Machine Learning
MLlib and Tensorflow

Amir H. Payberah
amir@sics.se

KTH Royal Institute of Technology
2016/10/10

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

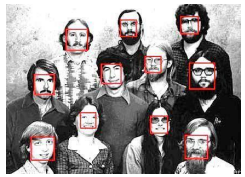
- Is this email spam or no spam?

Data and Knowledge

Data ⟶ Knowledge

- Is this email spam or no spam?

- Is there a face in this picture?

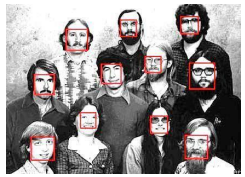


Data and Knowledge

Data ⟶ Knowledge

- Is this email spam or no spam?

- Is there a face in this picture?

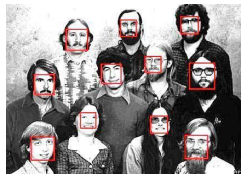


Data and Knowledge

Data ⟶ Knowledge

- Is this email spam or no spam?

- Is there a face in this picture?



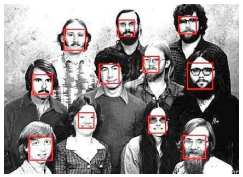
- Should I lend money to this customer given his spending behavior?

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

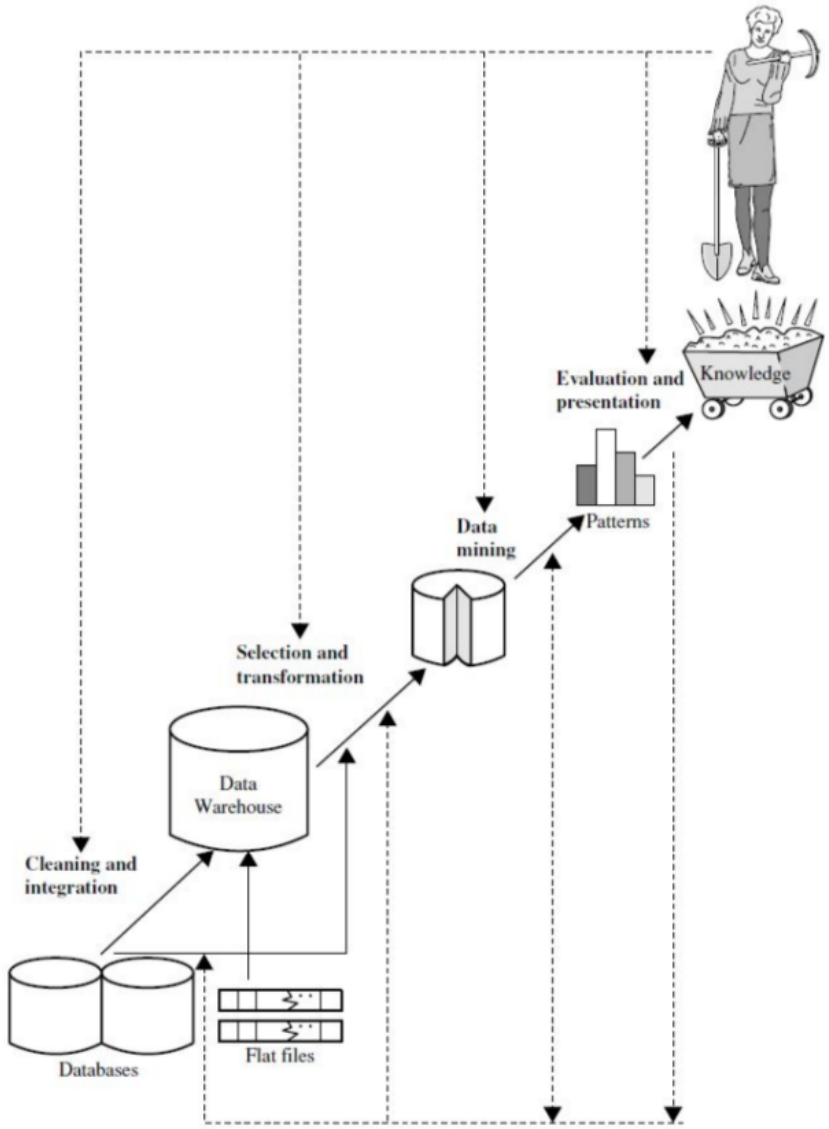
- Knowledge is not concrete

- 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!

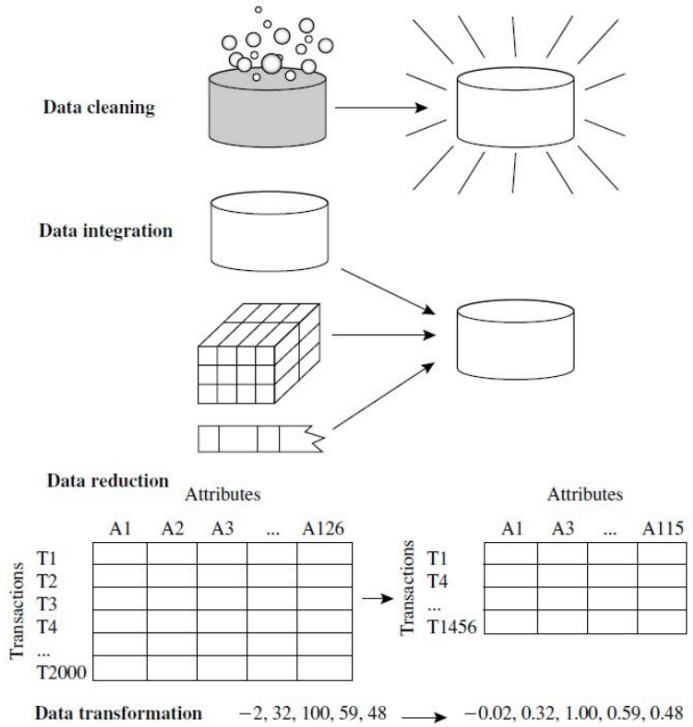
Knowledge Discovery from Data (KDD)

- 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.

Data Mining Functionalities

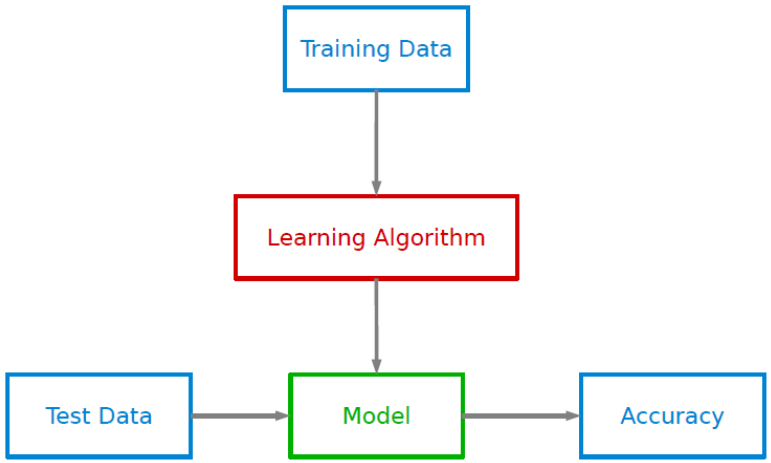
Data Mining Functionalities

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

Classification and Regression (Supervised Learning)

Supervised Learning (1/3)

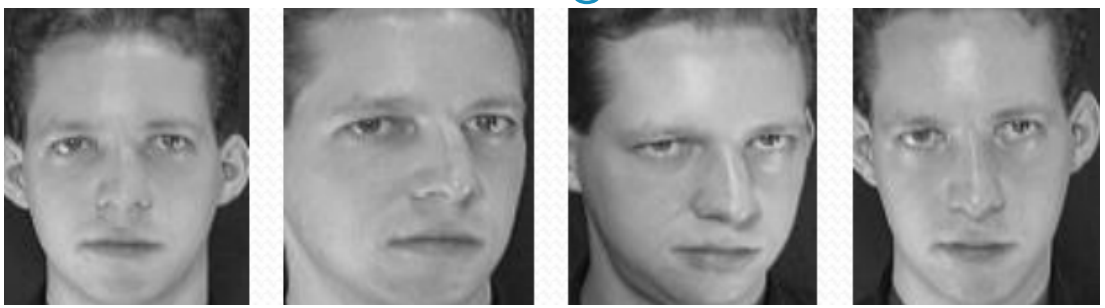
- Right answers are given.
  - Training data (input data) is labeled, e.g., spam/not-spam or a stock price at a time.
- A model is prepared through a training process.
- The training process continues until the model achieves a desired level of accuracy on the training data.



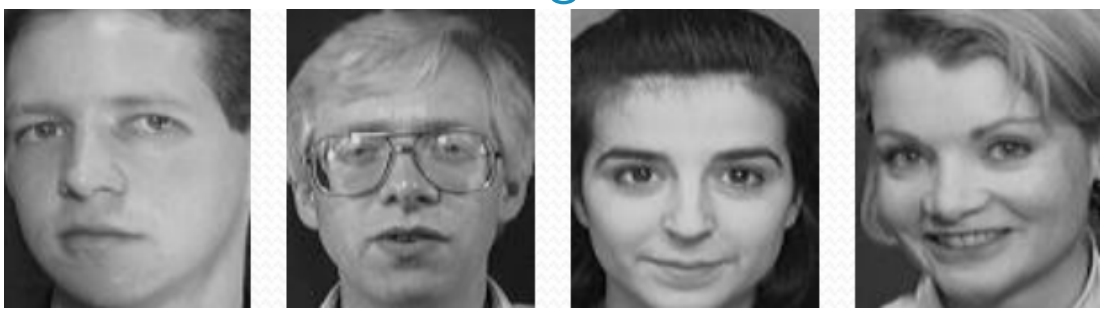
Supervised Learning (2/3)

- Face recognition

Training data



Testing data



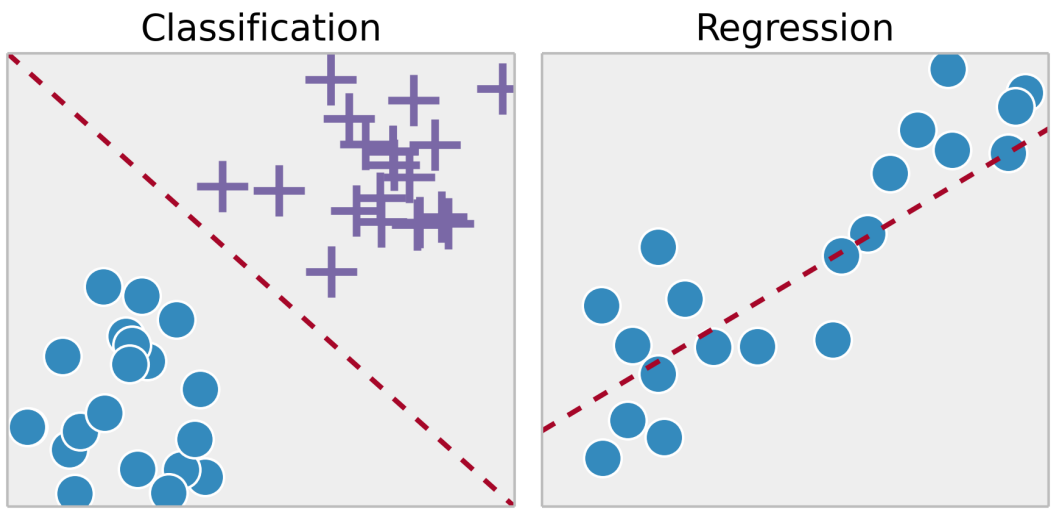
[ORL dataset, AT&T Laboratories, Cambridge UK]

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.



Classification/Regression Models

- Linear models
- Decision trees
- Naive Bayes models

Linear Models (1/2)

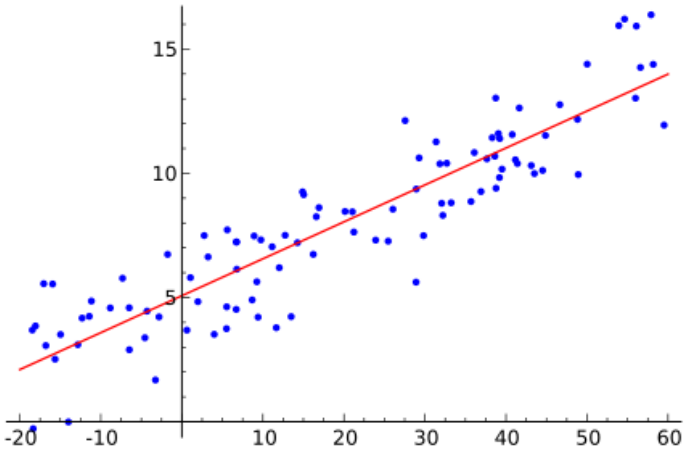
- Training dataset: $(x_1, y_1), \cdots, (x_n, y_n)$.
- $x_i = \langle x_{i1}, x_{i2}, \cdots, x_{im} \rangle$
- Model the target as a function of a linear predictor applied to the input variables: $y_i = g(\mathbf{w}^T \mathbf{x}_i)$.
  - E.g., $y_i = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_m x_{im}$

Linear Models (1/2)

- 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 $\min_{\mathbf{w} \in \mathbb{R}^m} \mathtt{f}(\mathbf{w})$

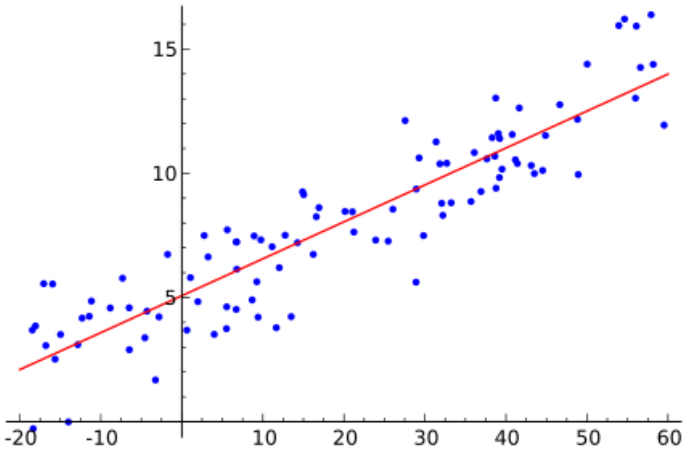
Linear Models (2/2)

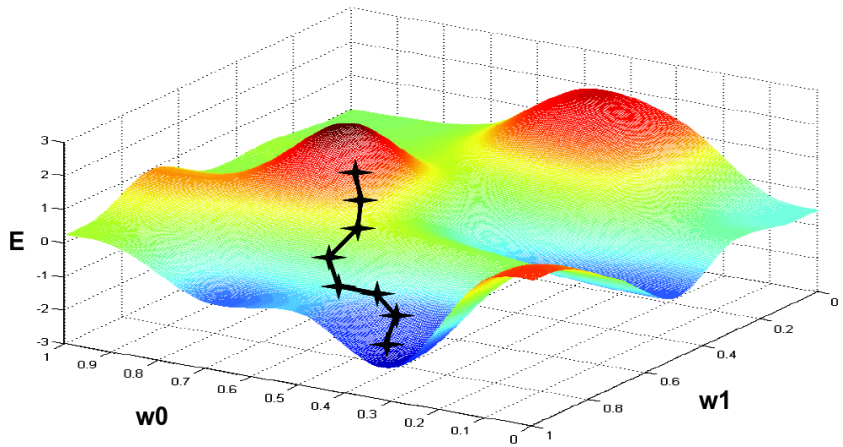
- $g(\mathbf{w}^T\mathbf{x}_i) = w_1x_{i1} + w_2x_{i2} + \cdots + w_mx_{im}$
- Loss function: minimizing squared different between predicted value and actual value: $L(g(\mathbf{w}^T\mathbf{x}_i), y_i) := \frac{1}{2}(\mathbf{w}^T\mathbf{x}_i - y_i)^2$



Linear Models (2/2)

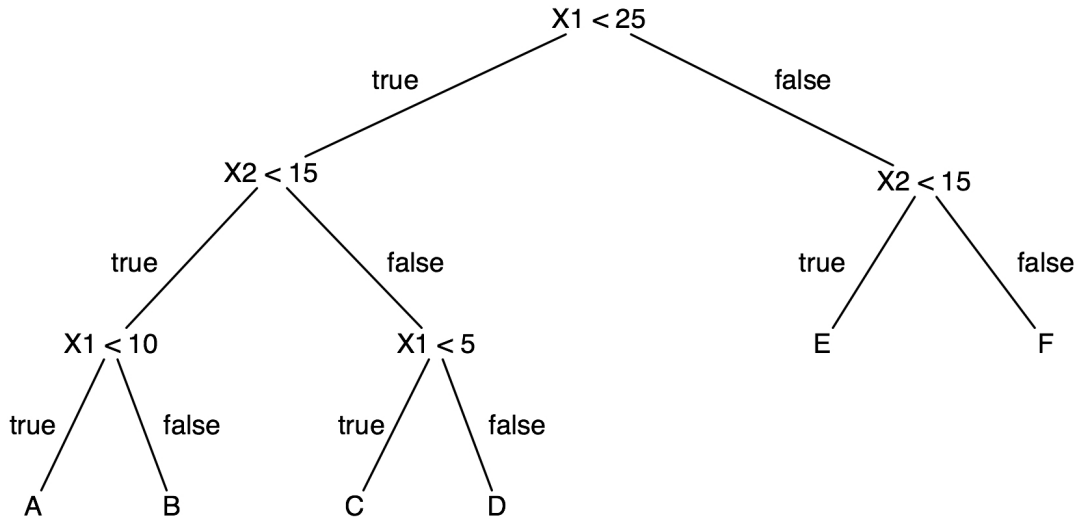
- $g(\mathbf{w}^T\mathbf{x}_i) = w_1x_{i1} + w_2x_{i2} + \cdots + w_mx_{im}$
- Loss function: minimizing squared different between predicted value and actual value: $L(g(\mathbf{w}^T\mathbf{x}_i), y_i) := \frac{1}{2}(\mathbf{w}^T\mathbf{x}_i - y_i)^2$
- Gradient descent





Decision Tree (1/2)

- 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.



Decision Tree (2/2)

- 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.

Naive Bayes (1/3)

- Using the probability theory to classify things.
- Find $\mathtt{p(y|x)}$.

Naive Bayes (1/3)

- Using the probability theory to classify things.
- Find $p(y|x)$.
- Replace $p(y|x)$ with $\frac{p(x|y)p(y)}{p(x)}$

Naive Bayes (2/3)

- 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 (3/3)

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 (3/3)

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 (3/3)

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 (3/3)

Officer Drew is female.

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

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

Clustering (Unsupervised Learning)

Clustering (1/4)

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

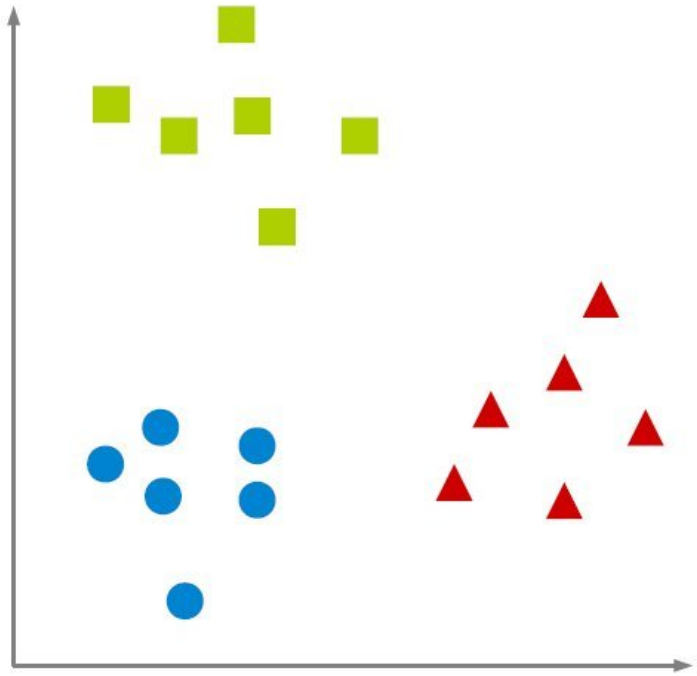
Clustering (1/4)

- 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.

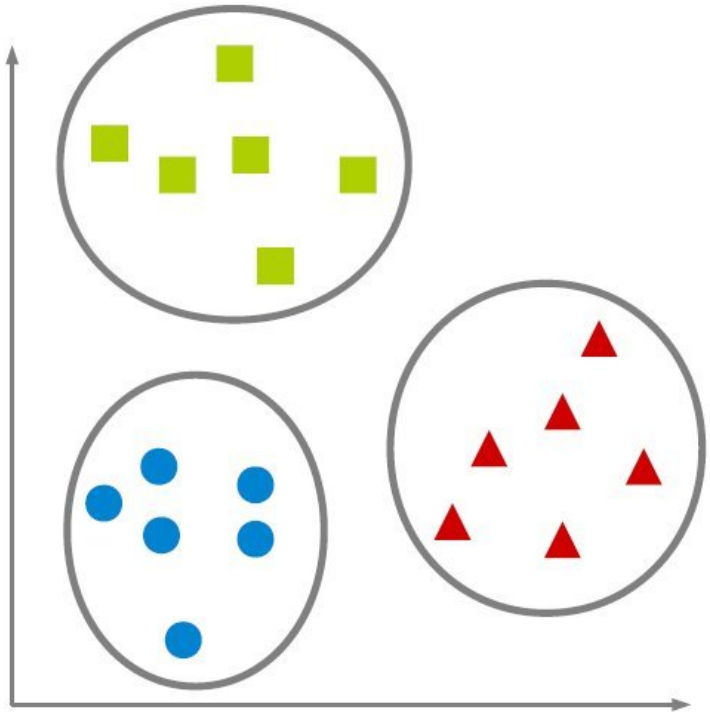
Clustering (1/4)

- 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.

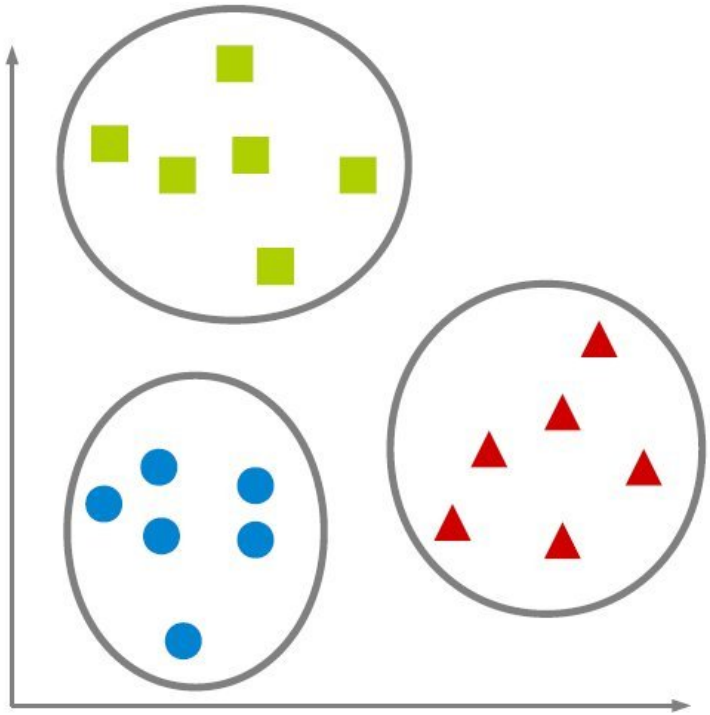
Clustering (2/4)



Clustering (2/4)



Clustering (2/4)

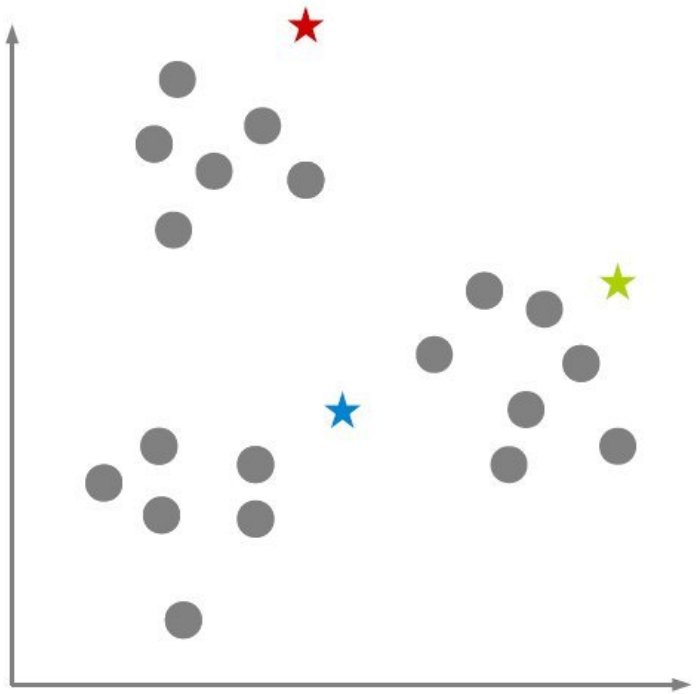


- k-means clustering is a popular method for clustering.

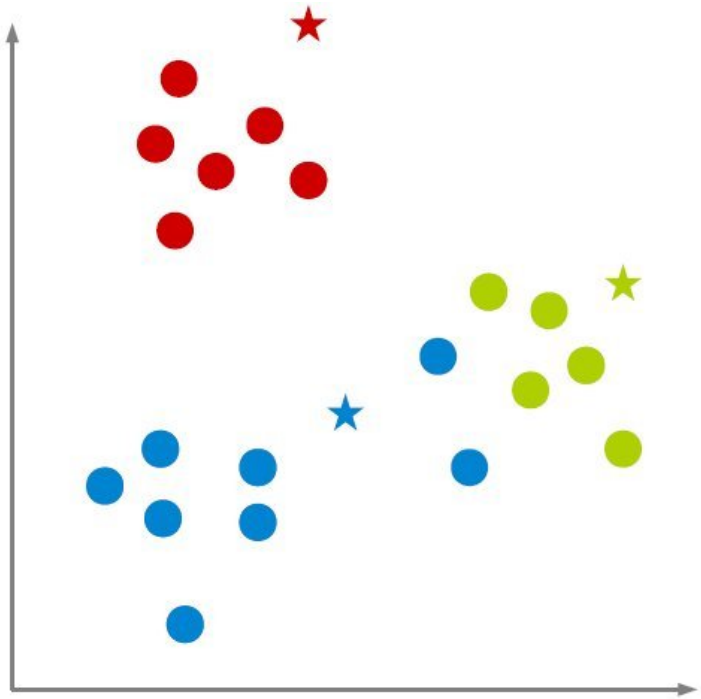
Clustering (3/4)

- 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.

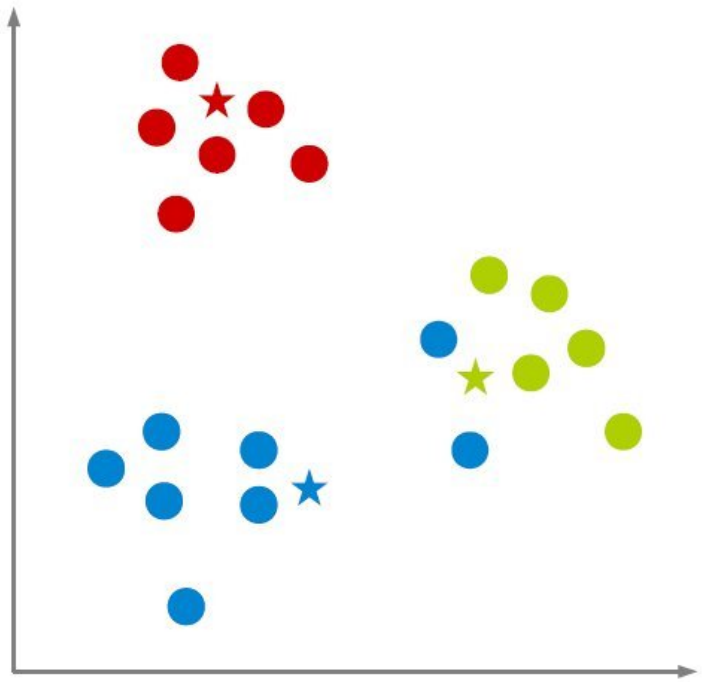
Clustering (4/4)



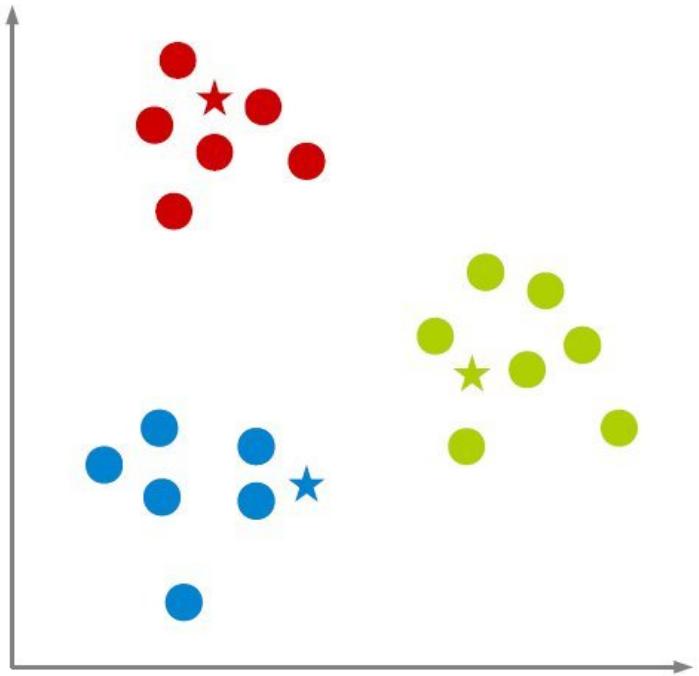
Clustering (4/4)



Clustering (4/4)



Clustering (4/4)



MLlib

MLlib 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)))
```

Supervised Learning in Spark

- Linear models
- Decision trees
- Naive Bayes models

Linear Models - Regression

```
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)

```
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 - 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 - 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

```
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))
```

Unsupervised Learning in Spark

- Kmeans clustering

KMeans Clustering

```
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)
```

Tensorflow

Tensorflow

- TensorFlow
  - An interface for expressing machine learning algorithms
  - An implementation for executing such algorithms
- Tensor: the data structure used.
- Flow: the computational model.

Programming Model - Dataflow

- A computation is described by a directed graph composed of a set of nodes.
- Tensors (values) flow along normal edges in the graph.
- Control dependencies are special edges with no data flows.

Programming Model - Operations and Kernels

- Operation: a node in the dataflow graph that represents an abstract computation.

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural-net building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

Programming Model - Operations and Kernels

- Operation: a node in the dataflow graph that represents an abstract computation.
- Operations can have attributes: they must be provided or inferred at graph-construction time.
  - To make operations polymorphic over different tensor element types, e.g., add of two tensors of type float or int32.

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural-net building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

Programming Model - Operations and Kernels

- Operation: a node in the dataflow graph that represents an abstract computation.
- Operations can have attributes: they must be provided or inferred at graph-construction time.
  - To make operations polymorphic over different tensor element types, e.g., add of two tensors of type float or int32.
- Kernel: a particular implementation of an operation.
  - Can be run on a particular type of device, e.g., CPU or GPU.

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural-net building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

Programming Model - Tensors (1/4)

- Tensor: the data structure used.
- Dense numerical arrays of the same datatype.
- Specified by:
  - Rank
  - Shape
  - Type

Programming Model - Tensors (2/4)

- Rank: the level of nesting

Rank	Math entity	Python example
0	Scalar (magnitude only)	s = 483
1	Vector (magnitude and direction)	v = [1.1, 2.2, 3.3]
2	Matrix (table of numbers)	m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
3	3-Tensor (cube of numbers)	t_3 = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]
n	n-Tensor (you get the idea)	t_n = [[...[1],[2]...],[...[3],[4]...]]

Programming Model - Tensors (3/4)

- Shape: the number of elements for each dimension.

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar: `4711`
1	[D0]	1-D	A 1-D tensor with shape [5]: `[1,2,3,4,5]`

2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4]:
			`[[1, 2, 3, 4],`
			`[5, 6, 7, 8],`
			`[9,10,11,12]]`

3	[D0, D1, D2]	3-D,	A 3-D tensor with shape [2, 4, 3]:
			`[[[ 1, 2, 3],`
			`[ 4, 5, 6],`
			`[ 7, 8, 9],`
			`[10,11,12]],`
			`[[13,14,15],`
			`[16,17,18],`
			`[19,20,21],`
			`[22,23,25]]]`

n	D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

Programming Model - Tensors (4/4)

- Type: the datatype of the numbers stored in the tensor.

Data type	Python type	Description
DT_FLOAT, DT_DOUBLE	tf.float32, tf.float64	32, 64 bits floating point.
DT_INT8, DT_INT16, DT_INT32, DT_INT64	tf.int8, tf.int16, tf.int32, tf.int64	8, 16, 32, 64 bits signed integer.
DT_UINT8	tf.uint8	8 bits unsigned integer.
DT_STRING	tf.string	Variable length byte arrays. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64, DT_COMPLEX128	Tf.complex64, tf.complex128	Complex number made of two 32, 64 bits floating points: real and imaginary parts.
DT_QINT8, DT_QINT32	tf.qint8, tf.qint32	8, 32 bits signed integer used in quantized Ops.
DT_QUINT8	tf.quint8	8 bits unsigned integer used in quantized Ops.

Programming Model - Variables

- Most tensors do not survive past a single execution of the graph.

Programming Model - Variables

- Most tensors do not survive past a single execution of the graph.
- Variable: a special kind of operation that returns a handle to a persistent mutable tensor that survives across executions of a graph.
- Handles to these persistent mutable tensors can be passed to a handful of special operations, e.g., `AssignAdd` (equivalent to $+=$) that mutate the referenced tensor.

Programming Model - Sessions

- Session: clients programs interact with the TensorFlow system by creating a Session.

Programming Model - Sessions

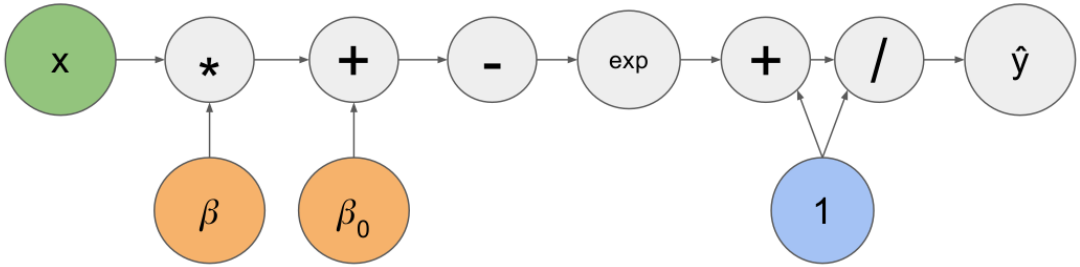
- Session: clients programs interact with the TensorFlow system by creating a Session.
- Extend method: to augment the current graph managed by the session with additional nodes and edges.
- Run method: to compute the transitive closure of all nodes that must be executed to compute the outputs.

Programming Model - Sessions

- Session: clients programs interact with the TensorFlow system by creating a Session.
- Extend method: to augment the current graph managed by the session with additional nodes and edges.
- Run method: to compute the transitive closure of all nodes that must be executed to compute the outputs.
- Usually a Session is set up once, and then execute the full graph or a few distinct subgraphs thousands or millions of times via Run calls.

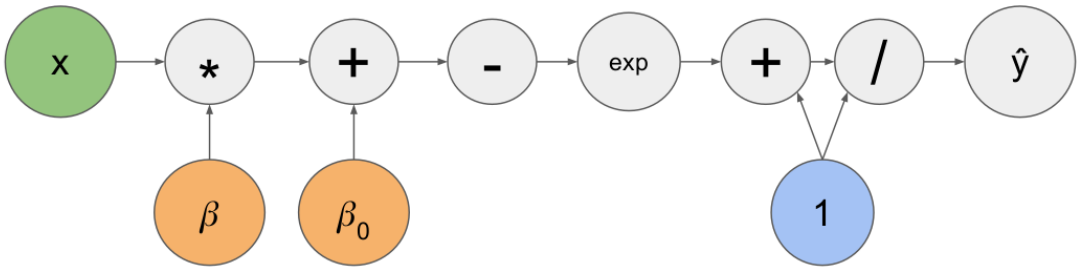
Example - Logistic Regression

- $\hat{y} = \frac{1}{1+e^{-(\beta_0+\beta.x)}}$



Example - Logistic Regression

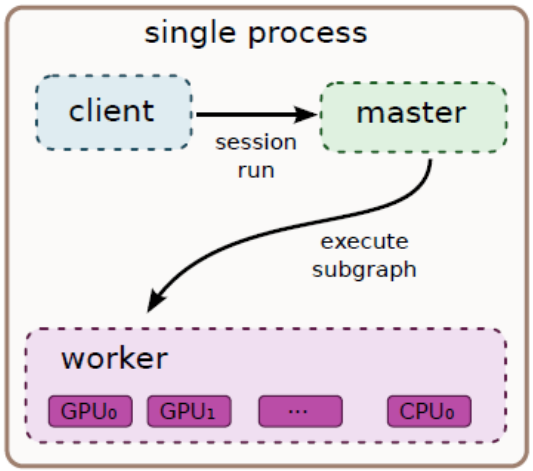
- $\hat{y} = \frac{1}{1+e^{-(\beta_0+\beta.x)}}$

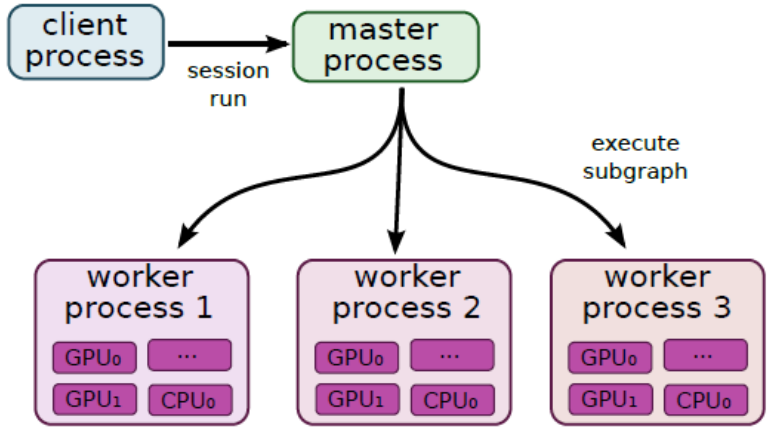



```
import tensorflow as tf

x = tf.placeholder(tf.float32, [1]) # add a placeholder node for the input x

beta0 = tf.Variable(tf.zeros(1)) # the bias
beta = tf.Variable(tf.ones(1)) # the coefficient for x

log_odds = beta0 + x * beta # perform the linear regression step
y_hat = 1 / (1 + tf.exp(-log_adds)) # apply the logistic function
initializer = tf.initialize_all_variables()
with tf.Session() as sess:
  sess.run(initializer)
  pred = sess.run(y_hat, feed_dict={x:[1]}) # evaluate the y_hat node
  print(pred)
```

Implementation

- Master-worker model
- Two implementations:
  - Local (single machine)
  - Distributed (multi machines)





Summary

Summary

- Data mining: classification, clustering, frequent patterns, anomaly
- MLlib and Tensorflow

Questions?