Updated: 2021-03-29

5.4
Machine Learning & MLlib

Northeastern University

What is Machine Learning?

- Machine learning ("ML") is:
  - when we teach (or train) a software system to perform repetitive tasks such as classification while only providing the system the vaguest idea of how to perform the work;
  - then running that system so that we don't have to spend the time doing it ourselves.
- Other names have been used for this, such as ***expert systems***, even ***AI*** generally.

ML Components

- Raw data
- Ingester (also known as *ETL* process: extract-transform-load) which can take the form of:
  - a batch ingester; or
  - a streaming ingester (mini-batches separated in the time domain with ***exactly-once*** guarantee).
- Trainer:
  - training and validation datasets in the form of rows of feature vectors with, optionally, labels;
  - configured algorithm, e.g. linear regression, perceptron.
- Model: the end result of training (and validating).
- Runner:
  - query processor which transforms a query into a feature vector;
  - model runner (applies the feature vector to the model).

ML types

- Supervised learning:
  - Features are selected by the supervisor;
  - Training/validation sets are *labeled* and supervisor trains the system to do its own labeling
  - Example: classification, regression.
- Unsupervised learning:
  - The training/validation sets are *unlabeled;*
  - We seek *patterns* or *insights* from the data.
  - Example: clustering.
- Deep learning:
  - Features are selected by the system;
  - Example: multi-layer neural nets, “Watson”.

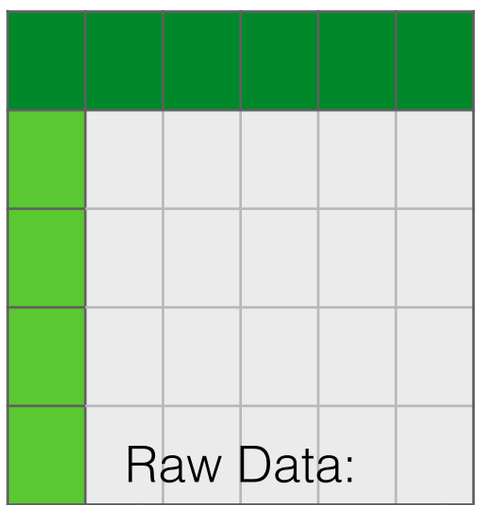
What is ML really doing?

- It is trying to “fit” a model to the training data that you provide;
- By “fit”, we mean that a set of inferences made by the model should have *minimal variance* from the true facts;
- Variance is the square of the Euclidean distance from the point inferred (predicted) from the actual point, i.e. $\sum (y_i - x_i)^2$ where ***i*** is over all the “dimensions”/features.
- All ML algorithms are essentially minimizing this total variance in one way or another.

Some notes on entropy

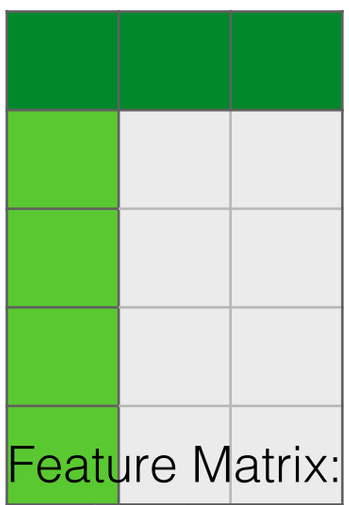
- Entropy in information theory is, like its thermodynamics counterpart, a measure of disorder (unknown variables);
- A ML system starts out with large entropy and that entropy is reduced through training
  - Each feature (a column in the training set) yields some information based on the range of values it can assume:
    - But this can be diminished by “mutual information” with another feature;
    - the information provided by a feature is also proportional to the ***number of rows of quality data*** provided;
    - But each feature adds an extra degree of freedom to the model (or you can think of it as another dimension)—this ***increases*** the total entropy of the model
- Bottom line: additional features that provide little or no information content actually have a positive (i.e. bad) effect on the entropy of the system
- Very very approximate rule of thumb: you need at least 1000 rows for every feature that you plan to model.

Strategy for (Supervised) ML



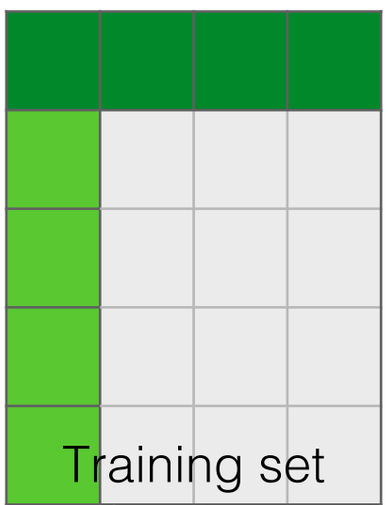
Raw Data:
rows (one per case);
columns (observations)

Feature Set



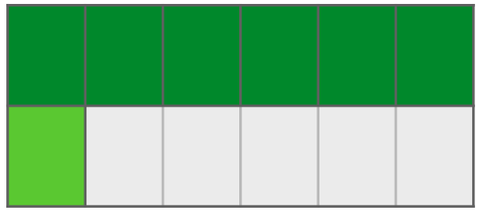
Feature Matrix:
rows (unchanged);
columns (features)

Label



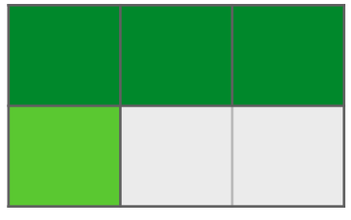
Training set
rows (unchanged);
cols: (labeled features)

Select



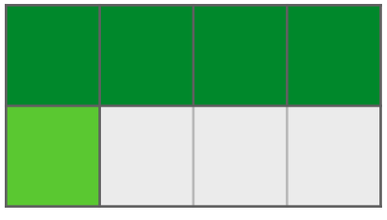
Validation Set
subset of rows

Feature Set



Feature Matrix:
rows (one per case);
columns (features)

Run



Validated model

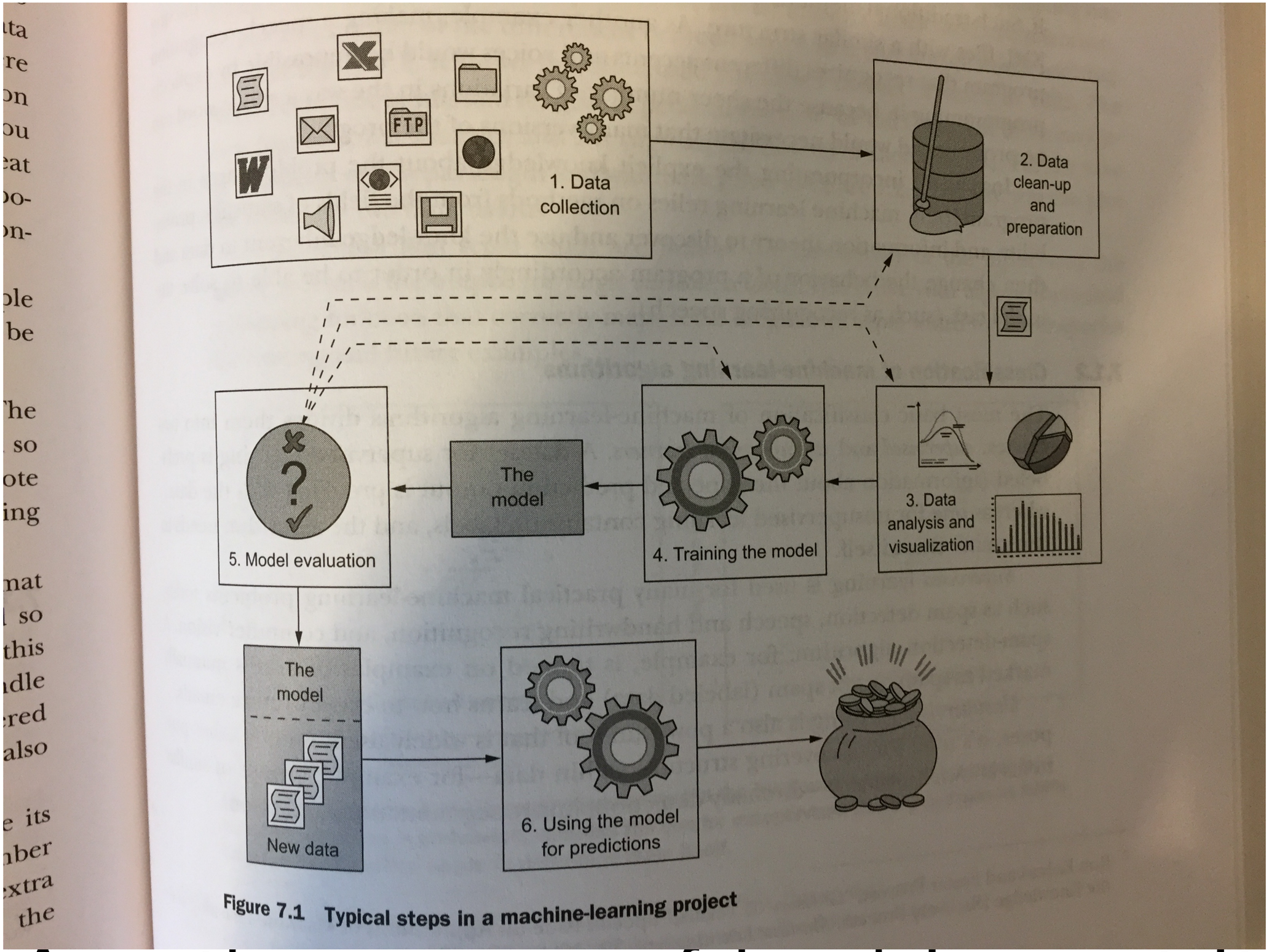


Figure 7.1 Typical steps in a machine-learning project

Another way of looking at it

Figure 7.1 from ***Spark in Action***

Data and Features

- Scaling:

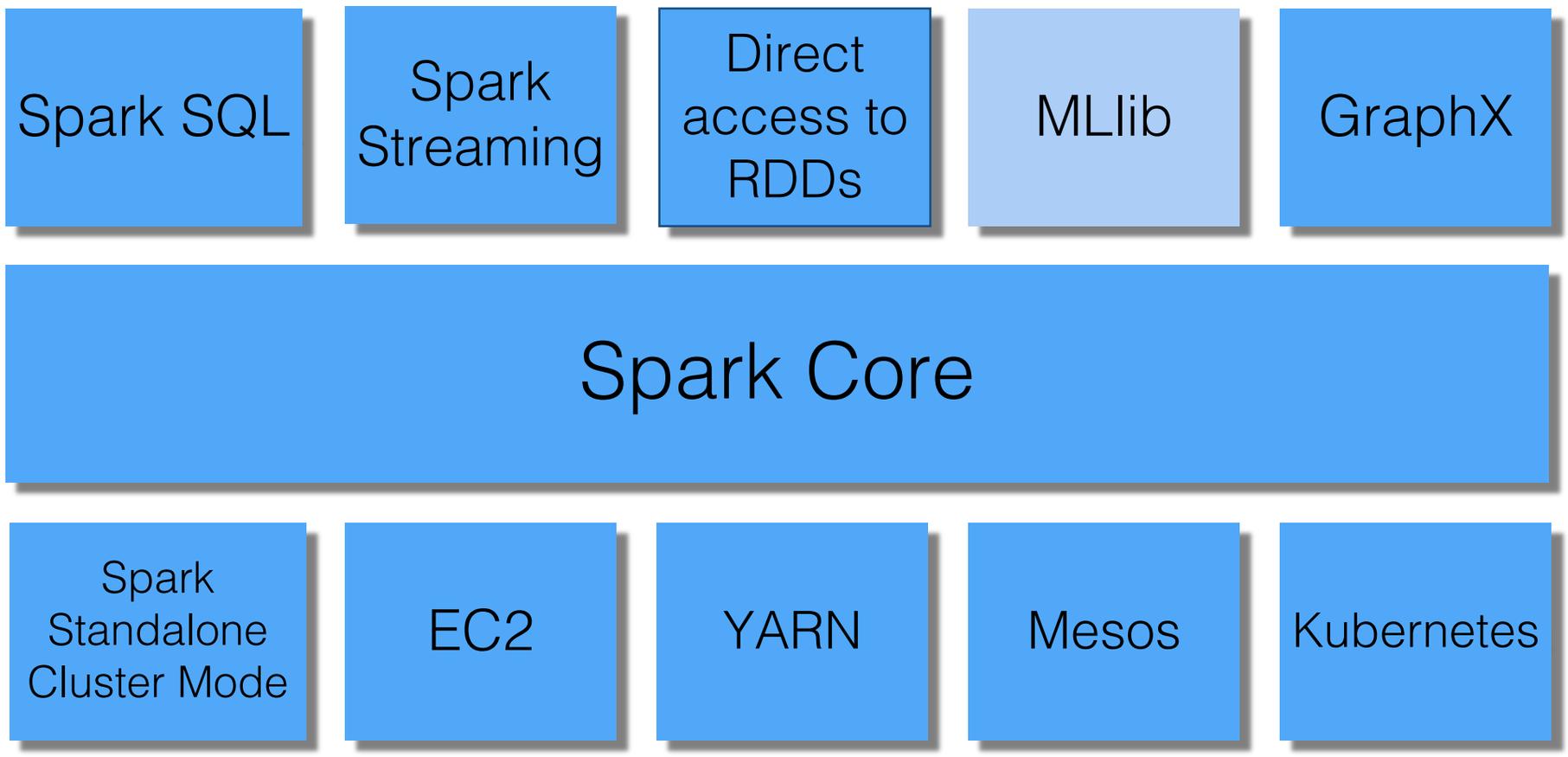
  Standardizes features by removing the mean and scaling to unit variance using column summary statistics on the samples in the training set.

- Statistics:
  - *Statistics.colStats(rdd)* gives column statistics for an RDD of vectors
  - *Statistics.corr(rdd, method)* computes correlation matrix
  - *Statistics.chiSqTest(rdd)* computes independence test for each feature
- Principal Component Analysis (PCA)
  - reduces dimensionality of features to a few uncorrelated dimensions
  - For example, you might reduce to two dimensions and then perform clustering to see if your data appears to divide naturally.

Feature Selection

- Feature selection is the essence of modeling
  - Suppose you have 100 observations but that your model (e.g. Neural Net, SVM, whatever) operates best with 10 inputs/dimensions.
  - You will choose 10 features from your 100 observations (x_j):
  - $f_i = f(x_0, x_1, \ldots x_n)$
  - $\mathbf{I}(f_i;f_j) \simeq 0$ for all i,j where $\mathbf{I}(X;Y)$ is the ***mutual information*** of X & Y
    - i.e. f_i and f_j should be, as far as possible, ***independent***.
  - Furthermore, x_j should be clean and unbiased: there should be no x_j values which are out of the x_j domain and:
    - x_j values should be uniformly distributed over its domain.
- Principle Component Analysis (PCA) will do a decent job of feature selection for you.

Spark Platform



Spark 3.1

- Since Spark 2.0, the primary API is via *DataFrames* ("Spark ML"), although the RDD-based API is extant.
- Features:
  - Learning algorithms for classification, regression, clustering and collaborative filtering,
  - Featurization: PCA, etc,
  - Pipelines
  - Persistence: models, etc.
  - Utilities: built-in image reading, basic statistics.

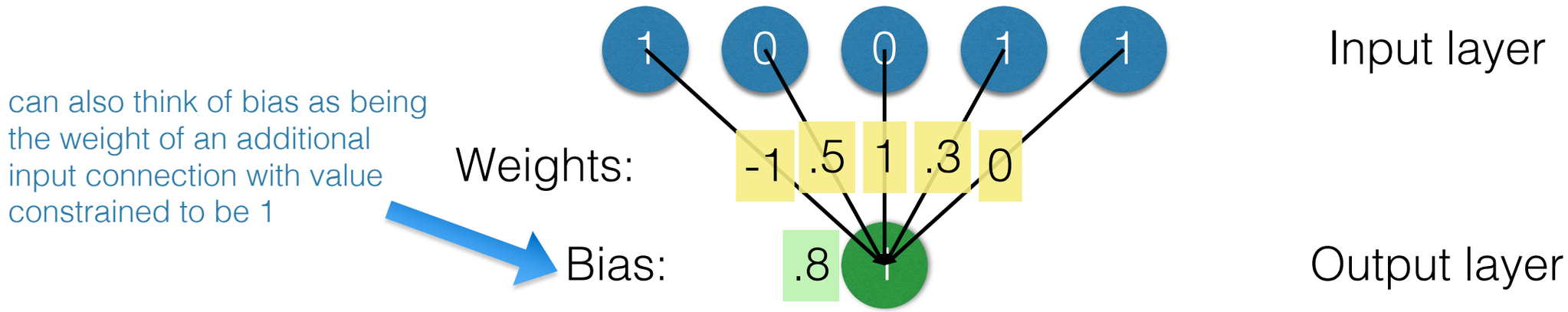
Introduction to (artificial) neural networks

Perceptron Classifier

- What is an Artificial Neural Network (ANN or “Neural Net”)?
  - It’s a general purpose network of quasi-analog “neurons,” which pass and combine “messages” to other neurons. The parameters of the connections and neurons are mutable—that’s to say they can learn.
- What is a Perceptron?
  - A ***perceptron*** is a very simple form of ANN
    - with neurons arranged in layers such that its “feed-forward” connections are formed only between layers (not between neurons of the same layer)
    - each neuron outputs a binary signal (the “message”)
    - one of the first types of neural net to be built (1957)
    - and able to ***classify*** input sets with one or more (orthogonal) labels
    - for each neuron, output is calculated as follows:
      - $f(x) = 1$ if $w \cdot x + b > 0$; otherwise 0 ← dot product
      - where $\boldsymbol{w}$ is the weight vector of connections from the previous layer, and
      - where $\boldsymbol{x}$ is the (binary) set of outputs at the previous layer,
      - and b is the bias

Training the perceptron

- Two-layer perceptron:

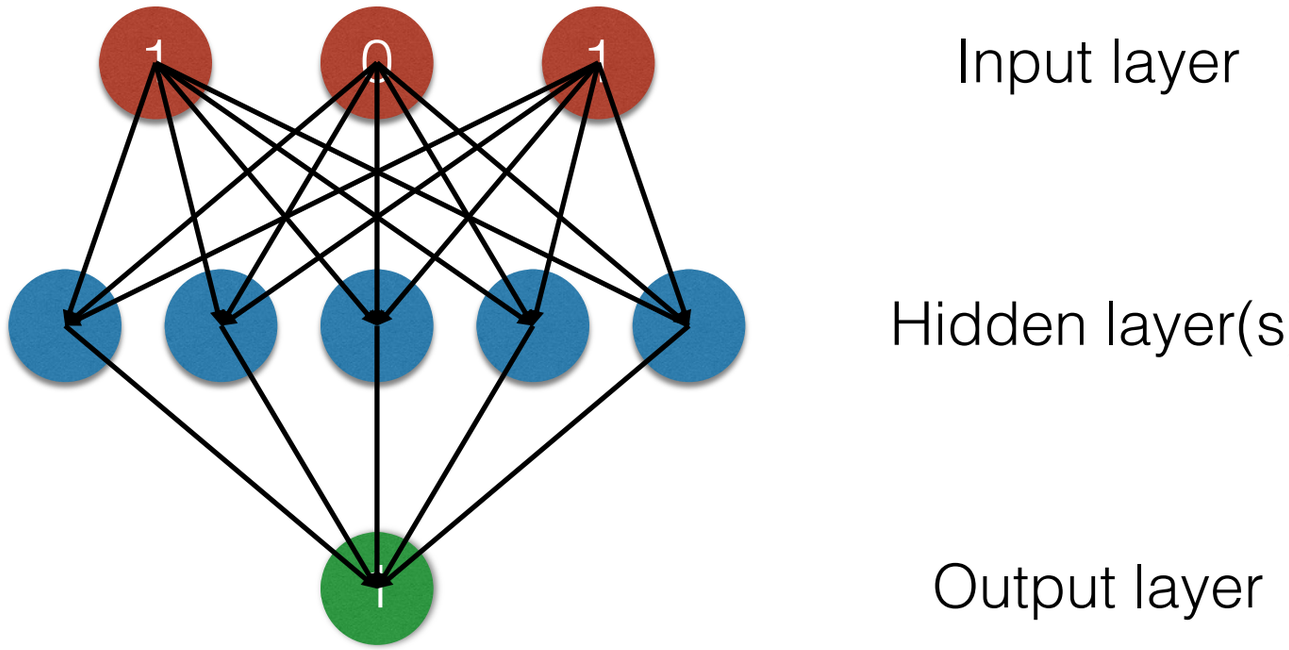


- The weights start out at random (and bias values usually zero):
  - As each new input set is labeled (supervised learning), the weights/bias are adjusted (by back-propagation of errors) until output values match as closely as possible.
  - Provided the input is *linearly separable**, the perceptron will converge (with some set of weights). The optimally stable solution is known as a *support vector machine*.

* if a hyperplane can be drawn separating the inputs

Solving non-linearly-separable problems

- Use a multi-layer perceptron:



- Now you can solve problems such as XOR

Back-propagation, etc.

- Remember our Newton-Raphson convergence from the first week or two?
  - Essentially, perceptrons use an N-dimensional version of the same thing where N is the number of weights (including bias) that must be adjusted and where the function is linear.
- In practice, perceptrons aren't strictly binary at each neuron
  - Hidden neurons use "sigmoid" function;
  - Output neurons use "softmax" function.

Perceptron code

```scala
package edu.neu.coe.scala.spark.nn
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.ml.classification.MultilayerPerceptronClassifier
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.sql.Row
object PerceptronClassifier extends App {
    val conf = new SparkConf().setAppName("perceptron")
    val sc = new SparkContext(conf)
    val sqlContext = new org.apache.spark.sql.SQLContext(sc)
    val sparkHome = "/Applications/spark-1.5.1-bin-hadoop2.6/"
    val trainingFile = "data/mllib/sample_multiclass_classification_data.txt"
    // this is used to implicitly convert an RDD to a DataFrame.
    import sqlContext.implicits._
    // Load training data
    val data = MLUtils.loadLibSVMFile(sc, s"$sparkHome$trainingFile").toDF()
    // Split the data into train and test
    val splits = data.randomSplit(Array(0.6, 0.4), seed = 1234L)
    val train = splits(0)
    val test = splits(1)
    // specify layers for the neural network:
    // input layer of size 4 (features), two intermediate of size 5 and 4 and output of size 3 (classes)
    val layers = Array[Int](4, 5, 4, 3)
    // create the trainer and set its parameters
    val trainer = new MultilayerPerceptronClassifier()
           .setLayers(layers)
           .setBlockSize(128)
           .setSeed(1234L)
           .setMaxIter(100)
    // train the model
    val model = trainer.fit(train)
    // compute precision on the test set
    val result = model.transform(test)
    val predictionAndLabels = result.select("prediction", "label")
    val evaluator = new MulticlassClassificationEvaluator()
           .setMetricName("precision")
    println("Precision:" + evaluator.evaluate(predictionAndLabels))
}
```

input file: label followed by four input neuron values

```
1 1:-0.222222 2:0.5 3:-0.762712 4:-0.833333
1 1:-0.555556 2:0.25 3:-0.864407 4:-0.916667
1 1:-0.722222 2:-0.166667 3:-0.864407 4:-0.833333
1 1:-0.722222 2:0.166667 3:-0.694915 4:-0.916667
0 1:0.166667 2:-0.416667 3:0.457627 4:0.5
1 1:-0.833333 3:-0.864407 4:-0.916667
2 1:-1.32455e-07 2:-0.166667 3:0.220339 4:0.0833333
2 1:-1.32455e-07 2:-0.333333 3:0.0169491 4:-4.03573e-08
1 1:-0.5 2:0.75 3:-0.830508 4:-1
0 1:0.611111 3:0.694915 4:0.416667
```

result:

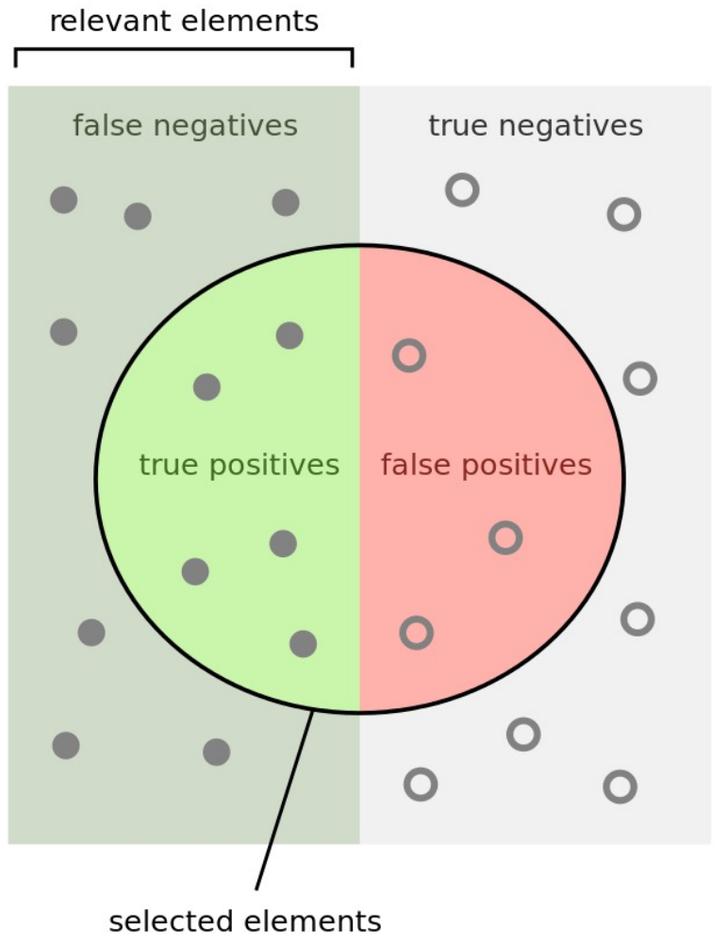
```
+----------+-----+
|prediction|label|
+----------+-----+
|       1.0|  1.0|
|       1.0|  1.0|
|       1.0|  1.0|
|       1.0|  1.0|
|       1.0|  1.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  0.0|
|       2.0|  2.0|
|       0.0|  2.0|
|       1.0|  1.0|
|       0.0|  0.0|
|       1.0|  1.0|
|       1.0|  1.0|
|       0.0|  0.0|
|       2.0|  2.0|
|       1.0|  1.0|
|       0.0|  0.0|
|       0.0|  0.0|
+----------+-----+
```

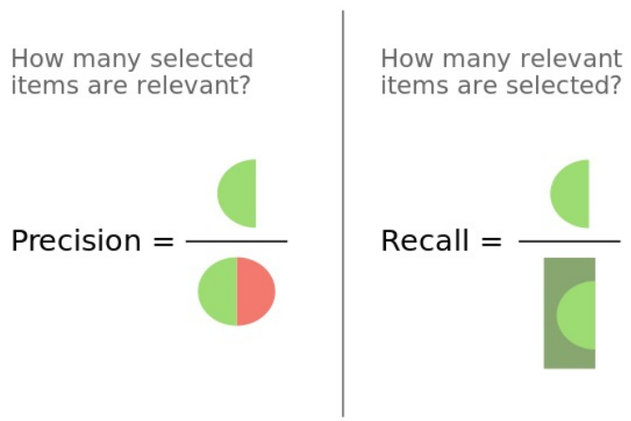
Why Neural Nets?

- I believe that ANNs are primarily useful because they:
  - are very general;
  - do not require much advance modeling thought;
  - are resistant to mutual information (co-variance);
  - can be used with very large feature sets and training sets—
    - in fact, multi-layer ANNs essentially do the feature selection for you;
    - can be used to gain insights for better modeling.
- What to avoid:
  - without sophisticated optimization (gradient descent, etc.) algorithms, convergence may be slow—or worse
  - MLlib uses L-BFGS (limited memory version of Broyden–Fletcher–Goldfarb–Shanno algorithm)

Classification

- Measurements
  - Precision/Recall (applicable for two classes)
    - e.g. for a recall of 50%, we want a precision of 60%
- Confusion matrix (for > two classes)





Spark MLlib tools for modeling

- In addition to *RDD*, MLlib uses the following types:

1. Vector: dense and sparse

```
import org.apache.spark.mllib.linalg.{Vector, Vectors}
Vectors.dense(1.0,2.0,3.0…)
Vectors.dense(array)
Vectors.sparse(4, Array(0,2), Array(1.0,2.0))
```

2. Matrix:

```
import org.apache.spark.mllib.linalg.{Matrix, Matrices}

// Create a dense matrix ((1.0, 2.0), (3.0, 4.0), (5.0, 6.0))
val dm: Matrix = Matrices.dense(3, 2, Array(1.0, 3.0, 5.0, 2.0, 4.0, 6.0))

// Create a sparse matrix ((9.0, 0.0), (0.0, 8.0), (0.0, 6.0))
val sm: Matrix = Matrices.sparse(3, 2, Array(0, 1, 3), Array(0, 2, 1), Array(9, 6, 8))
```

3. LabeledPoint

```
new LabeledPoint(label: Double, features: Vector)
```

4. Rating

5. Model

```
e.g. MultilayerPerceptronClassificationModel
```

Models, Algorithms
(but see latest Spark doc)

- Classification
  - Logistic regression
    - Binomial logistic regression
    - Multinomial logistic regression
  - Decision tree classifier
  - Random forest classifier
  - Gradient-boosted tree classifier
  - Multilayer perceptron classifier
  - Linear Support Vector Machine
  - One-vs-Rest classifier (a.k.a. One-vs-All)
  - Naive Bayes
  - Factorization machines classifier
- Regression
  - Linear regression
  - Generalized linear regression
    - Available families
  - Decision tree regression
  - Random forest regression
  - Gradient-boosted tree regression
  - Survival regression
  - Isotonic regression
  - Factorization machines regressor
- Linear methods
- Factorization Machines
- Decision trees
  - Inputs and Outputs
    - Input Columns
    - Output Columns
- Tree Ensembles
  - Random Forests
    - Inputs and Outputs
      - Input Columns
      - Output Columns (Predictions)
  - Gradient-Boosted Trees (GBTs)
    - Inputs and Outputs
      - Input Columns
      - Output Columns (Predictions)

Working with Text

- Stemming: use NLTK
  - alternatively could simply make all lower-case and remove punctuation
- Feature extraction using TF-IDF
  - use *HashingTF* model
  - Compute the IDF
  - Then the TF-IDF vectors
- Now do classification/regression to, for example, distinguish between spam and ham.
  - Use LogisticRegressionModel, for example

TF-IDF

- Term frequency—inverse document frequency
  - See https://en.wikipedia.org/wiki/Tf–idf
  - Karen Spärck Jones developed the idea in 1972
  - Essentially tf-idf measures the ***contribution to relevant information*** summed over all terms in a document. Common words (as measured by their frequency in some ***corpus*** of documents) are discounted while rare words are emphasized.
- Often used with k-means clustering to cluster text documents
- See spam filter in our repository (taken from ***Learning Spark***—Karau et al).

Zeppelin Notebook Interpreter Search in your notebooks Connected

Spam Analyzer

default

FINISHED

```
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.feature.HashingTF
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD

val spam = sc.textFile("/Users/scalaprof/bigdatascalaclass/SparkApp/spam.txt")
val norm = sc.textFile("/Users/scalaprof/bigdatascalaclass/SparkApp/normal.txt")
val tf = new HashingTF(10000)
val spamFeatures = spam.map(email => tf.transform(email.split(" ")))
val normFeatures = norm.map(email => tf.transform(email.split(" ")))
val posExamples = spamFeatures.map(f => LabeledPoint(1, f))
val negExamples = normFeatures.map(f => LabeledPoint(0, f))
val trainingData = posExamples.union(negExamples)
trainingData.cache()
val model = new LogisticRegressionWithSGD().run(trainingData)
val posTest = tf.transform("Subject: Cheap Stuff From: <omg.fu> O M G GET cheap stuff by sending money to Robin Hillyard".split(" "))
val negTest = tf.transform("Subject: Spark From: Robin Hillyard<scalaprof@gmail.com> Hi Adam, I started studying Spark the other day".split(" "))
println(s"Prediction for positive test example: ${model.predict(posTest)}")
println(s"Prediction for negative test example: ${model.predict(negTest)}")
```

```
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.feature.HashingTF
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD
spam: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1] at textFile at <console>:27
norm: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at textFile at <console>:26
tf: org.apache.spark.mllib.feature.HashingTF = org.apache.spark.mllib.feature.HashingTF@1701e4c1
spamFeatures: org.apache.spark.rdd.RDD[org.apache.spark.mllib.linalg.Vector] = MapPartitionsRDD[4] at map at <console>:30
normFeatures: org.apache.spark.rdd.RDD[org.apache.spark.mllib.linalg.Vector] = MapPartitionsRDD[5] at map at <console>:30
posExamples: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[6] at map at <console>:33
negExamples: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[7] at map at <console>:32
trainingData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = UnionRDD[8] at union at <console>:40
res3: trainingData.type = UnionRDD[8] at union at <console>:40
model: org.apache.spark.mllib.classification.LogisticRegressionModel = org.apache.spark.mllib.classification.LogisticRegressionModel: intercept = 0.0, numFeatures = 10000, numClasses = 2, thr
eshold = 0.5
posTest: org.apache.spark.mllib.linalg.Vector = (10000,[71,77,79,454,702,1371,2752,3066,3159,3290,3700,3707,4351,5726,6372,7023,9552],[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])
negTest: org.apache.spark.mllib.linalg.Vector = (10000,[73,575,2337,2752,3066,4605,4801,4849,5693,5726,9228,9401,9776],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,1.0,1.0,1.0])
Prediction for positive test example: 1.0
```

Spam filter