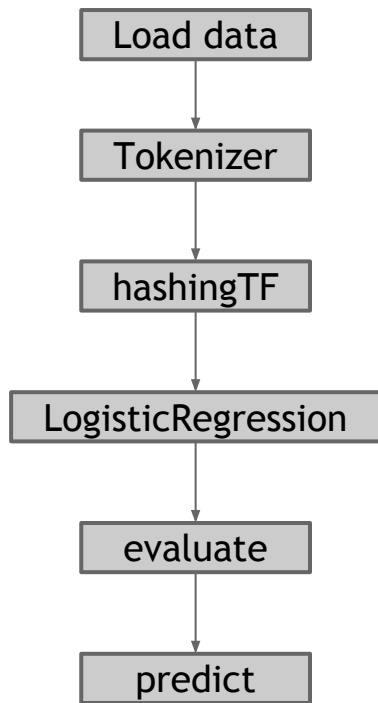
label	Int
words	Seq[String]

Feature transform



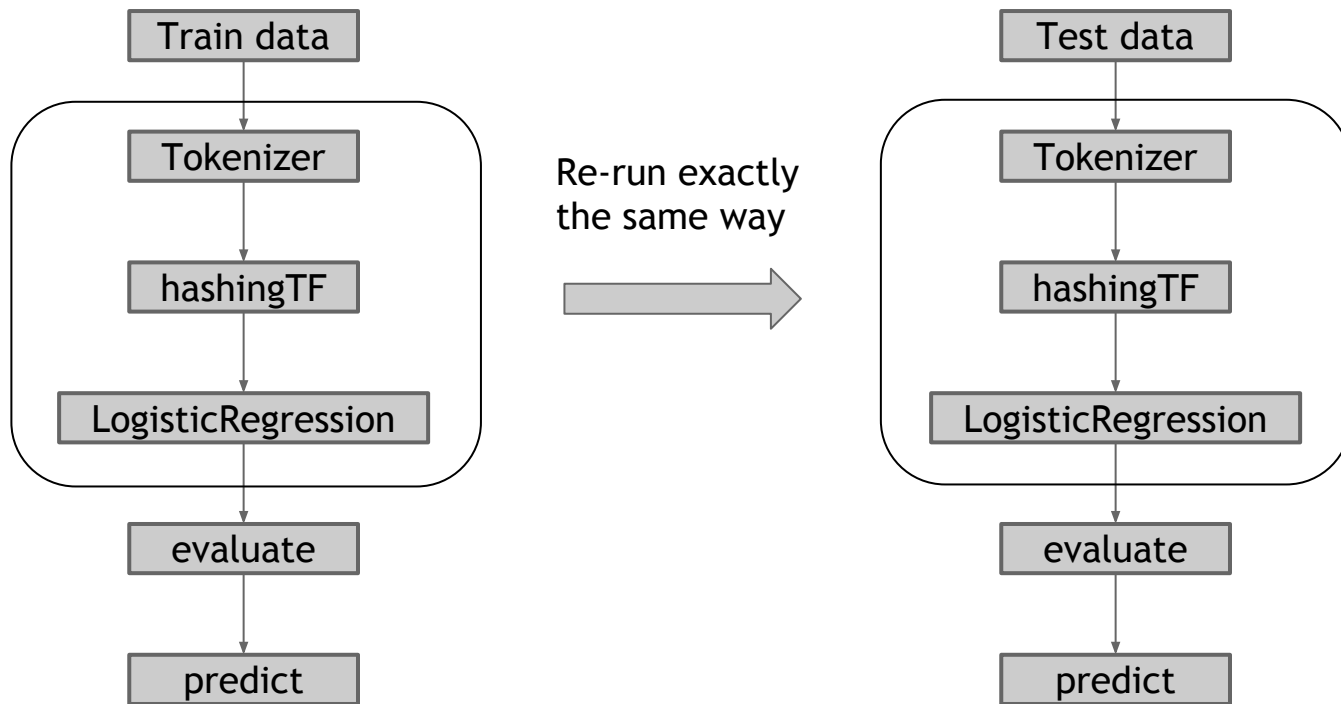
label	Int
words	Vector

Train and evaluate model



label	Int
features	Vector
prediction	Int

Train and evaluate model

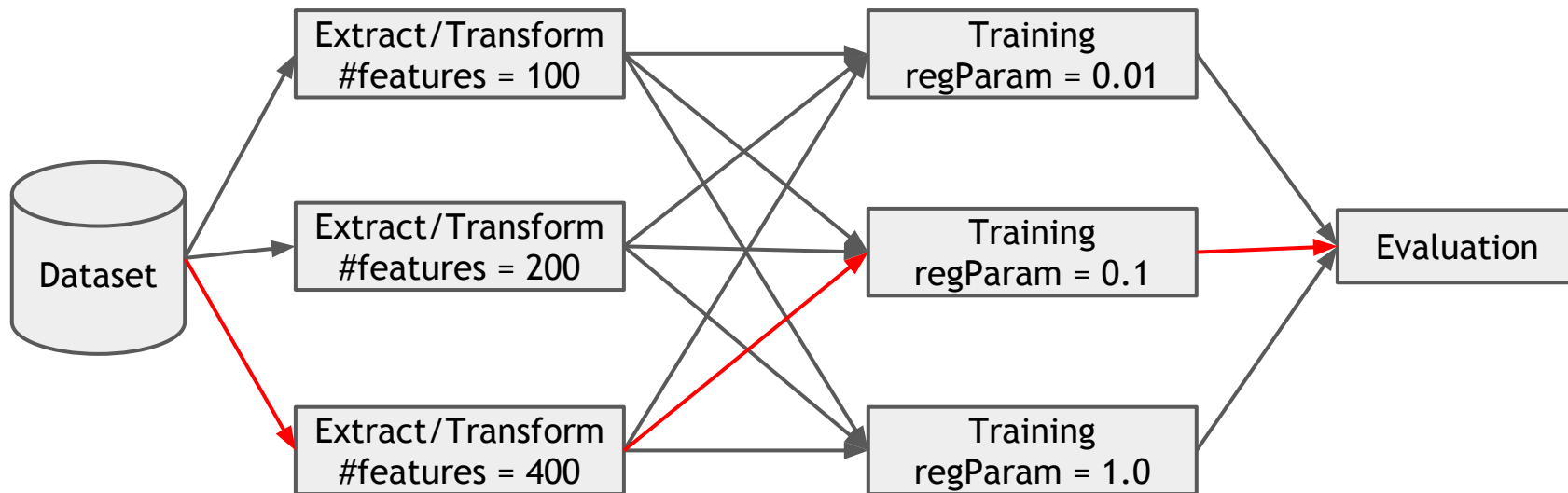


Concise code

```
val tokenizer = new Tokenizer()  
    .setInputCol("text")  
    .setOutputCol("words")  
val hashingTF = new HashingTF()  
    .setNumFeatures(1000)  
    .setInputCol(tokenizer.getOutputCol())  
    .setOutputCol("features")
```

```
val lr = new LogisticRegression()  
    .setMaxIter(10)  
    .setRegParam(0.01)  
val pipeline = new Pipeline()  
    .setStages(Array(tokenizer, hashingTF, lr))  
val model = pipeline.fit(trainingDataset)  
model.transform(testDataset)
```

Hyperparameter tuning



Cross validation

Given:

Estimator

Parameter grid

Evaluator

Find best parameters or models

```
// Build a parameter grid.
val paramGrid = new ParamGridBuilder()
  .addGrid(hashingTF.numFeatures, Array(10, 20, 40))
  .addGrid(lr.regParam, Array(0.01, 0.1, 1.0))
  .build()

// Set up cross-validation.
val cv = new CrossValidator()
  .setNumFolds(3)
  .setEstimator(pipeline)
  .setEstimatorParamMaps(paramGrid)
  .setEvaluator(new BinaryClassificationEvaluator)

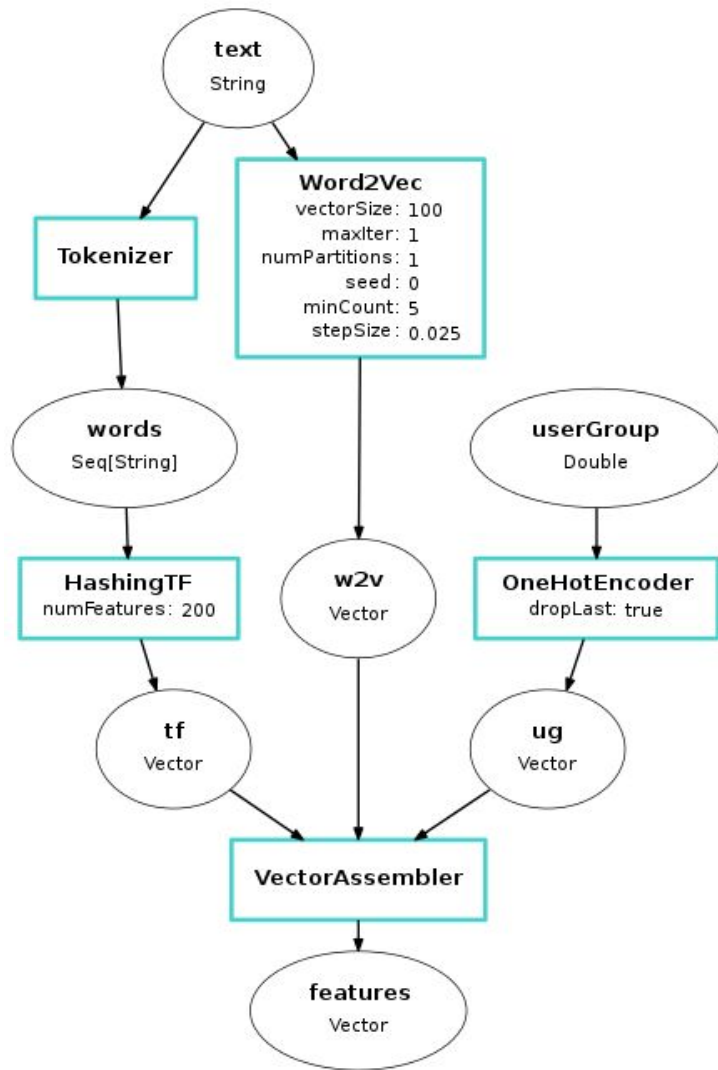
// Fit a model with cross-validation.
val cvModel = cv.fit(trainingDataset)
```

Feature Transformers

Transformer	Description	scikit-learn
Binarizer	Threshold numerical feature to binary	Binarizer
Bucketizer	Bucket numerical features into ranges	
ElementwiseProduct	Scale each feature/column separately	
HashingTF	Hash text/data to vector. Scale by term frequency	FeatureHasher
IDF	Scale features by inverse document frequency	TfidfTransformer
Normalizer	Scale each row to unit norm	Normalizer
OneHotEncoder	Encode k-category feature as binary features	OneHotEncoder

Transformer	Description	scikit-learn
PolynomialExpansion	Create higher-order features	PolynomialFeatures
RegexTokenizer	Tokenize text using regular expressions	(part of text methods)
StandardScaler	Scale features to 0 mean and/or unit variance	StandardScaler
StringIndexer	Convert String feature to 0-based indices	LabelEncoder
Tokenizer	Tokenize text on whitespace	(part of text methods)
VectorAssembler	Concatenate feature vectors	FeatureUnion
VectorIndexer	Identify categorical features, and index	
Word2Vec	Learn vector representation of words	

```
tok = Tokenizer(inputCol="text", outputCol="words")
htf = HashingTF(inputCol="words", outputCol="tf",
numFeatures=200)
w2v = Word2Vec(inputCol="text", outputCol="w2v")
ohe = OneHotEncoder(inputCol="userGroup",
outputCol="ug")
va = VectorAssembler(inputCols=["tf", "w2v", "ug"],
outputCol="features")
pipeline = Pipeline(stages=[tok,htf,w2v,ohe,va])
```



Algorithm Coverage

	Discrete	Continuous
Supervised	Classification LogisticRegression(with Elastic-Net) SVM DecisionTree RandomForest GBT NaiveBayes MultilayerPerceptron OneVsRest	Regression LinearRegression(with Elastic-Net) DecisionTree RandomForest GBT AFTSurvivalRegression IsotonicRegression
Unsupervised	Clustering KMeans GaussianMixture LDA PowerIterationClustering	Dimensionality Reduction, matrix factorization PCA SVD ALS WLS

Algorithm optimized for distributed computing

ALS

Online statistic algorithm

WLS vs L-BFGS (4096 features and 1 billion features)

Use in-place operation for vector/matrix, i.e. update existing vectors rather than creating new vector

Standardization on LiR and LoR

Linear vs Non-linear model in Spark

Linear model	Non-linear model
LogisticRegression LinearRegression AFTSurvivalRegression SVM NaiveBayes	DecisionTree RandomForest GBT MultilayerPerceptron

Non-linear relationships of covariates

Many observations and variables, non-linear relationships

Non-linear and non-parametric models are popular solutions, but they are slow and difficult to interpret

Spark ML solution:

- Automated feature generation with polynomial mappings

- Regularized regressions with various performance optimizations

For some problems, linear methods with feature engineering are as good as nonlinear kernel methods, and with better performance

User cases

Recommend engine with ALS

Classification such as users churn, users behavior prediction

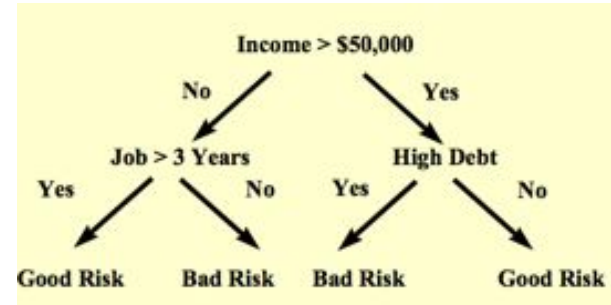
Clustering users based on geographic locations with DBSCAN

Provide R formula in R/Scala/Python for financial users

The credit scoring problem

Traditional credit scoring systems aim at deciding upon the creditworthiness of applicants using characteristics e.g. age, marital status, amount on savings account, macroeconomic, ...

The problem is usually tackled using classification techniques, e.g. logistic regression, neural networks, decision trees, ...



The credit scoring survival analysis problem

But: Time to default is also very important:

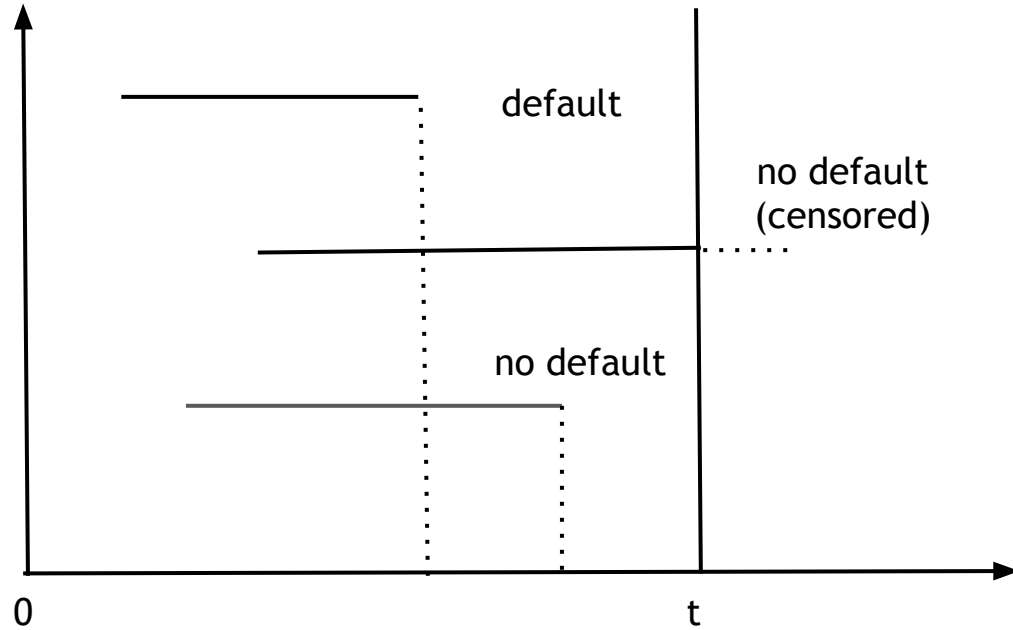
- to decide upon length of time of loan
- for debt provisioning purposes
- decide upon increase or decrease of credit limit
- to monitor a client's repayment behaviour

Traditional classification techniques not appropriate to handle this problem

- linear regression ?
- no, because of censored data

Use survival analysis methods originating from medicine

Time to default in credit risk



Log-linear survival regression model([SPARK-8518](#))

n iid random variables $\{(Y_1, X_1, \delta_1), \dots, (Y_n, X_n, \delta_n)\}$

$Y = \min\{T, C\}$ is the observed maturity

- T is the time to default
- C is the time to the end of the study or anticipated cancellation of the credit

$\delta = I(T \leq C)$ is the indicator of noncensoring(default)

X is a vector of explanatory covariates

We assume that there exists an unknown relationship between T and X , T and C are conditionally independent given X

$$\log T = \alpha + \beta^T X + \sigma \epsilon$$

Loss and gradient function

likelihood function

$$l(\beta, \sigma) = \sum_{i=1}^n [-\delta_i \log \sigma + \delta_i \log f_0(\epsilon_i) + (1 - \delta_i) \log S_0(\epsilon_i)] \quad \text{where} \quad \epsilon_i = \frac{\log t_i - x'_i \beta}{\sigma}$$

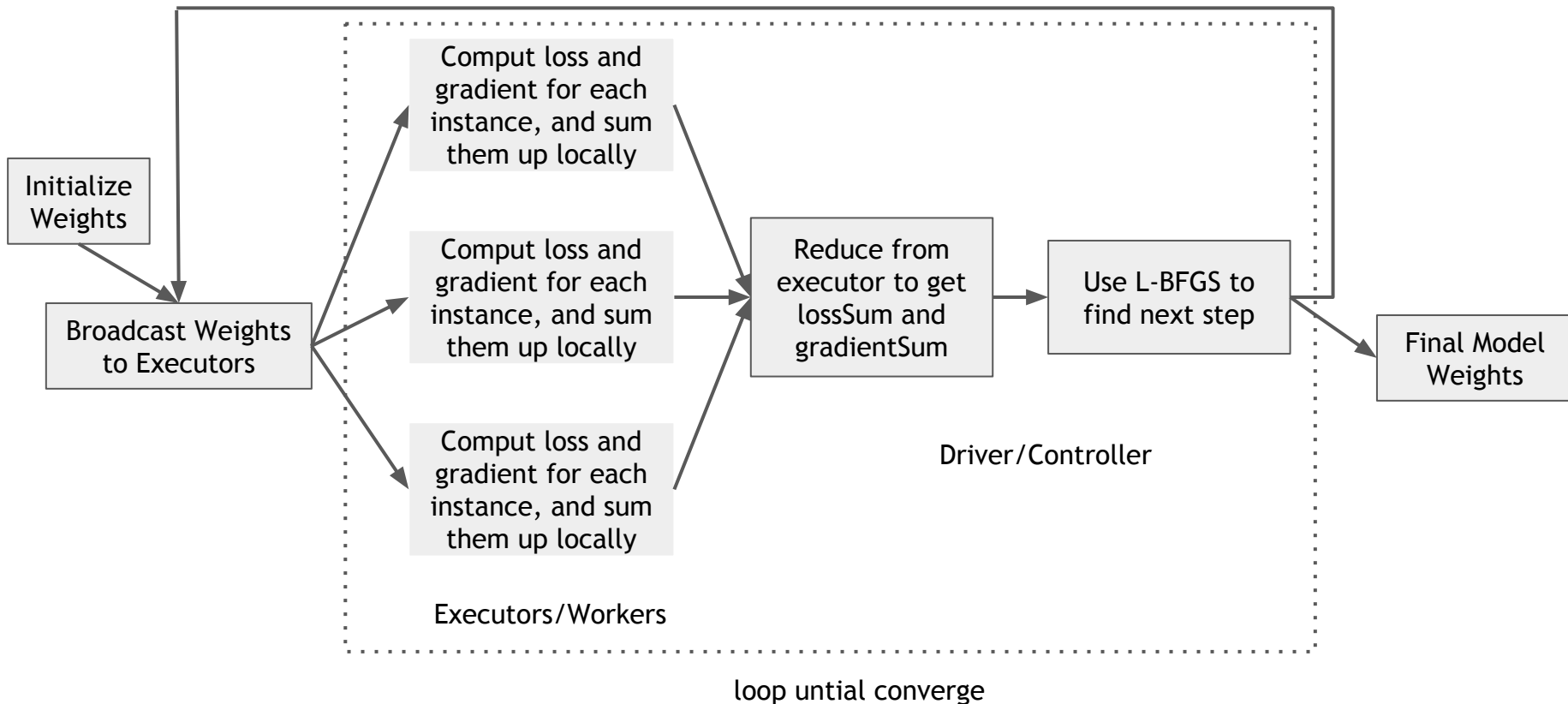
loss function

$$l(\beta, \sigma) = - \sum_{i=1}^n [\delta_i \log \sigma - \delta_i \epsilon_i + e^{\epsilon_i}]$$

gradient function

$$\frac{\partial(-l)}{\partial \beta} = \sum_{i=1}^n [\delta_i - e^{\epsilon_i}] \frac{x_i}{\sigma} \qquad \frac{\partial(-l)}{\partial(\log \sigma)} = \sum_{i=1}^n [\delta_i + (\delta_i - e^{\epsilon_i}) \epsilon_i]$$

AFTSurvivalRegression([SPARK-8518](#))



Benchmark

Dataset:

- 15288 records, can fit by R
- 80 million records, over 50 GB, hard to train in a single machine

Use survival model to generate default probabilities at various points in time.

Compare the results achieved using survival models with respect to classical parametric models (Logistic Regression of Spark ML).

To compare the logistic model with the survival analysis the time to event and censored observation in the datasets are transformed to a good-bad variable. The following procedure is used:

$$Good/Bad = \begin{cases} 1 \text{ (bad), } tte < (365 * \text{years}) \text{ AND } cens = 0 \\ 0 \text{ (good), otherwise} \end{cases}$$

Where years is the number of years the PD is estimated for, tte is the time to default and cens is a flag if the observation is censored (1) or defaulted (0).

Pipeline

Outliers detection

Data validation

Bucketizer

Omit the correlated variables

OneHotEncoder





VectorAssembler

Model training with cross validation and evaluation

Model coefficients - The same as R !

	Value
(Intercept)	1.3773
landline	0.5926
noSecondaryEducation	-0.9811
onlySecondaryEducation	-0.2876
higherThanSecondaryEducation	0.0014
employedLt1year	-0.9613
employedBt1and5year	-0.7782
employedGt5year	0.0038
paymentOnAccountFromSameBank	1.5536
paymentOther	0.0002
Log(scale)	-0.0236

Performance comparison

	Tools	R Logistic Regression	R AFT Survival Regression	Spark ML Logistic Regression	Spark AFT Survival Regression
Dataset1	ROC/AUC	0.75	0.77	0.75	0.77
	KS	0.43	0.43	0.43	0.43
Dataset2	ROC/AUC			0.72	0.74
	KS			0.39	0.38

Challenge

The AFT model is fully described by a distribution of an error term. The AFT model will always follow this distribution, therefore once e.g. log-normal distribution is chosen, it cannot be used to describe more peaks in a hazard function.

In practice, the proportion of defaulted credits is small, the proportion of censored data is large.

Conclusion

More and more algorithms and functions are moving to Spark ML/MLlib

Spark ML/MLlib is moving to traditional IT area

We are actively Spark contributors and users

We are hiring !

Thank you

梁堰波@明略数据

@DataScientist

ybliang8@gmail.com