Strata+Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

https://github.com/jayantshekhar/strata-2016

Building machine-learning apps with Spark

MLlib, ML Pipelines, and GraphX

Jayant / Amandeep / Krishna / Vartika

strataconf.com

#StrataHadoop

Agenda

Overview	10 min (9:00-9:10)	
Lab Environment Setup	15 min (9:10-9:25)	IntelliJ/Scala IDE for Eclipse/Zeppelin
MLlib	90 min (9:25-10:55)	Overview, Linear Regression, Random Forest, Custering, Recommendations, FPG, Text Analytics
Break	10 min (10:55-11:05)	
GraphX	50 min (11:05-11:55)	Overview, Exploring Structures, Community-Affiliation, Algorithms, The AlphaGo Community, Wikepedia Page Rank
ML Pipelines	15 min (11:55-12:10)	CrossValidation
Streaming MLlib	10 min (12:10-12:20)	Streaming K-Means
Closing	10 min (12:20-12:30)	

Download stuff if you haven't yet

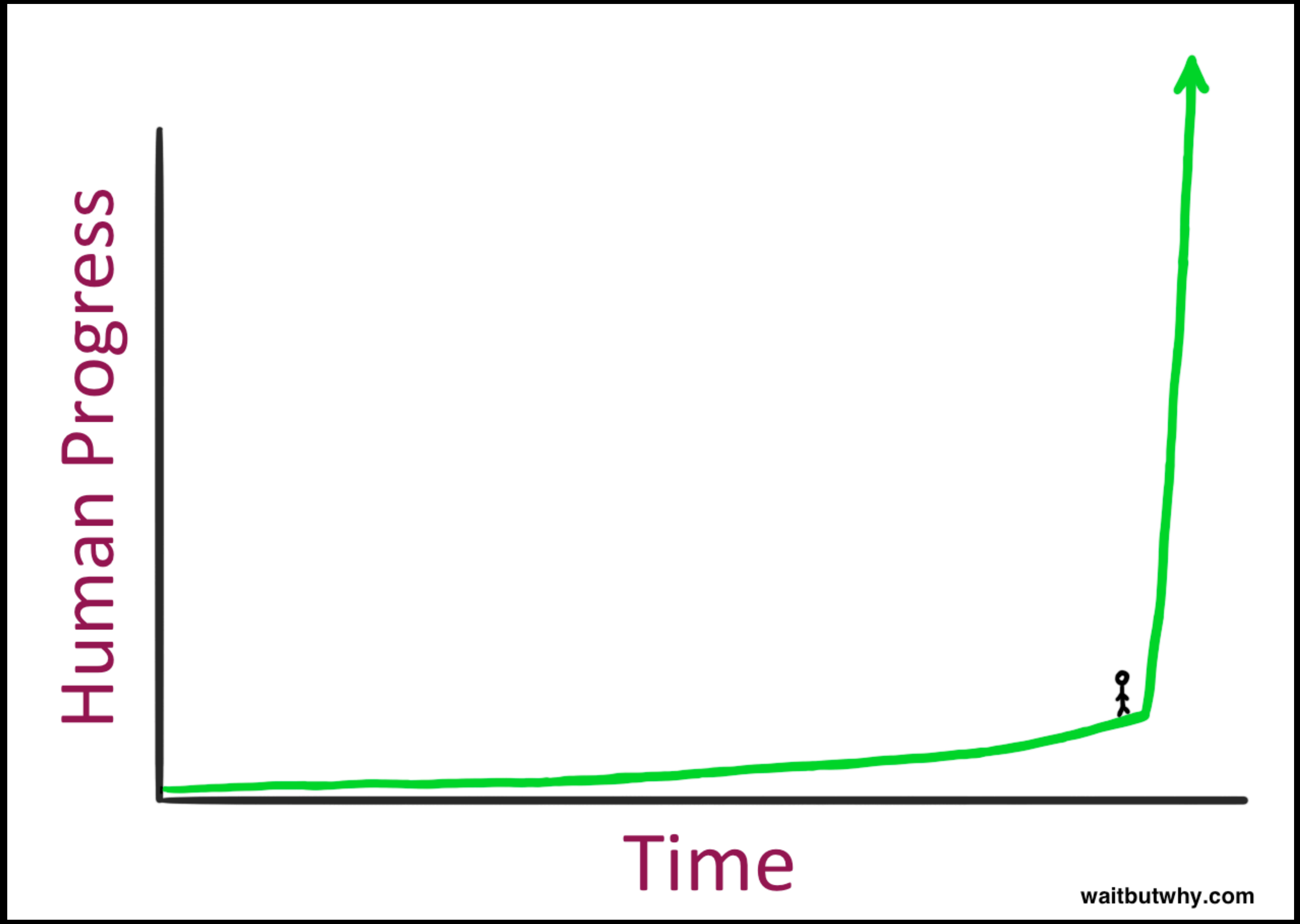
- http://conferences.oreilly.com/strata/hadoop-big-data-ca/public/schedule/detail/46943
- https://github.com/jayantshekhar/strata-2016

Your speakers

- Krishna Shankar
- Jayant Shekar
- Vartika Singh
- Supporting cast: Amandeep Khurana

Why?

- *We are on the edge of change comparable to the rise of human life on Earth. – Vernor Vinge (Prof, SDSU)*
- AI is fast becoming a reality
  - ANI (Narrow)
  - AGI (General)
  - ASI (Super Intelligence)
- We are currently in ANI stage
- To go to AGI and ASI
  - More compute power
  - Better algorithms and systems
- Reference: http://waitbutwhy.com/2015/01/artificial-intelligence-revolution-1.html



Machine Learning & Big Data

- Better systems => ML @ scale
  - Bigger training set => better models => better accuracy
  - Can’t be cost prohibitive
- Spark ecosystem
  - MLlib
  - GraphX
- Others out there
  - H20
  - Dato
  - Graphlab

What's a ML app?

- Collect data
- Clean data – make it usable
- Build model
- Train model
- Test model
- Use model
  - Apply to data at rest (historical)
  - Apply to making decisions as data comes in (current / future)

- MLlib

MLlib

Overview	05 min	
Linear Regression	15 min	Predict House Prices
Random Forest	10 min	Titanic Predict Survival
Clustering	20 min	Topic Modeling on newsgroup data with LDA
Recommendations	10 min	Movie Lens Ratings and Recommendations
FPG	05 min	Shopping Cart Analysis
Text Analytics	25 min	Mood Of the Nation/Mood of the Presidential debates

- Data Types
- Basic Statistics
- Feature Extraction & Transformation

- Local Vector
- Labeled Point
- Local Matrix
- Distributed Matric

- Summary Statistics
- Correlations
- Stratified Sampling
- Hypothesis Testing
- Random Data Generation

- TF-IDF
- Word2Vec
- Tokenizer
- OneHotEncoder
- n-gram

- Classification & Regression
  - Linear Models (SVMs, Linear Regression, Logistic Regression)
  - Naïve Bayes
  - Decision Tree
  - Ensembles
    - Random Forests
    - Gradient Boosted Trees
- Collaborative Filtering
  - ALS
- Frequent Pattern Mining
- Clustering
  - K-Means
- Dimensionality Reduction
  - SVD
  - PCA
- PMML model export

- Linear Methods
  - Linear Regression

Linear Regression

Regularizer

- Defines the trade off between minimizing the loss function and minimizing the model complexity

Loss Function

- Measures the error of the model on the training data.
- Squared Loss, which is a convex function

Ordinary or Linear Least squares → No regularization

Ridge Regression → L2 Regularization

Lasso → L1 Regularization

Goodness of fit – Mean Squared Error

Overfitting

Housing Prices Prediction

Removed the header row when reading
Reformatted the data to have it in format:
Housing Price, <feature attribute values, space separated>

- **Data**
  - **13 continuos attributes, 1 binary valued attribute**

    MEDV ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT

    24.00,18.00 2.310 0 0.5380 6.5750 65.20 4.0900 1 296.0 15.30 396.90 4.98

    21.60,0.00 7.070 0 0.4690 6.4210 78.90 4.9671 2 242.0 17.80 396.90 9.14

    34.70,0.00 7.070 0 0.4690 7.1850 61.10 4.9671 2 242.0 17.80 392.83 4.03

    33.40,0.00 2.180 0 0.4580 6.9980 45.80 6.0622 3 222.0 18.70 394.63 2.94
- Target Variable
  - MEDV
- Predictor Variables
  - ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT

More about Data

- Attribute Information
  - CRIM per capita crime rate by town
  - ZN proprotion of residential land zoned for lots over 25,000 sq.ft
  - INDUS prop of non-retail business over town
  - CHAS charles river dummy variable
  - NOX nitric oxide concentrates
  - RM average number of rooms per dwelling
  - DIS weighted distance to five Boston employment centers
  - RAD index of accessibility to highways

  ….

  …..

  https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.names

Housing Prices Prediction - Code

- **Read the data as a text file and convert it to an RDD of LabeledPoints Data types**

```
val parsedData = data.map { line =>
      val parts = line.split(',')
      val label = parts(0).toDouble
      val features = parts(1).split(' ').map(_.toDouble)
      LabeledPoint(label, Vectors.dense(features))
   }.cache()
```

- **Normalize the data across feature vectors using StandardScaler**

```
Import org.apache.spark.mllib.feature.StandardScaler
val scaler = new StandardScaler(withMean = true, withStd =
true).fit(parsedData.map(x => x.features))
```

Housing Prices Prediction - Code

- Tuning
  - Set Intercept to be True

```
val algorithm = new LinearRegressionWithSGD()
 algorithm.setIntercept(true)
```

  - Play with Number of iterations and step size

```
algorithm.optimizer.setNumIterations(numIterations).setStepSize(stepSize)
```

- Train the model and see the result

```
val model = algorithm.run(scaledData)

scaledData.take(5).foreach{ x=> println(s"Predicted: ${model.predict(x.features)}, Label: ${x.label}")}
```

- Titanic Survival Prediction
  - Random Forest

Titanic

Remove the header row when reading

- **Data**

PassengerId,Survived,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked

1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5 21171,7.25,,S

2,1,1,"Cumings, Mrs. John Bradley (Florence Briggs Thayer)",female,38,1,0,PC 17599,71.2833,C85,C

3,1,3,"Heikkinen, Miss. Laina",female,26,0,0,STON/O2. 3101282,7.925,,S

- Target Variable
  - Survived
- Predictor Variables
  - Pclass, Sex, Age, Fare

Titanic DataSet

VARIABLE DESCRIPTIONS:

survival	Survival (0 = No; 1 = Yes)
pclass	Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
name	Name
sex	Sex
age	Age
sibsp	Number of Siblings/Spouses Aboard
parch	Number of Parents/Children Aboard
ticket	Ticket Number
fare	Passenger Fare
cabin	Cabin
embarked	Port of Embarkation

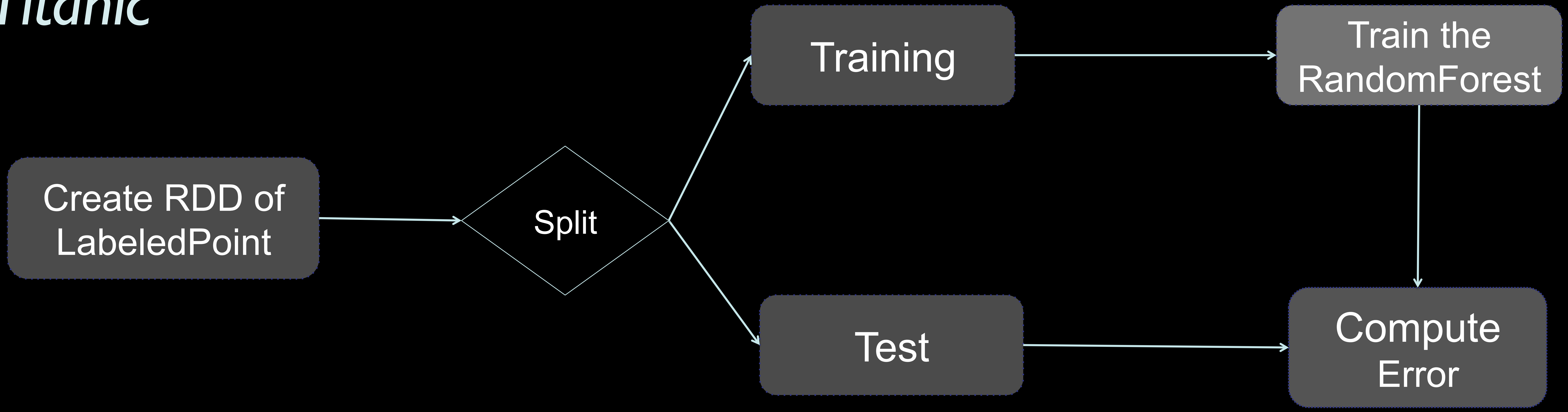
(C = Cherbourg; Q = Queenstown; S = Southampton)

NOTES:

Pclass is a proxy for socio-economic status (SES)
1st ~ Upper; 2nd ~ Middle; 3rd ~ Lower

Age is in Years; Fractional if Age less than One (1)
If the Age is Estimated, it is in the form xx.5

Titanic




```
root
 |-- PassengerId: string (nullable = true)
 |-- Survived: string (nullable = true)
 |-- Pclass: string (nullable = true)
 |-- Name: string (nullable = true)
 |-- Sex: string (nullable = true)
 |-- Age: string (nullable = true)
 |-- SibSp: string (nullable = true)
 |-- Parch: string (nullable = true)
 |-- Ticket: string (nullable = true)
 |-- Fare: string (nullable = true)
 |-- Cabin: string (nullable = true)
 |-- Embarked: string (nullable = true)
```

Random Forest

- **numTrees**: Number of trees in the forest.
- **maxDepth**: Maximum depth of each tree in the forest.
- **categoricalFeaturesInfo**: Specifies which features are categorical and how many categorical values each of those features can take. This is given as a map from feature indices to feature arity (number of categories). Any features not in this map are treated as continuous.
  - E.g., Map(0 -> 2, 4 -> 10) specifies that feature 0 is binary (taking values 0 or 1) and that feature 4 has 10 categories (values {0, 1, ..., 9}). Feature indices are 0-based: features 0 and 4 are the 1st and 5th elements of an instance’s feature vector.

- Tree 0:
- If (feature 0 in {0.0})
- If (feature 4 <= 8.7125)
- If (feature 3 <= 0.0)
- If (feature 2 <= 0.0)
- Predict: 0.0
- Else (feature 2 > 0.0)
- Predict: 0.0
- Else (feature 3 > 0.0)
- If (feature 1 <= 0.42)
- Predict: 1.0
- Else (feature 1 > 0.42)
- Predict: 0.0
- Else (feature 4 > 8.7125)
- If (feature 1 <= 14.0)
- If (feature 2 <= 2.0)
- Predict: 1.0
- Else (feature 2 > 2.0)
- Predict: 0.0
- Else (feature 1 > 14.0)

- Tree 1:
- If (feature 0 in {0.0})
- If (feature 4 <= 9.8375)
- If (feature 4 <= 7.8958)
- If (feature 4 <= 7.8292)
- Predict: 0.0
- Else (feature 4 > 7.8292)
- Predict: 0.0
- Else (feature 4 > 7.8958)
- If (feature 2 <= 0.0)
- Predict: 0.0
- Else (feature 2 > 0.0)
- Predict: 1.0
- Else (feature 4 > 9.8375)
- If (feature 3 <= 0.0)
- If (feature 4 <= 26.0)
- Predict: 0.0
- Else (feature 4 > 26.0)
- Predict: 0.0
- Else (feature 3 > 0.0)

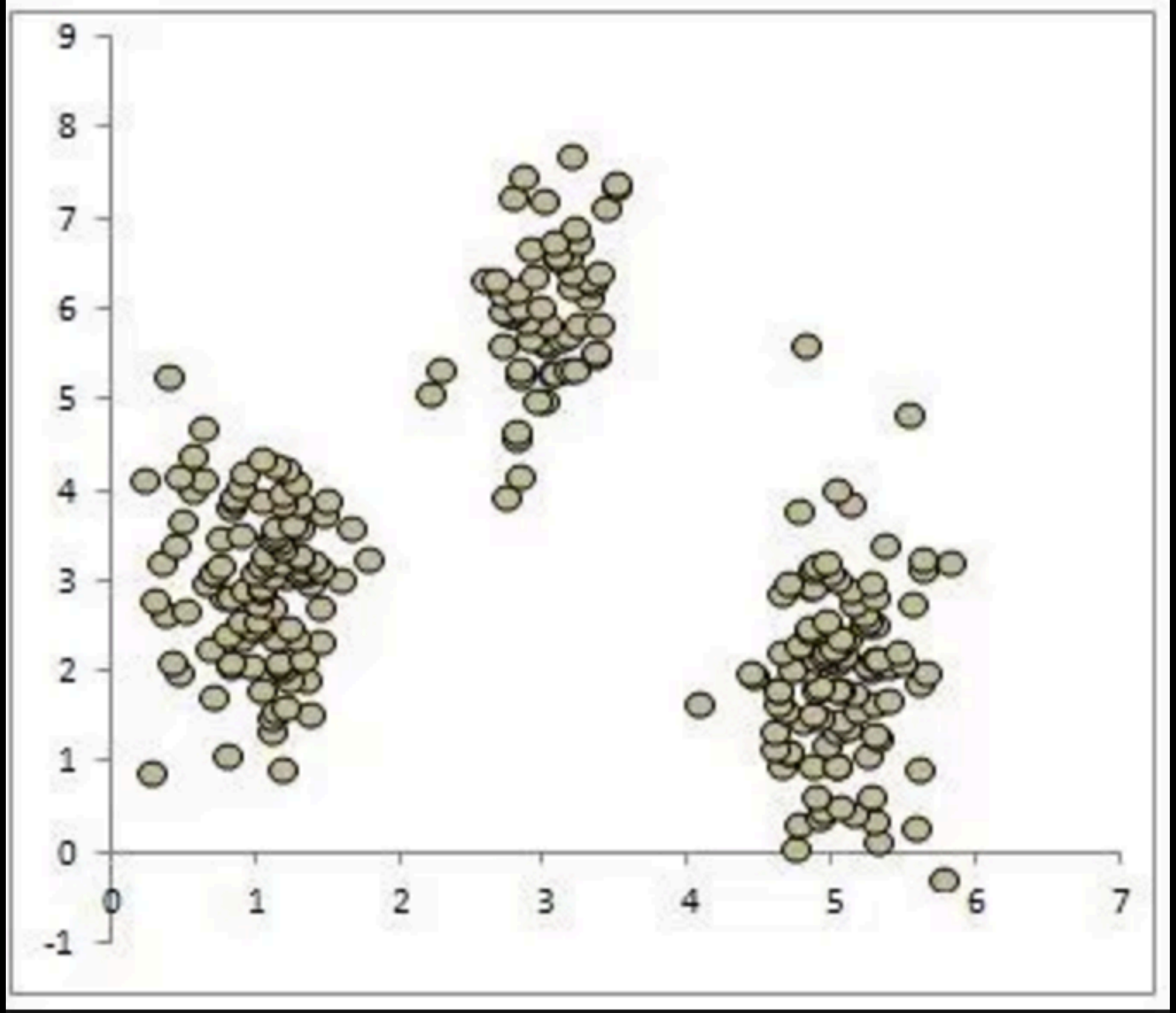
- Clustering

- Topic Modeling – News Group Data

Clustering

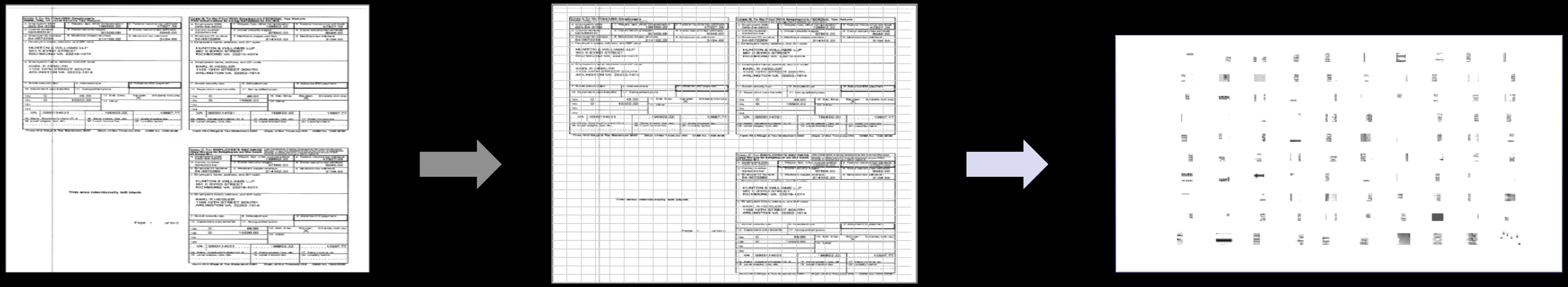
- K-Means
- EM
- GMM
- LDA
- Streaming K-Means

Clustering - Motivation



- A good clustering has predictive power.
- Predictions while uncertain, are useful, because we believe that the underlying cluster labels are meaningful and can help us take meaningful actions.

Clustering - Motivation



- Compression and Vector Quantization

Clustering - Motivation

- Failures of the cluster model may highlight interesting objects that deserve special attention, a.k.a outliers.
- Dimensionality reduction.
- Compression

K-means – ml.lib

- Parameters
  - K: Number of clusters.
  - maxIterations: maximum number of iterations to run
  - initializationMode: random or via K-Means||
  - Runs: number of sets to run
  - initializationSteps: number of steps in the K-Means|| algorithm
  - Epsilon: threshold
  - initialModel: optional – set of clusters used for initialization

K-means – sample run

- Imports
  - org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
  - org.apache.spark.mllib.linalg.Vectors

- Loading the data and Cache

- Specifying params
- Compute cost
- Saving and Loading the model

EM

- Maximizes lower bound on the log likelihood: Each step guaranteed to improve our answer until convergence
- Iterative method for finding maximum likelihood or maximum a posteriori estimates of parameters in statistical models, where the model depends on unobserved data.

Gaussian Mixture Modelling

- Distance function – Expectation Maximization
- A more Bayesian approach.
- For *k* clusters, and each component is represented as $z_k \sim N(\mu_k, \Sigma_k)$
- Each data point is x is generated from one of these components with a certain probability ($\Sigma_k = 1$ for each x)
- Parameters:
  - K: Number of desired clusters
  - convergenceTol: maximum change in log-likelihood at which we consider convergence achieved
  - maxIterations: Maximum number of iterations to perform before reaching convergence
  - initialModel: Optional starting point.

LDA – A topic model

- Infers topics from a collection of documents
- Topic correspond to cluster centers, and documents correspond to examples(rows) in a dataset.
- Topics and documents both exist in a feature space, where feature vectors are vectors of word counts.
- Distance function – uses statistical model of how text documents are generated.

LDA – Optimizers

- EMLDAOptimizer
  - Uses EM on likelihood function
  - Stores comprehensive results in *DistributedLDAModel*
  - Desirable to keep *maxIterations* greater than 50.
- OnlineLDAOptimizer
  - Iterative mim-batch sampling for online variational inference and is generally memory friendly
  - Stores the inferred topics in LocalLDAModel
  - docConcentration: Assymetric priors can be used.

Code Walkthrough - LDA

- Read the input documents and the stopwords File

```
val rawTextRDD = sc.wholeTextFiles(inputDir).map(_._2)
val stopwords = sc.textFile(stopWordFile).collect
```

- Tokenize the documents

```
val tokens = new RegexTokenizer()
            .setGaps(false)
            .setPattern("\\w+")
            .setMinTokenLength(4)
            .setInputCol("text")
            .setOutputCol("words")
            .transform(docDF)
```

- Split the data and Generate the model

```
val filteredTokens = new StopWordsRemover()
                .setStopWords(stopwords)
                .setCaseSensitive(false)
                .setInputCol("words")
                .setOutputCol("filtered")
                .transform(tokens)
```

Code Walkthrough - LDA

- Define the Estimator and train a model – CountVectorizer: Performs a vocabulary extraction and transforms it into

```
val cvModel = new CountVectorizer()
        .setInputCol("filtered")
        .setOutputCol("features")
        .setVocabSize(vocabSize)
        .fit(filteredTokens)
```

- Run the LDA

```
val lda = new LDA()
        .setOptimizer(new OnlineLDAOptimizer()
        .setMiniBatchFraction(math.min(1.0, mbf)))
        .setK(numTopics)
        .setMaxIterations(30)
        .setDocConcentration(-1) ) // Default symmetric doc-topic prior
        .setTopicConcentration(-1) // Default topic-word prior

val ldaModel = lda.run(countVectors)
```

Sample Output- LDA

TOPIC 0	
gatech	0.009160678955551409
engr	0.007574289447944401
theory	0.006716068304197075
prism	0.006568455199953412
cantaloupe	0.006281138868875978
uoknor	0.006240530630839462 5
boeing	0.006108729953431769
atlanta	0.006105897707959385 4
georgia	0.005408553307510841

TOPIC 2	
talk	0.011332796061488924
religion	0.0112257919741886 7
people	0.011175304719441202
misc	0.006154609606306365
atheism	0.006154507977337078
jesus	0.005934418956112216
news	0.005635678269501505
writes	0.005513754989295043
abortion	0.005407887588198011

TOPIC 7	
culture	0.015014415450966085
turkish	0.013406554534281 23
armenian	0.008732574491207 52
politics	0.007518673925730 33
armenians	0.007264617787511 53
talk	0.006808253370125685 5
jews	0.006743333373614261
people	0.006675121118441786
soviet	0.006597592366267302
history	0.005324235523883088

LDA – Optimizers

- EMLDAOptimizer
  - Uses EM on likelihood function
  - Stores comprehensive results in *DistributedLDAModel*
  - Desirable to keep *maxIterations* greater than 50.
- OnlineLDAOptimizer
  - Iterative mim-batch sampling for online variational inference and is generally memory friendly
  - Stores the inferred topics in LocalLDAModel
  - docConcentration: Assymetric priors can be used.

- Recommendations
  - Movie Lens Ratings

MovieLens 100K Dataset

Stable benchmark dataset. 100,000 ratings from 1000 users on 1700 movies. Released 4/1998.

- README.txt
- ml-100k.zip (size: 5 MB, checksum)
- Index of unzipped files

Permalink: http://grouplens.org/datasets/movielens/100k/

MovieLens 1M Dataset

Stable benchmark dataset. 1 million ratings from 6000 users on 4000 movies. Released 2/2003.

- README.txt
- ml-1m.zip (size: 6 MB, checksum)

Permalink: http://grouplens.org/datasets/movielens/1m/

MovieLens 10M Dataset

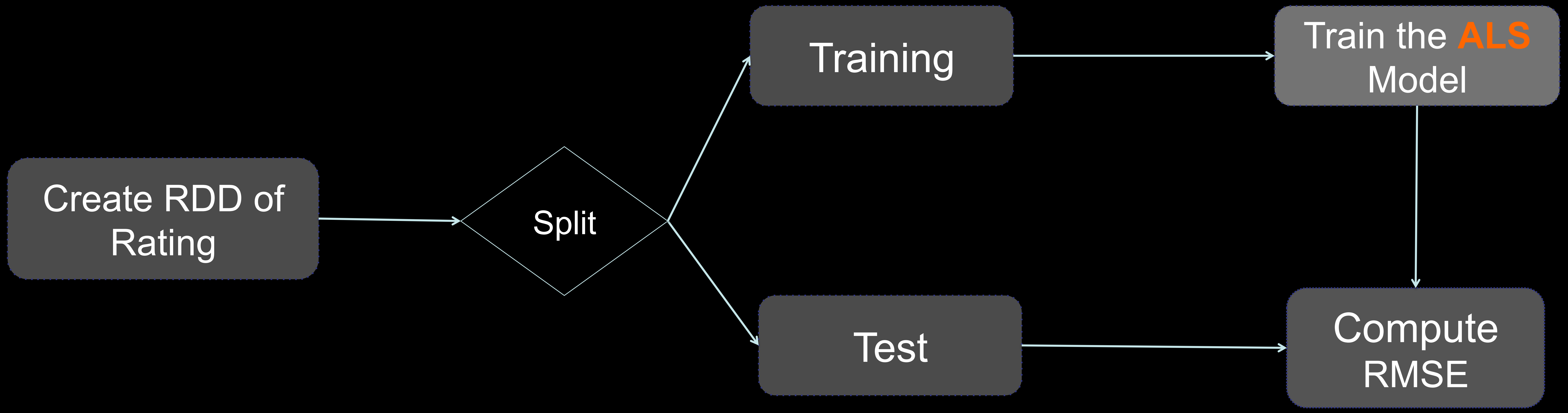
Stable benchmark dataset. 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. Released 1/2009.

- README.html
- ml-10m.zip (size: 63 MB, checksum)

MovieLens

Userid, movie id, rating

0::2::3
0::3::1
0::5::2



Recommendations with ALS

- Fill in the missing entries of a user-item association matrix
- *numBlocks* is the number of blocks used to parallelize computation (set to -1 to auto-configure).
- *rank* is the number of latent factors in the model.
- *iterations* is the number of iterations to run.
- *lambda* specifies the regularization parameter in ALS.
- *implicitPrefs* specifies whether to use the *explicit feedback* ALS variant or one adapted for *implicit feedback* data.
- *alpha* is a parameter applicable to the implicit feedback variant of ALS that governs the *baseline* confidence in preference observations.

discover actual shopping behavior

- Frequent Pattern Mining
  - FPG

Frequent Pattern Mining

- Mllib has parallel implementation of FP-Growth
  - minSupport: the minimum support for an itemset to be identified as frequent. For example, if an item appears 3 out of 5 transactions, it has a support of 3/5=0.6.
  - numPartitions: the number of partitions used to distribute the work.

FPGrowth

Create RDD of ArrayList<String>

r z h k p
z y x w v
u t s
s x o n r
x z y m t
s q e

Run FPGrowth

Print Results

[s], 3
[s,x], 3
[s,x,z], 2
[s,z], 2
[r], 3
[r,x], 2
[r,z], 2
[y], 3
[y,s], 2
[y,s,x], 2

- ML Pipelines

- 15 min

Typical Machine Learning Workflow

- Data Ingest Load
- Feature Extraction
- Training/Tuning the model
- Prediction/Inference

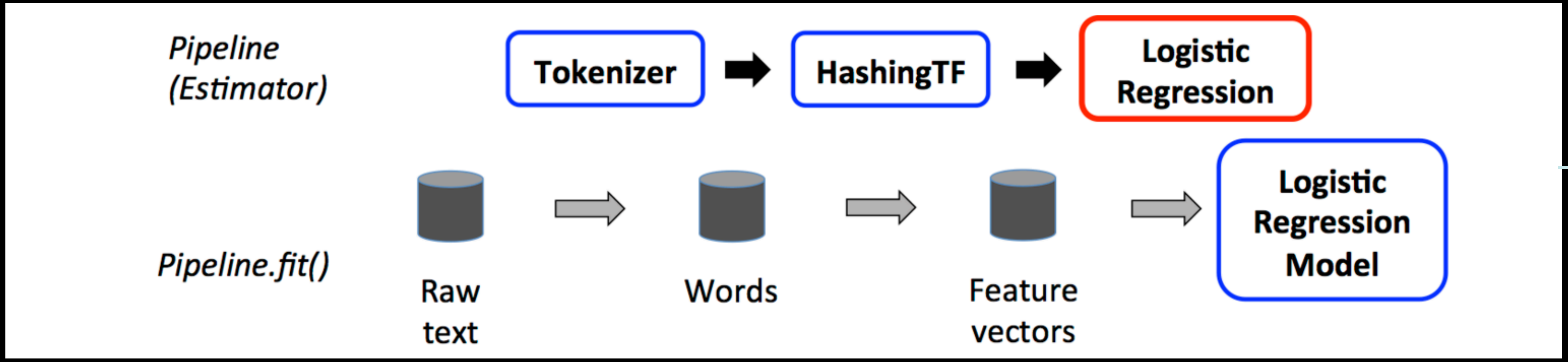
- Split each document's text into words.
- Convert each document's words into a numerical feature vector.
- Learn a prediction model using the feature vectors and labels.

ML Pipelines

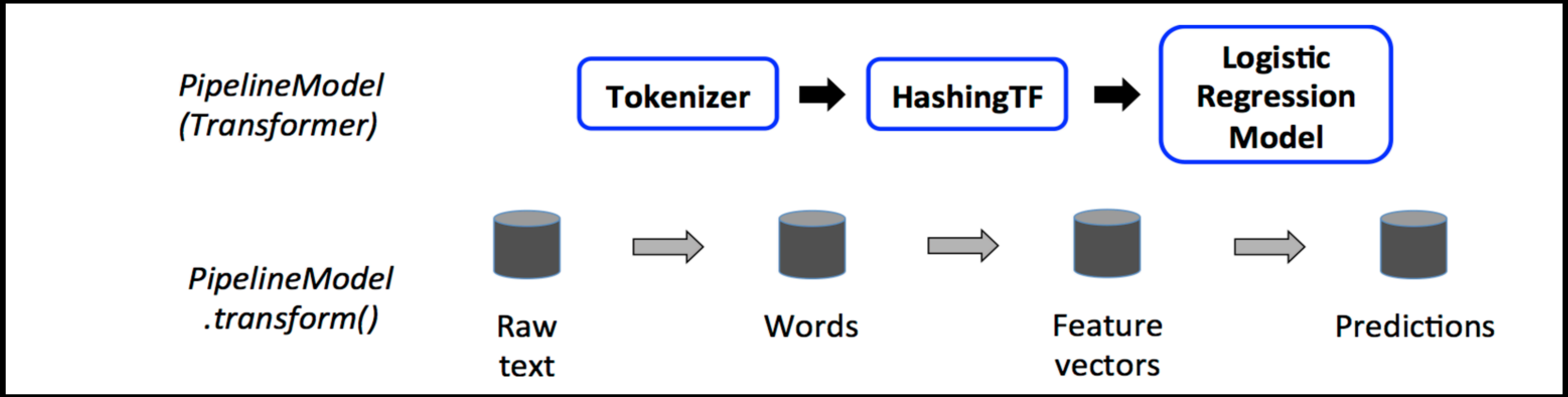
- Facilitates a quick and easy assembly and configuration of practical machine learning pipelines.
- Are like DAG of nodes – sequence of stages – Estimators and Transformers.
- Can be saved and loaded when needed to be applied in real time to new data.
- Parameter Tuning
- Flexible coding and Easy debugging – Use DataFrames

How it Works

- Input DataFrame is transformed as it passes through stages
- Transformer Stages - .transform()
- Estimator Stages - .fit() – produces a transformer
- Help ensure that training and test data go through identical feature processing steps
- Linear and Non-linear pipelines as long a data flow graph forms a DAG
- Only run time checking



- First two stages are transformers – append cols to DF
- .fit() generates a PipelineModel, which is a transformer



- PipelineModel has same number of stages as the original Pipeline, but all Estimators in the original pipeline have become transformers.
- .transform triggers prediction

Code Walkthrough - Pipelines

- Data
  - Housing data: data/housing/Housing.csv
  - Contains Categorical Variables
    - Use One-Hot-Encoding
- Algorithm
  - Continue with LinearRegression

Code Walkthrough - Pipelines

- Read Data and map it to a constructor

```
case class X(
		id: String ,price: Double, lotsize: Double, bedrooms: Double,
		bathrms: Double,stories: Double, driveway: String,recroom: String,
		fullbase: String, gashw: String, airco: String, garagepl: Double, prefarea: String)
……....

  val data = sc.textFile(input).map(_.split(","))
		.map( x => ( X(
		 x(0), x(1).toDouble, x(2).toDouble, x(3).toDouble, x(4).toDouble, x(5).toDouble, x(6), x(7), x(8), x(9), x(10),
x(11).toDouble, x(12) ))).toDF()
```

- Define Categorical Variables

```
val categoricalVariables = Array("driveway","recroom", "fullbase", "gashw", "airco", "prefarea")
```

Code Walkthrough - Pipelines

- Define Pipeline Stages – Convert Categorical variables to Continuos values

```
val categoricalIndexers: Array[org.apache.spark.ml.PipelineStage] =
        categoricalVariables.map(i => new StringIndexer()
        .setInputCol(i).setOutputCol(i+"Index"))

  val categoricalEncoders: Array[org.apache.spark.ml.PipelineStage] =
        categoricalVariables.map(e => new OneHotEncoder()
        .setInputCol(e + "Index").setOutputCol(e + "Vec"))

  val assembler = new VectorAssembler()
        .setInputCols( Array(
         "lotsize", "bedrooms", "bathrms", "stories", "garagepl","drivewayVec", "recroomVec", "fullbaseVec",
         "gashwVec","aircoVec", "prefareaVec"))
```

- Define the estimator

```
val lr = new LinearRegression()
        .setLabelCol("price")
        .setFeaturesCol("features")
        .setMaxIter(1000)
        .setSolver("l-bfgs")
```

Code Walkthrough - Pipelines

- Get the stages together

```
val steps = categoricalIndexers ++
            categoricalEncoders ++
            Array(assembler, lr)
```

- Define the pipeline with above stages

```
val pipeline = new Pipeline()
                .setStages(steps)
```

- Define the estimator

```
val tvs = new TrainValidationSplit()
          .setEstimator( pipeline )
          .setEvaluator( new RegressionEvaluator().setLabelCol("price") )
          .setEstimatorParamMaps(paramGrid)
          .setTrainRatio(0.75)
```

- Split the data and Generate the model

```
val Array(training, test) = data.randomSplit(Array(0.75, 0.25), seed = 12345)

  val model = tvs.fit(training)
```

Model Selection

- Cross Validation
  - Uses CrossValidator class – Estimator, a set of ParamMaps and an Evaluator
  - Data split into folds.
  - Evaluator: RegressionEvaluator, BinaryClassificationEvaluator and MultiClassClassificationEvaluator
  - Can use ParamGridBuilder utility to provide multiple parameters for optimal tuning.
  - Can be expensive, but well established method for parameter tuning.
- Train Validation Split
  - Only evaluates each combination of parameters once, as opposed to k times as done by CrossValidator
  - Splits data into training and testing

Code Walkthrough - Pipelines

- Define the model Model Selector

```
val cv = new CrossValidator()
      .setEstimator( pipeline )
      .setEvaluator( new RegressionEvaluator().setLabelCol("price"))
      .setEstimatorParamMaps( paramGrid )
      .setNumFolds(5)
```

- Split the data and Generate the model

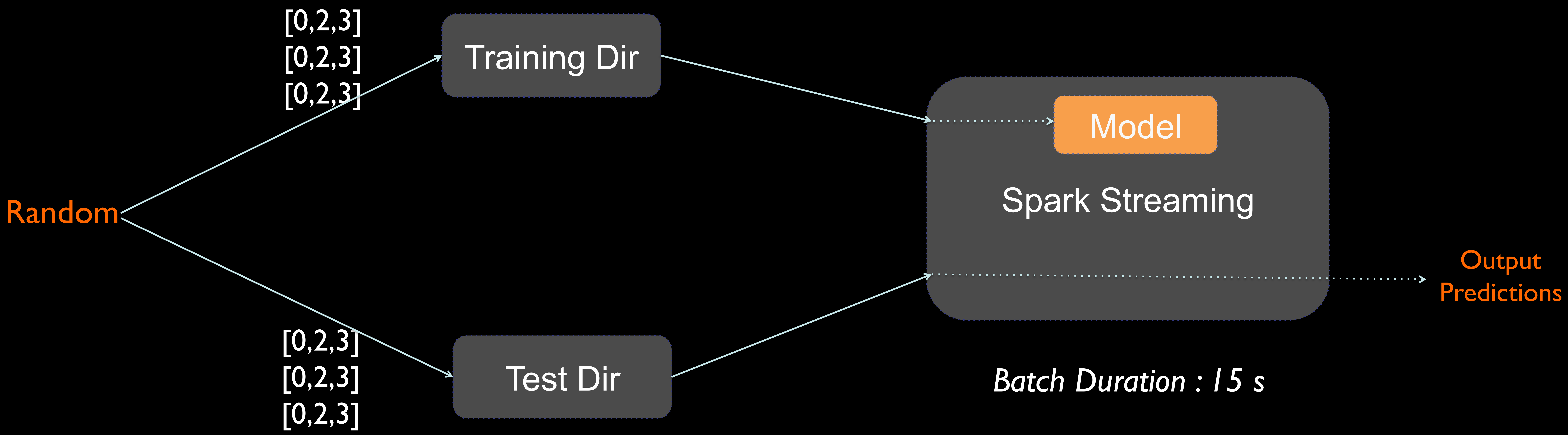
```
val Array(training, test) = data.randomSplit(Array(0.75, 0.25), seed = 12345)

val cvModel = cv.fit(training)
```

- Streaming MLlib
- 10 min


Kafka
Flume
HDFS/S3
Kinesis
Twitter
Spark
Streaming
HDFS
Databases
Dashboards


input data
stream
Spark
Streaming
batches of
input data
Spark
Engine
batches of
processed data

Streaming K-Means



Estimate clusters on one stream of data and make predictions on another stream

Streaming K-Means

- Each training point should be formatted as **[x1, x2, x3]**
- Test data point should be formatted as **(y, [x1, x2, x3])**, where y is some useful label or identifier (e.g. a true category assignment).
- Anytime a text file is placed in **../trainingDir** the model will update
- Anytime a text file is placed in **../testDir** they would be processed to produce predictions using the current model
- The decay can be specified using a halfLife parameter, which determines the correct decay factor such that, for data acquired at time t, its contribution by time t + halfLife will have dropped to 0.5