Création d’un assistant d'achat personnel à l'aide de Brain.js et Node.js

Cet atelier met en place un système simple de recommandation où les utilisateurs se voient recommander des produits en fonction de leur nom. Il montre comment intégrer un système de recommandation basé sur un réseau neuronal dans une application Express.js.

Référence :

<https://hackernoon.com/build-a-personal-shopping-assistant-using-brainjs-and-nodejs>

Etapes préliminaires

1. Installer Python

<https://www.python.org/downloads/>

1. Exécuter :

npm init

1. **Mettez à jour node-gyp**

npm install -g node-gyp

1. Faire le rebuild

npm rebuild

1. Installer les packages

npm install express mongoose

npm install --save brain.js@1.4.10

Créer les Models



const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({

  name: String,

  image: String,

  categories: [String],

  colors: [String],

  sizes: [String],

  // etc

});

module.exports = mongoose.model('Product', productSchema);



const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

  name: String,

  likes: [{type: mongoose.Schema.Types.ObjectId, ref: 'Product'}],

  dislikes: [{type: mongoose.Schema.Types.ObjectId, ref: 'Product'}]

});

module.exports = mongoose.model('User', userSchema);

Exécuter les Seeds pour remplir la BD



const Product = require('../models/product.model')

const mongoose = require('mongoose');

// Connexion à la base données

mongoose.connect("mongodb://127.0.0.1:27017/DataBaseBrain")

    .then(() => {console.log("DataBase Successfully Connected");})

    .catch(err => { console.log("Unable to connect to database", err); process.exit(); });

// Sample products

const products = [

    {

      name: 'green shirt',

      image: 'shirt1.jpg',

      categories: ['tops', 'shirts'],

      colors: ['green']

    },

    {

      name: 'skirt',

      image: 'skirt.jpg',

      categories: ['bottoms', 'skirts'],

      colors: ['black'],

      sizes: ['S', 'M', 'L']

    },

    {

      name: 'dress',

      image: 'dress.jpg',

      categories: ['tops', 'shirts'],

      colors: ['blue']

    },

    {

      name: 'djeans',

      image: 'shirt1.jpg',

      categories: ['buttom', 'pants'],

      colors: ['red']

    },

    {

      name: 'red shirt',

      image: 'shirt1.jpg',

      categories: ['tops', 'shirts'],

      colors: ['red']

    },

  ]

  // Populate database

  Product.insertMany(products).then(()=> {

    console.log('Database Product collection populated!');// Success

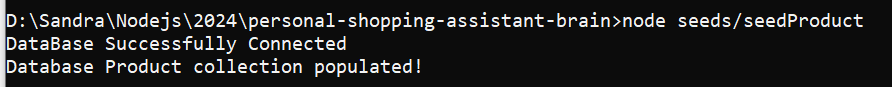
}).catch((error)=> {

    console.log(error)     // Failure

});

  Exécuter dans le terminal :

node seeds/seedProduct.js





const User = require('../models/user.model');

const Product = require('../models/product.model');

const mongoose = require('mongoose');

  // Connexion à la base données

  mongoose.connect("mongodb://127.0.0.1:27017/DataBaseBrain")

  .then(() => {console.log("DataBase Successfully Connected");})

  .catch(err => { console.log("Unable to connect to database", err); process.exit(); });

const allProducts = async () => {

  try {

    const articles = await Product.find();

    return articles;

  } catch (error) {

    console.log(error.message);

    return null;

  }

};

const populateDatabase = async () => {

  const users = []

  try {

    // Fetch products from the database

    await allProducts().then((products) =>{

      console.log(products)

      // Sample users with product references

      users.push({

        name: 'Mia Doe',

        likes: [products[0].\_id, products[1].\_id]

      })

      users.push(

      {

        name: 'Jiji Doe',

        likes: [products[0].\_id, products[1].\_id] ,

        dislikes: [products[2].\_id]

      })

      users.push(

        {

          name: 'Alan Doe',

          likes: [products[4].\_id] ,

          dislikes: [products[0].\_id, products[1].\_id]

        })

      users.push(

          {

            name: 'Emma Doe',

            likes: [products[1].\_id] ,

            dislikes: [products[4].\_id]

          })

      users.push(

            {

              name: 'Alex Doe',

              dislikes: [products[1].\_id]

            })

      users.push(

              {

                name: 'Sarah Doe',

                dislikes: [products[1].\_id]

              })

    })

     // Populate the User collection

     await User.insertMany(users);

     console.log('Database User collection populated!'); // Success

  }

    catch (error) {

      console.log("Error:", error);

    }

};

// Execute the function to populate the database

populateDatabase();

Exécuter dans le terminal :

node seeds/seedUser.js



Ecrire le code de la recommandation



const brain = require('brain.js');

const express = require('express');

const router = express.Router();

const Product = require("../models/product.model");

const User = require('../models/user.model');

const network = new brain.recurrent.LSTM();

// Function to fetch users and products from the database

const fetchData = async () => {

  try {

    const users = await User.find();

    const products = await Product.find();

    return { users, products };

  } catch (error) {

    console.log(error.message);

    return null;

  }

};

const trainNetwork = async () => {

  try {

    const data = await fetchData();

    if (!data || !data.users || !data.products) {

      throw new Error("Could not retrieve users or products");

    }

    const { users, products } = data;

    if (users.length < 2 || products.length < 3) {

      throw new Error("Insufficient users or products for training");

    }

    // Prepare training data using product names

    const trainingData = [

      { input: users[0].name, output: products[0].name },

      { input: users[0].name, output: products[1].name },

      { input: users[1].name, output: products[2].name }

    ];

    // Log training data for debugging

    console.log('Training Data:', trainingData);

    // Train the network

    network.train(trainingData, {

      iterations: 2000,

      log: true,

      logPeriod: 100

    });

    console.log("Network trained successfully");

    return network;

  } catch (error) {

    console.log("Error during training:", error);

    return null;

  }

};

const initialize = async () => {

  try {

    await trainNetwork();

  } catch (error) {

    console.log("Initialization error:", error);

  }

};

// Encode user preferences as a simple input

function encodeUser(userDoc) {

  return userDoc.name;  // Simplified for this example

}

router.get('/:userId', async (req, res) => {

  try {

    const user = await User.findById(req.params.userId);

    if (!user) {

      return res.status(404).json({ message: "User not found" });

    }

    const input = encodeUser(user);

    await initialize();

    const prediction = network.run(input);

    // Log predictions for debugging

    console.log('Predictions:', prediction);

    // Find the matching product by name

    const recommendedProduct = await Product.findOne({ name: prediction });

    if (recommendedProduct) {

      res.status(200).json([recommendedProduct]);  // Return as array to maintain consistency

    } else {

      res.status(200).json([]);  // Return empty array if no match found

    }

  } catch (error) {

    console.log(error.message);

    res.status(500).json({ message: "Server error" });

  }

});

module.exports = router;

Explication :

Ce code met en place un système de recommandation utilisant un réseau neuronal récurrent (RNN) LSTM avec la bibliothèque brain.js, intégré dans une application Express.js.

Initialisation du Réseau Neuronal avec const network = new brain.recurrent.LSTM() : on crée un nouveau réseau neuronal LSTM.

fetchData : Une fonction asynchrone pour récupérer les utilisateurs et les produits depuis la base de données.

trainingData est un tableau d'objets qui représente les exemples d'entraînement pour le réseau de neurones. Chaque objet a une propriété input qui est le nom d'un utilisateur et une propriété output qui est le nom d'un produit.

trainNetwork : Une fonction asynchrone pour entraîner le réseau neuronal avec les données récupérées avec fetchData.

On vérifie si les données ont été récupérées correctement.

On prépare les données d'entraînement en associant les noms des utilisateurs aux noms des produits.

On entraîne le réseau avec les données d'entraînement et les paramètres spécifiés (nombre d'itérations, journalisation).

network.train est une méthode qui entraîne le réseau de neurones avec les données fournies.

trainingData est passé en argument à cette méthode pour indiquer les exemples d'entraînement.

Le deuxième argument est un objet de configuration qui définit les paramètres de l'entraînement :

* Iterations : 2000 spécifie que le réseau doit passer par les données d'entraînement 2000 fois.
* Log : true indique que des informations de journalisation doivent être affichées pendant l'entraînement.
* logPeriod : 100 signifie que des informations de journalisation doivent être affichées toutes les 100 itérations.

Initialize : Initialise le système de recommandation en appelant trainNetwork pour entraîner le réseau.

encodeUser : Une fonction qui encode les préférences de l'utilisateur en entrée pour le réseau neuronal. Dans cet exemple, elle renvoie simplement le nom de l'utilisateur.

Les réseaux de neurones ne peuvent pas traiter directement des chaînes de caractères. Il y a donc un processus de prétraitement qui encode les noms des utilisateurs et des produits en vecteurs numériques.

On définit, ensuite, une route GET avec un paramètre d'identifiant utilisateur.

On cherche l'utilisateur avec l'identifiant fourni dans la base de données.

Si l'utilisateur est trouvé, on encode le nom de l'utilisateur avec encodeUser, on exécute la prédiction du réseau neuronal avec l'entrée encodée, et on retrouve le produit recommandé dans la base de données en fonction de la prédiction.

Si un produit recommandé est trouvé, on renvoie un statut 200 avec une réponse JSON contenant le produit recommandé.

Si aucun produit recommandé n'est trouvé, on renvoie un statut 200 avec un tableau JSON vide.

Exécuter le serveur



const express = require('express');

const mongoose = require('mongoose');

const app = express();

// Connect to the database

mongoose.connect("mongodb://127.0.0.1:27017/DataBaseBrain")

    .then(() => {

        console.log("Database Successfully Connected");

    })

    .catch(err => {

        console.log("Unable to connect to database", err);

        process.exit();

    });

// Middleware to parse JSON

app.use(express.json());

// Simple route to check if the server is working

app.get('/', (req, res) => {

    res.send('Shopping assistant is ready!');

});

//Appel de routes

const productRoutes= require('./routes/productrecommendation.routes');

app.use('/api/productsRecomm',productRoutes);

// Start the server

app.listen(3001, () => {

    console.log(`Server is listening on port ${3001}`);

});

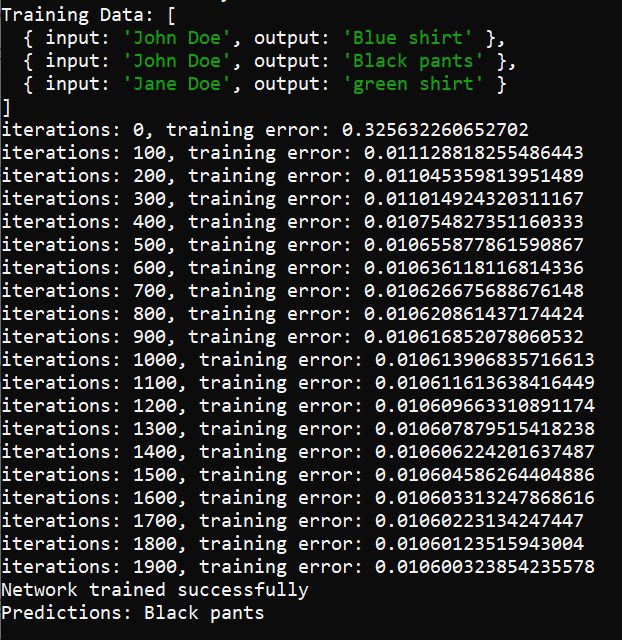
module.exports = app;

Tester le code

Exécuter la requête GET dans Thunder Client avec l’url (en donnant un \_id récupéré de la BD en paramètre) :

<http://localhost:3001/api/productsRecomm/6664730151b6553b3f65ee4b>

Dans le terminal on peut suivre le processus



Dans Thunder Client on verra :

