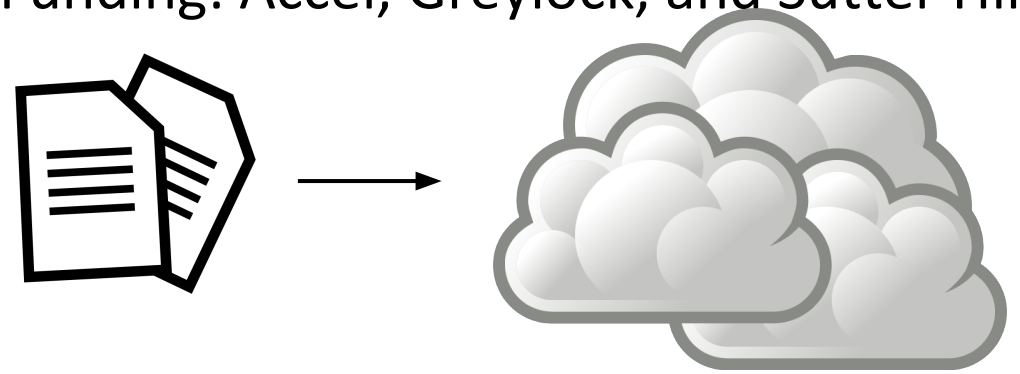
Scala type classes and machine learning

David Andrzejewski

Bay Area Scala Enthusiasts - 1/14/2013, 1/16/2013

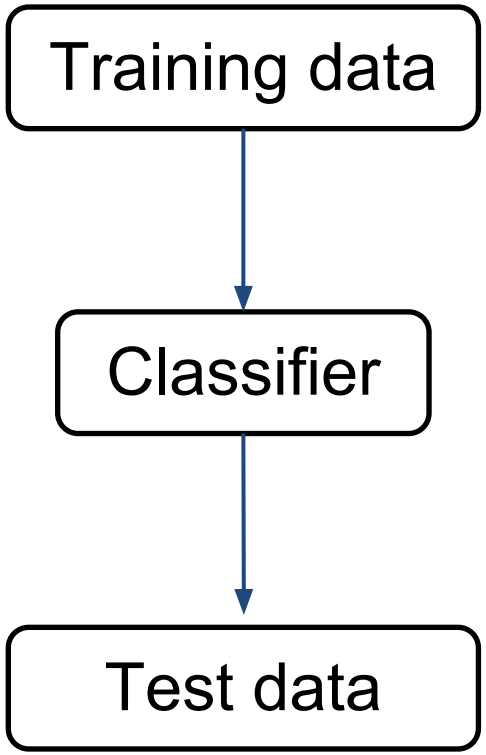
Context: Sumo Logic

- Mission: Transform Machine Data into IT and Business Insights
- Offering: First enterprise-class cloud-based log management and analytics service
- Funding: Accel, Greylock, and Sutter Hill Ventures



- **Advanced analytics**
- **Scala**

Machine learning example: spam classification

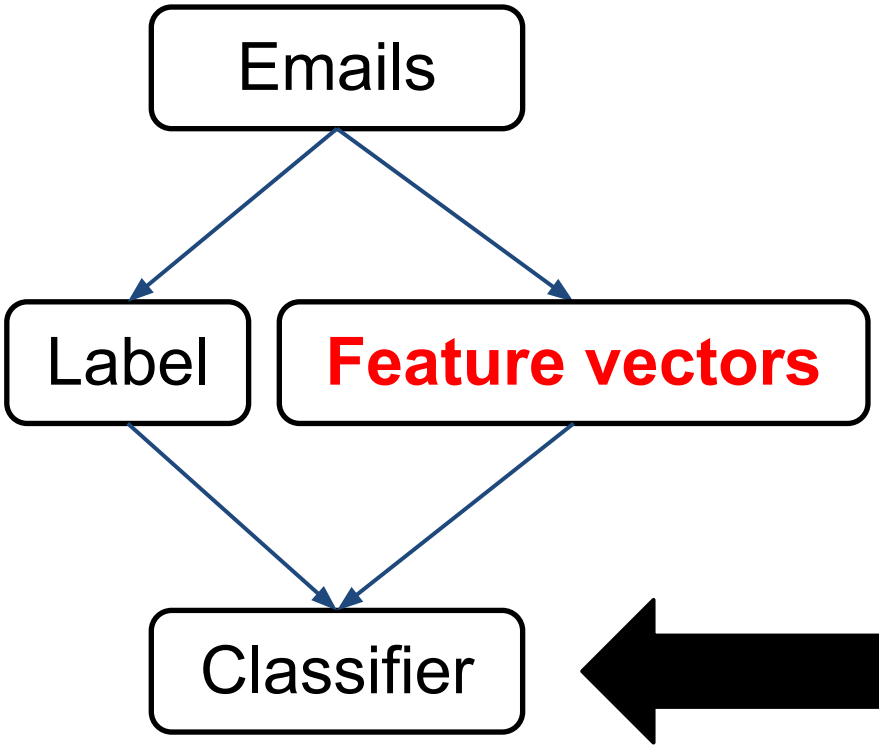


Hand-labeled e-mails

Naive Bayes

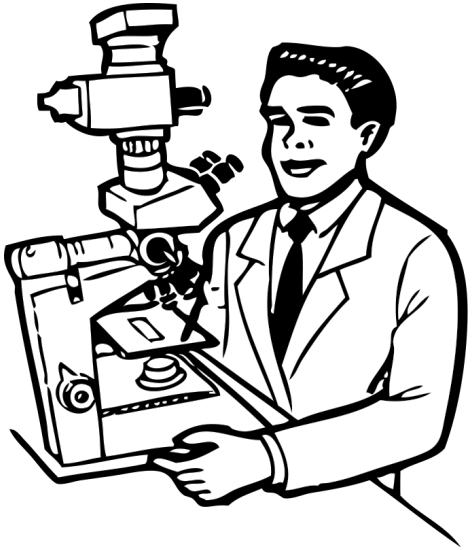
Real e-mails "in the wild"

Machine learning example: more details



Actually important

Fun and exciting



Feature vectors: machine learning

- Machine learning on **feature vectors**
  - "Bob, your new project idea is awful."
  - [1.3, 7.5, 0.0, 0.0, -4.2, ...]

- **Crucial** design decision
  - single words vs ngrams?
  - token vs character?
  - counts vs binary?
  - binary presence / absence?
  - weighting schemes (TF-IDF)?
  - smoothing?

Feature vectors: software engineering

- Machine learning code **only cares about features**

- Input formats for popular libraries
  - scikits.learn (Python): NumPy arrays
  - weka (Java): **Instance** interface
  - mahout (Hadoop-Java): **Vector** interface

- Common integration issues
  - caller bookkeeping
  - not type-safe / parameterized
  - wrappers not easily composable

Polymorphism via type class pattern

- **is-a** type contract **[T <: FeatureVector]**
  - inheritance / interfaces / sub-typing
  - "type T is a calculator of its feature vector"
- **has-a** type contract **[T : FeatureVector]**
  - type class pattern
  - "type T has a calculator of its feature vector"
- Classic Scala examples: Numeric and Ordering

Context bound [T : FeatureVector]

```
trait FeatureVector[T] {
  def features(x: T): Array[Double]
}

class LinearClassifier[T : FeatureVector](weights: Array[Double]) {
  val featureMapping = implicitly[FeatureVector[T]]

  def innerProduct(x: Array[Double], y: Array[Double]): Double = {
    (0 until x.size).map{i => x(i) * y(i)}.sum
  }

  def classify(x: T): Boolean = {
    innerProduct(featureMapping.features(x), weights) > 0
  }
}
```

Context bound (de-sugared)

```
trait FeatureVector[T] {
  def features(x: T): Array[Double]
}

//class LinearClassifier[T : FeatureVector](weights: Array[Double]) {
//  val featureMapping = implicitly[FeatureVector[T]]
class LinearClassifier[T](weights: Array[Double])
                         (implicit featureMapping: FeatureVector[T]) {

  def innerProduct(x: Array[Double], y: Array[Double]): Double = {
    (0 until x.size).map{i => x(i) * y(i)}.sum
  }

  def classify(x: T): Boolean = {
    innerProduct(featureMapping.features(x), weights) > 0
  }
}
```

Context bound: caller usage

```
implicit object CharValueFeatures extends FeatureVector[String] {
  def features(x: String): Array[Double] = {
    val a = x.filter(_ == 'a').size.toDouble
    val b = x.filter(_ == 'b').size.toDouble
    Array[Double](a,b)
  }
}

val classifier = new LinearClassifier[String](Array(-1.0, 0.5))
println("%s classified as %b".format("aaaB", classifier.classify
("aaaB")))
```

```
aaaB classified as false
```

"manual override"

```
object CharCaseFeatures extends FeatureVector[String] {
    def features(x: String): Array[Double] = {
      val upper = x.filter(_.isUpper).size.toDouble
      val lower = x.filter(_.isLower).size.toDouble
      Array[Double](upper,lower)
    }
}

val classifier =
     new LinearClassifier[String](Array(-1.0, 0.5))(CharCaseFeatures)

println("%s classified as %b".format("aaaB", classifier.classify
("aaaB")))
```

```
aaaB classified as true
```

sumologic

Why use this (or not)?

- Advantages
  - extend code you don't own
  - cleaner than inheritance / wrapping
  - easily substitute evidence parameters
  - avoid "over-stuffing" value types (eg, case classes)
  - leverage type checking
  - parameter-passing vs type annotation

- Disadvantages
  - implicits can be tricky
  - unfamiliar code idiom

- Other applications
  - serialization

References

- Shameless plug: Sumo Logic Free (500 MB / day)
  - https://www.sumologic.com/signup/
- Scala type class pattern
  - Sumo Logic blog post (+ references)
    - http://www.sumologic.com/blog/company/scala-at-sumo-type-classes-with-a-machine-learning-example
  - Seth Tisue talk at nescala 2012
    - http://www.youtube.com/watch?v=yYo0gANYViE
  - Scala in Depth by Joshua D. Suereth
- Other related stuff (I think?)
  - Haskell typeclasses
  - Clojure protocols
  - Scalaz

sumologic