Continuous Distribution Goodness-of-Fit in MLlib

Jose Cambronero

Data's portrait

Data can come in many shapes and forms, with an equally large amount of ways to describe them. Most people have dealt with averages and standard deviation. These are in effect *shorthand* for that description. Sometimes we might want more information, more perspective into that data's profile. In a situation like that we might look at other measures, such as skewness and kurtosis. However, sometimes we can provide a much neater description for data by stating it comes from a given distribution, which not only tells us things like the average value that we should expect, but effectively gives us the data's "recipe" so that we can compute all sorts of neat information from it.

Armed with this information we can go out in the world and try our hand at many tasks: we can perform anomaly detection by analyzing the likelihood of any new values under that distribution; we can seek to better understand natural phenomena by analyzing their statistical properties; we can validate our modeling logic and improve our ability to model processes; the opportunities are endless.

Innocent until proven guilty (read: null until proven alternate!)

We might have various distributions that we think fit our data. For example, we might have just fit a linear regression to a dataset, and believe that our residuals are white noise (i.e. standard normal), and we would now like some assurance that this is the case and we're not missing some important features in our model. Or we might have measured the wait time for the 6 train in NYC at the City Hall stop (protip: it's forever…just come to terms with that) and we'd like to model this using an exponential distribution. We now have the task of testing whether our empirical data actually follow these theoretical distributions.

It is often difficult to conclude that a given hypothesis is correct. However, the converse, concluding that a given hypothesis is wrong, is actually much simpler. This idea might initially seem a bit counterintuitive, but a simple example can clear things up. Let's say a bad back is a common (clearly not professional!) diagnosis for back pain. If I have back pain, it is possible that I might have a bad back. It is also possible that I simply sat for too long. However, if I don't have back pain, it is fairly certain that I don't have a bad back. This drives the underlying intuition behind the concept of a null hypothesis and statistical tests for that hypothesis.

We can use this hammer and nail, a statistic and a hypothesis test, to do something similar with our potential distributions. In fact, there are various standard statistical tests that allow us to analyze whether a given sample might come from a given theoretical distribution (or alternatively, whether 2 samples come from the same, unknown, distribution). We'll discuss 2 of these in depth in this blog post: the Kolmogorov-Smirnov test, and the Anderson-Darling test. While "passing" the test doesn't guarantee that the data comes from that distribution, failing it is a pretty surefire way to demonstrate that it does not.

Statistics: Goodness-of-Fit tests

Kolmogorov-Smirnov

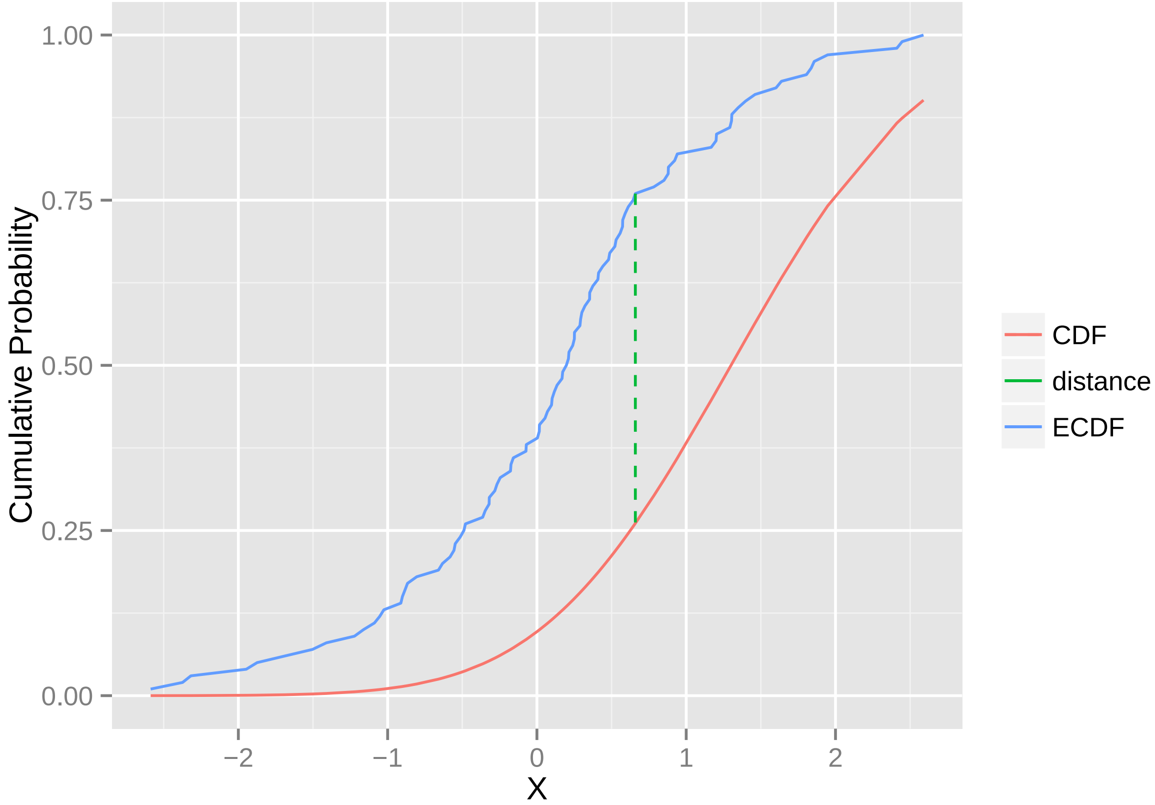
This might be one of the most popular goodness-of-fit tests for continuous data out there, with an implementation in pretty much every popular analytics platform. The intuition behind the test centers around comparing the largest deviation between the cumulative distribution at a given value X under the theoretical distribution and the empirical cumulative distribution.

The Kolmogorov-Smirnov statistic for the 1-sample, 2-sided statistic is defined as

$$D = \max_{i=1}^{n}(\Phi(Z_i) - \frac{i-1}{n}, \frac{i}{N} - \Phi(Z_i))$$

where N is the size of the sample, lZ is the sorted sample, and Φ represents the cumulative distribution function for the theoretical distribution that we want to test.

The general intuition is captured by the graphic below. The test tries to capture the largest difference between the 2 curves. Then, given the distribution of the statistic itself, we can make some claims regarding how likely such a distance would be assuming that the null hypothesis (i.e. that the data comes from the distribution) holds.



One of the main appeals of this test centers around the fact that the test is distribution-agnostic. By that we mean that the statistic can be calculated and compared the same way regardless of the theoretical distribution we are interested in comparing against (or the actual distributions of the 1 or 2 samples, depending on the test variant being performed).

Anderson-Darling

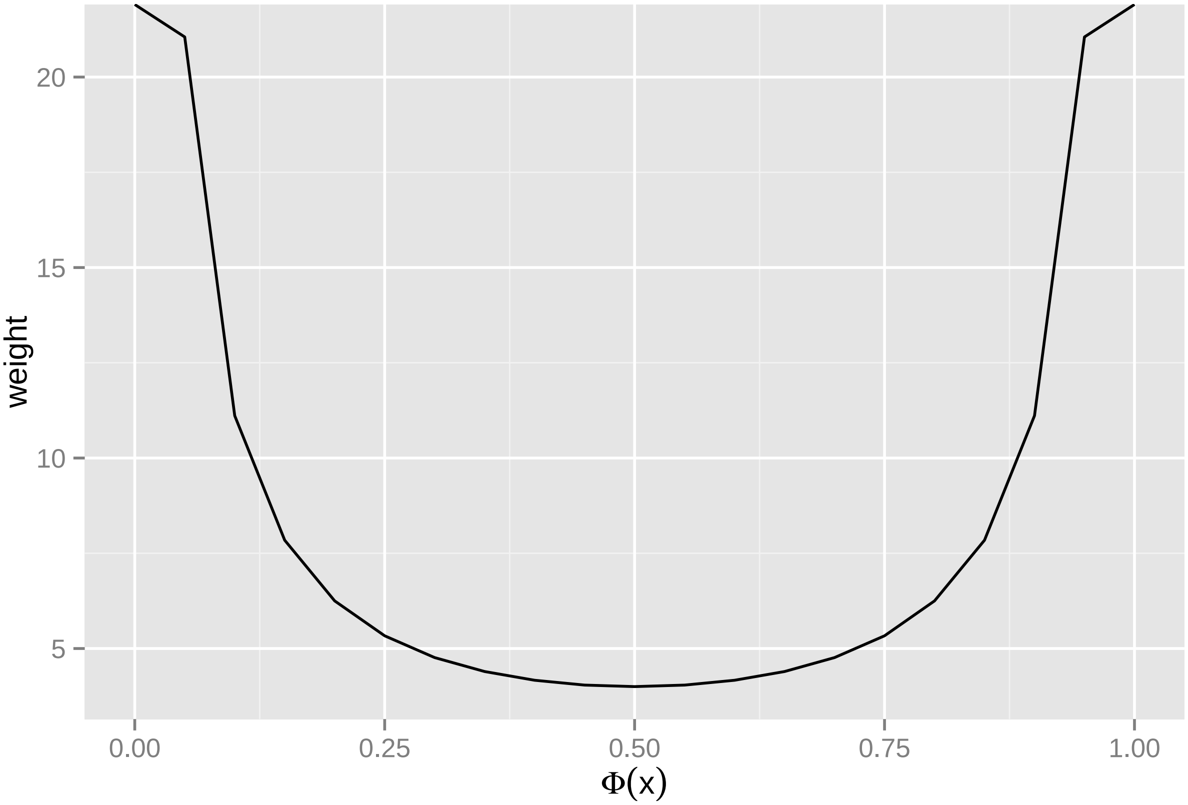
Anderson-Darling is often proposed as an alternative to the Kolmogorov-Smirnov statistic, with the advantage that it is better suited to identify departures from the theoretical distribution at the tails, and is more robust towards the nuances associated with estimating the distribution's parameters directly from the sample that we're trying to test.

In contrast to the Kolmogorov-Smirnov test, the Anderson-Darling test for 1-sample has critical values (i.e. the reference points that we will use to accept or dismiss our null hypothesis) that depend on the distribution we are testing against.

The informal intuition behind Anderson-Darling is that *not all distances are equal*, meaning, that a deviation of size m between the empirical CDF curve and the CDF curve should carry different importance depending on where it happens. Indeed, a small deviation at the start of the CDF or the end could signal that the sample doesn't really stem from that distribution. Why is that? Well, consider that the CDF tends to 0 at the left, and 1 at the right, so there **shouldn't** be any real discrepancies there, for the most part.

Specifically, the Anderson-Darling test weighs the square of the deviations by $\frac{1}{\Phi(x)(1-\Phi(x))}$, where Φ is once again the CDF.

The graph below shows the point we made earlier, departures at the tails are weighed more heavily, and contribute to the overall sum, even if they're nominally small.



Let's take a look at the formula definition of Anderson-Darling (for brevity sake, we will solely consider the computational formula, not the formal version which involves an integral from $-\infty$ to ∞).

$$A = -N - \frac{1}{N}\sum_{i=1}^{n}(2i+1)[\ln(\Phi(Z_i)) + \ln(1-\Phi(Z_{n+1-i}))]$$

Implementations

Distilling distributed distribution (statistics) (now say that three times fast...)

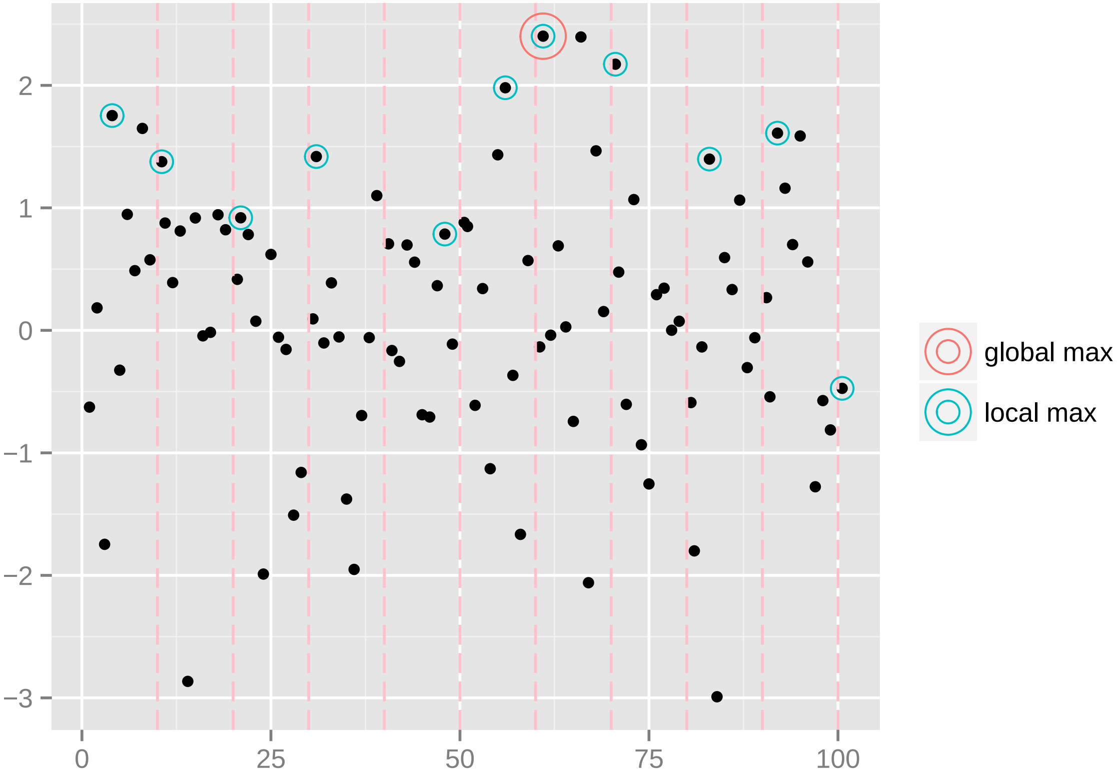
Calculating both of these statistics is extremely straightforward when performed in memory. If we have all our data, we can simply traverse it as needed and calculate what we need at each point. Performing the same calculations in a distributed setting can take a bit more craftiness.

Cloudera has contributed distributed implementations of these tests to MLlib. You can now find in the Statistics object: the 1-sample, 2-sided Kolmogorov-Smirnov test; the 2-sample, 2-sided, Kolmogorov-Smirnov test; the 1-sample Anderson-Darling test; and the k-sample Anderson-Darling test.

The remainder of this blog focuses on explaining the implementation details, along with some use-case examples for 2 of these tests: the 2-sample Kolmogorov-Smirnov test, and the 1-sample Anderson-Darling test. We've chosen these two as they arguably provide the most interesting implementations.

Locals and Globals

The general intuition behind the strategy applied to solve these problems is: we can often arrive at a global value by working with local values. For example, consider the problem of finding the global maximum in the graph below. We could keep track of the last seen maximum as we traverse the graph from left to right. But, if the graph were divided into parts, we could just as easily simply take the maximum of each part (local maximum, and potentially the global maximum) and then compare at the end and pick the winner.

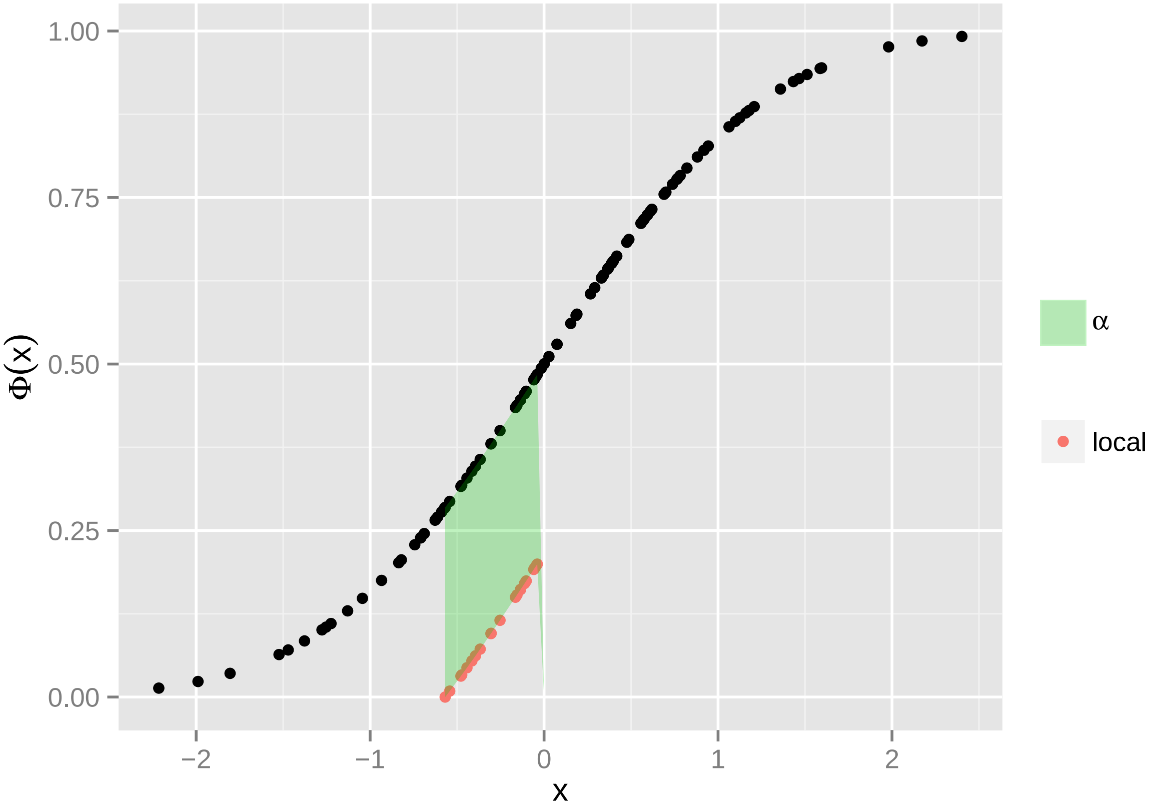


2-sample Kolmogorov-Smirnov test

The 2-sample variant of the KS test allows us to test the hypothesis that both samples stem from the same distribution. Its definition is a nice, straightforward extension of the 1-sample variant: we are looking for the largest distance between the 2 empirical CDF curves. The process for generating the 2 curves is quite simple. You create a co-sorted group Z, and then for each element Z_i you calculate 2 values, y_{i1} and y_{i2}, which are simply the number of elements in each respective sample with a value less than or equal to Z_i, divided by the number of elements in Z. This in effect is the CDF curve for Z assuming we view sample 1 and sample 2 as some sort of *step-wise theoretical CDFs*. It makes sense then, that we could in turn compare these curves to test whether the 2 CDFs (and thus the distributions) are the same.

Implementing this in-memory is very easy. The paragraph above effectively gives you the steps. But now let's think about calculating this in a distributed fashion. We are going to have to combine the data and sort it to create Z. There is no way for us to get around that. Once we have that, however, we'd like to minimize the amount of unnecessary shuffles (recall, this requires writing out data to disk, which is expensive) and jobs (which isn't quite a shuffle, but still consumes resources).

If we compute the empirical CDFs under each sample within a partition, the values for sample 1 will be off by α and the values for sample 2 will be off by β. However, note that all values for each sample within that partition will be off by that same amount. Since both are off by constants, the vertical distance between them is also off by a constant $(\alpha - \beta)$. The graph below shows this idea for the case of 1 sample. The 2-sample case is analogous.

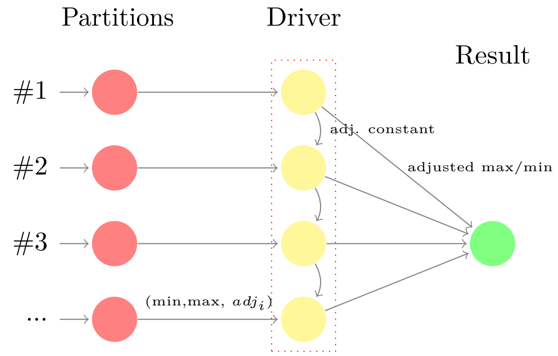


This fact allows us to simply concentrate on finding the largest deviation within each partition, adjusting these later by the approproriate constant (a is really a_i for each partition i, similarly for b), and then comparing the results for the maximum.

The general strategy will aid us well in other implementations and the gist of it can be summarized as: calculate as much as you can within each partition and communicate as little as you can so that you can adjust at then end. So in the case of the 2 sample KS test we want each partition to compute

- The maximum and minimum distance between the local empirical CDFs
- The count of elements in sample 1
- The count of elements in sample 2

We actually combine the last 2 points into 1 adjustment constant, to minimize the number of elements we need to keep track of. The diagram below visualizes the steps described prior.



TODO: add an example here.

Note that the test is meant for continuous distributions, so any ties in ranking will affect the test's power.

1-sample Anderson-Darling test

The implementation of the 1-sample Anderson-Darling test provides yet another productive strategy: algebraic transformations that isolate global vs local components can be useful in maximizing how much we can calculate locally and how much information we need to carry to adjust globally later. The Anderson-Darling statistic is frequently formulated as shown below:

$$A = -N - \frac{1}{N}\sum_{i=1}^{n}(2i+1)[\ln(\Phi(Z_i)) + \ln(1-\Phi(Z_{n+1-i}))]$$

By having a term that involved Z_i and Z_{n+1-i}, this requires that we have access to elements that may or may not be in the same partition. But we can clearly reformulate this in a way that every term in the statistic solely deals with Z_i, thus getting around this issue. We can re-index the second terms so that the sum is based on element i, rather than $n+1-i$. The final form of the statistic is then:

$$A = -N - \frac{1}{N}\sum_{i=1}^{n}(2i+1)\ln(\Phi(Z_i) + (2(n-i)+1)\ln(1-\Phi(Z_i))]$$

For our example, we'll test how well an exponential distribution fits the waiting time between arrivals of the NYC 6 Subway train (downtown bound) at different stops. Such a test might be a natural first line of attack for someone looking to model delays in the subway. For example, if the tests tell us that an exponential distribution cannot be ruled out, then we might decide to use the exponential's CDF to identify inordinately long times, and warn users about potential issues on the track.

We'll be using the May 2011 MTA ATS data. The files are available as csv downloads, with each row providing: the date, train identifier, direction of the train, timestamp of the event, the type of event (arrival or departure), route identifier, stop identifier, and track identifier.

We'll gloss over the parsing of the data here, but you can see the full code at ADDLINK.

```
object AndersonDarlingMTA {
  def main(a: Array[String]): Unit = {
    val sc = new SparkContext("local", "test")
    // TODO: replace with hdfs path in cluster
    val path = "/Users/josecambronero/Projects/gof/data/mta_may_2011"
    val mtaRaw = sc.textFile(path).sample(false, 0.3, 3L) // just a sample for now, since local
    val parsed = mtaRaw.map(x => Try(parseTrainEvent(x))).filter(_.isSuccess).map(_.get)
    // my train: 6, downtown, arrival times :)
    val train6Down= parsed.filter { x =>
      x.train == 6 && x.dir == 1 && x.eventType == 1
    }.sortBy(_.timestamp) // go ahead and sort now


    val stops = train6Down.map(_.stop).distinct
    val results = stops.map { stop =>
      val stopData = train6Down.filter(_.stop == stop).map(_.timestamp)
      // to calculate time span between arrivals we need to process elements sequentially
      // (to some extent). To take advantage of spark, we want to parallelize though
      // so the compromise is: processes partitions in parallel, and operate sequentially
      // within each partition. This means that we need to drop 1 span per partition (the first
      // one), and we drop any observations that are alone in a partition.
      val waitingTimes = stopData.mapPartitions { part =>
        val local = part.toArray
        local.zip(local.drop(1)).map { case (pt, t) =>
          (t.getMillis - pt.getMillis).toDouble / 6e4
        }.iterator
      }.filter(_ > 0)
      // We estimate the MLE for exponential (i.e. the mean of the data)
      val expDist = new ExponentialDistribution(waitingTimes.mean())
      // Just using KS as a place holder, this should be AD, since it is robust
      // against estimating parameters from the data that we're testing
      val result = Statistics.kolmogorovSmirnovTest(
        waitingTimes,
        (span: Double) => expDist.cumulativeProbability(span)
        )
      (stop, result)
    }
  }
```

While in this blog post we have used the Scala API, all these tests have also been made available in PySpark, allowing data scientists to calculate familiar statistics on big-data sized samples. As usual, all feedback is welcome. Happy hypothesis-testing!