Spark Machine Learning

Amir H. Payberah
amir@sics.se

SICS Swedish ICT
June 30, 2016

Data


Data
Actionable Knowledge

That is roughly the problem that Machine Learning addresses!

Data and Knowledge

Data ⟶ Knowledge

- Is this email spam or no spam?

Data and Knowledge

Data ⟶ Knowledge

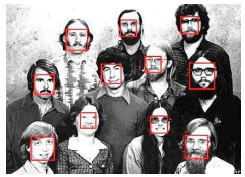
- Is this email spam or no spam?

Data and Knowledge

Data ⟶ Knowledge

- Is this email spam or no spam?

- Is there a face in this picture?

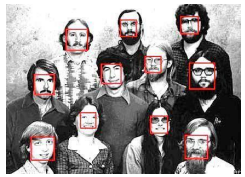


Data and Knowledge

Data ⟶ Knowledge

- Is this email spam or no spam?

- Is there a face in this picture?

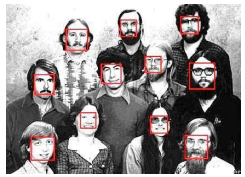


Data and Knowledge

Data ⟶ Knowledge

- Is this email spam or no spam?

- Is there a face in this picture?



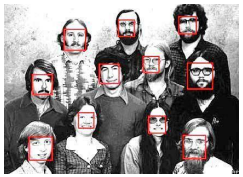
- Should I lend money to this customer given his spending b

Data and Knowledge

Data ⟶ Knowledge

- Is this email spam or no spam?

- Is there a face in this picture?



- Should I lend money to this customer given his spending behaviour?

Data and Knowledge

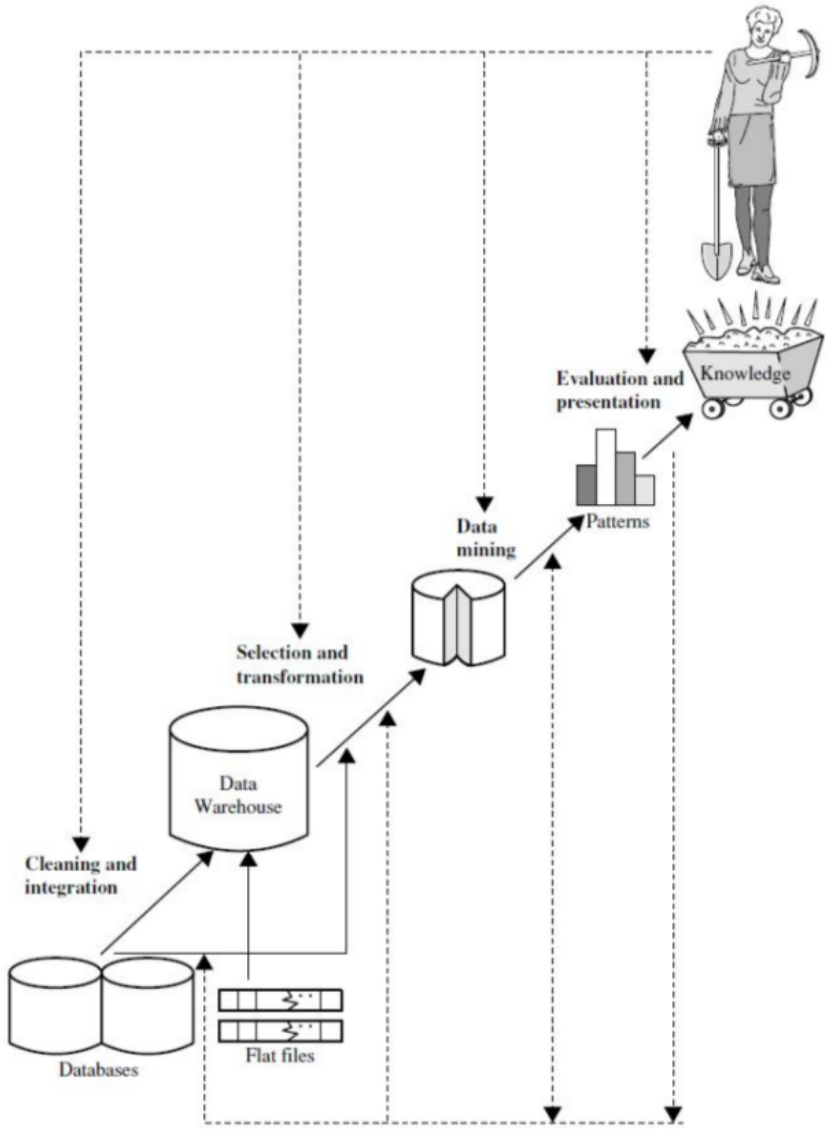
- Knowledge is not concrete

- Spam is an abstraction
- Face is an abstraction
- Who to lend to is an abstraction

You do not find spam, faces, and financial advice in datasets, you just find bits!

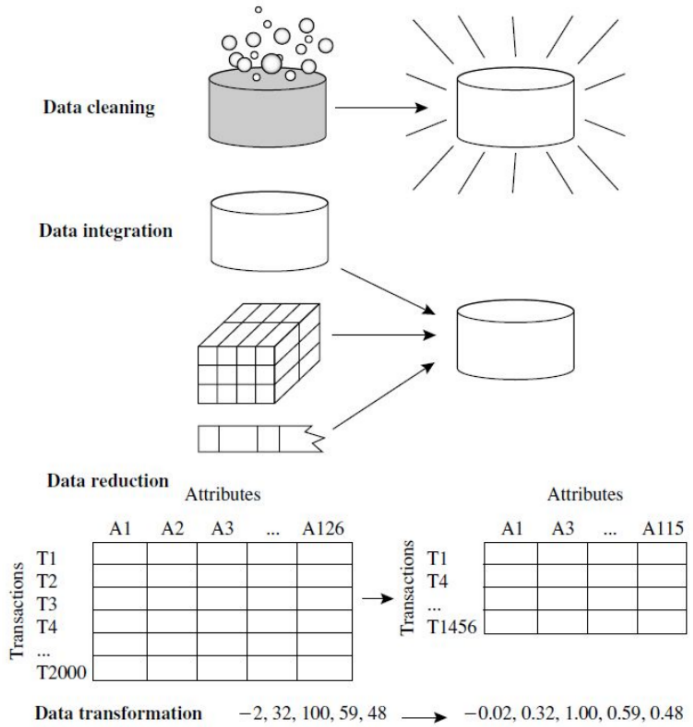
Knowledge Discovery from Data (KDD)

- Preprocessing
- Data mining
- Result validation



KDD - Preprocessing

- Data cleaning
- Data integration
- Data reduction, e.g., sampling
- Data transformation, e.g., normalization



KDD - Mining Functionalities

- Classification and regression (supervised learning)
- Clustering (unsupervised learning)
- Mining the frequent patterns
- Outlier detection

KDD - Result Validation

- Needs to evaluate the performance of the model on some criteria.
- Depends on the application and its requirements.

MLlib - Data Types

Data Types - Local Vector

- Stored on a single machine
- Dense and sparse
  - Dense (1.0, 0.0, 3.0): [1.0, 0.0, 3.0]
  - Sparse (1.0, 0.0, 3.0): (3, [0, 2], [1.0, 3.0])

```
val dv: Vector = Vectors.dense(1.0, 0.0, 3.0)

val sv1: Vector = Vectors.sparse(3, Array(0, 2), Array(1.0, 3.0))
val sv2: Vector = Vectors.sparse(3, Seq((0, 1.0), (2, 3.0)))
```

Data Types - Labeled Point

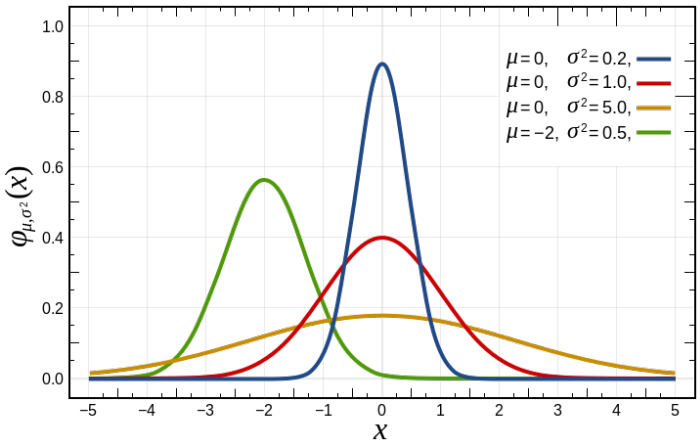
- A local vector (dense or sparse) associated with a label.
- `label`: label for this data point.
- `features`: list of features for this data point.

```
case class LabeledPoint(label: Double, features: Vector)

val pos = LabeledPoint(1.0, Vectors.dense(1.0, 0.0, 3.0))

val neg = LabeledPoint(0.0, Vectors.sparse(3, Array(0, 2), Array(1.0, 3.0)))
```

MLlib - Preprocessing

Data Transformation - Normalizing Features

- To get data in a standard Gaussian distribution: $\frac{x - mean}{sqrt(variance)}$




```
val features = labelData.map(_.features)

val scaler = new StandardScaler(withMean = true, withStd = true).fit(features)

val scaledData = labelData.map(lp => LabeledPoint(lp.label,
                                        scaler.transform(lp.features)))
```

MLlib - Data Mining

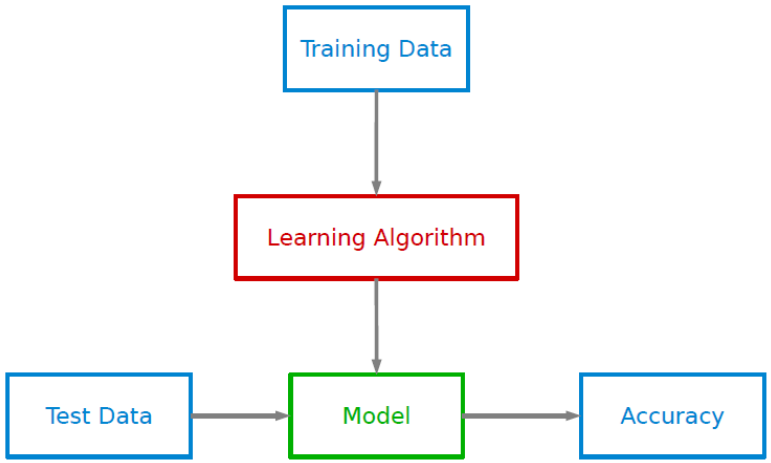
Data Mining Functionalities

- Classification and regression (supervised learning)
- Clustering (unsupervised learning)
- Mining the frequent patterns
- Outlier detection

Classification and Regression (Supervised Learning)

Supervised Learning (1/3)

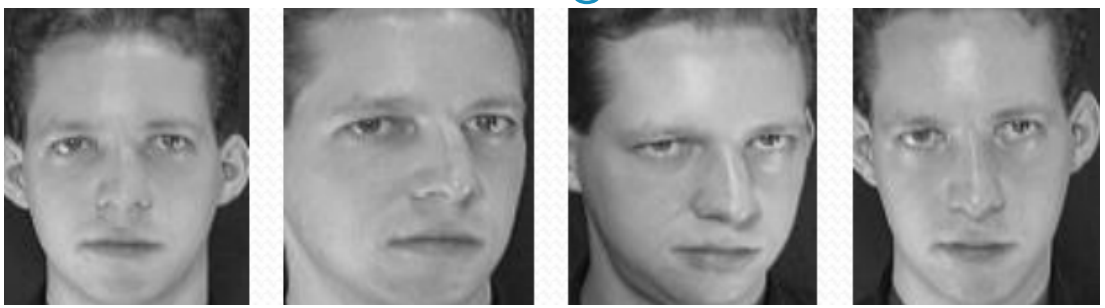
- Right answers are given.
  - Training data (input data) is labeled, e.g., spam/not-spam or a stock price at a time.
- A model is prepared through a training process.
- The training process continues until the model achieves a desired level of accuracy on the training data.



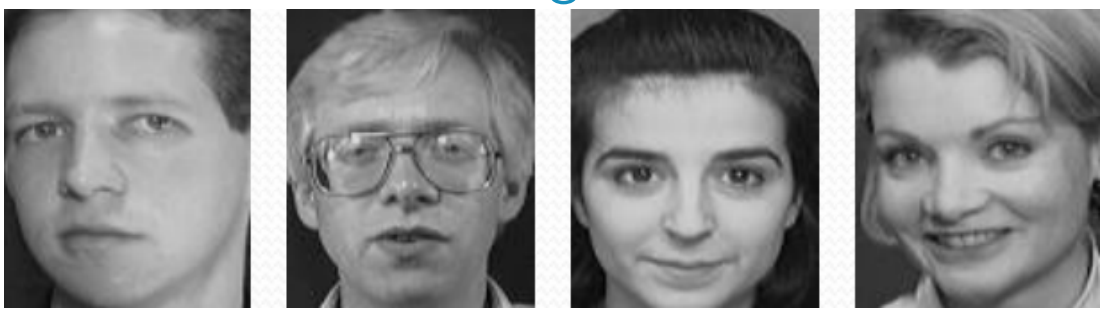
Supervised Learning (2/3)

- Face recognition

Training data



Testing data



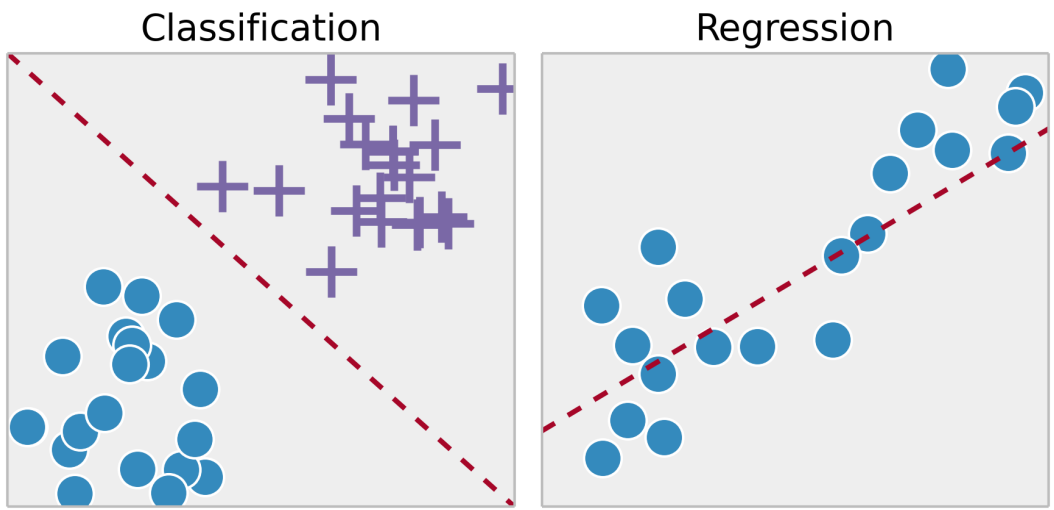
[ORL dataset, AT&T Laboratories, Cambridge UK]

Supervised Learning (3/3)

- Set of N training examples: $(x_1, y_1), \cdots, (x_n, y_n)$.
- $x_i = \langle x_{i1}, x_{i2}, \cdots, x_{im} \rangle$ is the feature vector of the ith example.
- y_i is the ith feature vector label.
- A learning algorithm seeks a function $y_i = f(X_i)$.

Classification vs. Regression

- Classification: the output variable takes class labels.
- Regression: the output variable takes continuous values.



Types of Classification/Regression Models in Spark

- Linear models
- Decision trees
- Naive Bayes models

Linear Models

Linear Models

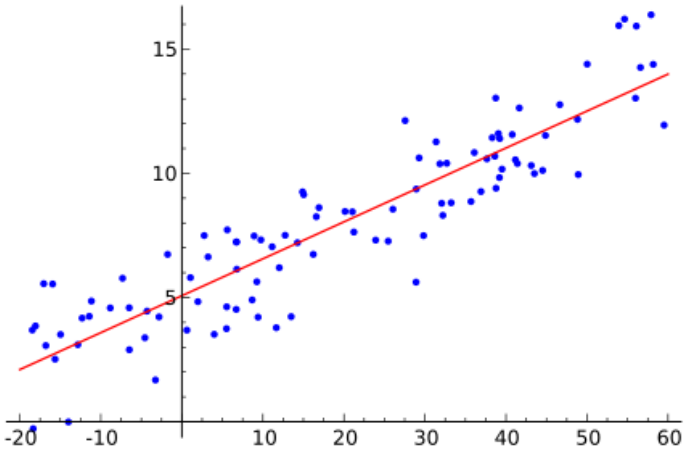
- Training dataset: $(x_1, y_1), \cdots, (x_n, y_n)$.
- $x_i = \langle x_{i1}, x_{i2}, \cdots, x_{im} \rangle$
- Model the target as a function of a linear predictor applied to the input variables: $y_i = g(\mathbf{w}^T \mathbf{x}_i)$.
  - E.g., $y_i = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_m x_{im}$

Linear Models

- Training dataset: $(\mathtt{x}_1, \mathtt{y}_1), \cdots, (\mathtt{x}_n, \mathtt{y}_n)$.
- $\mathtt{x}_i = \langle \mathtt{x}_{i1}, \mathtt{x}_{i2}, \cdots, \mathtt{x}_{im} \rangle$
- Model the target as a function of a linear predictor applied to the input variables: $\mathtt{y}_i = \mathtt{g}(\mathbf{w}^T \mathbf{x}_i)$.
  - E.g., $\mathtt{y}_i = \mathtt{w}_1 \mathtt{x}_{i1} + \mathtt{w}_2 \mathtt{x}_{i2} + \cdots + \mathtt{w}_m \mathtt{x}_{im}$
- Loss function: $f(\mathbf{w}) := \sum_{i=1}^{n} L(g(\mathbf{w}^T \mathbf{x}_i), y_i)$
- An optimization problem $\min_{\mathbf{w} \in \mathbb{R}^m} \mathtt{f}(\mathbf{w})$

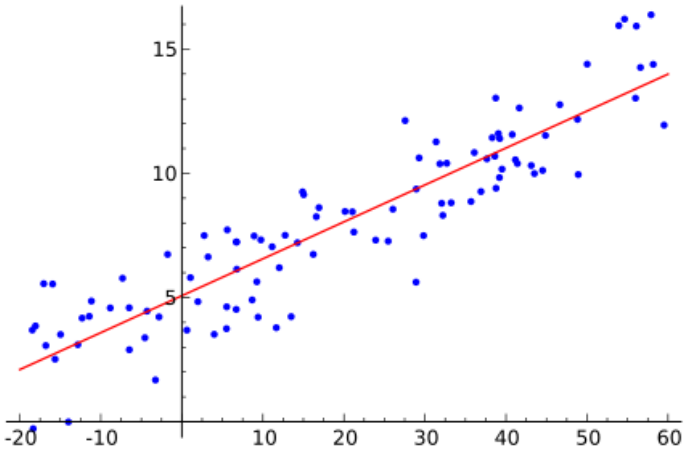
Linear Models - Regression (1/2)

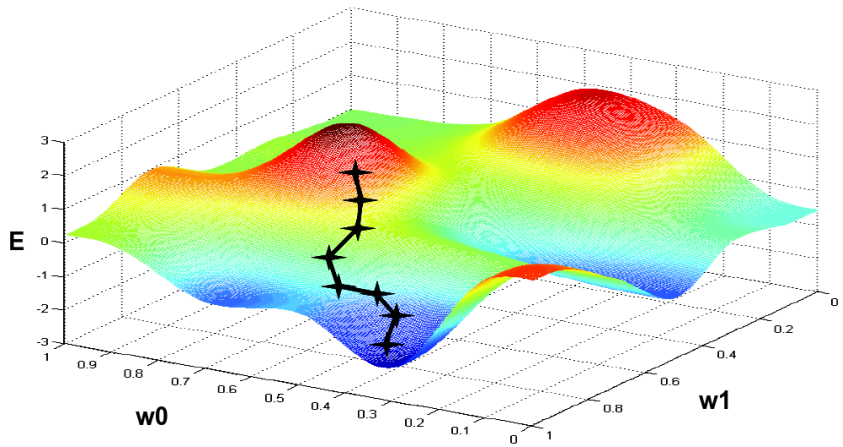
- $g(\mathbf{w}^T\mathbf{x}_i) = w_1x_{i1} + w_2x_{i2} + \cdots + w_mx_{im}$
- Loss function: minimizing squared different between predicted value and actual value: $L(g(\mathbf{w}^T\mathbf{x}_i), y_i) := \frac{1}{2}(\mathbf{w}^T\mathbf{x}_i - y_i)^2$



Linear Models - Regression (1/2)

- $g(\mathbf{w}^T\mathbf{x}_i) = w_1x_{i1} + w_2x_{i2} + \cdots + w_mx_{im}$
- Loss function: minimizing squared different between predicted value and actual value: $L(g(\mathbf{w}^T\mathbf{x}_i), y_i) := \frac{1}{2}(\mathbf{w}^T\mathbf{x}_i - y_i)^2$
- Gradient descent



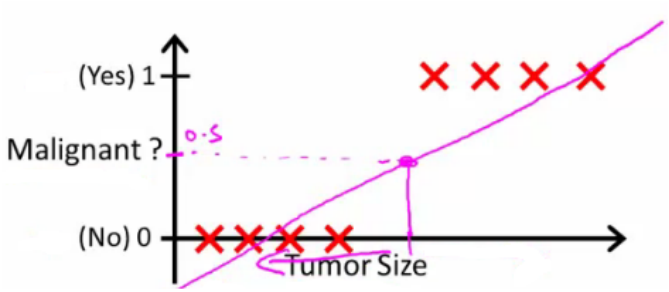


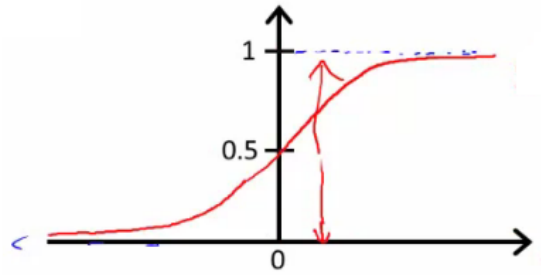
Linear Models - Regression (2/2)

```
val data: RDD[LabeledPoint] = ...
val splits = labelData.randomSplit(Array(0.7, 0.3))
val (trainigData, testData) = (splits(0), splits(1))

val numIterations = 100
val stepSize = 0.00000001
val model = LinearRegressionWithSGD
  .train(trainigData, numIterations, stepSize)

val valuesAndPreds = testData.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
```

Linear Models - Classification (Logistic Regression) (1/2)

- Binary classification: output values between 0 and 1
- $g(\mathbf{w}^T\mathbf{x}) := \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$ (sigmoid function)
- If $g(\mathbf{w}^T\mathbf{x}_i) > 0.5$, then $y_i = 1$, else $y_i = 0$





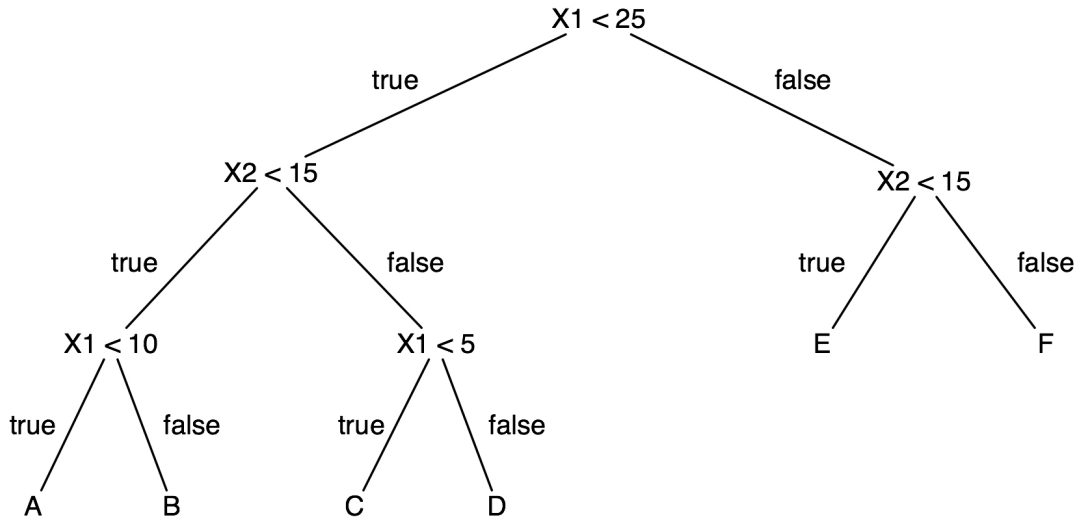
Linear Models - Classification (Logistic Regression) (2/2)

```
val data: RDD[LabeledPoint] = ...
val splits = labelData.randomSplit(Array(0.7, 0.3))
val (trainigData, testData) = (splits(0), splits(1))

val model = new LogisticRegressionWithLBFGS()
  .setNumClasses(10)
  .run(trainingData)

val predictionAndLabels = testData.map { point =>
  val prediction = model.predict(point.features)
  (prediction, point.label)
}
```

Decision Tree

Decision Tree

- A greedy algorithm.
- It performs a recursive binary partitioning of the feature space.
- Decision tree construction algorithm:
  - Find the best split condition (quantified based on the impurity measure).
  - Stops when no improvement possible.



Impurity Measure

- Measures how well are the two classes separated.
- The current implementation in Spark:
  - Regression: `variance`
  - Classification: `gini` and `entropy`

Stopping Rules

- The node depth is equal to the `maxDepth` training parameter.
- No split candidate leads to an information gain greater than `minInfoGain`.
- No split candidate produces child nodes which each have at least `minInstancesPerNode` training instances.

Decision Tree - Regression

```
val data: RDD[LabeledPoint] = ...
val splits = data.randomSplit(Array(0.7, 0.3))
val (trainingData, testData) = (splits(0), splits(1))

val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "variance"
val maxDepth = 5
val maxBins = 32

val model = DecisionTree.trainRegressor(trainingData,
  categoricalFeaturesInfo, impurity, maxDepth, maxBins)

val labelsAndPredictions = testData.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
```

Decision Tree - Classification

```
val data: RDD[LabeledPoint] = ...
val splits = data.randomSplit(Array(0.7, 0.3))
val (trainingData, testData) = (splits(0), splits(1))

val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "gini"
val maxDepth = 5
val maxBins = 32

val model = DecisionTree.trainClassifier(trainingData, numClasses,
  categoricalFeaturesInfo, impurity, maxDepth, maxBins)

val labelAndPreds = testData.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
```

Random Forest

- Train a set of decision trees separately.
- The training can be done in parallel.
- The algorithm injects randomness into the training process, so that each decision tree is a bit different.

Random Forest - Regression

```
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val numTrees = 3
val featureSubsetStrategy = "auto"
val impurity = "variance"
val maxDepth = 4
val maxBins = 32

val model = RandomForest.trainRegressor(trainingData, categoricalFeaturesInfo,
  numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins)

val labelsAndPredictions = testData.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
```

Random Forest - Classification

```
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val numTrees = 3
val featureSubsetStrategy = "auto"
val impurity = "gini"
val maxDepth = 4
val maxBins = 32

val model = RandomForest.trainClassifier(trainingData, numClasses,
  categoricalFeaturesInfo, numTrees, featureSubsetStrategy, impurity,
  maxDepth, maxBins)

val labelAndPreds = testData.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
```

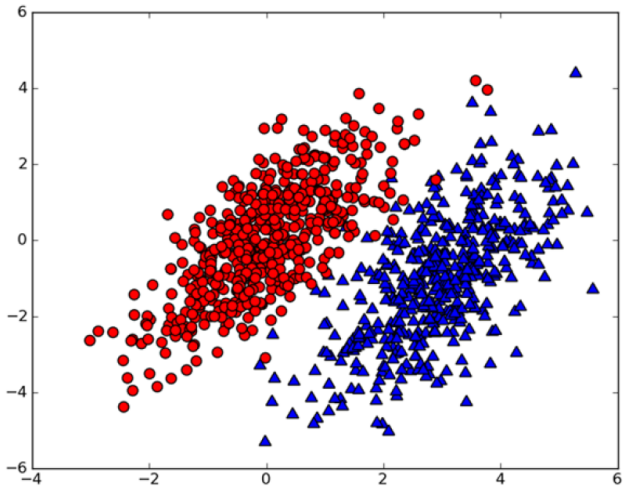
Naive Bayes

Naive Bayes (1/4)

- Using the probability theory to classify things.
- Assumption: independency between every pair of features.

Naive Bayes (1/4)

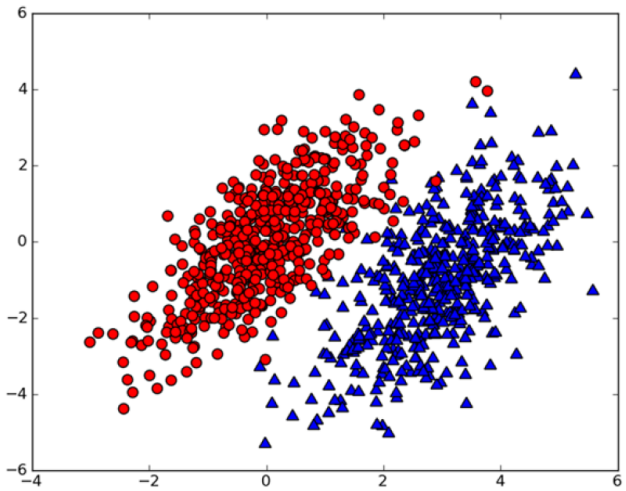
- Using the probability theory to classify things.
- Assumption: independency between every pair of features.



- y_1: circles, and y_2: triangles.
- (x_1, x_2) belongs to y_1 or y_2?

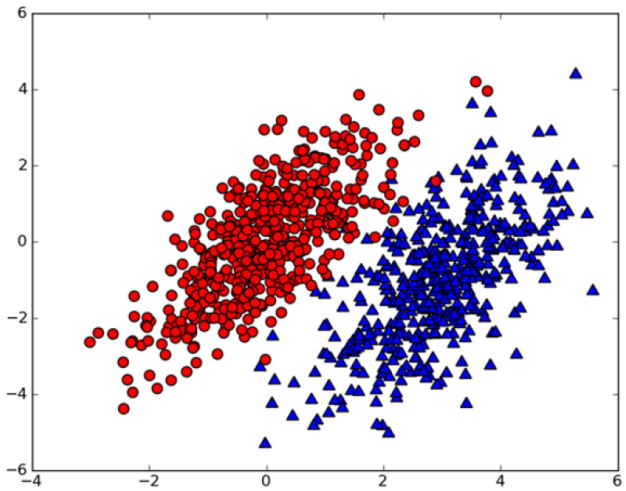
Naive Bayes (2/4)

- (x_1, x_2) belongs to y_1 or y_2?
- If $p(y_1|x_1, x_2) > p(y_2|x_1, x_2)$, the class is y_1.
- If $p(y_1|x_1, x_2) < p(y_2|x_1, x_2)$, the class is y_2.



Naive Bayes (2/4)

- (x_1, x_2) belongs to y_1 or y_2?
- If $p(y_1|x_1, x_2) > p(y_2|x_1, x_2)$, the class is y_1.
- If $p(y_1|x_1, x_2) < p(y_2|x_1, x_2)$, the class is y_2.



- Replace $p(y|x)$ with $\frac{p(x|y)p(y)}{p(x)}$

Naive Bayes (3/4)

- Bayes theorem: $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$
- $p(y|x)$: probability of instance x being in class y.
- $p(x|y)$: probability of generating instance x given class y.
- $p(y)$: probability of occurrence of class y
- $p(x)$: probability of instance x occurring.

Naive Bayes (4/4)

Is officer Drew male or female?

Name	Sex
Drew	Male
Claudia	Female
Drew	Female
Drew	Female
Alberto	Male
Karin	Female
Nina	Female
Sergio	Male

Naive Bayes (4/4)

Name	Sex
Drew	Male
Claudia	Female
Drew	Female
Drew	Female
Alberto	Male
Karin	Female
Nina	Female
Sergio	Male

- $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$
- $p(\texttt{male}|\texttt{drew}) = ?$
- $p(\texttt{female}|\texttt{drew}) = ?$

Naive Bayes (4/4)

Name	Sex
Drew	Male
Claudia	Female
Drew	Female
Drew	Female
Alberto	Male
Karin	Female
Nina	Female
Sergio	Male

- $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$
- $p(\texttt{male}|\texttt{drew}) = \frac{p(\texttt{drew}|\texttt{male})p(\texttt{male})}{p(\texttt{drew})} = \frac{\frac{1}{3} \times \frac{3}{8}}{\frac{3}{8}} = 0.33$
- $p(\texttt{female}|\texttt{drew}) = \frac{p(\texttt{drew}|\texttt{female})p(\texttt{female})}{p(\texttt{drew})} = \frac{\frac{2}{5} \times \frac{5}{8}}{\frac{3}{8}} = 0.66$

Naive Bayes (4/4)

Officer Drew is female.

Name	Sex
Drew	Male
Claudia	Female
Drew	Female
Drew	Female
Alberto	Male
Karin	Female
Nina	Female
Sergio	Male

- $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$
- $p(\texttt{male}|\texttt{drew}) = \frac{p(\texttt{drew}|\texttt{male})p(\texttt{male})}{p(\texttt{drew})} = \frac{\frac{1}{3} \times \frac{3}{8}}{\frac{3}{8}} = 0.33$
- $p(\texttt{female}|\texttt{drew}) = \frac{p(\texttt{drew}|\texttt{female})p(\texttt{female})}{p(\texttt{drew})} = \frac{\frac{2}{5} \times \frac{5}{8}}{\frac{3}{8}} = 0.66$

Naive Bayes

```
val data: RDD[LabeledPoint] = ...
val splits = data.randomSplit(Array(0.7, 0.3))
val (trainingData, testData) = (splits(0), splits(1))

val model = NaiveBayes.train(trainingData, lambda = 1.0,
  modelType = "multinomial")

val predictionAndLabel = test.map(p =>
  (model.predict(p.features), p.label))
```

Clustering (Unsupervised Learning)

Clustering (1/5)

- Clustering is a technique for finding similarity groups in data, called clusters.

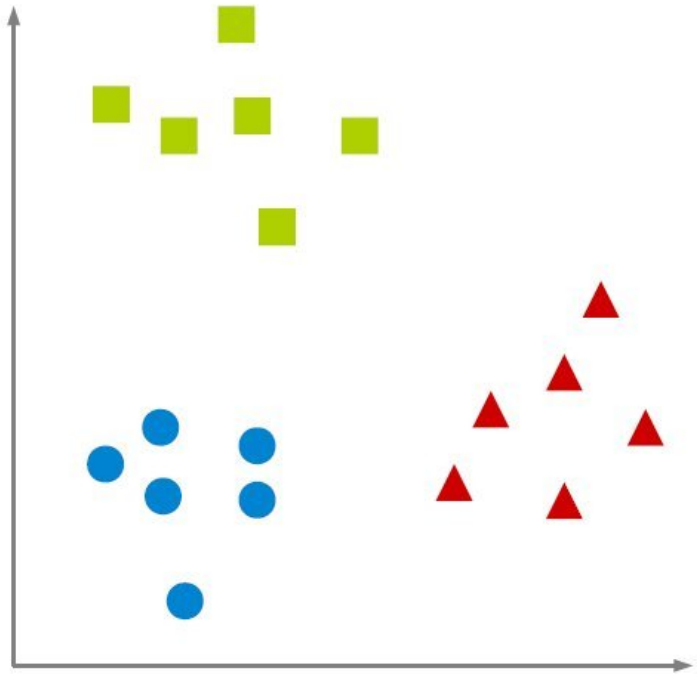
Clustering (1/5)

- Clustering is a technique for finding similarity groups in data, called clusters.
- It groups data instances that are similar to each other in one cluster, and data instances that are very different from each other into different clusters.

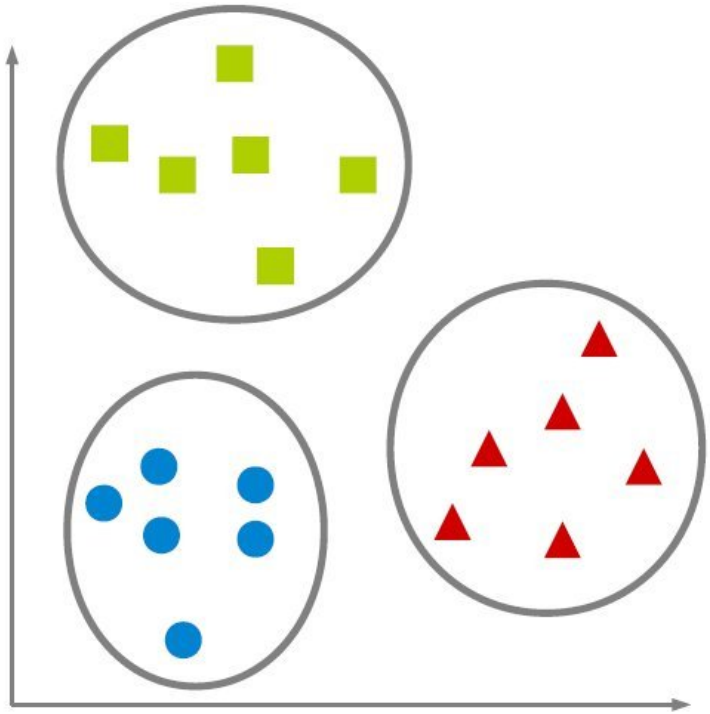
Clustering (1/5)

- Clustering is a technique for finding similarity groups in data, called clusters.
- It groups data instances that are similar to each other in one cluster, and data instances that are very different from each other into different clusters.
- Clustering is often called an unsupervised learning task as no class values denoting an a priori grouping of the data instances are given.

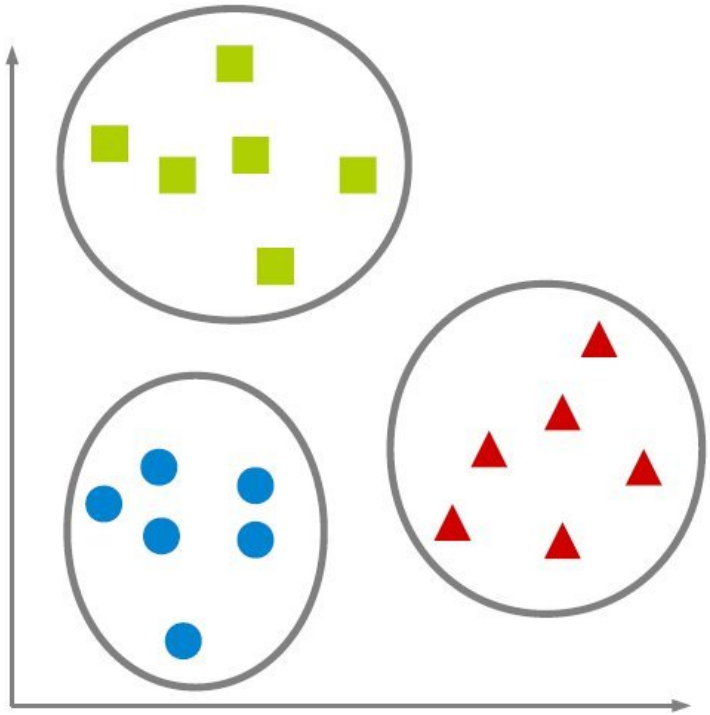
Clustering (2/5)



Clustering (2/5)



Clustering (2/5)

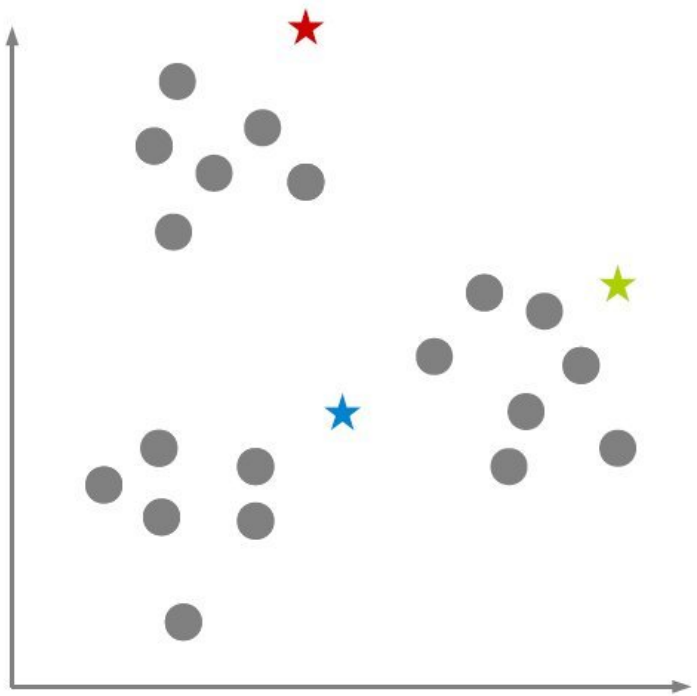


- k-means clustering is a popular method for clustering.

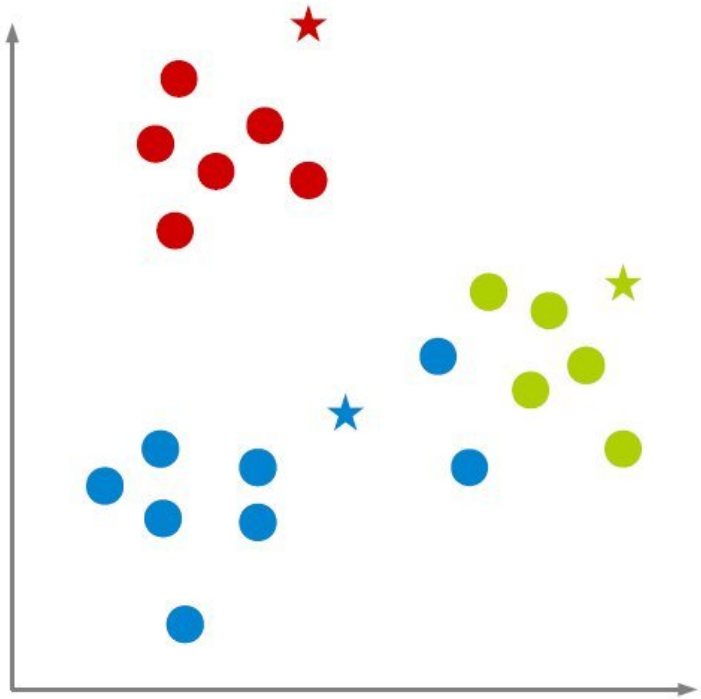
Clustering (3/5)

- K: number of clusters (given)
  - One mean per cluster.
- Initialize means: by picking k samples at random.
- Iterate:
  - Assign each point to nearest mean.
  - Move mean to center of its cluster.

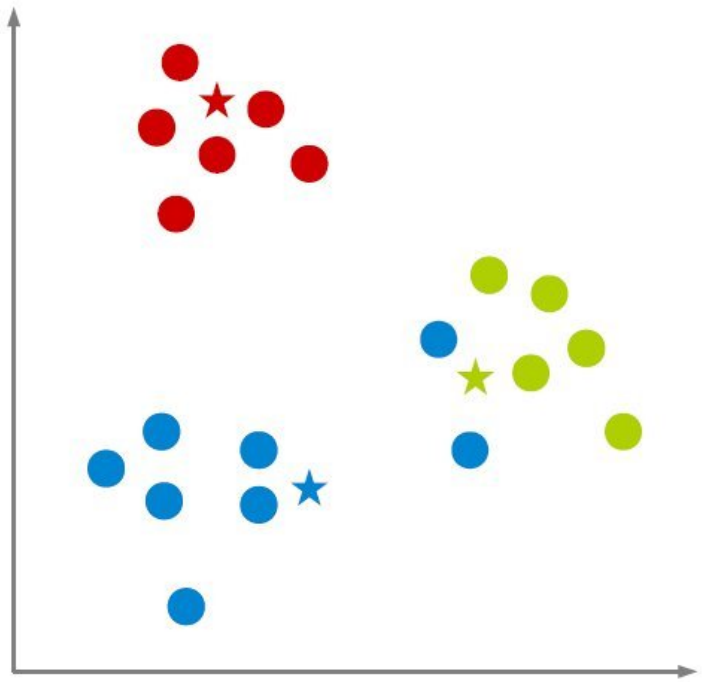
Clustering (4/5)



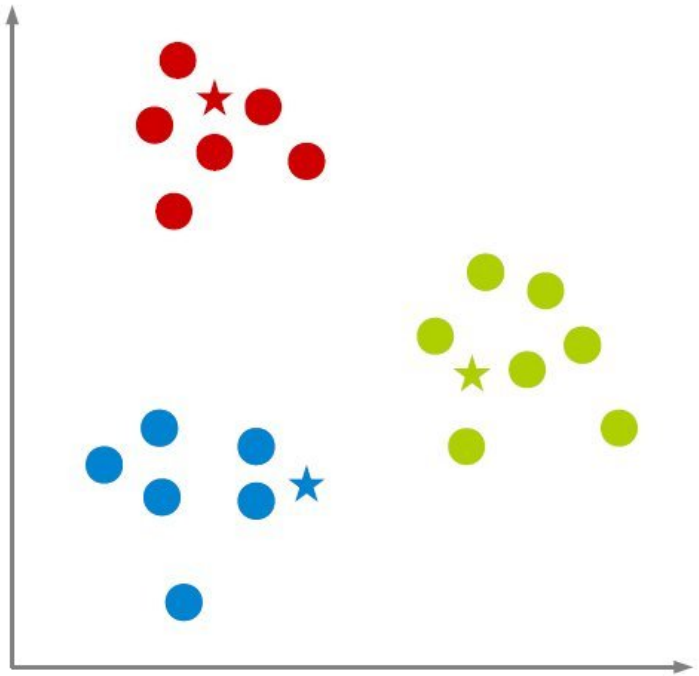
Clustering (4/5)



Clustering (4/5)



Clustering (4/5)



Clustering (5/5)

```
val data: RDD[LabeledPoint] = ...

val numClusters = 2
val numIterations = 20
val clusters = KMeans.train(data, numClusters, numIterations)

// Evaluate clustering by computing Within Set Sum of Squared Errors
val WSSSE = clusters.computeCost(data)
println("Within Set Sum of Squared Errors = " + WSSSE)
```

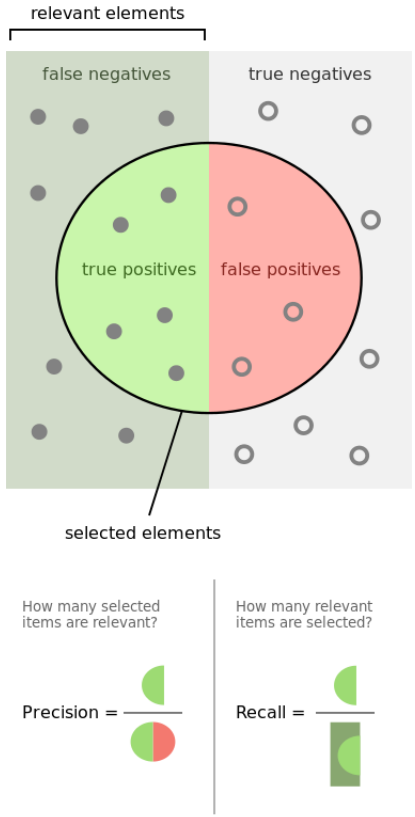
MLlib - Result Validation

Classification Model Evaluation

- There exists a true output and a model-generated predicted output for each data point.
- The results for each data point can be assigned to one of four categories:
  1. True Positive (TP): label is positive and prediction is also positive
  2. True Negative (TN): label is negative and prediction is also negative
  3. False Positive (FP): label is negative but prediction is positive
  4. False Negative (FN): label is positive but prediction is negative

Binary Classification (1/2)

- Precision (positive predictive value): the fraction of retrieved instances that are relevant.
- Recall (sensitivity): the fraction of relevant instances that are retrieved.
- F-measure: $(1+\beta^2)\frac{precision.recall}{\beta^2.precision+recall}$



Binary Classification (2/2)

```
val model = new LogisticRegressionWithLBFGS()
  .setNumClasses(2)
  .run(trainingData)

val predictionAndLabels = testData.map { point =>
  val prediction = model.predict(point.features)
  (prediction, point.label)
}

val metrics = new BinaryClassificationMetrics(predictionAndLabels)

val precision = metrics.precisionByThreshold
precision.foreach { case (t, p) => println(s"Threshold: $t, Precision: $p") }

val recall = metrics.recallByThreshold
recall.foreach { case (t, r) => println(s"Threshold: $t, Recall: $r") }

val beta = 0.5
val fScore = metrics.fMeasureByThreshold(beta)
fScore.foreach { case (t, f) => println(s"Threshold: $t, F-score: $f") }
```

Regression Model Evaluation (1/2)

- Mean Squared Error (MSE): $\frac{\sum_{i=0}^{N-1}(y_i-\hat{y}_i)^2}{N}$
- Root Mean Squared Error (RMSE): $\sqrt{\frac{\sum_{i=0}^{N-1}(y_i-\hat{y}_i)^2}{N}}$
- Mean Absolute Error (MAE): $\sum_{i=0}^{N-1}|y_i-\hat{y}_i|$

Regression Model Evaluation (2/2)

```
val numIterations = 100
val model = LinearRegressionWithSGD.train(trainigData, numIterations)

val valuesAndPreds = testData.map{ point =>
  val prediction = model.predict(point.features)
  (prediction, point.label)
}

val metrics = new RegressionMetrics(valuesAndPreds)

println(s"MSE = ${metrics.meanSquaredError}")
println(s"RMSE = ${metrics.rootMeanSquaredError}")
println(s"MAE = ${metrics.meanAbsoluteError}")
```

Summary

Summary

- Preprocessing: cleaning, integration, reduction, transformation
- Data mining: classification, clustering, frequent patterns, anomaly
- Result validation

Questions?