|  |
| --- |
| **Travaux Pratiques**  **FI BDCC S4 : Systèmes Multi Agents et Intelligence Artificielle Distribuée** TP: Machines Learning, DeepLearning(Mohamed HAMMANE) |

# Introduction :

Le Machine learning (apprentissage automatique) est la technologie la plus ancienne et la plus simple. Elle s’appuie sur un algorithme qui adapte lui-même le système à partir des retours faits par l’humain. La mise en place de cette technologie implique l’existence de données organisées. Le système est ensuite alimenté par des données structurées et catégorisées lui permettant de comprendre comment classer de nouvelles données similaires. En fonction de ce classement, le système exécute ensuite les actions programmées. Il sait par exemple identifier si une photo montre un chien ou un chat et classer le document dans le dossier correspondant.

Après une première phase d’utilisation, l’algorithme est optimisé à partir des feedbacks du développeur, qui informent le système des classifications erronées et lui indiquent les bonnes catégories.

Le Deep learning (apprentissage profond) n’a pas besoin de données structurées. Le système fonctionne à partir de plusieurs couches de réseaux neuronaux, qui combinent différents algorithmes en s’inspirant du cerveau humain. Ainsi, le système est capable de travailler à partir de données non structurées.

Cette approche est particulièrement adaptée pour les tâches complexes, lorsque tous les aspects des objets à traiter ne peuvent pas être catégorisés en amont. Le système du Deep learning identifie lui-même les caractéristiques discriminantes. Dans chaque couche, il recherche un nouveau critère spécifique de l’objet, qui sert de base pour décider de la classification retenue pour l’objet à la fin du processus.

Important : avec le Deep learning, le système identifie lui-même les caractéristiques discriminantes des données, sans avoir besoin d’une catégorisation préalable. Le système n’a pas besoin d’être entraîné par un développeur. Il évalue lui-même le besoin de modifier le classement ou de créer des catégories inédites en fonction des nouvelles données.

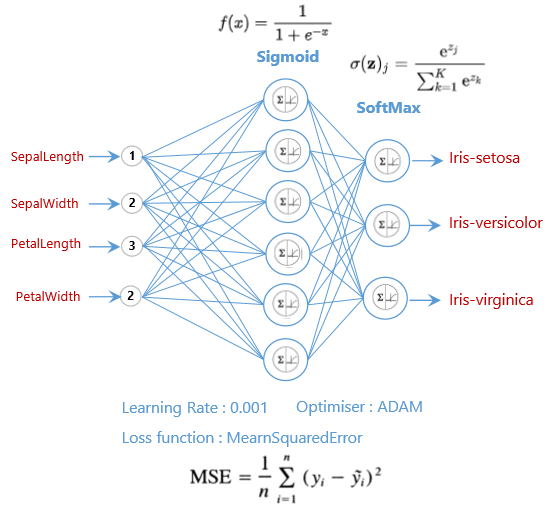
Tandis que le Machine learning fonctionne à partir d’une base de données contrôlable, le Deep learning a besoin d’un volume de données bien plus considérable. Le système doit disposer de plus de 100 millions d’entrées pour donner des résultats fiables.

Par ailleurs, la technologie nécessaire pour le Deep learning est plus sophistiquée. Elle exige plus de ressources IT et s’avère nettement plus coûteuse que le Machine learning : elle n’est donc pas intéressante, du moins à l’heure actuelle, pour une utilisation de masse par les entreprises.

# Enoncé :

On souhaite créer des applications qui utilisent DL4J (est une Framework open source (licence Apache) qui permet de construire, entraîner et tester une grande diversité d’algorithmes de Deep Learning (depuis les réseaux standard, jusqu’aux réseaux convolutionels, en passant par des architectures plus complexes). Il se base sur sa structure de données (Nd4j) permettant d’effectuer les opérations de l’algèbre linéaires sur les architectures massivement parallèles GPU et les architectures distribuées. Nd4jutilise du code natif (Cuda oblige), et alloue de l’espace or du tas Java. Ceci est impérativement à prendre en compte lorsque la volumétrie des données est importante. DL4J utilise DataVecpour la vectorisation et la transformation des données. ND4J est une bibliothèque de calcul scientifique et d'algèbre linéaire, écrite en langage de programmation Java et compatible avec d'autres langages tels que Scala et Clojure. Le ND4J a été versé à la fondation Eclipse en octobre 2017).

# Application 1 - Iris Data Set :



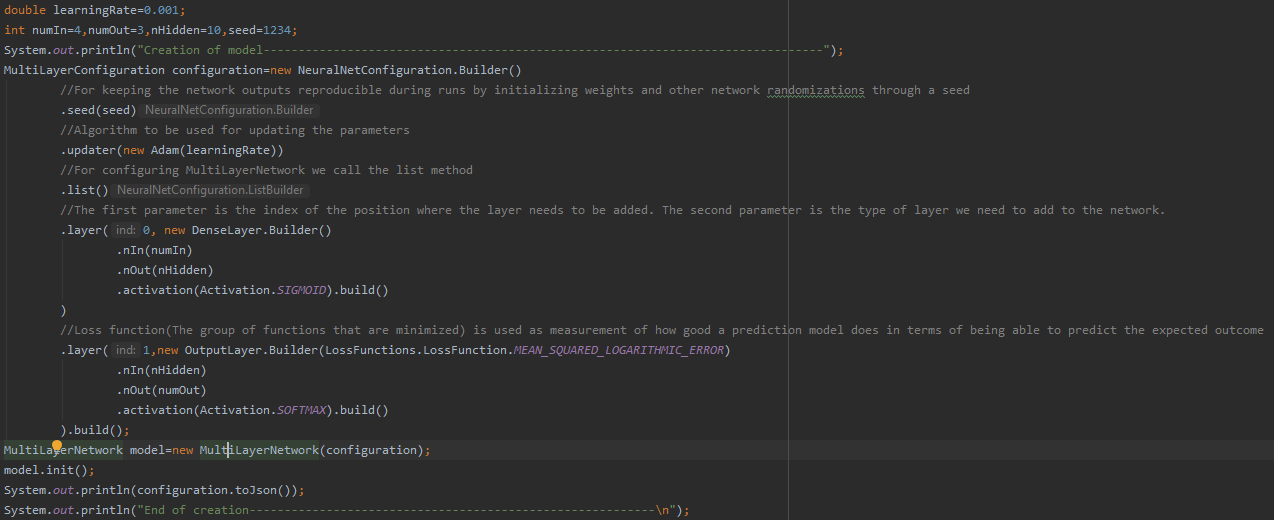


Dans cette application on veut créer un modèle qui va nous prédire le type des IRIS données.

Notre pom.xml est :

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.enset</groupId>  
 <artifactId>dl4j-iris-app</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.deeplearning4j</groupId>  
 <artifactId>deeplearning4j-core</artifactId>  
 <version>1.0.0-beta3</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.deeplearning4j</groupId>  
 <artifactId>deeplearning4j-ui\_2.11</artifactId>  
 <version>1.0.0-beta3</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.nd4j</groupId>  
 <artifactId>nd4j-native-platform</artifactId>  
 <version>1.0.0-beta3</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.nd4j</groupId>  
 <artifactId>nd4j-cuda-9.2-platform</artifactId>  
 <version>1.0.0-beta3</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-simple</artifactId>  
 <version>1.6.1</version>  
 </dependency>  
 </dependencies>  
  
</project>

## **Initialisation du model :**



## **Entraîner notre modèle :**

1. Pour entrainer notre modèle on va utiliser le fichier iris\_train.vsv qui contient (block après l’autre verticalement).

Labels

Labels

Labels

Features

Features

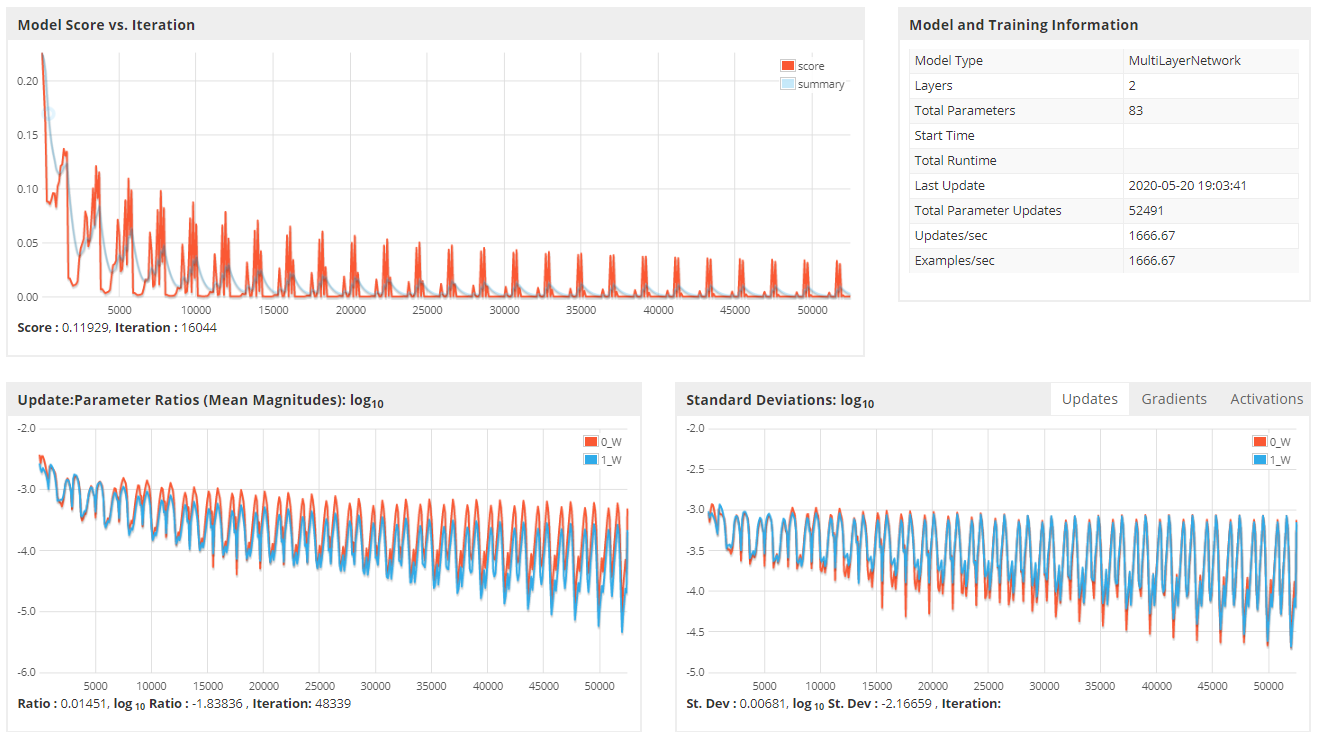
Features

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4.7 | 3.2 | 1.3 | 0.2 | 0 |  | 5.5 | 2.3 | 4 | 1.3 | 1 |  | 6.3 | 3.3 | 6 | 2.5 | 2 |
| 5 | 3.6 | 1.4 | 0.2 | 0 |  | 6.5 | 2.8 | 4.6 | 1.5 | 1 |  | 5.8 | 2.7 | 5.1 | 1.9 | 2 |
| 5.4 | 3.9 | 1.7 | 0.4 | 0 |  | 5.7 | 2.8 | 4.5 | 1.3 | 1 |  | 7.1 | 3 | 5.9 | 2.1 | 2 |
| 4.6 | 3.4 | 1.4 | 0.3 | 0 |  | 6.3 | 3.3 | 4.7 | 1.6 | 1 |  | 6.3 | 2.9 | 5.6 | 1.8 | 2 |
| 4.9 | 3.1 | 1.5 | 0.1 | 0 |  | 4.9 | 2.4 | 3.3 | 1 | 1 |  | 4.9 | 2.5 | 4.5 | 1.7 | 2 |
| 5.4 | 3.7 | 1.5 | 0.2 | 0 |  | 6.6 | 2.9 | 4.6 | 1.3 | 1 |  | 7.3 | 2.9 | 6.3 | 1.8 | 2 |
| 4.8 | 3.4 | 1.6 | 0.2 | 0 |  | 5.2 | 2.7 | 3.9 | 1.4 | 1 |  | 6.7 | 2.5 | 5.8 | 1.8 | 2 |
| 4.8 | 3 | 1.4 | 0.1 | 0 |  | 5.9 | 3 | 4.2 | 1.5 | 1 |  | 7.2 | 3.6 | 6.1 | 2.5 | 2 |
| 5.8 | 4 | 1.2 | 0.2 | 0 |  | 5.6 | 2.9 | 3.6 | 1.3 | 1 |  | 6.5 | 3.2 | 5.1 | 2 | 2 |
| 5.7 | 4.4 | 1.5 | 0.4 | 0 |  | 6.7 | 3.1 | 4.4 | 1.4 | 1 |  | 6.8 | 3 | 5.5 | 2.1 | 2 |
| 5.4 | 3.9 | 1.3 | 0.4 | 0 |  | 5.6 | 3 | 4.5 | 1.5 | 1 |  | 6.5 | 3 | 5.5 | 1.8 | 2 |
| 5.1 | 3.5 | 1.4 | 0.3 | 0 |  | 6.2 | 2.2 | 4.5 | 1.5 | 1 |  | 7.7 | 2.6 | 6.9 | 2.3 | 2 |
| 5.7 | 3.8 | 1.7 | 0.3 | 0 |  | 5.6 | 2.5 | 3.9 | 1.1 | 1 |  | 6 | 2.2 | 5 | 1.5 | 2 |
| 5.4 | 3.4 | 1.7 | 0.2 | 0 |  | 6.3 | 2.5 | 4.9 | 1.5 | 1 |  | 6.9 | 3.2 | 5.7 | 2.3 | 2 |
| 5.1 | 3.7 | 1.5 | 0.4 | 0 |  | 6.4 | 2.9 | 4.3 | 1.3 | 1 |  | 5.6 | 2.8 | 4.9 | 2 | 2 |
| 4.6 | 3.6 | 1 | 0.2 | 0 |  | 5.7 | 2.6 | 3.5 | 1 | 1 |  | 7.7 | 2.8 | 6.7 | 2 | 2 |
| 5.1 | 3.3 | 1.7 | 0.5 | 0 |  | 5.5 | 2.4 | 3.8 | 1.1 | 1 |  | 6.3 | 2.7 | 4.9 | 1.8 | 2 |
| 4.8 | 3.4 | 1.9 | 0.2 | 0 |  | 5.5 | 2.4 | 3.7 | 1 | 1 |  | 6.7 | 3.3 | 5.7 | 2.1 | 2 |
| 5 | 3.4 | 1.6 | 0.4 | 0 |  | 5.8 | 2.7 | 3.9 | 1.2 | 1 |  | 6.2 | 2.8 | 4.8 | 1.8 | 2 |
| 5.2 | 3.5 | 1.5 | 0.2 | 0 |  | 6 | 2.7 | 5.1 | 1.6 | 1 |  | 6.1 | 3 | 4.9 | 1.8 | 2 |
| 5.2 | 3.4 | 1.4 | 0.2 | 0 |  | 5.4 | 3 | 4.5 | 1.5 | 1 |  | 6.4 | 2.8 | 5.6 | 2.1 | 2 |
| 4.7 | 3.2 | 1.6 | 0.2 | 0 |  | 6 | 3.4 | 4.5 | 1.6 | 1 |  | 7.4 | 2.8 | 6.1 | 1.9 | 2 |
| 5.4 | 3.4 | 1.5 | 0.4 | 0 |  | 6.7 | 3.1 | 4.7 | 1.5 | 1 |  | 7.9 | 3.8 | 6.4 | 2 | 2 |
| 5.2 | 4.1 | 1.5 | 0.1 | 0 |  | 6.3 | 2.3 | 4.4 | 1.3 | 1 |  | 6.3 | 2.8 | 5.1 | 1.5 | 2 |
| 5.5 | 4.2 | 1.4 | 0.2 | 0 |  | 5.6 | 3 | 4.1 | 1.3 | 1 |  | 6.1 | 2.6 | 5.6 | 1.4 | 2 |
| 4.9 | 3.1 | 1.5 | 0.2 | 0 |  | 5.5 | 2.6 | 4.4 | 1.2 | 1 |  | 7.7 | 3 | 6.1 | 2.3 | 2 |
| 5 | 3.2 | 1.2 | 0.2 | 0 |  | 6.1 | 3 | 4.6 | 1.4 | 1 |  | 6.3 | 3.4 | 5.6 | 2.4 | 2 |
| 4.9 | 3.6 | 1.4 | 0.1 | 0 |  | 5.8 | 2.6 | 4 | 1.2 | 1 |  | 6.4 | 3.1 | 5.5 | 1.8 | 2 |
| 5 | 3.5 | 1.3 | 0.3 | 0 |  | 5 | 2.3 | 3.3 | 1 | 1 |  | 6 | 3 | 4.8 | 1.8 | 2 |
| 4.4 | 3.2 | 1.3 | 0.2 | 0 |  | 5.6 | 2.7 | 4.2 | 1.3 | 1 |  | 5.8 | 2.7 | 5.1 | 1.9 | 2 |
| 5.1 | 3.8 | 1.9 | 0.4 | 0 |  | 5.7 | 3 | 4.2 | 1.2 | 1 |  | 6.7 | 3.3 | 5.7 | 2.5 | 2 |
| 4.8 | 3 | 1.4 | 0.3 | 0 |  | 5.7 | 2.9 | 4.2 | 1.3 | 1 |  | 6.7 | 3 | 5.2 | 2.3 | 2 |
| 4.6 | 3.2 | 1.4 | 0.2 | 0 |  | 6.2 | 2.9 | 4.3 | 1.3 | 1 |  | 6.5 | 3 | 5.2 | 2 | 2 |
| 5.3 | 3.7 | 1.5 | 0.2 | 0 |  | 5.1 | 2.5 | 3 | 1.1 | 1 |  | 6.2 | 3.4 | 5.4 | 2.3 | 2 |
| 5 | 3.3 | 1.4 | 0.2 | 0 |  | 5.7 | 2.8 | 4.1 | 1.3 | 1 |  | 5.9 | 3 | 5.1 | 1.8 | 2 |

System.*out*.println("Training--------------------------------------------------------");  
RecordReader rr=new CSVRecordReader();  
rr.initialize(new FileSplit(new ClassPathResource("iris\_train.csv").getFile()));  
//Separate data  
int batchSize=1;  
int classIndex = 4;  
DataSetIterator dataSetTrain=new RecordReaderDataSetIterator(rr,batchSize,classIndex,numOut);  
while(dataSetTrain.hasNext()){  
 System.*out*.println("-----------------------------------------------------------");  
 DataSet dataSet = dataSetTrain.next();  
 System.*out*.println(dataSet.getFeatures());  
 System.*out*.println("///////////////////////////////////////////////////////////");  
 System.*out*.println(dataSet.getLabels());  
 System.*out*.println("-----------------------------------------------------------");  
}  
//Train our model 500 time  
int nEpochs=500;  
for (int i = 0; i <nEpochs ; i++) model.fit(dataSetTrain);  
System.*out*.println("End of training-------------------------------------------------");

## **Voir l’évolution de notre modèle graphiquement :**

*System.out.println("Monitoring learning-------------------------------------------\n");  
UIServer uiServer = UIServer.getInstance();  
InMemoryStatsStorage statsStorage=new InMemoryStatsStorage();  
uiServer.attach(statsStorage);  
model.setListeners(new StatsListener(statsStorage));  
System.out.println("End of monitoring---------------------------------------------\n");*



## **Evaluer le modèle :**

Pour évaluer le modèle on va utiliser le fichier : "iris\_test.csv" qui contient.

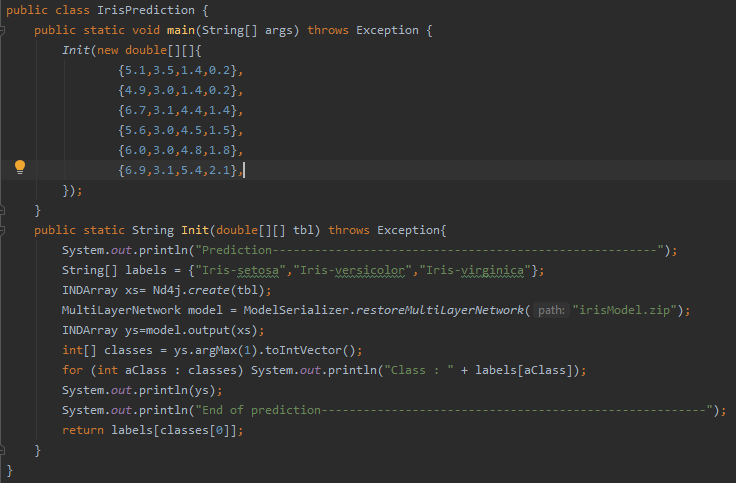
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5.1 | 3.5 | 1.4 | 0.2 | 0 |  | 7 | 3.2 | 4.7 | 1.4 | 1 |  | 6.5 | 3 | 5.8 | 2.2 | 2 |
| 4.9 | 3 | 1.4 | 0.2 | 0 |  | 6.4 | 3.2 | 4.5 | 1.5 | 1 |  | 7.6 | 3 | 6.6 | 2.1 | 2 |
| 4.6 | 3.1 | 1.5 | 0.2 | 0 |  | 6.9 | 3.1 | 4.9 | 1.5 | 1 |  | 6.4 | 2.7 | 5.3 | 1.9 | 2 |
| 5 | 3.4 | 1.5 | 0.2 | 0 |  | 5 | 2 | 3.5 | 1 | 1 |  | 5.7 | 2.5 | 5 | 2 | 2 |
| 4.4 | 2.9 | 1.4 | 0.2 | 0 |  | 6 | 2.2 | 4 | 1 | 1 |  | 5.8 | 2.8 | 5.1 | 2.4 | 2 |
| 4.3 | 3 | 1.1 | 0.1 | 0 |  | 6.1 | 2.9 | 4.7 | 1.4 | 1 |  | 6.4 | 3.2 | 5.3 | 2.3 | 2 |
| 5.1 | 3.8 | 1.5 | 0.3 | 0 |  | 5.8 | 2.7 | 4.1 | 1 | 1 |  | 7.7 | 3.8 | 6.7 | 2.2 | 2 |
| 5 | 3 | 1.6 | 0.2 | 0 |  | 5.9 | 3.2 | 4.8 | 1.8 | 1 |  | 7.2 | 3.2 | 6 | 1.8 | 2 |
| 4.8 | 3.1 | 1.6 | 0.2 | 0 |  | 6.1 | 2.8 | 4 | 1.3 | 1 |  | 7.2 | 3 | 5.8 | 1.6 | 2 |
| 5.5 | 3.5 | 1.3 | 0.2 | 0 |  | 6.1 | 2.8 | 4.7 | 1.2 | 1 |  | 6.4 | 2.8 | 5.6 | 2.2 | 2 |
| 4.4 | 3 | 1.3 | 0.2 | 0 |  | 6.6 | 3 | 4.4 | 1.4 | 1 |  | 6.9 | 3.1 | 5.4 | 2.1 | 2 |
| 5.1 | 3.4 | 1.5 | 0.2 | 0 |  | 6.8 | 2.8 | 4.8 | 1.4 | 1 |  | 6.7 | 3.1 | 5.6 | 2.4 | 2 |
| 4.5 | 2.3 | 1.3 | 0.3 | 0 |  | 6.7 | 3 | 5 | 1.7 | 1 |  | 6.9 | 3.1 | 5.1 | 2.3 | 2 |
| 5 | 3.5 | 1.6 | 0.6 | 0 |  | 6 | 2.9 | 4.5 | 1.5 | 1 |  | 6.8 | 3.2 | 5.9 | 2.3 | 2 |
| 5.1 | 3.8 | 1.6 | 0.2 | 0 |  | 5.5 | 2.5 | 4 | 1.3 | 1 |  | 6.3 | 2.5 | 5 | 1.9 | 2 |

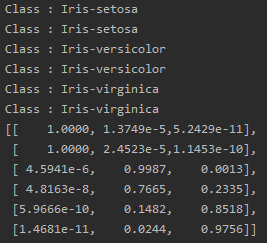
System.*out*.println("Model Evaluation------------------------------------------------");  
RecordReader rrTest=new CSVRecordReader();  
rrTest.initialize(new FileSplit(new ClassPathResource("iris\_test.csv").getFile()));  
DataSetIterator dataSetTest=new RecordReaderDataSetIterator(rrTest,batchSize,classIndex,numOut);  
Evaluation evaluation=new Evaluation();  
while(dataSetTest.hasNext()){  
 DataSet dataSet=dataSetTest.next();  
 INDArray features=dataSet.getFeatures();  
 INDArray targetLabels=dataSet.getLabels();  
 INDArray predictedLabels=model.output(features);  
 evaluation.eval(predictedLabels,targetLabels);  
}  
System.*out*.println(evaluation.stats());  
System.*out*.println("End of evaluation-----------------------------------------------");

Puis on va enregistrer nos informations pour ne pas perdre du temps chaque fois dans les étapes précédentes pour la prédiction, en utilisant :

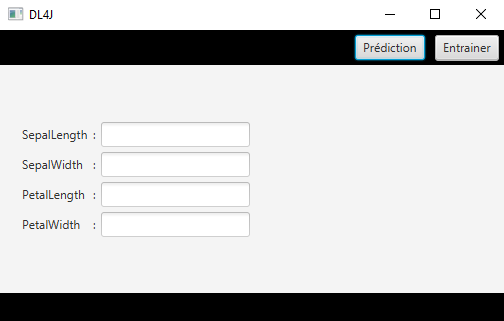
ModelSerializer.*writeModel*(model,"irisModel.zip",true);

## **Prédiction :**





## **Application JAVAFX :**



L’utilisateur clique sur entrainer puis il choisit le nombre de fois :



Maintenant il peut prédire :

