Errata

Chapter 1 Getting started /
Mathematical notation for the curious– Page 10

Variance

$$Var(X) = E[(\boldsymbol{x} - E[x])^2] = \frac{1}{n}\sum_{i=1}^{n}(x_i - E[X])^2$$

Unbiased sample variance

$$\widehat{Var}(X) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - E[X])^2$$

Chapter 1 Getting started /
Abstraction – Page 12

"Monoids are associative operations. For instance, if ts1, ts2, and ts3 are three time series, then the property ts1 + (ts2 + ts3) = (ts1 + ts2) + ts2 is true."

to be replaced by
"Monoids are associative operations. For instance, if ts1, ts2, and ts3 are three time series, then the property ts1 + (ts2 + ts3) = (ts1 + ts2) + ts3 is true"

Chapter 1 Getting started /
Generative models – Page 17

$$p(Y|X) = p(X|Y).p(Y)/p(X)$$

to be replaced by

$$p(Y|X) = p(X|Y).p(Y)/p(X)$$

Chapter 1 Getting started /
Preprocessing the data set – Page 30-31

One of the solutions is to create a class that computes the counters and the statistics on demand using, once again, the lazy values:

```
class Stats[T <% Double](private values: DVector[T])
  class _Stats(
    var minValue: Double,
    var maxValue: Double,
```

```
    var sum: Double,
    var sumSqr: Double)
  val stats = {
     val _stats = new _Stats(Double.MaxValue,Double.MinValue,0.0, 0.0)
     values.foreach(x => {
        if(x < _stats.minValue) _stats.minValue = x
        if(x > _stats.maxValue) _stats.maxValue = x
        _stats.sum += x
        _stats.sumSqr += x*x
     })
  }
...
```

The same approach is used to compute the multivariate normal distribution:

```
  def gauss: DblVector =
    values.map(x => {
      val y = x - mean
      INV_SQRT_2PI*Math.exp(-0.5*y*y /( stdDev* stdDev))/stdDev
    })
```

Chapter 2: Hello World! /

Dependency injection – Page 48

Overriding val with def

It is advantageous to override the declaration of a value with a defiition of a method with the same signature. Contrary to a value that locks the implementation of the value, a method can return a different value for each invocation:

```
trait PreProcessor { val validation = … }
trait MyValidator extends Validator { def validation
= … }
```

In Scala, a value declaration can be overridden by the method defiition, not vice versa.

To be replaced by

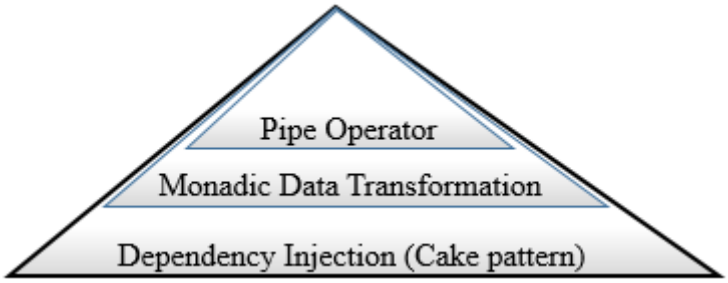
It is advantageous to override the definition of a method with a declaration of a value with the same signature. Contrary to a value which is assigned once for all during instantiation, a method can return different value for each invocation.

A def is a proc that can be a def, a val or a lazy val. Therefore, in Scala you should not override a value `declaration with a method definition with the same signature`

```
trait Validator { val g = (n: Int) => …}
trait MyValidator extends Validator { def g(n: Int) = ..}//WRONG
```

Chapter 2 Hello World! /

The computational framework – Page 44



Chapter 3 Data Preprocessing /

Simple moving average – Page 67

The simple moving average of a time series *{xt}* with a period *p* is computed as the average of the last *p* observations:

$$\tilde{x}_{t+1} = \frac{1}{p}\sum_{j=t-p+1}^{t} x_j$$

The computation is implemented iteratively using the following formula (1):

$$\tilde{x}_{t+1} = \tilde{x}_t + \frac{x_t - x_{t-p}}{p} \quad t \geq p \qquad \tilde{x}_{t+1} = 0 \quad t < p$$

Chapter 3 Data Preprocessing /

The weighted moving average – Page 69

The weighted moving average of a series *{xt}* with a period *p* and a normalized weights distribution *{αj}* is given by the following formula (2):

$$\tilde{x}_{t+1} = \frac{1}{p}\sum_{j=t-p+1}^{t} \alpha_{j-t+p-1} x_j \quad with \sum_{j=0}^{p-i} \alpha_j = 1 \quad for\ t \geq p$$

The exponential moving average on a series *{xt}* and a smoothing factor α is computed by the following iterative formula:

$$\tilde{x}_{t+1} = (1-\alpha)\tilde{x}_t + \alpha x_t$$

Chapter 3 Data Preprocessing /

Time series – Page 65

Here is a partial implementation of the XTSeries class. Comments, exceptions, argument validations, and debugging code are omitted in the code:

```
class XTSeries[@specialized(Double) T](label: String, arr:
Array[T]) { // 1
```

…

Chapter 3 Data Preprocessing /

The Kalman filter – Page 85

"In this respect, the Kalman filter shares some similarities with the **Hidden Markov models** (**HMM**) described in *Chapter 6, Regression and Regularization* [3:11]".

to be replaced by
"In this respect, the Kalman filter shares some similarities with the **Hidden Markov models** (**HMM**) described in *Chapter 7, Sequential Data Models* [3:11].
The section describes the **discrete-time Kalman** filter. The **continuous-time Kalman** filter uses ordinary differential equations for state and measurement equations."

Chapter 4 Unsupervised learning /
Gaussian mixture model – Page 119

The mixture model

If *{xi}* is a set of observed features associated with latent features *{zk}*, the probability for the feature *xi* given *zk* has a value *j*:

$$p(x_i|z_i = j)$$

he probability *p* is called the base distribution. If we extend to the entire model, *θ= {xi, zk}*, the conditional probability is defied as follows

$$p(x_i|\theta) = \sum_{j=1}^{J} \pi_j p(x_i|z_i = j) \quad with \quad \sum_{j=1}^{J} \pi_j = 1 \ \ and \ \pi_j \in [0,1]$$

Chapter 6 Sequential Data Models /
Numerical optimization – Page 191

Iterative approximation using the Taylor series is described as follows

$$f(x_i|w) - f(x_i|w^{(k)}) \sim \sum_{j=0}^{D-1} \frac{\partial f(x_i|w^{(k)})}{\partial w_j} (w - w^{(k)})$$

Chapter 7 Sequential Data Models /
The hidden Markov model / Baum-Welch estimator (EM) – Page 224

The maximum likelihood, **maxLikelihood** is computed as part of the constructor to ensure consistent state of the hidden Markov model.

```
var likelihood = frwrdBckwrdLattice
Range(0, state.maxIters) find( _ => {
  lambda.estimate(state, obs) //1
  val _likelihood = frwrdBckwrdLattice //2
  val diff = likelihood - _likelihood //3
  likelihood = _likelihood
  diff < eps //4
}) match {
   case Some(likeliHood) => state.lambda.normalize; likeliHood
```

Chapter 7 Sequential Data Models /
The hidden Markov model / Viterbi algorithm - Page 227 – 228

```
class ViterbiPath(_lambda: HMMLambda, _state: HMMState, _obs:
Array[Int]) extends HMMInference(_lambda, _state, _obs) {
   val maxDelta = (recurse(0), state.QStar())
   …
}
```

The recursive method that implements [M14] and [M15] steps is invoked by the constructor:

```
@scala.annotation.tailrec
def recurse(t: Int): Double = {
  if( t == 0)  initial   //1
  else {
     Range(0, lambda.getN).foreach( updateMaxDelta(t, _) ) //2
     if( t ==  obs.size-1) {  //6
       val idxMaxDelta = Range(0, lambda.getN).map(i => //[M14]
          (i, state.delta(t, i))).maxBy(_._2)
       state.QStar.update(t+1, idxMaxDelta._1) //7 [M15]
       idxMaxDelta._2
     }
     else
       recurse(t+1)
  }
}
```

Once initialized (line 1) for the first observation, the maximum value of delta and its state index are computed for each of the state (line 2) by the method, `updateMaxDelta`. Next, the index of the column of the transition matrix *A* corresponding to the maximum of delta is computed (line 3). The last step is to update the matrix psi (line 4) (with respect to delta (line 5)). Once the step t reaches the maximum number of observation labels (line 6), the optimum sequence of states *q** is computed [M15] (line 7). Ancillary methods are omitted.

```
def updateMaxDelta(t: Int, j: Int): Unit = {
  val idxMaxDelta = Range(0, lambda.getN).map(i => //3
     (i, state.delta(t-1, i)*lambda.A(i, j))).maxBy(_._2)
  state.psi += (t, j, idxDelta._1) //4
  state.delta += (t, j, idxDelta._2) //5
}
```

Chapter 7 Sequential Data Models /
The hidden Markov model /The hidden Markov model for time series - Page 232

"Moreover, moving averaging techniques, discrete Fourier transforms, and generic Kalman fiters require the states to be continuous with linear dependencies, although the extended Kalman filter can approximate nonlinear states."

to be replaced by

"Filtering techniques such as moving averaging techniques, discrete Fourier transforms, and Kalman filter applies to both discrete and continuous states while HMM does not. Moreover, the extended Kalman filter can estimate nonlinear states"

Chapter 8 Kernel Models and Support Vector Machines/
The non-separable case (soft margin) - Page 259

υ-SVM formulation of a linear SVM:

$$\min_{w,\rho,\xi}\left\{\frac{w^T w}{2}-\rho+\frac{1}{\upsilon n}\sum_{i-0}^{n-1}\xi_i\right\}$$
$$\xi_i \geq 0, \quad y_i(w^T x+w_0) \geq \rho-\xi_i \quad \forall i$$

Chapter 10 Genetic Algorithm /
Crossover / Population - Page 347

The **geneticIndices** method (line 5) computes the relative indices of the crossover bit in the chromosomes and genes:

```
def geneticIndices(prob: Double): GeneticIndices = {
  var idx = (prob*chromosomeSize).floor.toInt
  val chIdx = if(idx == chromosomeSize) chromosomeSize-1 else idx
  idx = (prob*geneSize).floor.toInt
  val gIdx = if(idx == geneSize) geneSize-1 else idx
  GeneticIndices(chIdx, gIdx)
}
```

Chapter 11 Reinforcement Learning /
Reinforcement learning /Action-value iterative update - Page 372

Q-Learning value-action updating formula

$$\hat{Q}_t^{\pi} = Q_t^{\pi} + \alpha\left[r_{t+1} + \gamma \max_{a_{t+1}} Q^{\pi}(s_{t+1}, a_{t+1}) - Q_t^{\pi}\right]$$

Chapter 12 Scalable Framework /
Akka / The Master actor - Page 420

The receive message handler processes only two types of messages: Start from the client code and Completed from the workers, as shown in the following code:

```
override def receive = {
  case s: Start => split
  case msg: Completed => {
    if(aggregator.size >= partitioner.numPartitions-1) {
      aggregate
      workers.foreach( context.stop(_) )
    }
    aggregator.append(msg.xt.toArray)
  }
  case Terminated(sender) => {
    if(aggregator.size >= partitioner.numPartitions-1) {
```

```
      context.stop (self)
      context.system.shutdown
    }
  }
}
```

Chapter 12 Scalable Framework /

Akka / Master with routing – Page 422

```
override def receive = {
  case msg: Start => split
  case msg: Completed => {
    if(aggregator.size >= partitioner.numPartitions-1) {
      aggregate
      context.stop(router)
    }
    aggregator.append(msg.xt.toArray)
  }
  case Terminated(sender) => {
    if( aggregator.size >= partitioner.numPartitions-1) {
      context.stop(self)
      context.system.shutdown
    }
  }
}
```