Hypothesis Test Using Kolmogorov-Smirnov Algorithm

by Md. Rezaul Karim · Nov. 27, 16 · Big Data Zone

In statistical learning, the Kolmogorov-Smirnov test is used to check whether 2 samples follow the same distribution. In this article, I will show how to perform statistical hypothesis test using Spark machine learning API (aka **Spark MLlib**). The **Kolmogorov-Smirnov (KS)** algorithm will be used to reveal the hypothesis testing with a step-by-step by example on the well known **Haberman's survival dataset.**

However, before going into deeper, some background knowledge is mandated for them who are new to the area of statistical machine learning. Furthermore, a technical detail on the hypothesis testing using **Pearson's chi-squared test** algorithm can be found in my previous article here.

What Is Hypothesis Testing?

A **statistical hypothesis** [1] is a hypothesis that can be tested based on an observation modeled via a set of random variables. A **statistical hypothesis test** is, therefore, a method of statistical inference – where, more than one experimental datasets are compared. These datasets can be obtained by sampling and compared against a synthetic dataset from an idealized model [1].

In other words, a hypothesis is proposed and carried out for finding the statistical relationship between two datasets. More technically, the findings are compared as an alternative to an idealized null hypothesis that proposes no relationship between two datasets [1]. However, this comparison is deemed **statistically insignificant** if the relationship between the datasets would be an unlikely realization of the null hypothesis according to a threshold probability. The threshold probability is also called the ***significance level*** [1, 2].

The process of differentiating between the **null hypothesis** and **alternative hypothesis** [3] is aided by identifying two conceptual types of errors – also called type 1 & type 2 errors. Additionally, a parametric limit is specified on how many type 1 errors will be permitted[1, 3]. One of the most common selection techniques is based on either **Akaike information criterion** or **Bayes factor** [1] algorithm. Consequently, hypothesis testing is useful as well powerful tool in statistics to determine:

i) Whether a result is **statistically significant**
ii) Whether a result is **occurred by chance or not**

Because of the confirmatory nature, a statistical hypothesis test is also called **confirmatory data analysis.** Correspondingly**,** it can be contrasted with **exploratory data analysis** that might not have pre-specified hypotheses [2, 3].

Hypothesis Testing and the Confirmatory Data Analysis

In statistics, **exploratory data analysis(EDA),initial data analysis** (IDA) and **confirmatory data analysis (CDA)** are three important aspects to reaching out a statistical

confirmatory data analysis (CDA) are three important aspects to reaching out a statistical hypothesis with necessary data analytical solution. An EDA based data analysis was first promoted by **John Tukey** [4] to encourage the statisticians to explore the data, and possibly formulate hypotheses that could lead to new data collection and experiments [1]. As a result, the **EDA** is an approach for analyzing data to summarize their main characteristics, usually done by visual methods like graphs, plot or interactive visualization [5].

However, EDA is different from that of the IDA. The former is focused on checking assumptions required for model fitting and hypothesis testing. The latter deals with handling missing values and making transformations of variables using popular algorithms like TF-IDF, Vector assembler or Indexer etc. as transformers [1]. Here comes the essentiality of hypothesis testing in statistics to determine whether a result is **statistically significant**. Alternatively, to determine if a particular result has occurred by chance or naturally.

Note that EDA needs to be performed before the CDA. The reason is natural, since if you want to perform the confirmatory test, you will be needing to specify the related statistic like mean, median, mode and standard deviation etc. Therefore, in a nutshell, statistical hypothesis testing is sometimes called CDA.

How Is It Performed?

According to the documentation provided by Wikipedia [1], the following two types of reasoning are considered for performing the hypothesis test:

Initial Hypothesis-based Approach

There is an ***initial research hypothesis*** for which the truth is always ***unknown.*** Now the below steps are typically iterated sequentially:

- The first step is to state the relevant ***null*** and ***alternative hypotheses***
- The second step is to consider the ***statistical assumptions*** (i.e, **statistical independenc**e) being made about the sample in doing the test
- Decide which test is appropriate, and state the relevant **test statistic** say **T**
- From the assumptions, derive the distribution (i.e., most of the cases ***normal distribution***) of the test statistic under the ***null hypothesis***
- Select a ***significance level*** to say defined by **α**which is the probability threshold below for which the ***null hypothesis*** will be ***rejected*** (common values are 5% and 1%)
- Compute the ***observed value*** t_{obs} from the observations of the test statistic **T** *in step 3*
- Now based on the value, decide whether to ***reject*** the ***null hypothesis*** in favor of the ***alternate hypothesis*** or ***accept*** the hypothesis.

Note that, the assumption in step 2 is ***non-trivial*** since ***invalid assumptions*** will tend the results of the test **invalid**.

Alternate Hypothesis-based Approach

Here typically step 1 to 3 are iterated as an **initial hypothesis as** stated above. After that the following steps are performed:

- Compute the observed value t_{obs} from the observations of the test statistic *T in step 4*
- Calculate the ***p-value*** [3], which is the ***probability under the null hypothesis***. More technically, the value of sampling the test statistic at least as extreme as that which was observed
- Reject the **null hypothesis**, in favor of the **alternative hypothesis**, if and only if the ***p-value*** is less than the significance level (the ***selected probability***) threshold.

Now to compute the ***p-value***, here I describe two ***rules of thumbs*** based on several sources [1-3, 5]:

- If the p-value is **p > 0.05** (i.e., 5%), accept your hypothesis. Note that a deviation is small enough to take the hypothesis to an acceptance level. A p-value of 0.6, for example, means that there is a 60% probability of any deviation from expected result. However, this is within the range of an acceptable deviation.
- If the p-value is **p < 0.05**, reject your hypothesis by concluding that some factors other than by chance are operating for the deviation to perfect.

Similarly, a ***p-value*** of 0.01 means that there is only a 1% chance that this deviation is due to chance alone, which means that other factors must be involved that need to be addressed [5]. Note that the ***p-value*** may vary for your case depending upon the data quality, dimension, structure and types etc [2].

In a nutshell, if the **p-Value < 0.05** (significance level), we reject the **null hypothesis** that they are drawn from the same distribution. In other words, p < 0.05 implies **x** (independent variable) and **y** (dependent variable) from different distributions.

Hypothesis Testing and Spark

As already discussed that the current implementation of Spark (i.e., Spark 2.0.2) provides the hypothesis testing facility on the static as well as dynamic (i.e., streaming) data. Consequently**,** the **Spark MLlib** API currently supports **Pearson's chi-squared (χ2) tests** for goodness of fit and independence for batch or static dataset. Secondly, `Spark MLlib` provides a 1-sample, 2-sided implementation of the **KS** test for equality of probability distributions. That to be discussed in the next section.

The third type of hypothesis testing that Spark provides support is using the streaming data. The `Spark MLlib` provides online implementations of some tests to support use cases like the **A/B testing** [8]. Theis test may be performed on **Spark Discrete Streaming** that takes the streaming of type Boolean and double –i.e., **DStream**[(Boolean, Double)] [5]. Where, the first element of each tuple indicates control group (false) or treatment group (true) and the second element is the value of an observation.

Kolmogorov-Smirnov Testing With Spark

By providing the name of a theoretical distribution(currently solely supported for the **normal distribution**) and its parameters, or a function to calculate the **cumulative distribution** according to a given **theoretical distribution**, the user can test the **null hypothesis** that their sample is drawn from that distribution [5].

In the case that the user tests against the normal distribution (`i.e., distName="norm"`), but does not provide distribution parameters, the test initializes to the standard normal distribution and logs an appropriate message. However, while performing the **KS test**, we need to specify the value of the **mean** and **standard deviation** of the dataset as well. Here is how the **p-value** is explained in Spark implementation (as Java-like pseudocode notation):

```
String pValueExplain = null;
if (pValue <= 0.01)
  pValueExplain = "Very strong presumption against null hypothesis"+ nullHypothesis;
else if (0.01 < pValue && pValue <= 0.05)
  pValueExplain= "Strong presumption against null hypothesis"+ ullHypothesis;
else if (0.05 < pValue && pValue <= 0.1)
  pValueExplain= "Low presumption against null hypothesis"+ nullHypothesis;
else
  pValueExplain = "No presumption against null hypothesis" + nullHypothesis;
```

Here we get to know the outcome of the assumption against the **p-value** using the **pValueExplain** variable and its value changing over time across the data objects.

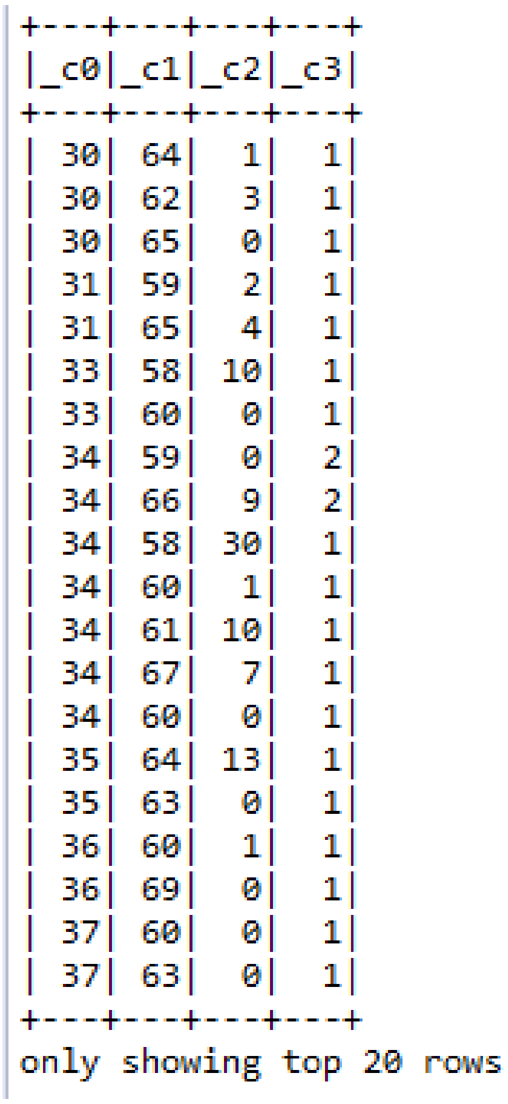
In this example, I will show a step-by-step example how to perform hypothesis testing using **KolmogorovSmirnov** test on **Haberman survival dataset** [6]. Now, before moving into Spark example, let's

KolmogorovSmirnov test on **Haberman survival dataset** [6]. Now, before moving into Spark example, let's explore the dataset first.

Dataset Exploration

Haberman's Survival Dataset(HSD) can be downloaded from the **UCI machine learning dataset repository** at [6]. As per the dataset description at [7], the **HSD** contains cases from a study that was conducted between 1958 and 1970 at the **University of Chicago's Billings Hospital** on the survival of patients who had undergone surgery for **breast cancer**. The number of instances is 306 and the umber of attributes is 4 (including the class attribute). Here is the related information regarding the 4 attributes:

1. Age of patient at time of operation (numerical)
2. Patient's year of operation (numerical)
3. Number of positive axillary nodes detected (numerical)
4. Survival status (class attribute)
   1 = the patient survived 5 years or longer
   2 = the patient died within 5 year



```
+---+---+---+---+
|_c0|_c1|_c2|_c3|
+---+---+---+---+
| 30| 64|  1|  1|
| 30| 62|  3|  1|
| 30| 65|  0|  1|
| 31| 59|  2|  1|
| 31| 65|  4|  1|
| 33| 58| 10|  1|
| 33| 60|  0|  1|
| 34| 59|  0|  2|
| 34| 66|  9|  2|
| 34| 58| 30|  1|
| 34| 60|  1|  1|
| 34| 61| 10|  1|
| 34| 67|  7|  1|
| 34| 60|  0|  1|
| 35| 64| 13|  1|
| 35| 63|  0|  1|
| 36| 60|  1|  1|
| 36| 69|  0|  1|
| 37| 60|  0|  1|
| 37| 63|  0|  1|
+---+---+---+---+
only showing top 20 rows
```

Figure 1 shows the top-level view of the Haberman's survival dataset.

Spark MLlib Solution for the KolmogorovSmirnov Test

The Spark based solution demonstrated here has 6 steps from data loading, parsing, test set preparation, testing and finally result from analysis:

Step-1: Load required packages and APIs

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaDoubleRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.mllib.stat.Statistics;
import org.apache.spark.mllib.stat.test.KolmogorovSmirnovTestResult;
```

Step-2: Create Spark context

```
SparkConf conf = new SparkConf().setAppName("JavaHypothesisTestingKolmogorovSmirnovTestExample").setMa
 JavaSparkContext jsc = new JavaSparkContext(conf);
```

Step-3: Load and parse the dataset to create an ArrayList of Double

```
        String path = "input/haberman.data";//Paht of the data source
        BufferedReader br = new BufferedReader(new FileReader(path));
        String line = null;
        List<Double> myList = new ArrayList<Double>(); // ArrayList of Double
        while ((line = br.readLine()) != null) { //Reading the file line by line
            String[] tokens = line.split(",");
            myList.add(Double.parseDouble(tokens[0])); //Pushing the 1st value
            myList.add(Double.parseDouble(tokens[1])); //Pushing the 2nd value
            myList.add(Double.parseDouble(tokens[2])); //Pushing the 3rd value
            myList.add(Double.parseDouble(tokens[3])); //Pushing the 4th value
            }
```

Step-4: Preparing the test data

It is to be noted that the **KS test** algorithm accepts and performs the statistical analysis on JavaDoubleRDDs as a **normal distribution**. Consequently, a JavaDoubleRDD can be generated by parallelizing the above list as ArrayList. Therefore, first let's convert the above list as list of double values as follows:

```
Double [] list = myList.toArray(new Double[myList.size()]);
```

Now that we have the list of doubles and we can prepare the JavaDoubleRDD as follows:

```
JavaDoubleRDD data = jsc.parallelizeDoubles(Arrays.asList(list));
```

Step-5: Perform the Kolmogorov-Smirnov Test

Perform the **hypothesis** test using the KolmogorovSmirnov algorithm on the test dataset we prepared in the previous step. Also, we need to explicitly specify the distribution as normal. Further, we also need to specify the mean and the standard deviation of the test set we prepared.

```
KolmogorovSmirnovTestResult testResult = Statistics.kolmogorovSmirnovTest(data, "norm", 35.0, 1.5);
```

Note that here the distribution of the test data (i.e., training data in this case) is normal. The 3rd and 4th parameters are mean and standard deviation (SD) respectively.

The requirement of these parameters signifies why an exploratory analysis is needed to be performed before the confirmatory data analysis. For the brevity and simplicity, we assume the mean as 35.0 and SD as 1.5. Readers should calculate the arithmetic mean and the standard deviation of the distribution to get more accurate result.

Step-6: Print the test results

Note the summary of the test including the p-value, test statistic, and null hypothesis if our p-value indicates significance, we can reject the null hypothesis that goes as follows:

```
System.out.println(testResult);
```

Fort the above test configuration, you should receive the below like results:

Kolmogorov-Smirnov test summary:
degrees of freedom = 0
statistic = 0.49957093966680316
pValue = 2.747246874434 95E-11
The Very strong presumption against null hypothesis: Sample follows the theoretical distribution.

Conclusion

From the above result, we got a larger p-value. Therefore, the initial hypothesis can be accepted. However, for an in-depth analysis on the p-value, readers, are suggested to read the documentation provided at [1-3]. Moreover, selecting the proper dataset and doing the hypothesis test prior applying the hyperparameter tuning are recommended too. Readers are also suggested to find more on statistical learning with Spark MLlib at [5].

The maven friendly **pom.xml** file, associated source codes and **Haberman's survival dataset** can be downloaded from my **GitHub** repository here at [8]. In my next article, I will show how to perform the A/B like testing on real-time streaming data.

Like This Article? Read More From DZone

Hypothesis Test Using Pearson's Chi-squared Test Algorithm

Random Forest as a Regressor: A Spark-based Solution

Random Forest as a Classifier: A Spark-based Solution

Free DZone Refcard
Data Warehousing

Topics: SPARK 2.0.0 , BIG DATA ANALTICS , STATISTICAL LEARNING , MACHINE LEARNING , BIG DATA