How to Eliminate Surprises In Your Data

Anne DeCusatis and Idrees Khan
Northeast Scala Symposium 2020
The example pipeline

I’m gonna take my [data] to the old town [pipeline],
gonna [process data] til I can’t no more
Example pipeline

- Input: a dataset that contains all track listens for a given day

```json
{
    "name": "TrackPlay",
    "namespace": "com.spotify.surprises.schema",
    "type": "record",
    "fields": [
        {
            "name": "trackId",
            "type": "string"
        },
        {
            "name": "country",
            "type": "string"
        },
        {
            "name": "msPlayed",
            "type": "long"
        }
    ]
}
```
Example pipeline

- Input: a dataset that contains played songs
- Output: a dataset that contains the most-played songs

```scala
class TrackPlayCountJob {
  def main(cmdlineArgs: Array[String]): Unit = {
    val (sc, args) = ContextAndArgs(cmdlineArgs)

    sc.avroFile[TrackPlay](args("trackPlays"))
      .map(tp => (tp.getTrackId.toString, 1L))
      .groupByKey
      .map(t => (t._1, t._2.sum))
      .map { case (t, c) => s"$t	$c" }
      .saveAsTextFile(args("output"))

    sc.close()
  }
}
```
The pipeline doesn’t exist in isolation

[data] queen, feel the beat of the [new hire team]
Open source tools for data quality

We’ll float on, good [tools] are on the way
Integration testing

I want your love and I want your revenge
You and me could write a [pipeline test]
Example pipeline

- Input: a dataset that contains played songs
- Output: a dataset that contains the most-played songs
- Unit test: example inputs have expected outputs
Ratatool - Scalacheck

- Our team is writing pipeline logic... need to write tests
  - Unit tests
  - Integration tests
  - Property-based tests?
```scala
private def mockTrackPlay(trackId: String, msPlayed: Long) = {
  TrackPlay.newBuilder()
    .setTrackId(trackId)
    .setMsPlayed(msPlayed)
    .setCountry("US")
    .build
}
```

```scala
private def mockTrackPlay(trackId: String, msPlayed: Long) = {
  specificRecordOf[TrackPlay]
    .amend(Gen.const(trackId))(_.setTrackId)
    .amend(Gen.const(msPlayed))(_.setMsPlayed)
    .sample.get
}
```

```scala
mockTrackPlay("trackId", 600000L)
// {"trackId": "trackId", "country": "仆様(1)", "msPlayed": 600000}
```
End to end tests - quickly

I’m the [DAG] guy....
Ratatool - Sampler

- Want to run pipeline end-to-end
- We have 232M MAU, so lots of track plays per day
- Pipelines can cost a lot of money or take a long time
- Downsample input to reduce iteration time

```bash
ratatool bigSampler avro
--in gs://bucket/input_tracks.avro
--out gs://bucket/sampled_tracks.avro
--sample=0.01
```
Ratatool - Sampler
Where are you running the end to end tests?

And I’m here to remind you of the mess you left when you [wrote to prod]
Test Environment

- Now it’s time for our team to run their pipeline!
- Common issues & mistakes in the development lifecycle can create conflict or confusion
  - Multiple engineers testing at the same time
  - Engineers may accidentally publish test data to production
    - This can propagate downstream
  - Testing resources can eat up production quotas
  - Engineers can forget to clean up testing data
Testing changes in your dataset output

Turn and face the strange
```scala
object TrackPlayCountJob {
  def main(cmdlineArgs: Array[String]): Unit = {
    val (sc, args) = ContextAndArgs(cmdlineArgs)

    val trackPlays = sc.avroFile[TrackPlay](args("trackPlays"))
      .map(tp => (tp.getTrackId.toString, 1L))
      .groupByKey
      .map(t => (t._1, t._2.sum))
      .map { case (t, c) => s"$t\t$c" }
    .saveAsTextFile(args("output"))

    sc.close()
  }
}
```

```scala
object TrackPlayCountJob {
  def main(cmdlineArgs: Array[String]): Unit = {
    val (sc, args) = ContextAndArgs(cmdlineArgs)

    sc.avroFile[TrackPlay](args("trackPlays"))
      .map(_.getTrackId)
      .countByValue
    .map { case (t, c) => s"$t\t$c" }
    .saveAsTextFile(args("output"))

    sc.close()
  }
}
```
Ratatool - Diffy

- Pipeline updated to be more performant
- Have we broken anything?

```
ratatool bigDiffy
  --input-mode=avro
  --key=track_id
  --lhs=gs://bucket/unoptimized.avro
  --rhs=gs://bucket/optimized.avro
  --output=gs://bucket/diff
```
Testing content in your dataset output

How was I supposed to know that something wasn't right here?
Validation

- How do we have confidence in what the data actually contains?
  - Are our TrackIDs actually TrackIDs?
  - How many invalid countries do I have?
Validation

- Have Avro record containing fields with avro types
- Records can have many fields to be validated
- Records can have Nesting or Repeated fields
- Primitive data types can represent many different kinds of data
  - A String could be a Track ID or a Country Code
- Many Data Engineers spread across different teams who have different expectations of their data
- Want to provide simple API for the pipeline author
Validation

trait ValidationType {
  def checkValid: Boolean
}

case class CountryCode(protected val data: String) extends ValidationType {
  override def checkValid: Boolean = Locale.getISOCountries.contains(data)
}

trait Validator[A <: ValidationType] {
  def validate(a: PreValidation[A]): PostValidation[A]
}
Validation

class ValidationTypeValidator[A <: ValidationType] extends Validator[A] {
  override def validate(data: PreValidation[A]): PostValidation[A] =
    data.validate
}

implicit def vtv[T <: ValidationType]: Validator[T] = new ValidationTypeValidator[T]
object Validator {
    type Typeclass[T] = Validator[T]

    def combine[T](caseClass: CaseClass[Validator, T]): Validator[T] = new Validator[T] {
        override def validate(a: PreValidation[T]) : PostValidation[T] = {
            val mapped: Seq[PostValidation[T]] = caseClass.parameters.map(param =>
                param.typeclass.validate(param.dereference(a.data)))

            val record = caseClass.rawConstruct(mapped)

            if (mapped.exists(_.isInvalid)) {
                Invalid(record)
            } else {
                Valid(record)
            }
        }
    }
}
Validation

```scala
implicit class SCollectionValidator[T](sc: SCollection[T])(implicit vr: Validator[T]) {
  def validate(): SCollection[PostValidation[T]] = {
    sc.applyTransform(ParDo.of(new ValidatorDoFn[T](vr)))
  }
}
```

```scala
implicit class SCollectionConverter[GR <: GenericRecord](sc: SCollection[GR]) {
  def fromAvro[T](implicit c: AvroConverter[T]): SCollection[T] = {
    sc.applyTransform(ParDo.of(new FromAvroConverterDoFn(c)))
  }
}
```
Validation

- Magnolia can be used to derive Validator Typeclasses for our records
- Can also be used to derive Converters to/from Avro

```scala
object TrackPlayCountJob {
  def main(cmdlineArgs: Array[String]): Unit = {
    val (sc, args) = ContextAndArgs(cmdlineArgs)
    sc.avroFile[TrackPlay](args("trackPlays"))
      .fromAvro[TrackPlayCC]
      .validate()
      .map(_.trackId)
      .countByValue
      .map { case (t, c) => s"$t\t$c" }
      .saveAsTextFile(args("output"))
    sc.close()
  }
}

case class TrackPlayCC(
  trackId: TrackId,
  country: Country,
  msPlayed: PosNum
)
```
How do we know if upstream has changed?

I just took a DNA test, turns out I'm 100% that [data]
Statistical Profiling

- Our pipeline is running End-to-End
- How can we have a high level view of the actual production inputs/outputs?
  - What is the distribution of msPlayed?
  - How many distinct countries do I have?
  - What are the most common tracks?
Focus on easy to use tools
Spotify open source:
  https://github.com/spotify/ratatool
  https://github.com/spotify/elitzur (new in March 2020)
  https://github.com/spotify/scio
  https://twitter.com/SpotifyEng

Other open source:
  https://github.com/propensive/magnolia
  https://github.com/typelevel/scalacheck

Anne:
  anned@spotify.com, twitter @precisememory, github anne-decusatis

Idrees:
  idreess@spotify.com, twitter @idreesxkhan, github idreeskhan

Songs in this talk: https://spoti.fi/2pMA6qu