Configurando via Maven
propriedades do projeto

```
<properties>
	<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
	<jdk.version>1.8</jdk.version>
	<spark.version>2.3.0</spark.version>
</properties>
```


Configurando via Maven
dependências

```
<dependencies>
	<dependency>
	   <groupId>org.apache.spark</groupId>
	   <artifactId>spark-core_2.11</artifactId>
	   <version>${spark.version}</version>
	</dependency>
```

- repetir dependências do Spark com artifactId
  - spark-sql_2.11 – spark-mllib_2.11 – spark-graphx_2.11
  - spark-yarn_2.11 – spark-network-shuffle_2.11
  - spark-streaming-flume_2.11

```
	<dependency>
	   <groupId>com.databricks</groupId>
	   <artifactId>spark-csv_2.11</artifactId>
	   <version>1.3.0</version>
	</dependency>
```

Configurando via Maven
dependências

```
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
</dependencies>
```


Configurando via Maven
plugins

```
<build>
  <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-eclipse-plugin</artifactId>
      <version>2.9</version>
      <configuration>
        <downloadSources>true</downloadSources>
        <downloadJavadocs>false</downloadJavadocs>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.5.1</version>
      <configuration>
        <source>${jdk.version}</source>
        <target>${jdk.version}</target>
      </configuration>
    </plugin>
```

Configurando via Maven
plugins

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>
    <version>2.4.3</version>
    <configuration>
       <shadeTestJar>true</shadeTestJar>
    </configuration>
 </plugin>
```


Configurando via Maven
plugins

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>2.4.1</version>
    <configuration>
       <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
       </descriptorRefs>
       <archive>
          <manifest>
             <mainClass>dl.App2_MLP_Spark</mainClass>
          </manifest>
       </archive>
    </configuration>
```

Configurando via Maven
plugins

```
            <executions>
               <execution>
                  <id>make-assembly</id>
                  <phase>package</phase>
                  <goals>
                     <goal>single</goal>
                  </goals>
               </execution>
            </executions>
         </plugin>
      </plugins>
   </build>
```


Importações

java.io
- Serializable

java.util
- HashMap
- Map

Importações

org.apache.spark
- api.java.JavaRDD
- ml
- classification
  - MultilayerPerceptronClassificationModel
  - MultilayerPerceptronClassifier
- evaluation.MulticlassClassificationEvaluator
- mllib
- feature.StandardScalerModel
- linalg
  - Vector
  - Vectors
- stat
  - MultivariateStatisticalSummary
  - Statistics
- util.MLUtils

Importações

org.apache.spark.sql
- *
- api.java.UDF1
- types.DataTypes

scala
- Option
- Some
- Tuple2
- Tuple3

static org.apache.spark.sql.functions
- callUDF
- col

Criando Classe Auxiliar : Par de Vetores

para armazenar rótulo survived e suas características normalizadas

```
public static class VectorPair implements Serializable {
    private double label;
    private Vector features;

    public VectorPair(double label, Vector features) {
        this.label = label;
        this.features = features;
    }
    public VectorPair() {}
    public Vector getFeatures() { return this.features; }
    public void setFeatures(Vector features) { this.features = features; }
    public double getLabel() { return this.label; }
    public void setLabel(double label) { this.label = label; }
}
```


Dados da Classe

```
private static SparkSession spark;
private static MultivariateStatisticalSummary summary = null;
private static double mean_fare_column;
private static double mean_age_column;
private static StandardScalerModel scaler;
private static Dataset<Row> trainingData;
private static Dataset<Row> validationData;
```

Main

```
public static void main(String[] args) {
    spark = createSparkSession();
    spark.sparkContext().setLogLevel("ERROR");
        // para reduzir mensagens de INFO
    Dataset<Row> trainingDataSet = doETL_TrainingData();
    Dataset<Row> scaled_trainingDataset
            = scaleTrainingData(trainingDataSet);
    MultilayerPerceptronClassificationModel model
            = trainNetwork(scaled_trainingDataset);
    evaluateNetwork(model);
    Dataset<Row> resultDF = testNetwork(model);
    resultDF.write().format("com.databricks.spark.csv")
                .option("header", true).save("result/result.csv");
}
```


Criando uma Sessão Spark

```
private static SparkSession createSparkSession() {
    return SparkSession
            .builder()
            .master("local[*]")
            .config("spark.sql.warehouse.dir", "spark")
            .appName("App2_MLP_Spark")
            .getOrCreate();
}
```

Leitura dos Dados de Treinamento

```
private static Dataset<Row> readTrainingData() {
      Dataset<Row> input_data = spark.sqlContext()
            .read()
            .format("com.databricks.spark.csv")
            .option("header", "true")
            .option("inferSchema", "true")
            .load("data/train.csv");
      input_data.show();
   return  input_data;
}
```


UDFs para Conversão para Valores Numéricos

Portas de Embarque – Sexo

```
private static UDF1<String,Option<Integer>> normEmbarked=(String d) ->
   {
      if (null == d return Option.apply(null);
      else if (d.equals("S")) return Some.apply(0);
            else if (d.equals("C")) return Some.apply(1);
                  else return Some.apply(2);
      }
}

private static UDF1<String, Option<Integer>> normSex = (String d) -> {
     if (null == d) return Option.apply(null);
     else if (d.equals("male")) return Some.apply(0);
           eles return Some.apply(1);
      };
}
```

ETL dos Dados de Treinamento
lendo dados de treinamento

```
private static Dataset<Row> doETL_TrainingData() {

   Dataset<Row> input_data = readTrainingData();
```


ETL dos Dados de Treinamento
seleção das colunas relevantes – conversão das não numéricas

```
   spark.sqlContext().udf().register
         ("normSex", normSex, DataTypes.IntegerType);
   spark.sqlContext().udf().register
         ("normEmbarked", normEmbarked, DataTypes.IntegerType);

   Dataset<Row> numeric_relevant_columns = input_data.select(
         col("Survived"),
         col("Fare"),
         callUDF("normSex", col("Sex")).alias("Sex"),
         col("Age"),
         col("Pclass"),
         col("Parch"),
         col("SibSp"),
         callUDF("normEmbarked",col("Embarked"))
                  .alias("Embarked"));
   numeric_relevant_columns.show();
```

ETL dos Dados de Treinamento

cálculos de valores médios de colunas com valores nulos

```
// convert DataSet into JavaRDD to calculate mean values of columns
JavaRDD<Vector> data_statistic
        = numeric_relevant_columns.rdd().toJavaRDD().map(row ->
Vectors.dense(
        row.<Double>getAs("Fare"),
        row.isNullAt(3) ? 0d : row.<Double>getAs("Age")));

summary = Statistics.colStats(data_statistic.rdd());

mean_fare_column = summary.mean().apply(0);
mean_age_column = summary.mean().apply(1);
```


ETL dos Dados de Treinamento

UDFs para substituição de valores nulos por valores médios

```
UDF1<String, Option<Double>> normFare = (String d) -> {
    if (null == d) return Some.apply(mean_fare_column);
    else return Some.apply(Double.parseDouble(d));
};

UDF1<String, Option<Double>> normAge = (String d) -> {
    if (null == d) return Some.apply(mean_age_column);
    else return Some.apply(Double.parseDouble(d));
};
```

ETL dos Dados de Treinamento

substituindo valores nulos por valores médios

```
spark.sqlContext().udf().register
        ("normFare", normFare, DataTypes.DoubleType);
spark.sqlContext().udf()
      .register("normAge", normAge, DataTypes.DoubleType);
Dataset<Row> without_nulls_numeric_relevant_columns
  = numeric_relevant_columns.select(
        col("Survived"),
        callUDF("normFare", col("Fare").cast("string")).alias("Fare"),
        col("Sex"),
        callUDF("normAge", col("Age").cast("string")).alias("Age"),
        col("Pclass"),
        col("Parch"),
        col("SibSp"),
        col("Embarked"));
   without_nulls_numeric_relevant_columns.show();
   return without_nulls_numeric_relevant_columns;
}
```


Funções Auxiliares para Normalizar Colunas

```
private static Tuple3<Double, Double, Double> flattenPclass
      (double value) {
   Tuple3<Double, Double, Double> result;
   if (value == 1) result = new Tuple3<>(1d, 0d, 0d);
   else if (value == 2) result = new Tuple3<>(0d, 1d, 0d);
   else result =new Tuple3<>(0d, 0d, 1d);
   return result;
}

private static Tuple3<Double, Double, Double> flattenEmbarked
      (double value) {
   Tuple3<Double, Double, Double> result;
   if (value == 0) result = new Tuple3<>(1d, 0d, 0d);
   else if (value == 1) result = new Tuple3<>(0d, 1d, 0d);
   else result = new Tuple3<>(0d, 0d, 1d);
   return result;
}
```

Funções Auxiliares para Normalizar Colunas

```
private static Tuple2<Double, Double> flattenSex(double value) {
   Tuple2<Double, Double> result;
   if (value == 0) result = new Tuple2<>(1d, 0d);
   else result = new Tuple2<>(0d, 1d);
   return result;
}
```


Normalizando Colunas

```
public static Vector getScaledVector
   (double fare, double age, double pclass,  double sex,
    double embarked, StandardScalerModel scaler) {
      Vector scaledContinous
            = scaler.transform(Vectors.dense(fare, age));
      Tuple3<Double, Double, Double> pclassFlat = flattenPclass(pclass);
      Tuple3<Double, Double, Double> embarkedFlat
            = flattenEmbarked(embarked);
      Tuple2<Double, Double> sexFlat = flattenSex(sex);
       return Vectors.dense(scaledContinous.apply(0),
            scaledContinous.apply(1),
            sexFlat._1(), sexFlat._2(),
            pclassFlat._1(), pclassFlat._2(), pclassFlat._3(),
            embarkedFlat._1(),  embarkedFlat._2(), embarkedFlat._3());
 }
```

Normalizando Dados de Treinamento
calculando desvio padrão e média

```
private static Dataset<Row> scaleTrainingData
      (Dataset<Row> trainingDataSet) {
    Vector standard_deviation
      = Vectors.dense(Math.sqrt(summary.variance().apply(0)),
                      Math.sqrt(summary.variance().apply(1)));
    Vector mean = Vectors.dense(summary.mean().apply(0),
                                summary.mean().apply(1));
    scaler = new StandardScalerModel(standard_deviation, mean);
```


Normalizando Dados de Treinamento
codificadores para valores numéricos

```
Encoder<Integer> integerEncoder = Encoders.INT();
Encoder<Double> doubleEncoder = Encoders.DOUBLE();
Encoders.BINARY();
Encoder<Vector> vectorEncoder = Encoders.kryo(Vector.class);
Encoders.tuple(integerEncoder, vectorEncoder);
Encoders.tuple(doubleEncoder, vectorEncoder);
```

Normalizando Dados de Treinamento

par de vetores para rótulo e características normalizadas

```
JavaRDD<VectorPair> scaledRDD
            = trainingDataSet.toJavaRDD().map(row -> {
      VectorPair vectorPair = new VectorPair();
      vectorPair.setLabel(new Double(row.<Integer>getAs("Survived")));
      vectorPair.setFeatures(getScaledVector(
            row.<Double>getAs("Fare"),
            row.<Double>getAs("Age"),
            row.<Integer>getAs("Pclass"),
            row.<Integer>getAs("Sex"),
            row.isNullAt(7) ? 0d : row.<Integer>getAs("Embarked"), scaler));
      return vectorPair;
 });
```


Normalizando Dados de Treinamento

convertendo RDD do Vetor para Vetor Mlib

```
Dataset<Row> scaled_trainingDataset
        = spark.createDataFrame(scaledRDD, VectorPair.class);
scaled_trainingDataset.show();

return scaled_trainingDataset;
}
```

Treinando a Rede

convertendo Vetor Mlib para Vetor ML para utilizar MLP do Spark
dividindo dados entre treinamento e validação

```
private static MultilayerPerceptronClassificationModel
        trainNetwork(Dataset<Row> scaled_trainingDataset) {

    Dataset<Row> scaledData2
        = MLUtils.convertVectorColumnsToML(scaled_trainingDataset);

    Dataset<Row> data = scaledData2.toDF("features", "label");
    Dataset<Row>[] datasets
       = data.randomSplit(new double[]{0.80, 0.20}, 12345L);
    trainingData = datasets[0];
    validationData = datasets[1];
```


Treinando a Rede

configurando a rede e gerando o modelo da rede

```
    int[] layers = new int[] {10, 16, 32, 2};
    MultilayerPerceptronClassifier mlp
            = new MultilayerPerceptronClassifier()
           .setLayers(layers)
           .setBlockSize(128)
           .setSeed(1234L)
           .setTol(1E-8)
           .setMaxIter(1000);
    MultilayerPerceptronClassificationModel model
            =  mlp.fit(trainingData);
    return model;
}
```

Avaliando a Rede
gerando os avaliadores

```
private static void evaluateNetwork
        (MultilayerPerceptronClassificationModel model) {
    Dataset<Row> predictions = model.transform(validationData);
    predictions.show();

    MulticlassClassificationEvaluator evaluator
          = new MulticlassClassificationEvaluator()
          .setLabelCol("label").setPredictionCol("prediction");
    MulticlassClassificationEvaluator evaluator1
          = evaluator.setMetricName("accuracy");
    MulticlassClassificationEvaluator evaluator2
          = evaluator.setMetricName("weightedPrecision");
    MulticlassClassificationEvaluator evaluator3
          = evaluator.setMetricName("weightedRecall");
    MulticlassClassificationEvaluator evaluator4
          = evaluator.setMetricName("f1");
```


Avaliando a Rede
calculando e visualizando as métricas de avaliação

```
    double accuracy = evaluator1.evaluate(predictions);
    double precision = evaluator2.evaluate(predictions);
    double recall = evaluator3.evaluate(predictions);
    double f1 = evaluator4.evaluate(predictions);

    System.out.println("Accuracy = " + accuracy);
    System.out.println("Precision = " + precision);
    System.out.println("Recall = " + recall);
    System.out.println("F1 = " + f1);
    System.out.println("Test Error = " + (1 - accuracy));
}
```

Gerando Dados de Teste

tratando colunas não numéricas – lendo dados de teste

```
private static Dataset<Row> getTestData() {
   spark.sqlContext().udf().register
         ("normSex", normSex, DataTypes.IntegerType);
   spark.sqlContext().udf().register
         ("normEmbarked", normEmbarked, DataTypes.IntegerType);

   Dataset<Row> testData = spark.sqlContext()
         .read()
         .format("com.databricks.spark.csv")
         .option("header", "true")
         .option("inferSchema", "true")
         .load("data/test.csv");
```


Gerando Dados de Teste

selecionando colunas relevantes

```
        return testData.select(
            col("PassengerId"),
            col("Fare"),
            callUDF("normSex", col("Sex")).alias("Sex"),
            col("Age"),
            col("Pclass"),
            col("Parch"),
            col("SibSp"),
            callUDF("normEmbarked", col("Embarked")).alias("Embarked"));
   }

}
```

Testando a Rede

gerando os dados de teste
substituindo valores nulos (usando Map em vez de UDF)

```
private static Dataset<Row> testNetwork
        (MultilayerPerceptronClassificationModel model) {
    Dataset<Row> testData = getTestData();
    testData.show();

    Map<String, Object> m = new HashMap<String, Object>();
    m.put("Age", mean_age_column);
    m.put("Fare", mean_fare_column);

    Dataset<Row> testData2 = testData.na().fill(m);
    testData2.show();
```


Testando a Rede

criando pares de vetores com rótulo e características normalizadas

```
JavaRDD<VectorPair> testRDD = testData2.javaRDD().map(row -> {
    VectorPair vectorPair = new VectorPair();
    vectorPair.setLabel(row.<Integer>getAs("PassengerId"));
    vectorPair.setFeatures(getScaledVector(
            row.<Double>getAs("Fare"),
            row.<Double>getAs("Age"),
            row.<Integer>getAs("Pclass"),
            row.<Integer>getAs("Sex"),
            row.<Integer>getAs("Embarked"),
            scaler));
    return vectorPair;
});
```

Testando a Rede

criando DataFrame
convertendo vetores Mlib para ML
gerando a predição

```
    Dataset<Row> scaledTestData
          = spark.createDataFrame(testRDD, VectorPair.class);

    Dataset<Row> finalTestData
          = MLUtils.convertVectorColumnsToML(scaledTestData)
                    .toDF("features", "PassengerId");
    trainingData.show();
    finalTestData.show();

    Dataset<Row> resultDF = model.transform(finalTestData)
          .select("PassengerId", "prediction");
    resultDF.show();
    return resultDF;

}
```