TutorialKart

Classification using Naive Bayes in Apache Spark MLlib with Java

Apache Spark MLlib – Classification using Naive Bayes

Classification is a task of identifying the features of an entity and classifying the entity to one of the predefined classes/categories based on the previous knowledge.

Naive Bayes is one of the simplest methods used for classification. Naive Bayes Classifier could be built in scenarios where problem instances (/ examples / data set / training data) could be represented as feature vectors. And the distinctive feature of Naive Bayes is : it considers that features independently play a part in deciding the category of the problem instance i.e., Naive Bayes does not care about the correlation between features if present any. Despite the fact that many other classifiers beat out Naive Bayes, it is still sustaining in the machine learning community because it requires relatively small number of training data for estimating the parameters required for classification.

In this Apache Spark Tutorial, we shall learn to classify items using Naive Bayes Algorithm of Apache Spark MLlib in Java Programming Language.

Steps for Classification using Naive Bayes

Following is a step by step process to build a classifier using Naive Bayes algorithm of MLLib. You may setup Java Project with Apache Spark and follow the steps.

1. Configure Spark.

```
SparkConf sparkConf = new SparkConf().setAppName("NaiveBayesClassifierExample");
```

2. Start a spark context.

```
JavaSparkContext jsc = new JavaSparkContext(sparkConf);
```

3. Load Data and Split the data to be used for training and testing. The data file used in this example is present in the folder “**data**” in “**apache spark**“, downloaded from official website.

```
// provide path to data transformed as [feature vectors]
```

```
String path = "data/mllib/sample_libsvm_data.txt";
JavaRDD inputData = MLUtils.loadLibSVMFile(jsc.sc(), path).toJavaRDD();

// split the data for training (60%) and testing (40%)
JavaRDD[] tmp = inputData.randomSplit(new double[]{0.6, 0.4});
JavaRDD training = tmp[0]; // training set
JavaRDD test = tmp[1]; // test set
```

4. Train a Naive Bayes model.

```
NaiveBayesModel model = NaiveBayes.train(training.rdd(), 1.0);
```

5. Use the model to predict on the test data, and calculate accuracy.

```
JavaPairRDD<Double, Double> predictionAndLabel =
                test.mapToPair(p -> new Tuple2<>(model.predict(p.features()), p.label())
// calculate the accuracy
double accuracy =
                predictionAndLabel.filter(pl -> pl._1().equals(pl._2())).count() / (doub
```

6. Save the trained classifier model to local for future use.

```
model.save(jsc.sc(), "target/tmp/myNaiveBayesModel");
```

7. Stop the spark context.

```
jsc.stop();
```

Following is the complete Java program.

NaiveBayesClassifierExample.java

```
import scala.Tuple2;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.mllib.classification.NaiveBayes;
import org.apache.spark.mllib.classification.NaiveBayesModel;
import org.apache.spark.mllib.regression.LabeledPoint;
import org.apache.spark.mllib.util.MLUtils;
import org.apache.spark.SparkConf;

public class NaiveBayesClassifierExample {

    public static void main(String[] args) {

        // configure spark
        SparkConf sparkConf = new SparkConf().setAppName("JavaNaiveBayesExample")
                                             setMaster("local[2]").set("spark.executor.memor
```

```
                                        .setMaster( local[2] ).set( spark.executor.memor
        // start a spark context
        JavaSparkContext jsc = new JavaSparkContext(sparkConf);

        // provide path to data transformed as [feature vectors]
        String path = "data/mllib/sample_libsvm_data.txt";
        JavaRDD inputData = MLUtils.loadLibSVMFile(jsc.sc(), path).toJavaRDD();

        // split the data for training (60%) and testing (40%)
        JavaRDD[] tmp = inputData.randomSplit(new double[]{0.6, 0.4});
        JavaRDD training = tmp[0]; // training set
        JavaRDD test = tmp[1]; // test set

        // Train a Naive Bayes model
        NaiveBayesModel model = NaiveBayes.train(training.rdd(), 1.0);

        // Predict for the test data using the model trained
        JavaPairRDD<Double, Double> predictionAndLabel =
                test.mapToPair(p -> new Tuple2<>(model.predict(p.features()), p.label())
        // calculate the accuracy
        double accuracy =
                predictionAndLabel.filter(pl -> pl._1().equals(pl._2())).count() / (doub

        System.out.println("Accuracy is : "+accuracy);

        // Save model to local for future use
        model.save(jsc.sc(), "target/tmp/myNaiveBayesModel");

        // stop the spark context
        jsc.stop();
    }
}
```

Output

```
Accuracy is : 0.975609756097561
```

Conclusion

In this Apache Spark Tutorial, we learned how to solve Classification problem using Naive Bayes algorithm in Apache Spark MLlib.

Learn Apache Spark

- Apache Spark Tutorial
- Install Spark on Ubuntu
- Install Spark on Mac OS
- Scala Spark Shell - Example

- Python Spark Shell - PySpark
- Setup Java Project with Spark
- Spark Scala Application - WordCount Example
- Spark Python Application
- Spark DAG & Physical Execution Plan
- Setup Spark Cluster
- Configure Spark Ecosystem
- Configure Spark Application
- Spark Cluster Managers

Spark RDD

- Spark RDD
- Spark RDD - Print Contents of RDD
- Spark RDD - foreach
- Spark RDD - Create RDD
- Spark Parallelize
- Spark RDD - Read Text File to RDD
- Spark RDD - Read Multiple Text Files to Single RDD
- Spark RDD - Read JSON File to RDD
- Spark RDD - Containing Custom Class Objects
- Spark RDD - Map
- Spark RDD - FlatMap
- Spark RDD - Filter
- Spark RDD - Distinct
- Spark RDD - Reduce

Spark Dataseet

- Spark - Read JSON file to Dataset
- Spark - Write Dataset to JSON file
- Spark - Add new Column to Dataset
- Spark - Concatenate Datasets

Spark MLlib (Machine Learning Library)

Spark SQL

Spark Others