Strata + Hadoop WORLD

PRESENTED BY

O'REILLY cloudera

Unraveling data with Spark using machine learning

Vartika / Jayant /Jeff

strataconf.com

#StrataHadoop

About Us

Vartika Singh is a Solutions Architect at Cloudera with over 12 years of experience in applying machine learning techniques to big data problems.

Jayant is the CEO of Sparkflows.io with over 10 years of experience building big data products and applying machine learning.

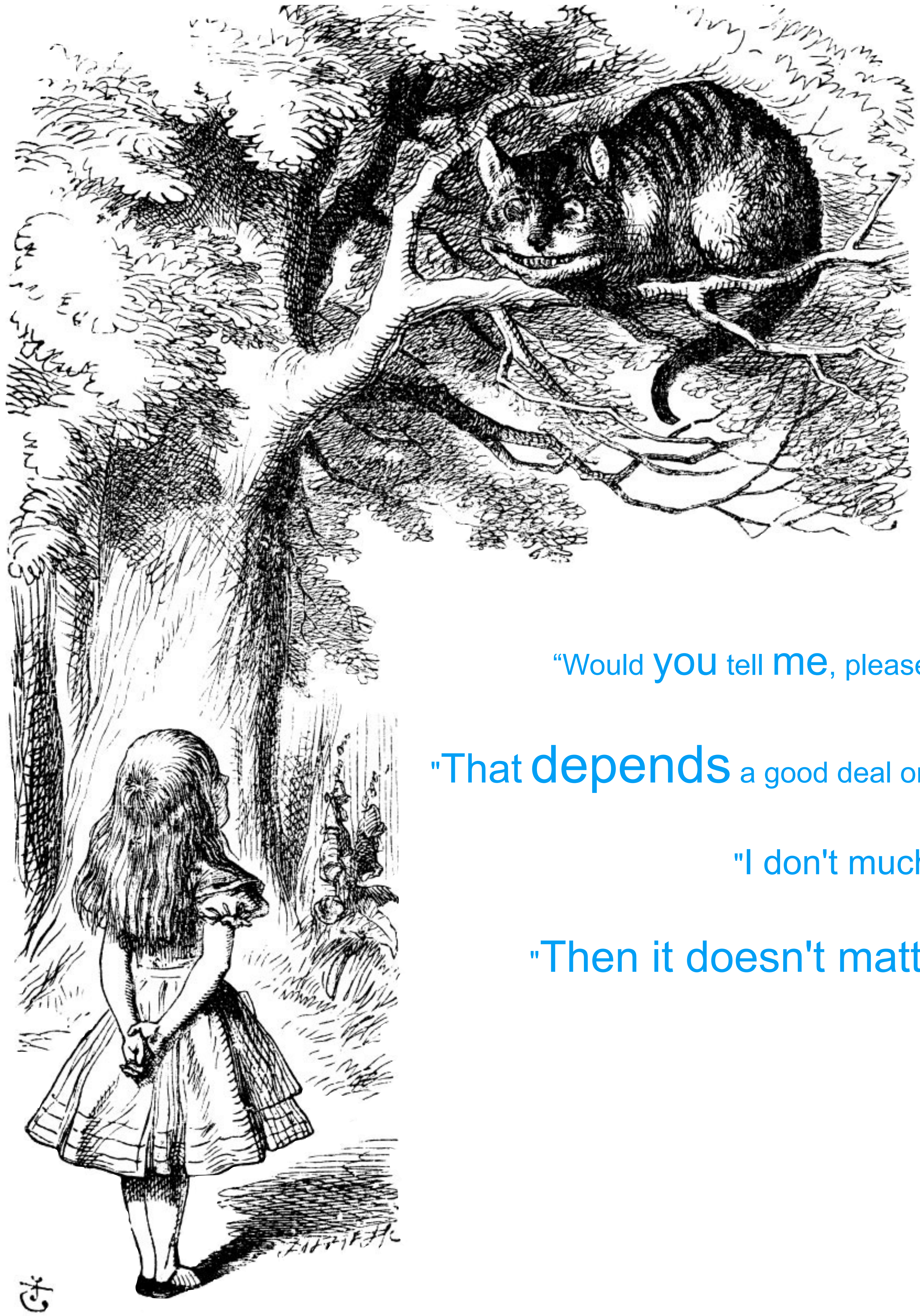
Prev : Cloudera / Yahoo / eBay

Jeff Shmain is a solution architect at Cloudera.
He has 16+ years of financial industry experience with a strong understanding of security trading, risk, and regulations. Over the last few years, Jeff has been helping various clients implement spark applications.

Download & Install

CDSW	Spark Shell
http://cdsw.cloudera.com/	▪ http://spark.apache.org/downloads.html ▪ spark-shell --driver-memory 2G --executor-memory 2G --num-executors 2 --executor-cores 2

https://github.com/WhiteFangBuck/strata-sanjose-2017



“Would you tell me, please, which road do I take?"

"That depends a good deal on where you want to get to."

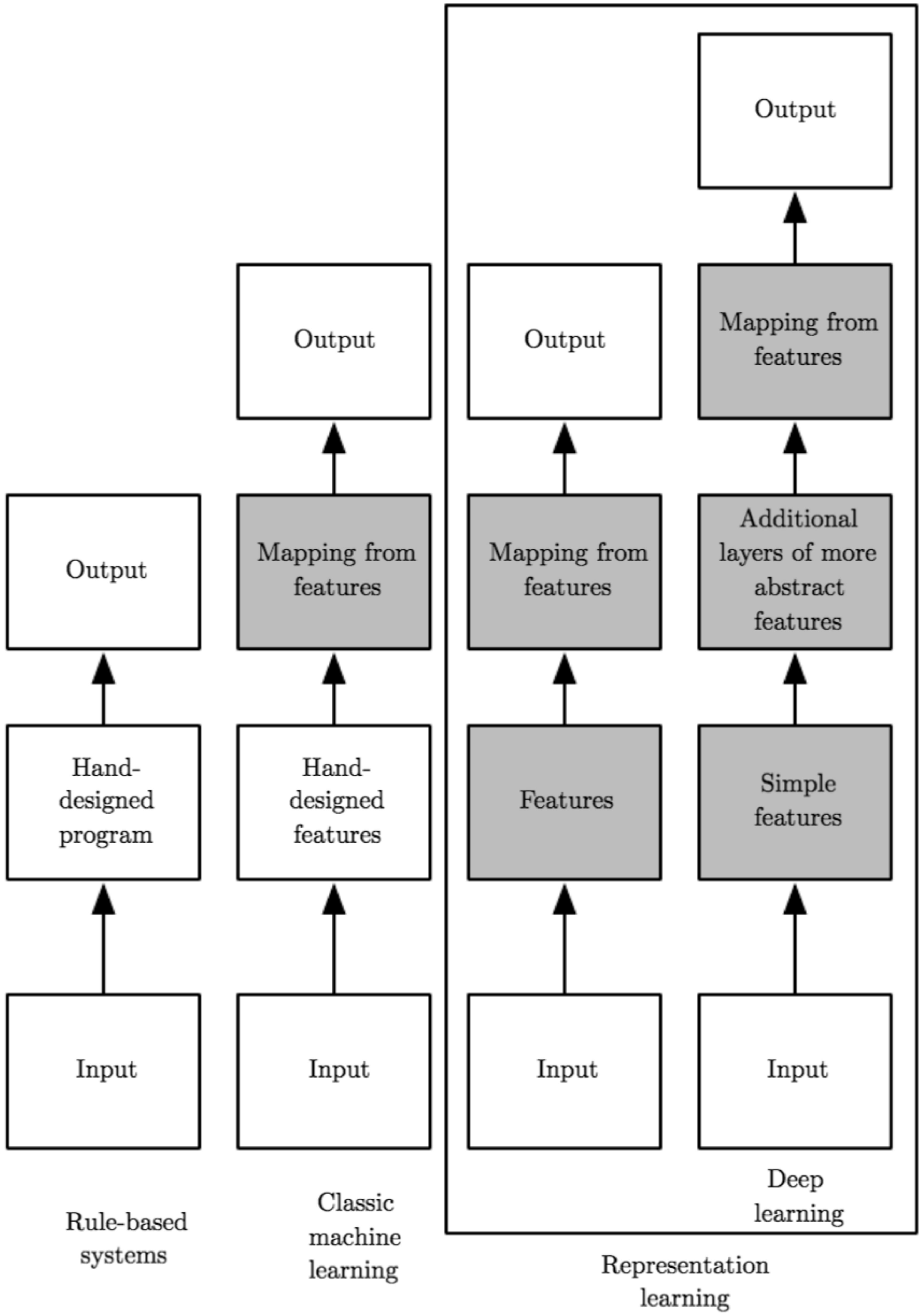
"I don't much care where –"

"Then it doesn't matter which way you go.”

A Data Scientists Nightmare

Which Algorithm???!!!!

- Class of algorithms
  - Supervised
  - Unsupervised
- Examples
  - Supervised: Classification, Regression
  - Unsupervised: Clustering



CDH Stack - Typical Machine Learning Workflow

- Data Ingest Load
- Feature Transformation and 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.

spark.ml

- 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.
- Hyperparameter Tuning
- Flexible coding and Easy debugging – Use DataFrames/DataSets

Feature Transformers

Scaling, converting, or modifying features

Working with DataSets/DataFrames

id	airco
1	no
2	no
..	..

- Immutable
- Adds the new data by creating a new DataFrame and appending the new column to it.

Transformation on The Column

id	airco	garagepl
1	no	1
2	no	0
..	..	..

Working with the data - StringIndexer/OneHotEncoder (VectorIndexer)

id	price	lot size	bedrooms	bathrooms	stories	driveway	recroom	fullbase	gashw	airco	garagepl	prefarea
1	42000	5850	3	1	2	yes	no	yes	no	no	1	no
2	38500	4000	2	1	1	yes	no	no	no	no	0	no
..	..	..	..	..	..	..	..	..	..	..	..	..

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

drivewayIndex	recroomIndex	fullbaseIndex	gashwIndex	aircoIndex	prefareaIndex
0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0
..	..	..	..	..	..

org.apache.spark.ml.feature.StringIndexer

drivewayVec	recroomVec	fullbaseVec	gashwVec	aircoVec	prefareaVec
(1,[0],[1.0])	(1,[0],[1.0])	(1,[],[])	(1,[0],[1.0])	(1,[0],[1.0])	(1,[0],[1.0])
(1,[0],[1.0])	(1,[0],[1.0])	(1,[0],[1.0])	(1,[0],[1.0])	(1,[0],[1.0])	(1,[0],[1.0])
..	..	..	..	..	..

org.apache.spark.ml.feature.OneHotEncoder

org.apache.spark.ml.feature.VectorIndexer

Transformers - Code Walk Through

```
val indexer = new StringIndexer()
  .setInputCol("category")
  .setOutputCol("categoryIndex")
  .fit(df)
val indexed = indexer.transform(df)
```

StringIndexer

```
val vIndexer = new
VectorIndexer()
  .setInputCol("features")
  .setOutputCol("indexed")
  .setMaxCategories(10)

val indexerModel =
vIndexer.fit(libSVMData)
```

VectorIndexer

```
val encoder = new OneHotEncoder()
  .setInputCol("categoryIndex")
  .setOutputCol("categoryVec")

val encoded = encoder.transform(indexed)
```

One Hot Encoder

Preparing the data - Tokenizer/RegExTokenizer

text	docId
Xref: cantaloupe....	0
|Path: cantaloupe....	1
..	..

org.apache.spark.ml.feature. RegexTokenizer

text	docId	words
Xref: cantaloupe....	0	[xref, cantaloupe...
|Path: cantaloupe....	1	[path, cantaloupe...
..	..	…

org.apache.spark.ml.feature. StopWordsRemover

text	docId	words	filtered
Xref: cantaloupe....	0	[xref, cantaloupe...	[comp, archives, ...
|Path: cantaloupe....	1	[path, cantaloupe...	[magnesium, club....
..	..	…	…

- gaps
- minTokenLength
- pattern
- toLowercase

Transformers - Code Walk Through

```
val tokenizer = new Tokenizer()
  .setInputCol(“sentence”)
  .setOutputCol("words")

val regexTokenizer = new RegexTokenizer()
  .setInputCol("sentence")
  .setOutputCol("words")
  .setPattern("\\W")

// alternatively
.setPattern("\\w+").setGaps(false)

val countTokens = udf { (words: Seq[String]) =>
words.length }

val tokenized =
tokenizer.transform(sentenceDataFrame)
```

Tokenizer

```
val regexTokenized = regexTokenizer
  .transform(sentenceDataFrame)
  .select("sentence", "words")
  .withColumn("tokens",
countTokens(col("words")))
```

RegExTokenizer

```
val remover = new
StopWordsRemover()
  .setInputCol("words")
  .setOutputCol("filtered")

remover.transform(regexTokenized)
```

One Hot Encoder

Tranformers

Hands On Exercise

Extracting features from “raw” data

…some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used.

—Pedro Domingos, in “A few useful things to know about Machine Learning.”

Preparing the data - CountVectorizer

text	docId
Xref: cantaloupe....	0
|Path: cantaloupe....	1
..	..

org.apache.spark.ml.feature. RegexTokenizer

text	docId	words
Xref: cantaloupe....	0	[xref, cantaloupe...
|Path: cantaloupe....	1	[path, cantaloupe...
..	..	…

org.apache.spark.ml.feature. StopWordsRemover

text	docId	words	filtered
Xref: cantaloupe....	0	[xref, cantaloupe...	[comp, archives, ...
|Path: cantaloupe....	1	[path, cantaloupe...	[magnesium, club,...
..	..	…	…

org.apache.spark.ml.feature. CountVectorizer

text	docId	words	filtered	features
Xref: cantaloupe...	0	[xref, cantaloupe...	[comp, archives, ...	(10000, [0,1,2,3,4...
|Path: cantaloupe...	1	[path, cantaloupe...	[magnesium, club,...	(10000, [0,1,2,3,4...
..	..	…	…	

Implementation details: The CountVectorizer

- Convert a collection of text documents to vectors of token counts.
- When an a-priori dictionary is not available
- The model produces sparse representations for the documents over the vocabulary, which can then be passed to other algorithms like LDA.

Tuning parameters: The CountVectorizer

- vocabSize : Top number of words ordered by term frequency across the corpus.
- minDF : Affects the fitting process by specifying the minimum number (or fraction if < 1.0) of documents a term must appear in to be included in the vocabulary.
- minTF: Filter to ignore rare words in a document. For each document, terms with frequency/count less than the given threshold are ignored.
- binary: If True, all nonzero counts (after minTF filter applied) are set to 1.

CountVectorizer - Code Walk Through

org.apache.spark.ml.feature. RegexTokenizer

```
val hashingTF = new HashingTF()
  .setInputCol("words")
  .setOutputCol("rawFeatures")
  .setNumFeatures(20)

val featurizedData = hashingTF
  .transform(wordsData)
```

org.apache.spark.ml.feature.IDF

```
val idf = new
IDF().setInputCol("rawFeatures").setOutputCol("features")
val idfModel = idf.fit(featurizedData)

val rescaledData = idfModel.transform(featurizedData)
```

org.apache.spark.ml.feature.CountVectorizer

```
val cvModel: CountVectorizerModel =
   new CountVectorizer()
  .setInputCol("words")
  .setOutputCol("features")
  .setVocabSize(3)
  .setMinDF(2).fit(wordsData)

val cvm = new CountVectorizerModel(
   Array("a", "b", "c"))
  .setInputCol("words")
  .setOutputCol("features")
```

Implementation details: The TF

- Both HashingTF and CountVectorizer can be used to generate the term frequency vectors.
- HashingTF utilizes the "hashing trick"
  - Hash function used here is "MurmurHash 3"
  - Avoids the need to compute a global term-to-index map
  - Suffers from potential hash collisions.
- CountVectorizer converts text documents to vectors of term counts.

Tuning parameters: The TF

- binary : Binary toggle to control term frequency counts. If true, all non-zero counts are set to 1. This is useful for discrete probabilistic models that model binary events rather than integer counts. (default = false)

- numFeatures : Number of features. Should be greater than 0. (default = 2^18)

Implementation details: The IDF

- The IDFModel takes feature vectors (generally created from HashingTF or CountVectorizer) and scales each column

- It down-weights columns which appear frequently in a corpus.

Tuning parameters: The IDF

- minDocFreq : The minimum number of documents in which a term should appear. Default: 0

Text Segmentation

- spark.ml doesn’t provide tools for text segmentation
- We refer users to the Stanford NLP Group and scalanlp/chalk.

TFIDF - Code Walk Through

org.apache.spark.ml.feature. RegexTokenizer

```
val hashingTF = new HashingTF()
  .setInputCol(“words")
  .setOutputCol(“rawFeatures")
  .setNumFeatures(20)

val featurizedData = hashingTF
  .transform(wordsData)
```

org.apache.spark.ml.feature.IDF

```
val idf = new
IDF().setInputCol("rawFeatures").setOutputCol("features")
val idfModel = idf.fit(featurizedData)

val rescaledData = idfModel.transform(featurizedData)
```

Preparing the data - Word2Vec

text	docId
Xref: cantaloupe....	0
|Path: cantaloupe....	1
..	..

org.apache.spark.ml.feature. RegexTokenizer

text	docId	words
Xref: cantaloupe....	0	[xref, cantaloupe...
|Path: cantaloupe....	1	[path, cantaloupe...
..	..	…

org.apache.spark.ml.feature. StopWordsRemover

text	docId	words	filtered
Xref: cantaloupe....	0	[xref, cantaloupe...	[comp, archives, ...
|Path: cantaloupe....	1	[path, cantaloupe...	[magnesium, club....
..	..	…	…

org.apache.spark.ml.feature.Word2Vec

text	docId	words	filtered	features
Xref: cantaloupe....	0	[xref, cantaloupe...	[comp, archives, ...	(10000, [0,1,2,3,4...
|Path: cantaloupe....	1	[path, cantaloupe...	[magnesium, club....	(10000, [0,1,2,3,4...
..	..	…	…	

Implementation details: Word2Vec

- The model maps each word to a unique fixed-size vector.
- The Word2VecModel transforms each document into a vector using the average of all words in the document

Tuning parameters: The word2vec

- maxIter : Maximum number of Iterations (>0)

- maxSentenceLength : Sets the maximum length (in words) of each sentence in the input data. Any sentence longer than this threshold will be divided into chunks of up to maxSentenceLength size. Default: 1000

- minCount : The minimum number of times a token must appear to be included in the word2vec model's vocabulary. Default: 5

- stepSize : Param for Step size to be used for each iteration of optimization (> 0).

- vectorSize : The dimension of the code that you want to transform from words. Default: 100

Implementation details: Word2Vec

documents	coordinate 1	coordinate 2	coordinate 3
[Hi, I, heard, about, Spark]	-0.008142343163490296	0.020513632893562 32	0.03255096450448036
[I, wish, Java, could, use, case, classes]	0.043090314205203734	0.035048123182994974,	0.023512658663094044
..[Logistic, regression, models, are, neat]	0.038572299480438235..	-0.03250147425569594	-0.01552378609776497

Vector Size 3

Vector Size 5

documents	coordinate 1	coordinate 2	coordinate 3	coordinate 4	coordinate 5
[Hi, I, heard, about, Spark]	0.018459798023104667	-0.027064743265509606	0.03365720957517624	-0.01668163686990738	-0.026146824297029525
[I, wish, Java, could, use, case, classes]	-0.04265850649348327	-0.009572108409234455	0.016981298769158975	0.008395011403730937	0.004702869004436901 5
..[Logistic, regression, models, are, neat]	0.015756532829254866	-0.012175573443528265	0.031459877640008925	0.022983803600072863	-0.001562454551458358 8

Word2Vec - Code Walk Through

org.apache.spark.ml.feature. RegexTokenizer

```
val hashingTF = new HashingTF()
  .setInputCol(“words")
  .setOutputCol(“rawFeatures")
  .setNumFeatures(20)

val featurizedData = hashingTF
  .transform(wordsData)
```

org.apache.spark.ml.feature.IDF

```
val idf = new
IDF().setInputCol("rawFeatures").setOutputCol("features")
val idfModel = idf.fit(featurizedData)

val rescaledData = idfModel.transform(featurizedData)
```

org.apache.spark.ml.feature.Word2Vec

```
val word2vec = new Word2Vec()

val model = word2vec.fit(rescaledData)
```

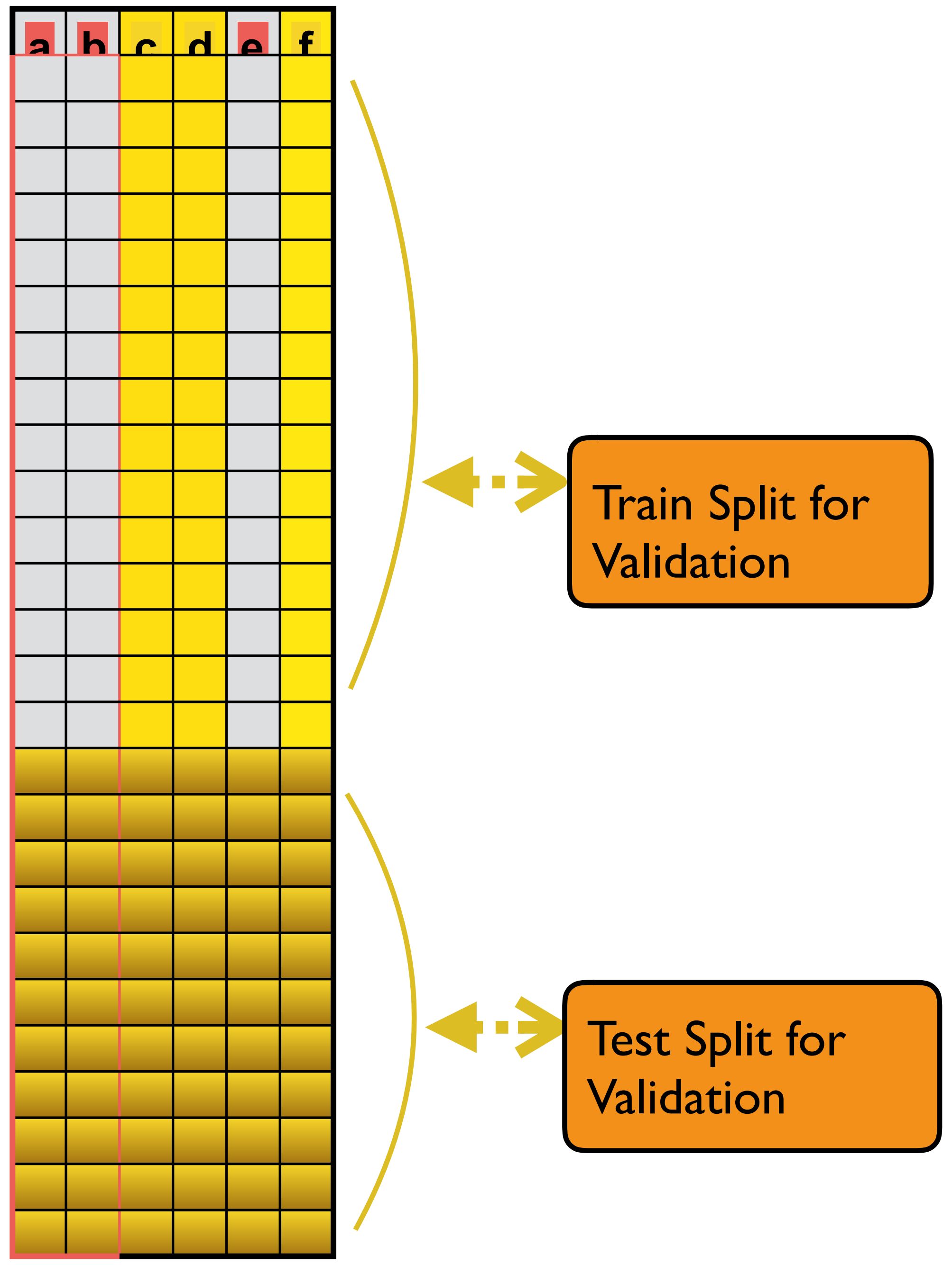
Extractors

Hands On Exercise

Model Selection and Tuning

Using data to find the best model or parameters for a given task.

Model Selection and Tuning



CrossValidation Split
- Split Data into folds
- Fits the estimator on all the folds
- After identifying the best parammap, it re-fits the estimator using the best param map

- Use data to find the best model or parameters for a given task.
- Can be done for single estimator or an entire pipeline.
- Require
  - Estimator -> algorithm to tune
  - ParamMap -> set of parameters to tune over
  - Evaluator -> metric to measure how well a fitted Model does on *holdout* data
- Can be done via CrossValidation or TrainValidationSplit
  - CrossValidation
    - Goes over multiple folds of data.
    - Slower but more reliable
  - TrainValidation split
    - Evaluate combination of parameters only once.
    - Faster but less reliable

Parameters: Model tuning

Common
estimator - estimator to be evaluated. Examples: LinearRegression, Pipeline, DecisionTreeRegressor
estimatorParamMaps - a grid of parameters to evaluate the model
evaluator - used to select hyper-parameters that maximize the validated metric. Examples: BinaryClassification Evaluator, RegressionEvaluator, MulticlassClassificationEvaluator

CrossValidator
numFolds - split input data into this many parts. Default: 3

TrainValidationSplit
trainRatio - split input data set into training and validation data with this ratio. Default: 0.75

TrainValidationSplit

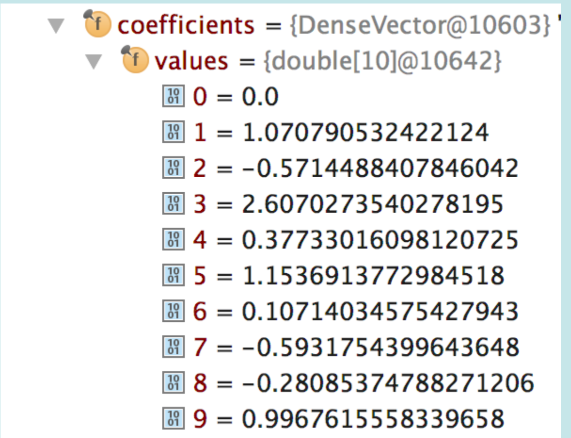
label	features
-9.49	(10 [0 1 2 3 4 5 6 7 8 9] [0.455 0.366 -0.383 -0.446 0.331 0.807 -0.262 -0.449 -0.073 0.566]
0.258	(10 [0 1 2 3 4 5 6 7 8 9] [0.839 -0.127 0.500 -0.227 -0.645 0.189 -0.580 0.652 -0.656 0.175]
-4.439	(10 [0 1 2 3 4 5 6 7 8 9] [0.503 0.142 0.160 0.505 -0.937 -0.284 0.636 -0.165 0.948 0.427]
-19.783	(10 [0 1 2 3 4 5 6 7 8 9] [-0.039 -0.417 0.900 0.641 0.273 -0.262 -0.279 -0.131 -0.085 -0.055]
-7.967	(10 [0 1 2 3 4 5 6 7 8 9] [-0.062 0.655 -0.698 0.668 -0.079 -0.439 -0.608 -0.641 0.731 -0.027]
-7.896	(10 [0 1 2 3 4 5 6 7 8 9] [-0.158 0.266 0.400 -0.369 0.143 -0.258 0.744 0.611 0.232 -0.251]
-8.465	(10 [0 1 2 3 4 5 6 7 8 9] [0.394 0.817 -0.608 0.618 0.256 -0.073 -0.389 0.080 0.270 -0.747]
2.121	(10 [0 1 2 3 4 5 6 7 8 9] [-0.005 -0.945 -0.927 -0.032 0.310 -0.208 0.880 -0.231 0.292 0.541]
1.072	(10 [0 1 2 3 4 5 6 7 8 9] [0.788 0.198 0.952 -0.846 0.550 -0.442 0.798 -0.252 -0.137 -0.335]
-13.772	(10 [0 1 2 3 4 5 6 7 8 9] [-0.370 -0.115 -0.807 0.490 -0.658 0.611 -0.720 -0.814 -0.946 0.097]
-5.082	(10 [0 1 2 3 4 5 6 7 8 9] [-0.436 0.935 0.809 -0.312 -0.972 0.619 0.043 0.670 0.167 0.376]
7.888	(10 [0 1 2 3 4 5 6 7 8 9] [0.113 -0.768 0.177 0.790 0.253 -0.235 0.807 0.667 -0.480 0.924]
14.323	(10 [0 1 2 3 4 5 6 7 8 9] [-0.205 0.147 -0.484 0.643 0.318 0.228 -0.024 -0.277 0.476 0.711]
-20.057	(10 [0 1 2 3 4 5 6 7 8 9] [-0.321 0.516 0.452 0.017 0.551 -0.248 0.726 0.394 -0.680 0.600]

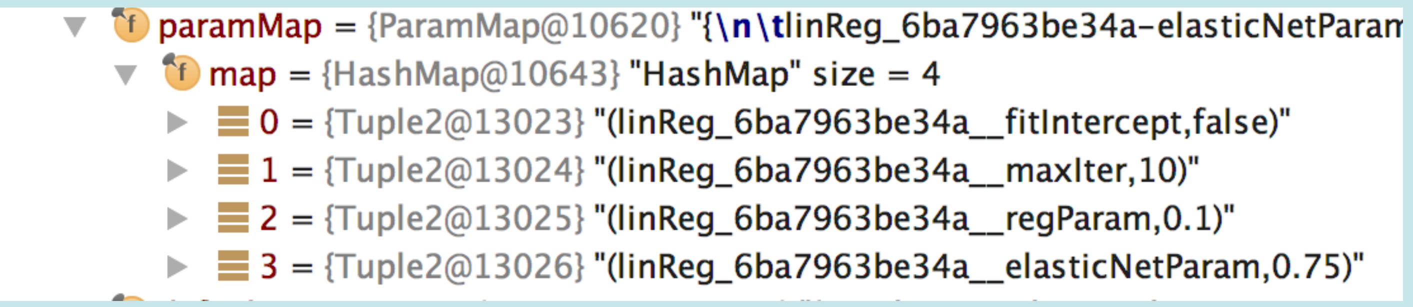
Param Grid				
	1	0.1	0.01	
0	(1,0,true)	(0.1,0,true)	(0.01,0,true)	TRUE
0.25	(1,0.25,true)	(0.1,0.25,true)	(0.01,0.25,true)	
0.5	(1,0.5,true)	(0.1,0.5,true)	(0.01,0.5,true)	
0.75	(1,0.75,true)	(0.1,0.75,true)	(0.01,0.75,true)	
0	(1,0,false)	(0.1,0,false)	(0.01,0,false)	FALSE
0.25	(1,0.25,false)	(0.1,0.25,false)	(0.01,0.25,false)	
0.5	(1,0.5,false)	(0.1,0.5,false)	(0.01,0.5,false)	
0.75	(1,0.75,false)	(0.1,0.75,false)	(0.01,0.75,false)	

org.apache.spark.ml.regression.LinearRegression

org.apache.spark.ml.evaluation.RegressionEvaluator

Best Model





TrainValidationSplit - Code Walk Through

org.apache.spark.ml.regression.LinearRegression

```
val lr = new LinearRegression()
      .setMaxIter(10)
```

org.apache.spark.ml.tuning.ParamGridBuilder

```
   val paramGrid = new ParamGridBuilder()
      .addGrid(lr.regParam, Array(0.1, 0.01, 1))
      .addGrid(lr.fitIntercept, Array(true, false))
      .addGrid(lr.elasticNetParam, Array(0.0, 0.25, 0.5, 0.75))
      .build()
```

org.apache.spark.ml.tuning.TrainValidationSplit

```
val trainValidationSplit = new TrainValidationSplit()
      .setEstimator(lr)
      .setEvaluator(new RegressionEvaluator)
      .setEstimatorParamMaps(paramGrid)
      .setTrainRatio(0.8)

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

CrossValidator

Param Grid			
	10	100	1000
0.01	(0.01,10	(0.01,10	(0.01,1000)
0.1	(0.1,10)	(0.1,100	(0.1,1000)

label	text
1	ad-jerry ad-bruckheimer ad-chase ad-premier
0	ad-rheumatoid ad-arthritis ad-expert ad-tip
0	ad-rheumatologist ad-anju ad-varghese ad-yonker
0	ad-siemen ad-water ad-remediation ad-water
0	ad-symptom ad-muscle ad-weakness ad-genetic
1	ad-animal ad-animal ad-wild ad-sa ad-official
0	ad-dr ad-enrico ad-fazzini ad-parkinson
0	ad-ulcerative ad-colitis ad-uc ad-quiz ad-ulcerative
0	ad-wellcentive ad-registry ad-web ad-base ad-preventive

tokenizer, hashingTF, idf, lr

BinaryClassificaitonEvaluator

(0.01,1000)

label	text
1	ad-free ad-raw ad-food ad-video ad-feature ad-mike
0	ad-north ad-shore ad-lij ad-cancer ad-center
1	ad-world ad-finest ad-alpaca ad-breeder
1	ad-vet ad-online ad-veterinarian ad-online
0	ad-gum ad-disease ad-treatment ad-dentist ad-near
1	ad-rabbit ad-guinea ad-pig ad-hutch ad-deal ad-save
0	ad-colitis ad-symptom ad-ulcerative ad-colitis ad-symptom ad-quiz
0	ad-disease ad-sign ad-symptom ad-cause ad-lung ad-respiratory
1	ad-pygmy ad-goat ad-pygmy ad-goat title-pygmy title-goat pygmy
1	ad-feed ad-supplement ad-buy ad-nutritional ad-supplement ad-

tokenizer, hashingTF, idf, lr

BinaryClassificaitonEvaluator

(0.1,1000)

Best Param Grid
(0.01,1000)
(0.1,1000)

label	text
1	ad-jerry ad-bruckheimer ad-chase ad-premier
0	ad-rheumatoid ad-arthritis ad-expert ad-tip
0	ad-rheumatologist ad-anju ad-varghese ad-yonker
0	ad-siemen ad-water ad-remediation ad-water
0	ad-symptom ad-muscle ad-weakness ad-genetic
1	ad-animal ad-animal ad-wild ad-sa ad-official
0	ad-dr ad-enrico ad-fazzini ad-parkinson
0	ad-ulcerative ad-colitis ad-uc ad-quiz ad-ulcerative
0	ad-wellcentive ad-registry ad-web ad-base ad-preventive
1	ad-free ad-raw ad-food ad-video ad-feature ad-mike
0	ad-north ad-shore ad-lij ad-cancer ad-center
1	ad-world ad-finest ad-alpaca ad-breeder
1	ad-vet ad-online ad-veterinarian ad-online
0	ad-gum ad-disease ad-treatment ad-dentist ad-near
1	ad-rabbit ad-guinea ad-pig ad-hutch ad-deal ad-save
0	ad-colitis ad-symptom ad-ulcerative ad-colitis ad-symptom ad-quiz
0	ad-disease ad-sign ad-symptom ad-cause ad-lung ad-respiratory
1	ad-pygmy ad-goat ad-pygmy ad-goat title-pygmy title-goat pygmy
1	ad-feed ad-supplement ad-buy ad-nutritional ad-supplement ad-

- tokenizer
- hashingTF
- idf
- lr

Binary Classification Evaluator

Best Model

CrossValidation - Code Walk Through

org.apache.spark.ml.Pipeline

```scala
val tokenizer = new Tokenizer()
    .setInputCol("text")
    .setOutputCol("words")

val hashingTF = new HashingTF()
    .setInputCol(tokenizer.getOutputCol)
    .setOutputCol("rawFeatures")

val idf = new IDF()
    .setInputCol("rawFeatures")
    .setOutputCol("features")

val lr = new LogisticRegression()
    .setMaxIter(10)

val pipeline = new Pipeline()
    .setStages(Array(tokenizer, hashingTF, idf, lr))
```

org.apache.spark.ml.tuning.ParamGridBuilder

```scala
val paramGrid = new ParamGridBuilder()
    .addGrid(hashingTF.numFeatures, Array(10, 100, 1000))
    .addGrid(lr.regParam, Array(0.1, 0.01))
    .build()
```

org.apache.spark.ml.tuning.CrossValidator

```scala
val cv = new CrossValidator()
    .setEstimator(pipeline)
    .setEvaluator(new BinaryClassificationEvaluator)
    .setEstimatorParamMaps(paramGrid)
    .setNumFolds(2)

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

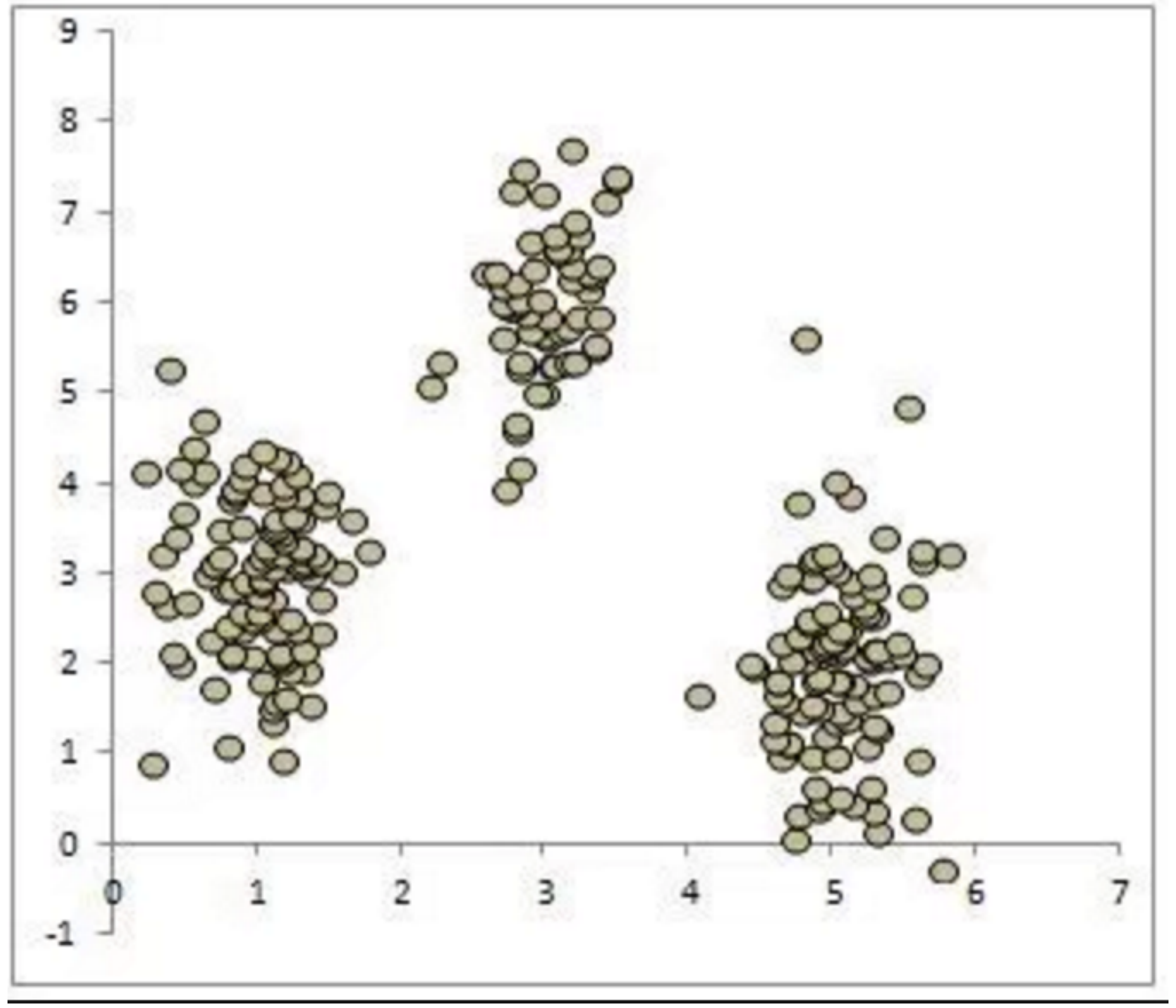
Model Tuning

Hands On Exercise

Clustering

(of data points) having similar numerical values.

Clustering



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

Tuning parameters: KMeans

featuresCol - dataframe column that specifies features
k - number of clusters to create. Default: 2
maxIter - maximum number of iterations
predictionCol - dataframe column that will have cluster ID prediction
tol - convergence tolerance
initMode (Advanced) - Initialization algorithm. **random** to chose random points for centroids. **k-means||** - (default) parallel variant of k-means++
initSteps (Advanced) - Number of steps for initialization mode. Default: 2

KMeans - Reading the Extracting Features

```
val inputData = session.read.option("header","true").option("inferSchema","true").csv(dataset)
```

Infer the schema from the header

Name	Day	Flight	Arrival Time	PayGrade
Kevin Doe	Saturday	VX 0906	2:30 PM	1
Darran Doe	Saturday	UA1248	6:00 PM	3
..	..	..	..	..

Generate Features

Name	Day	Flight	Arrival Time	PayGrade	Saturday	Sunday	Monday	dateFrac	Grade
Kevin Doe	Saturday	VX 0906	2:30 PM	1	1	0	0	0.6728	1
Darran Doe	Saturday	UA1248	6:00 PM	3	1	0	0	0.7935	3
..	..	..	..	..	..	..	..	..	..

org.apache.spark.ml.feature.VectorAssembler

Name	Day	Flight	Arrival Time	PayGrade	Saturday	Sunday	Monday	dateFrac	Grade	features
Kevin Doe	Saturday	VX 0906	2:30 PM	1	1	0	0	0.6728	1	[1.0,0.0,0.0,0.6728004600345026,1.0]
Darran Doe	Saturday	UA1248	6:00 PM	3	1	0	0	0.7935	3	[1.0,0.0,0.0,0.7935595169637722,3.0]
..	..	..	..	..	..	..	..	..	..	

KMeans - Scaling and Running Model

Name	Day	Flight	Arrival Time	PayGrade	Saturday	Sunday	Monday	dateFrac	Grade	features
Kevin Doe	Saturday	VX 0906	2:30 PM	1	1	0	0	0.6728	1	[1.0,0.0,0.0,0.6728004600345026,1.0]
Darran Doe	Saturday	UA1248	6:00 PM	3	1	0	0	0.7935	3	[1.0,0.0,0.0,0.7935595169637722,3.0]
..	..	..	..	..	..	..	..	..	..	..

org.apache.spark.ml.feature.MinMaxScaler

Name	Day	Flight	Arrival Time	PayGrade	Saturday	Sunday	Monday	dateFrac	Grade	features	scaled_features
Kevin Doe	Saturday	VX 0906	2:30 PM	1	1	0	0	0.6728	1	[1.0,0.0,0.0,0.6728004600345026,1.0]	[1.0,0.0,0.0,0.6935388263189093,0.0]
Darran Doe	Saturday	UA1248	6:00 PM	3	1	0	0	0.7935	3	[1.0,0.0,0.0,0.7935595169637722,3.0]	[1.0,0.0,0.0,0.8180201541197392,0.5]
..	..	..	..	..	..	..	..	..	..	..	..

org.apache.spark.ml.clustering.KMeans

Name	Day	Flight	Arrival Time	PayGrade	Saturday	Sunday	Monday	dateFrac	Grade	features	scaled_features	clusterId
Kevin Doe	Saturday	VX 0906	2:30 PM	1	1	0	0	0.6728	1	[1.0,0.0,0.0,0.6728004600345026,1.0]	[1.0,0.0,0.0,0.6935388263189093,0.0]	10
Darran Doe	Saturday	UA1248	6:00 PM	3	1	0	0	0.7935	3	[1.0,0.0,0.0,0.7935595169637722,3.0]	[1.0,0.0,0.0,0.8180201541197392,0.5]	13
..	..	..	..	..	..	..	..	..	..	..	..	..

KMeans - Code Walk Through

```
val dataset = "data/kmeans/flightinfo/flights_nofeatures.csv"
val inputData = session.read
    .option("header","true")
```

Read in the dataset

```
val isSat = udf {(x:String) => if (x.toLowerCase.equals("saturday")) 1 else 0}
val isSun = udf {(x: String) => if (x.toLowerCase.equals("sunday")) 1 else 0}
val isMon = udf {(x: String) => if (x.toLowerCase.equals("monday")) 1 else 0}

val transformedDay = inputData.withColumn("Saturday", isSat(inputData("Day")))
                     .withColumn("Sunday", isSun(inputData("Day")))
                     .withColumn("Monday", isMon(inputData("Day")))
val dayFract = udf {(x:String) =>
                 if (x == null)
                   0
                 else
                 {
                   val formatter = new java.text.SimpleDateFormat("h:m a")
                   val curr = formatter.parse(x).getTime.toDouble
                   val full = formatter.parse("11:59 PM").getTime.toDouble
                   curr/full
                 }
               }

val  toInt = udf {(s: String) =>
  s.toInt
}

val transformedTime = transformedDay.withColumn ("dateFract",dayFract(transformedDay("Arrival Time")))
                          .withColumn("Grade",toInt(transformedDay("PayGrade")))
```

```
val assembler = new VectorAssembler()
                   .setInputCols(Array("Saturday","Sunday","Monday","dateFract","Grade"))
                   .setOutputCol("features")

val featurizedData = assembler.transform(transformedTime)
```

Assemble into features vector

```
val scaler = new MinMaxScaler()
                   .setInputCol("features")
                     .setOutputCol("scaled_features")
val scalerModel = scaler.fit(featurizedData)
val scaledData = scalerModel.transform(featurizedData)
```

Scale the features

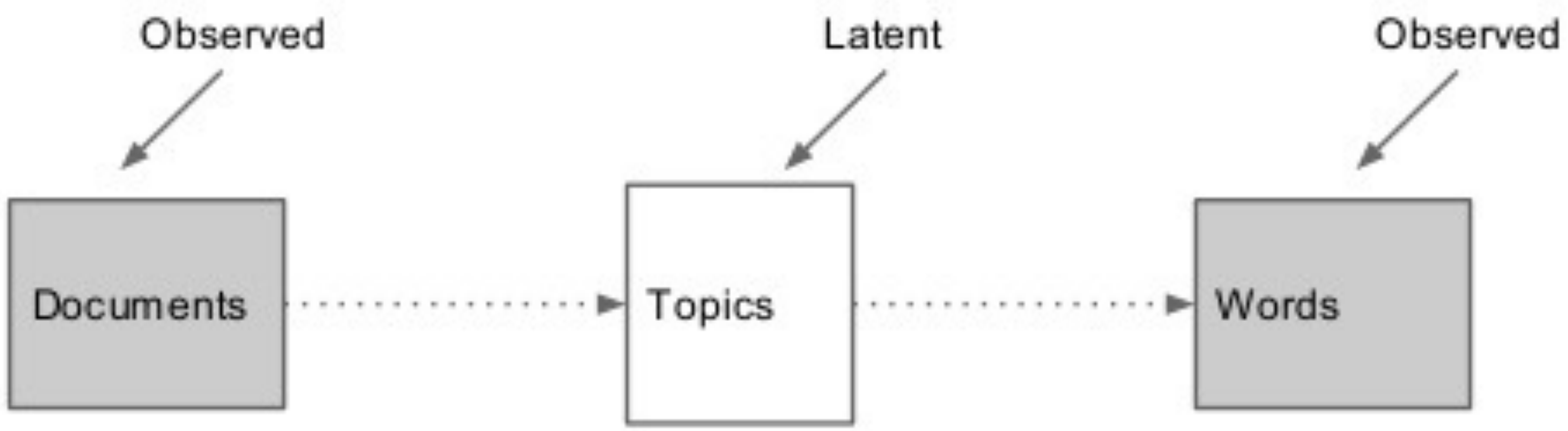
```
val kmeans = new KMeans()
  .setK(20)
  .setFeaturesCol("scaled_features")
  .setPredictionCol("clusterId")
val model = kmeans.fit(scaledData)
```

Generate Model

Hands On Exercise

Clustering - LDA

Goal of Topic Modeling



Documents are about several topics at the same time.
Topics are associated with different words.

Topics in the documents are expressed through the words that are used.

Tuning Parameters - LDA

- k : number of topics to be inferred
- maxIter: >0
- optimizer: online/em
- seed
- subSamplingRate: (for online only): should be adjusted in synch with LDA.maxIter so the entire corpus is used. Specifically, set both so that maxIterations * miniBatchFraction greater than or equal to 1.
- topicConcentration
- topicDistributionCol

Some house-keeping

- Get the word distribution
- Clean up the data
  - DataFrame => UDF => Word Distribution again to get a new stop word list

```
val cleanData = udf ({ text: String => text.replaceAll("[^A-Za-z ]", " ")})
```

- Splitting
- Stemming

Looking at the results

Topic 1	
space	0.012660059181
nasa	0.005330283688
program	0.005239824606
available	0.005144048115
system	0.004966346014
data	0.004792352055

Topic 2	
colorado	0.02660943268
guns	0.01395920562
ucsu	0.01308461546
udel	0.01298423116
firearms	0.01224593594
intercon	0.01082974947

Topic 3	
people	0.00774559688
turkish	0.00763194932
israel	0.00673282049
mideast	0.00664783679
jewish	0.00590779415
jews	0.00563967327

Topic 4	
news	0.020640614429
baseball	0.019538283096
sport	0.012547926648
game	0.010113930332
subject	0.009844881506
organization	0.009577969868

Topic 5	
duke	0.013943643237
team	0.009899371553
hockey	0.008075055082
sport	0.008021120554
news	0.007865084086
ulowell	0.007315961791
league	0.007098242372
baseball	0.006199597687

Topic 6	
rutgers	0.025988077707
christian	0.017065131308
religion	0.013231107212
writes	0.009754402317
talk	0.009659720709
lines	0.009513981142

Topic Modeling - Code Walk-through

```
val rawTextRDD =
spark.sparkContext.wholeTextFiles(inputDi
r).map(_._2)
val docDF = rawTextRDD
  .zipWithIndex.toDF("text", "docId")
```

Read the input documents and the stopwords File

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

Filter the content based on stop words

```
val ngram = new NGram()
   .setInputCol("filtered")
   .setOutputCol("ngrams")
   .transform(filteredTokens)
```

ngram

```
val lda = new LDA()
  .setOptimizer("online")
  .setK(numTopics)
  .setMaxIter(maxIterations)
```

LDA

LDA

Hands On Exercise

Classification

The action or process of classifying something according to shared qualities or characteristics.

Random Forests in spark.ml

- Support both binary and multi class classification and regression

- Use both Categorical and Continuous features

- Random forests handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.

- Implements random forests using the existing decision tree implementation.

Random Forests in spark.ml - Implementation

Training

- Since training of decisions trees is done separately, the training is done in parallel.
- Randomness injected into the training process includes:
  - Subsampling the original dataset on each iteration to get a different training set.
  - Considering different random subsets of features to split at each tree node

Prediction

- Aggregates predictions from its set of decision trees.
- Classification: Majority vote
- Regression: Averaging

Random Forests : Tuning parameters

- numTrees: Number of trees in the Forest => accuracy vs speed
- maxDepth: Expressive vs overfitting. More suitable for random forests than a single decision tree.
- subsamplingRate
- featuredSubsetStrategy - auto, all, onethird, sort, log2, n
- impurity: entropy, gini
- minInfoGain
- minInstancePerNode

Random Forests in ml : Input and Output Columns

Input Column	Types	Default	Description
labelCol	Double	“label”	Label to Predict
featuresCol	Vector	“features”	Feature Vector

Output Column	Types	Default	Description
predictionCol	Double	"prediction"	Predicted label
rawPredictionCol	Vector	"rawPrediction"	Vector of length # classes, with the counts of training instance labels at the tree node which makes the prediction
probabilityCol	Vector	"probability"	Vector of length # classes equal to rawPrediction normalized to a multinomial distribution

Random Forest Classifier - Code Walk Through

Split the dataset

```
val Array(trainingData, testData) = countVectors
  .randomSplit(Array(0.7, 0.3))
```

org.apache.spark.ml.feature.RandomForestClassifier

```
val rf = new RandomForestClassifier()
  .setLabelCol("indexedLabel")
  .setFeaturesCol("indexedFeature
```

org.apache.spark.ml.feature.IndexToString

```
val labelConverter = new IndexToString()
  .setInputCol("prediction")
  .setOutputCol("predictedLabel")
  .setLabels(labelIndexer.labels)
```

Evaluate the model

```
val evaluator = new MulticlassClassificationEvaluator()
  .setLabelCol("indexedLabel")
  .setPredictionCol("prediction")
  .setMetricName("accuracy")
```

Random Forest Classifier: DataSet

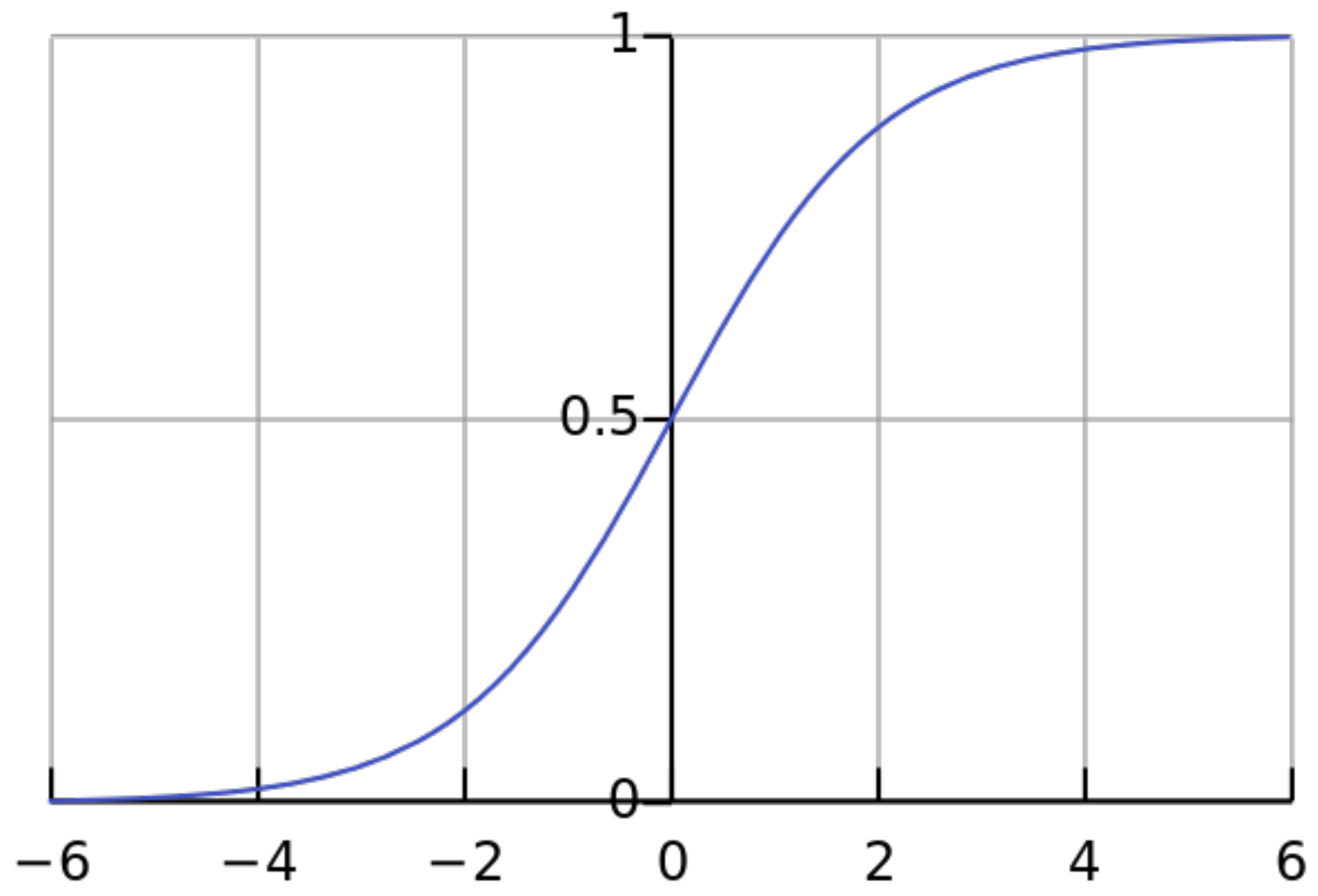
- Movie Review data set
- Exercise

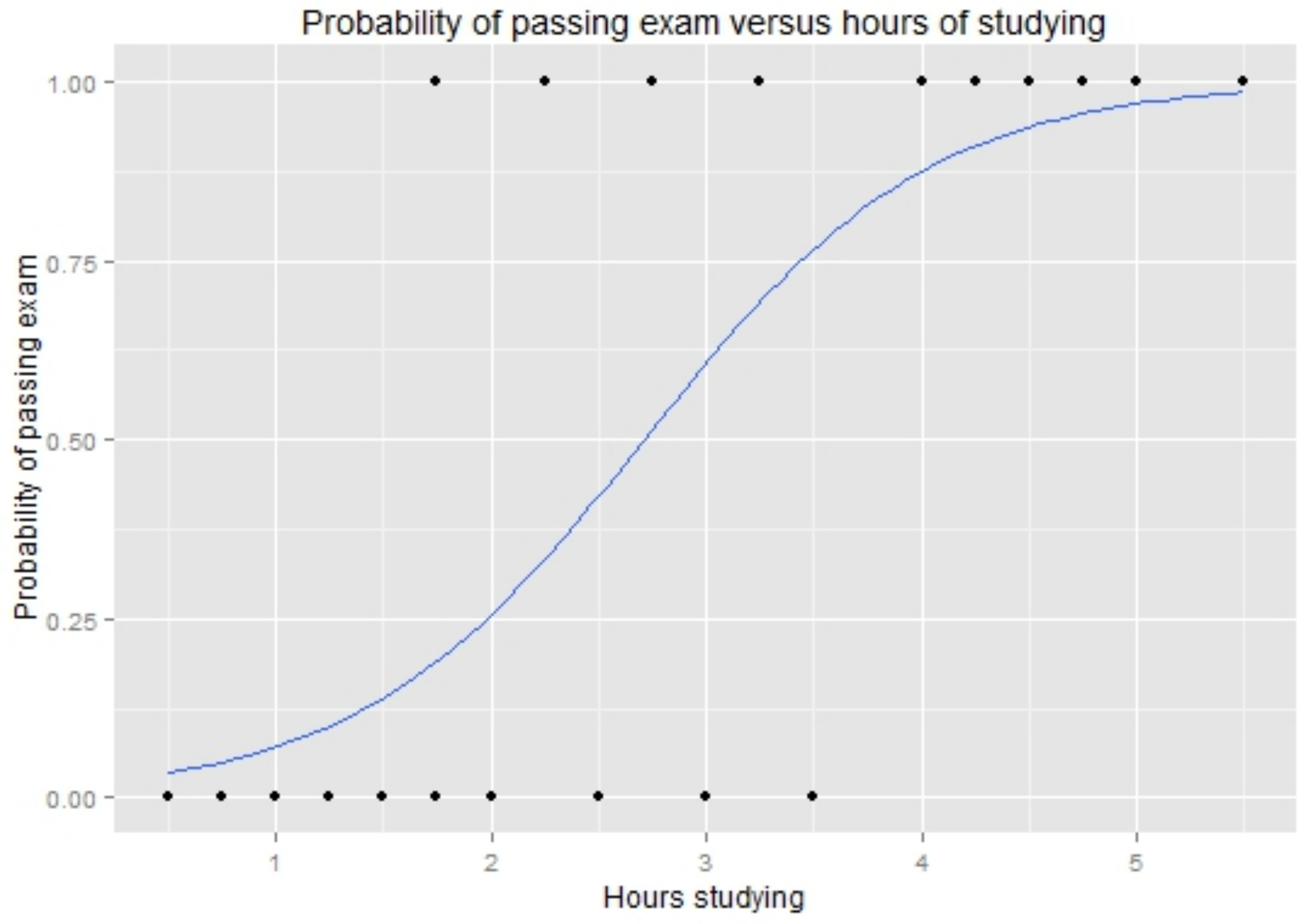
Random Forest Classifier

Hands On Exercise

Classification - Logistic Regression

- Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution.





Logistic Regression

- SMS Spam Collection Dataset
- https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection#

Look into the Data...

spam	message
StringType	StringType
ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
ham	Ok lar... Joking wif u oni...
spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's
ham	U dun say so early hor... U c already then say...
ham	Nah I don't think he goes to usf, he lives around here though
spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv
ham	Even my brother is not like to speak with me. They treat me like aids patent.

Logistic Regression - Code Walk-through

```
val customSchema = StructType(Array(
    StructField("spam", StringType, true),
    StructField("message", StringType, true)
  ))
val ds = spark.read.option("inferSchema", "true").option("delimiter",
"\t").schema(customSchema).csv("data/SMSSpamCollection.tsv")
```

Read the input documents

```
// tokenize
val tokenizer = new Tokenizer().setInputCol("message").setOutputCol("tokens")
val tokdata = tokenizer.transform(indexed)

// tf
val hashingTF = new HashingTF()
  .setInputCol("tokens").setOutputCol("tf")//.setNumFeatures(20)
val tfdata = hashingTF.transform(tokdata)

// idf
val idf = new IDF().setInputCol("tf").setOutputCol("idf")
val idfModel = idf.fit(tfdata)
val idfdata = idfModel.transform(tfdata)
```

Tokenize/TF/IDF the message column

```
val assembler = new VectorAssembler()
  .setInputCols(Array("idf"))
  .setOutputCol("features")

val assemdata = assembler.transform(idfdata)

// split
val Array(trainingData, testData) =
assemdata.randomSplit(Array(0.7, 0.3), 1000)
```

VectorAssembler & Split

```
val lr = new LogisticRegression()
  .setLabelCol("label")
  .setFeaturesCol("features")

// Fit the model
val lrModel = lr.fit(trainingData)

// predict
val predict = lrModel.transform(testData)

predict.show(100)

val evaluator = new BinaryClassificationEvaluator()
  //.setLabelCol("indexedLabel")
  .setRawPredictionCol("prediction")
  .setMetricName("precision")

val accuracy = evaluator.evaluate(predict)
```

Train, Predict, Evaluate the model

Input & Output Columns

Input Column	Types	Default	Description
labelCol	Double	“label”	Label to Predict
featuresCol	Vector	“features”	Feature Vector

Output Column	Types	Default	Description
predictionCol	Double	"prediction"	Predicted label
rawPredictionCol	Vector	"rawPrediction"	Vector of length # classes, with the counts of training instance labels at the tree node which makes the prediction
probabilityCol	Vector	"probability"	Vector of length # classes equal to rawPrediction normalized to a multinomial distribution

Tuning Parameters

maxIter - maximum number of Iterations
tol - convergence tolerance of Iterations
regParam - regularization parameter
threshold - threshold in binary classification

Results

spam	message	spam_idx	ner	tf	rawPrediction	probability	prediction
StringType	StringType	DoubleType	ArrayType(StringType,true)	org.apache.spark.mllib.linalg.VectorUDT@f71b0bce	org.apache.spark.mllib.linalg.VectorUDT@f71b0bce	org.apache.spark.mllib.linalg.VectorUDT@f71b0bce	DoubleType
ham	Ok lar... Joking wif u oni...	0.0	WrappedArray(ok, lar..., joking, wif, u, oni...)	(1000,[117,401,508,548,596,716],[1.0,1.0,1.0,1.0,1.0,1.0])	[41.59998777365775,-41.59998777365775]	[1.0,8.57738418290502E-19]	0.0
ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune	0.0	WrappedArray(as, per, your, request, 'melle, melle, (oru, minnaminunginte, nurungu, vettam)', has, been, set, as, your, callertune, for, all, callers., press, *9, to, copy, your, friends, callertune)	(1000,[63,66,122,123,267,329,359,434,573,577,673,685,707,762,798,806,820,877,917,943,955,962], [1.0,1.0,2.0,3.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0])	[23.079968654702505,-23.079968654702505]	[0.999999999905268,9.47320576757792E-11]	0.0
ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to	0.0	WrappedArray(as, per, your, request, 'melle, melle, (oru, minnaminunginte, nurungu, vettam)', has, been, set, as, your, callertune, for, all, callers.,	(1000,[63,66,122,123,267,329,359,434,573,577,673,685,707,762,798,806,820,877,917,943,955,962], [1.0,1.0,2.0,3.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0])	[23.079968654702505,-23.079968654702505]	[0.999999999905268,9.47320576757792E-11]	0.0

Telco Churn Prediction

- Telco Churn Prediction
- https://www.sgi.com/tech/mlc/db/churn.names
- https://www.sgi.com/tech/mlc/db/churn.all

Random Forest - Code Walk-through

```
val ds = spark.read.option("inferSchema",
"true").schema(customSchema).csv("data/churn.all")
```

Read the input documents

```
val indexer = new StringIndexer()
      .setInputCol("intl_plan")
      .setOutputCol("intl_plan_idx")
val indexed = indexer.fit(ds).transform(ds)
```

Index the intl_plan column

```
// vector assembler
val assembler = new VectorAssembler()
  .setInputCols(Array("account_length",
"intl_plan_idx", "number_vmail_messages",
"total_day_minutes", "total_day_calls"))
  .setOutputCol("features")

val assemdata = assembler.transform(churned)
```

Assemble the features

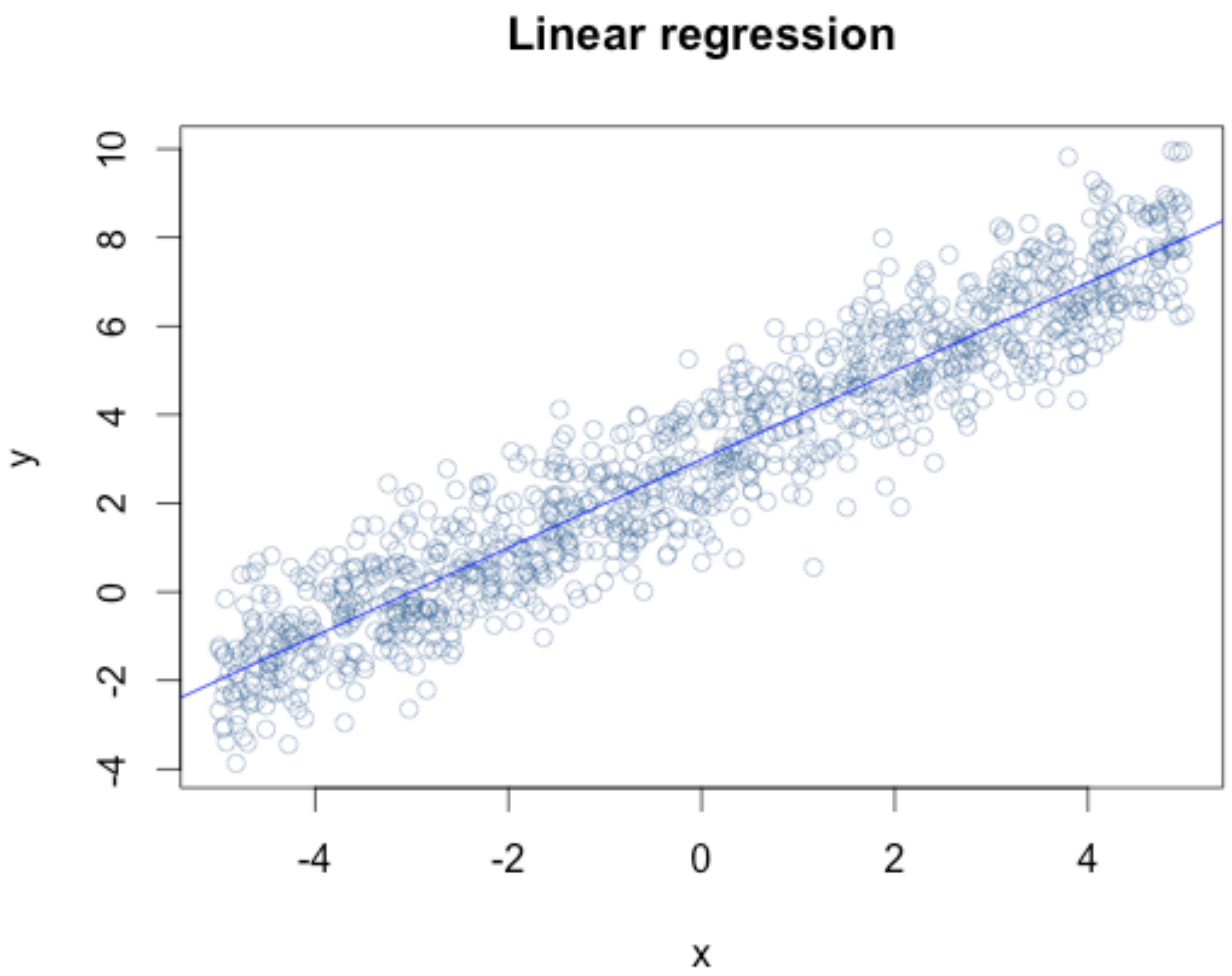
```
// Train a RandomForest model.
val rf = new RandomForestClassifier()
  .setLabelCol("churned_idx")
  .setFeaturesCol("features")
  .setNumTrees(10)

// Fit the model
val rfModel = rf.fit(trainingData)
```

Train the Model

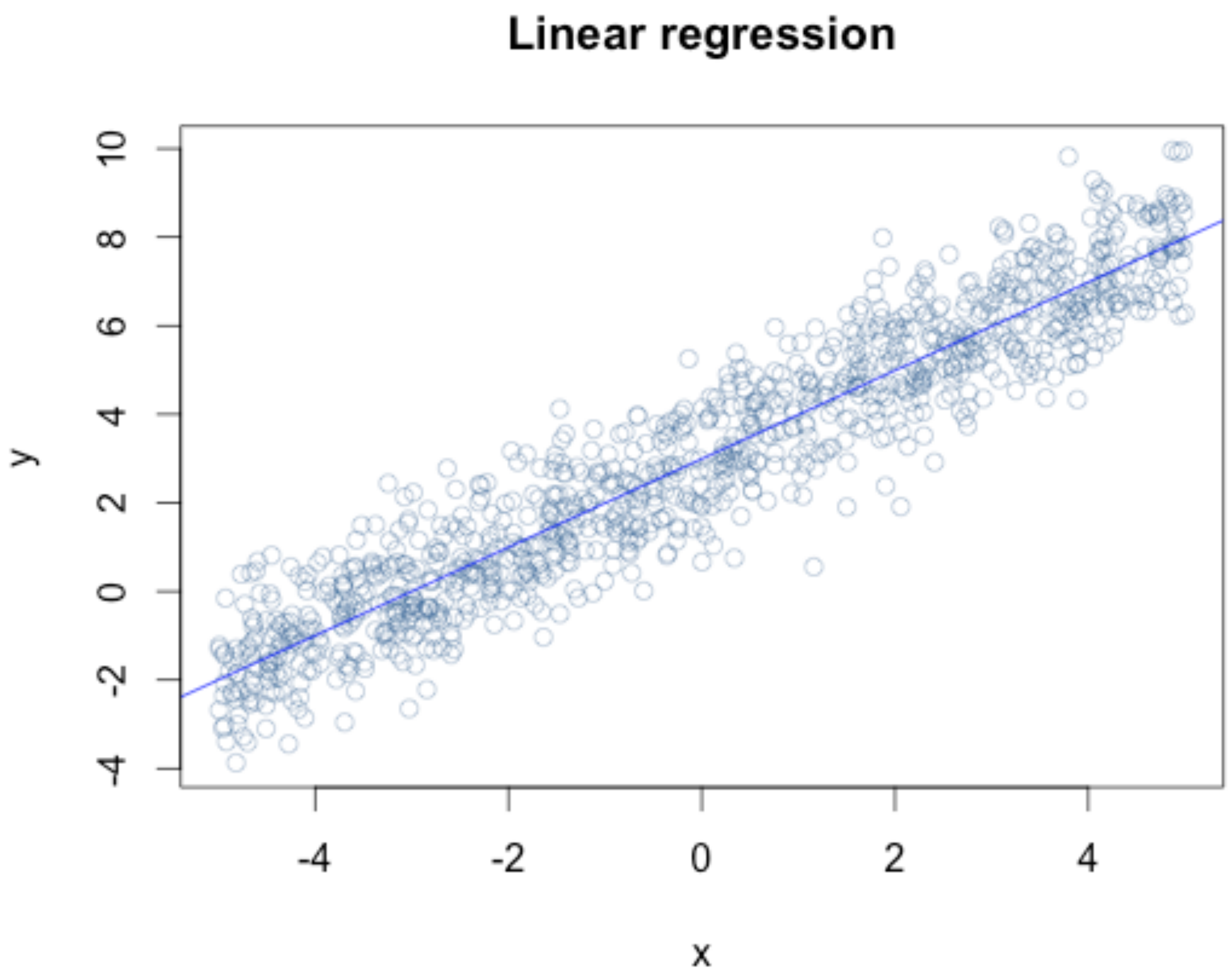
Regression

Regression allows you to make predictions from data by learning the relationship between features of your data and some observed, continuous-valued response.



Regression - Linear Methods

- Modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. => y^ = w` x , where w ∈ R(n), is a vector of parameters.



Generalized Linear Regression

- Specifications of Linear models where the response variable follows some distribution from the exponential family of distributions.

- Spark's GeneralizedLinearRegression interface allows for flexible specification of GLMs which can be used for various types of prediction problems including linear regression, Poisson regression, logistic regression, and others.

Linear Regression - Predict Housing Prices

```
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)
```

The case class to map the data in to

```
val paramGrid = new ParamGridBuilder()
        .addGrid(lr.regParam, Array(0.1, 0.01, 0.001))
        .addGrid(lr.fitIntercept)
        .addGrid(lr.elasticNetParam, Array(0.0, 1.0))
```

Construct the Parameter Grid

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

Form the Validator

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

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

Split and train the data

Gradient-boosted Trees Regression

- Gradient-boosted trees (GBTs) are a popular regression method using ensembles of decision trees.
- GBTs iteratively train decision trees in order to minimize a loss function. The spark.ml implementation supports GBTs for binary classification and for regression, using both continuous and categorical features.
- It produces a prediction model in the form of an ensemble of weak prediction models.

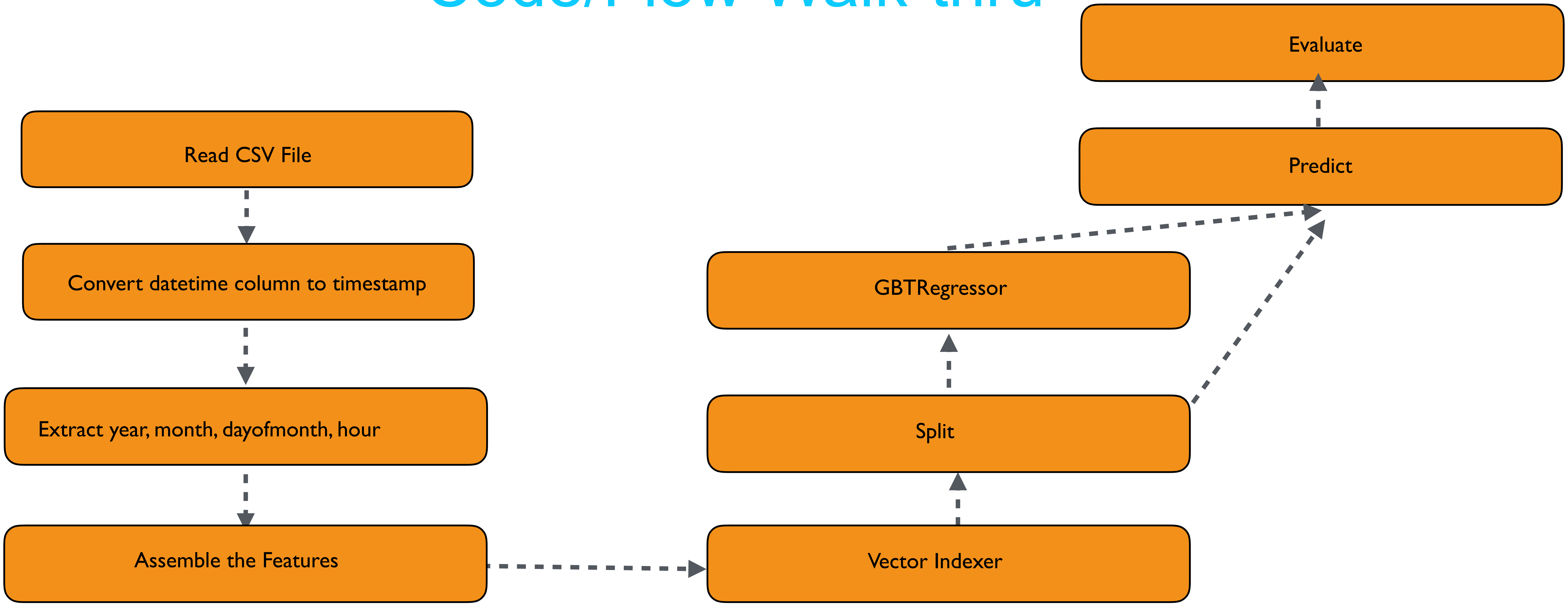
  - Bike Sharing Dataset
  - http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset

Look into the Data...

datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
TimestampType	IntegerType	IntegerType	IntegerType	IntegerType	DoubleType	DoubleType	IntegerType	DoubleType	IntegerType	IntegerType	IntegerType
2011-01-01 00:00:00.0	1	0	0	1	9.84	14.395	81	0.0	3	13	16
2011-01-01 01:00:00.0	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2011-01-01 02:00:00.0	1	0	0	1	9.02	13.635	80	0.0	5	27	32
2011-01-01 03:00:00.0	1	0	0	1	9.84	14.395	75	0.0	3	10	13
2011-01-01 04:00:00.0	1	0	0	1	9.84	14.395	75	0.0	0	1	1
2011-01-01 05:00:00.0	1	0	0	2	9.84	12.88	75	6.0032	0	1	1

https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset

Code/Flow Walk-thru



Input & Output Columns

Input Column	Types	Default	Description
labelCol	Double	“label”	Label to Predict
featuresCol	Vector	“features”	Feature Vector
predictionCol	Double	"prediction"	Prediction

Tuning Parameters

numIterations - number of trees in the Ensemble. Each Iteration produces one tree
learningRate - if the algorithm behavior seems unstable, decreasing this value may improve stability
loss - loss function

Prediction

datetime_month	datetime_dayofmonth	datetime_hour	count	feature_vector	feature_vector_index	prediction
IntegerType	IntegerType	IntegerType	DoubleType	org.apache.spark.mllib.linalg.VectorUDT@f71b0bce	org.apache.spark.mllib.linalg.VectorUDT@f71b0bce	DoubleType
1	1	1	40.0	[1.0,0.0,0.0,1.0,80.0,2011.0,1.0,1.0,1.0,9.02,13.635,0.0]	[0.0,0.0,0.0,0.0,80.0,0.0,0.0,0.0,1.0,9.02,13.635,0.0]	48.75757260342313
1	1	3	13.0	[1.0,0.0,0.0,1.0,75.0,2011.0,1.0,1.0,3.0,9.84,14.395,0.0]	[0.0,0.0,0.0,0.0,75.0,0.0,0.0,0.0,3.0,9.84,14.395,0.0]	15.842618061222538
1	1	3	13.0	[1.0,0.0,0.0,1.0,75.0,2011.0,1.0,1.0,3.0,9.84,14.395,0.0]	[0.0,0.0,0.0,0.0,75.0,0.0,0.0,0.0,3.0,9.84,14.395,0.0]	15.842618061222538
1	1	8	8.0	[1.0,0.0,0.0,1.0,75.0,2011.0,1.0,1.0,8.0,9.84,14.395,0.0]	[0.0,0.0,0.0,0.0,75.0,0.0,0.0,0.0,8.0,9.84,14.395,0.0]	96.54450534895048
1	1	9	14.0	[1.0,0.0,0.0,1.0,76.0,2011.0,1.0,1.0,9.0,13.12,17.425,0.0]	[0.0,0.0,0.0,0.0,76.0,0.0,0.0,0.0,9.0,13.12,17.425,0.0]	56.14313366729195
1	1	10	36.0	[1.0,0.0,0.0,1.0,76.0,2011.0,1.0,1.0,10.0,15.58,19.695,16.9979]	[0.0,0.0,0.0,0.0,76.0,0.0,0.0,0.0,10.0,15.58,19.695,7.0]	53.23957701518251
1	1	18	35.0	[1.0,0.0,0.0,3.0,88.0,2011.0,1.0,1.0,18.0,17.22,21.21,16.9979]	[0.0,0.0,0.0,2.0,88.0,0.0,0.0,0.0,18.0,17.22,21.21,7.0]	95.43205213150503
3	1	7	64.0	[1.0,0.0,1.0,1.0,50.0,2011.0,3.0,1.0,7.0,5.74,6.82,12.998]	[0.0,0.0,1.0,0.0,50.0,0.0,0.0,2.0,0.0,7.0,5.74,6.82,5.0]	115.62180696705387

Take aways

- Deep learning allows the computer to build complex concepts out of simpler concepts.

- Choices in CDH stack
  - SparkML
  - MLLib

- Other new exciting technologies
  - Tensor flow
  - Caffe

- Use the algorithms to unlock the value in data as oppose to lock yourself with a specific technology

- For Examples of Technology Application:
  - http://github.mtv.cloudera.com/DataScience/nlp

Thank you!!!!! :)

2:40pm–3:20pm Thursday, March 16, 2017

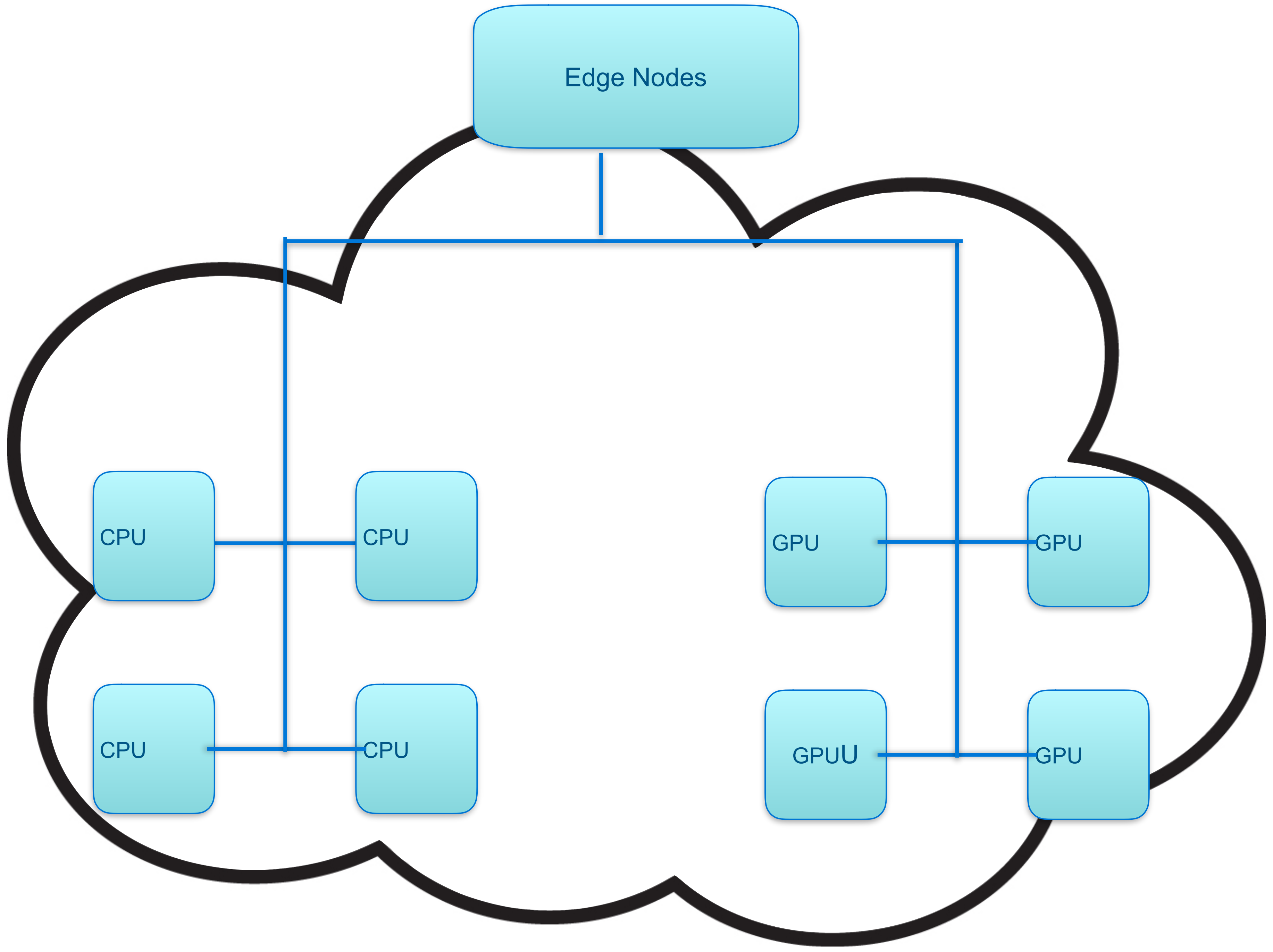
Ask me anything: Unraveling data with Spark using machine learning

Ask Me Anything

Location: 212 A-B

Vartika Singh (Cloudera), Jayant Shekhar (Sparkflows Inc.), Jeffrey Shmain (Cloudera)

CaffeOnSpark - Demo



- Launch Caffe engines on GPU devices or CPU devices within the Spark executor.
- Invokes a JNI layer with fine-grain memory management.
- Unlike traditional Spark applications, CaffeOnSpark executors communicate to each other via MPI allreduce style interface via TCP/Ethernet or RDMA/ Infiniband.
- This Spark+MPI architecture enables CaffeOnSpark to achieve similar performance as dedicated deep learning clusters.
- Checkpointing: CaffeOnSpark enables training state being snapshotted periodically, thus we could resume from previous state after a failure of a CaffeOnSpark job.

CaffeOnSpark - Demo

- Talk about Implementation details

CPU and GPU - Best of both worlds!

- CaffeOnSpark
- DeepLearning4j

- GPU is Optimized for taking huge batches of data and performing the same operation over and over
- GPUs are special purpose and can compute vector maths, matrix maths, pixel transforms and rendering jobs about 10-100x faster than the equivalent CPU

- Architecturally, the CPU is composed of just few cores with lots of cache memory that can handle a few software threads at a time.
- They excel in serial tasks, branching operations and file operations.