Hypothesis Test Using Pearson’s Chi-squared Test Algorithm

by Md. Rezaul Karim · Nov. 25, 16 · Big Data Zone

In this article, I will show how to perform statistical hypothesis test using Spark machine learning API (aka **Spark MLlib**). The will be used to reveal the hypothesis testing with a step-by-step by example on the well-known **Wisconsin Breast Cancer Prognosis (WBCP) dataset.**

In statistics, the Chi-squared test can be used to test if two categorical variables are dependent, by means of a contingency table. 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 **Kolmogorov-Smirnov (KS) 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].

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:

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].

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

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.

However, in this article, I will show hypothesis testing using the **Pearson's chi-squared (PCS) test** algorithm implemented in Spark. However, the **Kolmogorov-Smirnov (KS) test** will be discussed in my next article.

Goodness of Fit Test Results Using the Pearson's Chi-squared Test

In this test, the input data types determine whether the goodness of fit or the independence test is conducted. The goodness of fit test requires an input type of Vector; whereas, the independence test requires a Matrix as input. Spark MLlib also supports the input type RDD [LabeledPoint] to enable feature selection via **Chi-square independence tests** [3]. The probability of obtaining a test statistic result at least as extreme as the one that was actually observed, assuming that the **null hypothesis** is true [3].

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"+ $nullHypothesis;
else if (0.05 < pValue && pValue <= 0.1)
  pValueExplain= "Low presumption against null hypothesis"+ $nullHypothesis;
else
  pValueExplain = "No presumption against null hypothesis" + $nullHypothesis;
```

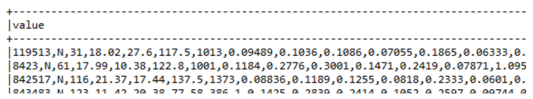
Here we get to know the outcome of the assumption against the **p-value** using the **pValueExplain** variable and its value changing over time and data objects.

Dataset Exploration

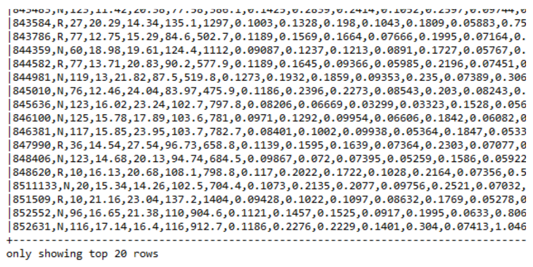
At first, we need to prepare dense vector from the categorical dataset like **Wisconsin Breast Cancer Prognosis** (WBCP) dataset. The details of the attributes found in **WBCP** dataset at [4] are:

- ID number
- Outcome (R = **recurrent**, N = **non-recurrent**)
- Time (recurrence time if field 2 => R, disease-free time if field 2 => N)
- 4 to 33: Ten real-valued features are computed for each cell nucleus: Radius, Texture, Perimeter, Area, Smoothness, Compactness, Concavity, Concave points, Symmetry and Fractal dimension. The thirty-four is Tumour size and the thirty-five is the Lymph node status as follows:
- Tumour size - diameter of the excised tumor in centimeters
- Lymph node status - the number of positive axillary lymph nodes.

These values are observed at the time of surgery from the year 1988 to 1995 and out of the 198 instances, 151 are non-recurring (N) and only 47 are recurring (R). Figure 1 shows a snapshot of the dataset. However, real cancer diagnosis and prognosis datasets like **COSMIC**, **ICGC** and **TCGA** nowadays contain many other features and fields in structured or unstructured ways [6].



```
+---------------------------------------------------------------------------
|value
+---------------------------------------------------------------------------
|119513,N,31,18.02,27.6,117.5,1013,0.09489,0.1036,0.1086,0.07055,0.1865,0.06333,0.
|8423,N,61,17.99,10.38,122.8,1001,0.1184,0.2776,0.3001,0.1471,0.2419,0.07871,1.095
|842517,N,116,21.37,17.44,137.5,1373,0.08836,0.1189,0.1255,0.0818,0.2333,0.0601,0.
|843483,N,123,11.42,20.38,77.58,386.1,0.1425,0.2839,0.2414,0.1052,0.2597,0.09744,0.
```



```
|843483,N,123,11.42,20.38,77.58,386.1,0.1425,0.2839,0.2414,0.1052,0.2597,0.09744,0
|843584,R,27,20.29,14.34,135.1,1297,0.1003,0.1328,0.198,0.1043,0.1809,0.05883,0.75
|843786,R,77,12.75,15.29,84.6,502.7,0.1189,0.1569,0.1664,0.07666,0.1995,0.07164,0.
|844359,N,60,18.98,19.61,124.4,1112,0.09087,0.1237,0.1213,0.0891,0.1727,0.05767,0.
|844582,R,77,13.71,20.83,90.2,577.9,0.1189,0.1645,0.09366,0.05985,0.2196,0.07451,0
|844981,N,119,13,21.82,87.5,519.8,0.1273,0.1932,0.1859,0.09353,0.235,0.07389,0.306
|845010,N,76,12.46,24.04,83.97,475.9,0.1186,0.2396,0.2273,0.08543,0.203,0.08243,0.
|845636,N,123,16.02,23.24,102.7,797.8,0.08206,0.06669,0.03299,0.03323,0.1528,0.056
|846100,N,125,15.78,17.89,103.6,781,0.0971,0.1292,0.09954,0.06606,0.1842,0.06082,0
|846381,N,117,15.85,23.95,103.7,782.7,0.08401,0.1002,0.09938,0.05364,0.1847,0.0533
|847990,R,36,14.54,27.54,96.73,658.8,0.1139,0.1595,0.1639,0.07364,0.2303,0.07077,0
|848406,N,123,14.68,20.13,94.74,684.5,0.09867,0.072,0.07395,0.05259,0.1586,0.05922
|848620,R,10,16.13,20.68,108.1,798.8,0.117,0.2022,0.1722,0.1028,0.2164,0.07356,0.5
|8511133,N,20,15.34,14.26,102.5,704.4,0.1073,0.2135,0.2077,0.09756,0.2521,0.07032,
|851509,R,10,21.16,23.04,137.2,1404,0.09428,0.1022,0.1097,0.08632,0.1769,0.05278,0
|852552,N,96,16.65,21.38,110,904.6,0.1121,0.1457,0.1525,0.0917,0.1995,0.0633,0.806
|852631,N,116,17.14,16.4,116,912.7,0.1186,0.2276,0.2229,0.1401,0.304,0.07413,1.046
+----------------------------------------------------------------------------------
only showing top 20 rows
```

Figure 1: Snapshot of the breast cancer prognosis data (partially shown)

Pearson’s Chi-squared Test With Spark MLlib

In this example, I will show 3 tests to explain how the hypothesis testing works using the PCS test:

- Goodness of fit result on dense vector created from the breast cancer prognosis dataset,
- Independence tests for a randomly created matrix
- The independence test on a contingency table from the cancer dataset.

Step-1: Load required packages

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.mllib.linalg.DenseVector;
import org.apache.spark.mllib.linalg.Matrices;
import org.apache.spark.mllib.linalg.Matrix;
import org.apache.spark.mllib.linalg.Vector;
import org.apache.spark.mllib.regression.LabeledPoint;
import org.apache.spark.mllib.stat.Statistics;
import org.apache.spark.mllib.stat.test.ChiSqTestResult;
import org.apache.spark.rdd.RDD;
import org.apache.spark.sql.SparkSession;
import com.example.SparkSession.UtilityForSparkSession;
```

Step-2: Create a Spark session

```java
static SparkSession spark = UtilityForSparkSession.mySession();
```

The implementation of the UtilityForSparkSession class is as follows:

```java
public class UtilityForSparkSession {
    public static SparkSession mySession() {
    SparkSession spark = SparkSession
               .builder()
               .appName("JavaHypothesisTestingOnBreastCancerData ")
               .master("local[*]")
              .config("spark.sql.warehouse.dir", "E:/Exp/")
              .getOrCreate();
        return spark;
    }
}
```

Step-3: Perform the goodness of fit test

The following line of codes collects the vectors that we created using the myVector() method:

```
Vector v = myVector();
```

Where the implementation of the myVector() method goes as follows:

```
public static Vector myVector() throws NumberFormatException, IOException {
BufferedReader br = new BufferedReader(new FileReader(path));
        String line = null;
        Vector v = null;
        while ((line = br.readLine()) != null) {
            String[] tokens = line.split(",");
            double[] features = new double[30];
            for (int i = 2; i < features.length; i++) {
                features[i-2] =
                       Double.parseDouble(tokens[i]);
            }
            v = new DenseVector(features);
        }
        return v;
    }
```

Now let’s compute the goodness of the fit. Note, if a second vector to test is not supplied as a parameter, the test run occurs against a uniform distribution automatically.

```
ChiSqTestResult goodnessOfFitTestResult = Statistics.chiSqTest(v);
```

Now let’s print the result of the goodness using:

```
System.out.println(goodnessOfFitTestResult + "\n");
```

Chi-squared test summary:

method: Pearson

degrees of freedom = 29

statistic = 24818.028940912533

pValue = 0.0

Very strong presumption against null hypothesis: observed follows the same distribution as expected

Since the **p-value** is very low (i.e., less than 5%) that is insignificance and consequently, we cannot accept the hypothesis based on the data.

Step-4: Independence test on contingency matrix

Before performing this test, let’s create a contingency 4x3 matrix randomly. Here, the matrix goes as follows: ((1.0, 3.0, 5.0, 2.0), (4.0, 6.0, 1.0, 3.5), (6.9, 8.9, 10.5, 12.6))

```
Matrix mat = Matrices.dense(4, 3, new double[] { 1.0, 3.0, 5.0, 2.0, 4.0, 6.0, 1.0, 3.5, 6.9, 8.9, 10.
```

Now let’s conduct the Pearson's independence test on the input contingency matrix:

```
ChiSqTestResult independenceTestResult = Statistics.chiSqTest(mat);
```

To evaluate the test result, let's print the summary of the test including the **p-value**, **degrees of freedom** and the related statistic as follows:

```
System.out.println(independenceTestResult + "\n");
System.out.println(independenceTestResult + "\n");
```

We got the following statistic as summarized follows:

Chi-squared test summary:
method: Pearson
degrees of freedom = 6
statistic = 6.911459343085576
pValue = 0.3291131185252161
No presumption against null hypothesis: the occurrence of the outcomes is statistically independent.

Since the **p-value** is significantly higher (i.e., 32%), we can now accept the hypothesis based on the contingency matrix. However, the size of the above matrix is very small which is clearly seen. Therefore, even though we have the required p-value, we should not accept the hypothesis but let's try the hypothesis test using the **independence test** on **contingency table** instead with a larger dimension.

Step-5: Independence test on contingency table
At first, let's create a contingency table by means of RDDs from the cancer dataset as follows:

```
static String path = "input/wpbc.data";
RDD<String> lines = spark.sparkContext().textFile(path, 2);
JavaRDD<LabeledPoint> linesRDD = lines.toJavaRDD().map(new Function<String, LabeledPoint>() {
        public LabeledPoint call(String lines) {
        String[] tokens = lines.split(",");
        double[] features = new double[30]; //30 features
        for (int i = 2; i < features.length; i++) {
        features[i - 2] = Double.parseDouble(tokens[i]); //Exclude first two column
                        }
        Vector v = new DenseVector(features);
        if (tokens[1].equals("R")) {
        return new LabeledPoint(1.0, v); // recurrent
            } else {
        return new LabeledPoint(0.0, v); // non-recurrent
                }
            }
        });
```

Well, we have constructed a contingency table from the raw (feature, label) pairs. Now let's conduct the test as **ChiSquaredTestResult** for every feature against the label.

```
ChiSqTestResult[] featureTestResults = Statistics.chiSqTest(linesRDD.rdd());
```

Now let's observe the test result against each column (i.e. for each 30 feature point) using the following code segment:

```
int i = 1;
for (ChiSqTestResult result : featureTestResults) {
System.out.println("Column " + i + ":");
System.out.println(result + "\n");
i++;
}
```

Column 1:
Chi-squared test summary:
method: Pearson
degrees of freedom = 94
statistic = 85.96752752672163
pValue = 0.7103468748120438

```
No presumption against null hypothesis: the occurrence of the outcomes is statistically independent.

Column 2:
Chi-squared test summary:
method: Pearson
degrees of freedom = 176
statistic = 180.50725658729024
pValue = 0.3921633980422585
No presumption against null hypothesis: the occurrence of the outcomes is statistically independent.

Column 3:
Chi-squared test summary:
method: Pearson
degrees of freedom = 192
statistic = 197.99999999999957
pValue = 0.3680602522293952
No presumption against null hypothesis: the occurrence of the outcomes is statistically independent.
.
.
Column 30:
Chi squared test summary:
method: pearson
degrees of freedom = 0
statistic = 0.0
pValue = 1.0
No presumption against null hypothesis: the occurrence of the outcomes is statistically independent.
```

Conclusion

From the above result, we can see that for some feature point (i.e. column) we got larger **p-value** compared to others. However, results vary on different columns. Readers are, therefore, suggested to select the proper dataset and do the hypothesis test prior applying the **hyperparameter tuning**. There is no concrete example in this regard since the result may vary against datasets you have. Readers are also suggested to find more on statistical learning with Spark MLlib at [3].

In my next article, I will show how to perform hypothesis testing using the **Kolmogorov-Smirnov Test**. The maven friendly **pom.xml** file, associated source codes and **breast cancer prognosis dataset** can be downloaded from my **GitHub** repository here.