Featran

Type Safe and Generic Feature Transformation in Scala

Fallon Chen, fallon@spotify.com

Feature Engineering

The process of using domain knowledge to transform features into a form useful in machine learning algorithms

Features

Measurable property or characteristic of phenomenon being observed

```
case class Song(id: Long,
                name: String,
                artist: String,
                genre: Option[String],
                numPlays: Long)

val songs = Seq(
Song(1, "Blackbird","The Beatles", Some("Rock"), 46522936),
Song(2, "Bleed", "Meshuggah", Some("Metal"), 8741978),
Song(3, "Bad and Bougee", "Migos", Some("Hip Hop"), 379732121),
Song(4, "Feet Don't Fail Me", "QOTSA", Some("Rock"), 5642096))
```

Features Useful for ML

Machine learning algorithms operate on collections of numbers (doubles, floats)

```scala
// collaborative filtering for recommendation systems
// Create 2d array of Doubles representing (user, song) pair
val userLiked = users
  .map(u => songs.map(s => if(u.favoriteSongs.contains(s)) 1.0 else 0.0)

// Factor out vectors representing taste
val (userVectors, songVectors) =
   secretSauceMatrixFactorization(songUserMatrix)

// .. compare userVectors, songVectors to make recommendations
```

How do we get from raw features like Song to ML algorithm friendly features like songVectors?

Feature Transformation

3 Basic Feature Transformations

- Binarizer
- Max-Min Scaler
- One Hot Encoding

Binarizer

Given a threshold, return 0 or 1 based on whether or not the value crosses the threshold

```
val threshold = 50000000L // 50M
val isPopular = songs.map(s => if(s.numPlays >= threshold) 1.0 else 0.0)
// isPopular: Seq[Double] = List(1.0, 1.0, 1.0, 1.0)
```

Min-Max Scaler

Given a min and a max, scale features to fit within that range

```scala
val min = 1.0
val max = 10.0
// find min and max song plays in one pass
val (songsMinPlays, songsMaxPlays) = songs
         .map(s => (s.numPlays.toFloat, s.numPlays.toFloat))
         .reduce((x, y) => (math.min(x._1, y._1), math.max(x._2, y._2)))
val minMax = songs.map { x =>
    ((x.numPlays - songsMinPlays) / (songsMaxPlays - songsMinPlays))
      * (max - min) + min
}
//minMax: Seq[Double] = List(1.9835267513990402, 1.074578134343028,
// 9.999999463558197, 1.0)
```

One Hot Encoding

Given a sequence of categories over all features, the feature is transformed into an sequence that has a 1 in the position matching its category and 0 otherwise

- Song("Bleed", "Meshuggah", "Metal", 8741978)
- Genres: ("Hip Hop", "Metal", "Rock")
- One Hot: (0 , 1 , 0)

One Hot Encoding

```
// Find unique ordered categories i.e. genres
val genres = songs.map(_.genre)
                  .map(Set(_))
                  .reduce(_ ++ _)
                  .toSeq.sorted
// genres: Seq[Option[String]] =
// ArrayBuffer(Some(Hip Hop), Some(Metal), Some(Rock))

// Calculate one hots
val oneHots = songs
               .map(s => genres.map(g => if(s.genre == g) 1.0 else 0.0))
//oneHots: Seq[Seq[Double]] = List(ArrayBuffer(0.0, 0.0, 1.0),
// ArrayBuffer(0.0, 1.0, 0.0), ArrayBuffer(1.0, 0.0, 0.0),
// ArrayBuffer(0.0, 0.0, 1.0))
```

Notice a pattern?

- Extract T => A e.g. Song => genre
- Aggregate
    - Map A => B e.g. genre to Set
    - Reduce B => B e.g. Set union
    - Map B => C e.g. Set to Seq in One Hot
- Transform A using C e.g. Song => Seq using genres

Transformer

- Aggregate
    - Map A => B e.g. genre to Set
    - Reduce B => B e.g. Set union
    - Map B => C e.g. Set to Seq in One Hot
- Transform A using C e.g. Song => Seq using genres

Featran

FeatureSpec

```
class FeatureSpec[T] {
    def required[A](f: T => A)(t: Transformer[A, _, _])
    def optional[A](f: T => Option[A], default: Option[A] = None)
                    (t: Transformer[A, _, _])
}
```

FeatureSpec

```
import com.spotify.featran._
import com.spotify.featran.transformers._

val featureSpec = FeatureSpec.of[Song]
    .required(_.numPlays)(Binarizer("isPopular", 50000000.0))
    .required(_.numPlays)(MinMaxScaler("numPlay_normalized"))
    .optional(_.genre)(OneHotEncoder("genre"))
```

FeatureExtractor

```scala
val featureExtractor = featureSpec.extract(songs)
// featureExtractor: com.spotify.featran.FeatureExtractor[Seq,Song]
```

FeatureExtractor - Input Types

FeatureSpec#extract[M[_]: CollectionType](input: M[T]): FeatureExtractor[M, T]

```
trait CollectionType[M[_]] { self =>
    def map[A, B](ma: M[A], f: A => B): M[B]
    def reduce[A](ma: M[A], f: (A, A) => A): M[A]
    def cross[A, B](ma: M[A], mb: M[B]): M[(A, B)] // map A with C
}
```

Featran provides definitions for Scala collections and as well as distributed collection types that extend this trait

FeatureExtractor - Transformed Features

- featureNames
- featureValues
- featureSettings

FeatureExtractor - Names

Names passed to the transformer

```
val featureSpec = FeatureSpec.of[Song]
  .required(_.numPlays.toDouble)(Binarizer("isPopular", 50000000.0))
  .required(_.numPlays.toDouble)(MinMaxScaler("numPlay_normalized"))
  .optional(_.genre)(OneHotEncoder("genre"))

val featureExtractor = featureSpec.extract(songs)

val featureNames = featureExtractor.featureNames
// featureNames: Seq[Seq[String]] =
// List(List(isPopular, numPlay_normalized,
//           genre_Hip Hop, genre_Metal, genre_Rock))
```

FeatureExtractor - Values

```scala
val featureValues = featureExtractor.featureValues[Seq[Double]]
// featureValues: Seq[Seq[Double]] = List(
//       List(0.0, 0.10928075401101646, 0.0, 0.0, 1.0),
//       List(0.0, 0.00828645992365073, 0.0, 1.0, 0.0),
//       List(1.0, 1.0, 1.0, 0.0, 0.0),
//       List(0.0, 0.0, 0.0, 0.0, 1.0))
```

FeatureExtractor - Values - Output Types

FeatureExtractor#featureValues[F: FeatureBuilder]: M[F]

```scala
trait FeatureBuilder[T] { self =>
  def init(dimension: Int): Unit
  def add(name: String, value: Double): Unit
  def skip(): Unit
  def result: T
  def map[U](f: T => U): FeatureBuilder[U] = new FeatureBuilder[U] {
  // ...
}
```

Featran provides these output types and more: Traversable[T], SparseVector[T], DenseVector[T], TensorFlow Example protobuf ...

FeatureExtractor#featureSettings

Save the results of the Aggregator step (map-reduce-map) for future use

```
val settings = featureExtractor.featureSettings
// settings: Seq[String] = List(
// [{"cls":"com.spotify.featran.transformers.Binarizer",...,"aggregators":
//   {"cls":"com.spotify.featran.transformers.MinMaxScaler", ... ,
//    "aggregators":"5642096.0,3.79732121E8,1.0"},
//   {"cls":"com.spotify.featran.transformers.OneHotEncoder", ... ,
//    "aggregators":"label:Hip+Hop,label:Metal,label:Rock"}])
```

FeatureExtractor#featureSettings

Use saved settings to skip Aggregator step

```
val ews = featureSpec.extractWithSettings(songs, settings)
// ews: FeatureExtractor[Seq,Song]
```

Other Features - Feature Rejection

Transformer provides separate collection with errors for each record

```
val validValues = featureExtractor.featureResults[Seq[DoubleSong]]
                                    .filter(_.rejections.isEmpty)
                                    .map(_.value)
```

Other Features - Combining FeatureSpecs

Compute multiple feature transformations over the same inputs

```scala
val fs1 = FeatureSpec.of[Song]
      .required(_.numPlays.toDouble)(Binarizer("isPopular", 50000000.0))
      .required(_.numPlays.toDouble)(MinMaxScaler("numPlay_normalized"))

val fs2 = FeatureSpec.of[Song]
      .optional(_.genre)(OneHotEncoder("genre"))

val multiFeatures = MultiFeatureSpec(fs1, fs2).extract(songs)
```

Other features - Feature Crossing

Cross values of a pair of transformers by specifying transformer and function for combining

Other features - Java API

Use Java when Scala isn't available

Available Transformers

- Binarizer
- Bucketizer
- Hash{OneHot,NHot,NHotWeighted}Encoder
- HeavyHitters
- Identity
- MaxAbsScaler
- MinMaxScaler
- {OneHot,NHot,NHotWeighted}Encoder
- PolynomialExpansion
- QuantileDiscretizer
- StandardScaler
- VectorIdentity
- VonMisesEvaluator

Why use Featran?

- Reproducibility of feature engineering
    - Clarity and flexibility in how you compose and build features
    - Ability to save intermediate stages using featureSettings
- Many common transformers provided out of the box
- Supports many input collection types and output feature formats

Want to know more?

https://github.com/spotify/featran

Thanks! Questions?