Implementación de un modelo predictivo en aplicación.

Javier Fernández

Enero 2024

Índice

1. Objetivo.

Desarrollar un modelo de aprendizaje profundo para predecir las ventas de camisetas de fútbol en una tienda online e implementarlo en una api.

2. Creación del modelo.

He creado 2 modelos, el primero de ellos pensado para secuencias de 1 elemento (es decir, en este modelo el tiempo pierde importancia) y el segundo para secuencias de 60 elementos (en este modelo el tiempo tiene un gran peso), para así poder compararlos y elegir el que mejor funcione. Para su creación, los pasos que he seguido han sido:

- **Importar librerías, módulos y funciones necesarias.**
- **Lectura, análisis descriptivo y análisis exploratorio de los datos.**
- **Limpieza y preparación de los datos.**
- **Creación, entrenamiento y validación de los modelos.**

Todos ellos los podemos encontrar explicados en detalle junto a su fragmento de código en el archivo .ipynb.

3. Implementación del modelo en una api.

3.1. Instalar Flask.

```
pip install flask
```

3.2. Crear una nueva aplicación Flask.

```
mkdir mi_aplicacion_flask
cd mi_aplicacion_flask
```

3.3. Crear un entorno virtual.

```
python -m venv venv
source venv/bin/activate  # Para activar el entorno
```

3.4. Instalar Flask en el entorno virtual.

```
pip install flask
```

3.5. Crear archivo `app.py` e `index.html`.

app.py:

```
from flask import Flask, render_template, jsonify, request
from keras.models import load_model
import numpy as np

app = Flask(__name__)

# Cargar el modelo entrenado
model = load_model("Pesos1_PracticaAA.best.hdf5")

@app.route('/')
def index():
    return render_template('index.html')

# Endpoint para predecir
@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Obtener datos de la solicitud
        data = request.get_json(force=True)
        user_sequence = np.array([data['user_sequence']])  # Adaptar la secuencia a la forma

        # Loguear la entrada
        print("Entrada al modelo:")
        print(user_sequence)

        # Realizar la predicción
        prediction = model.predict(user_sequence)  # Hacer la predicción

        # Loguear la salida
        print("Salida del modelo:")
        print(prediction)

        # Enviar la predicción como respuesta
        return jsonify({'prediction': prediction.tolist()})

    except Exception as e:
        return jsonify({'error': str(e)})

if __name__ == '__main__':
    app.run(debug=True)
```

La funcion index utiliza `render_template` para indicar que cuando un usuario acceda a la ruta '/', Flask debe mostrar la plantilla HTML llamada 'in-

dex.html'. La función `predict` obtiene datos de la solicitud, realiza la predicción y envia la respuesta JSON.

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>API Example</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
            text-align: center;
        }

        h1 {
            color: #333;
        }

        .input-container {
            display: flex;
            justify-content: space-between;
            margin-bottom: 10px;
        }

        .input-box {
            width: 48%;
            padding: 10px;
            box-sizing: border-box;
        }

        button {
            background-color: #4CAF50;
            color: white;
            padding: 10px 20px;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            font-size: 16px;
        }

        button:hover {
            background-color: #45a049;
```

```
        }

        p {
            margin-top: 20px;
            font-size: 18px;
        }
    </style>
</head>
<body>

    <h1>API Example</h1>

    <div class="input-container">
        <input id="countryInput" class="input-box" placeholder="País (ej: 1,0,0,0,0,0,...)">
        <input id="priceInput" class="input-box" placeholder="Precio por unidad">
    </div>

    <button onclick="predict()">Hacer Predicción</button>

    <p id="predictionResult"></p>

    <script>
        function parseInput(input1, input2) {
            var countryString = input1.trim();
            var country = countryString.split(',').map(Number);  // Convertir la cadena de n
            var price = parseFloat(input2.trim());

            if (isNaN(price)) {
                price = 0; // Manejar el caso en que el precio no sea un número válido
            }

            return country.concat(price);
        }

        function predict() {
            var countryInput = document.getElementById('countryInput').value;
            var priceInput = document.getElementById('priceInput').value;

            var userSequence = parseInput(countryInput, priceInput);

            fetch('/predict', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ 'user_sequence': userSequence })
```

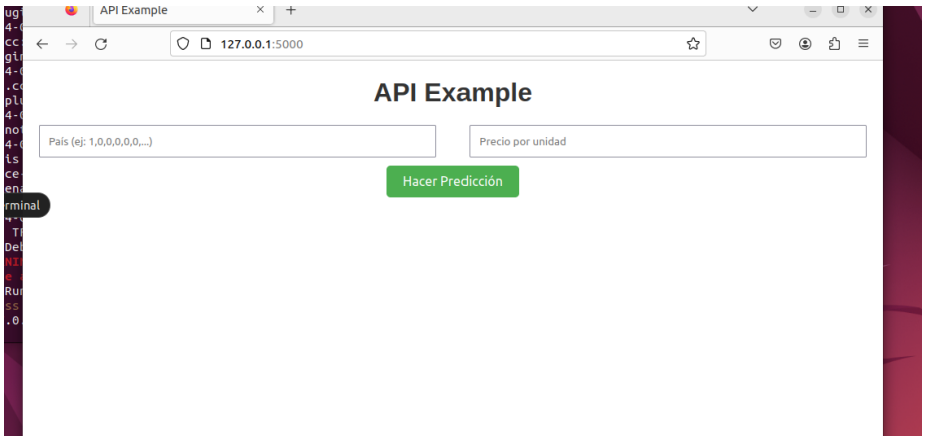
```
            })
            .then(response => response.json())
            .then(data => {
                if ('prediction' in data) {
                    document.getElementById('predictionResult').innerHTML = 'Predicción: ' +
                } else {
                    document.getElementById('predictionResult').innerHTML = 'Error en la pre
                }
            })
            .catch(error => {
                console.error('Error:', error);
                document.getElementById('predictionResult').innerHTML = 'Error en la predic
            });
        }
    </script>

</body>
</html>
```

En el script de js (que es lo verdaderamente importante del archivo html), la función `parseInput` se encarga de procesar la entrada del usuario, mientras que la función `predict` obtiene los valores de entrada, realiza una solicitud POST a la ruta /predict del servidor Flask y maneja la respuesta, actualizando el contenido de la página con el resultado de la predicción o mostrando un mensaje de error en caso de fallo.

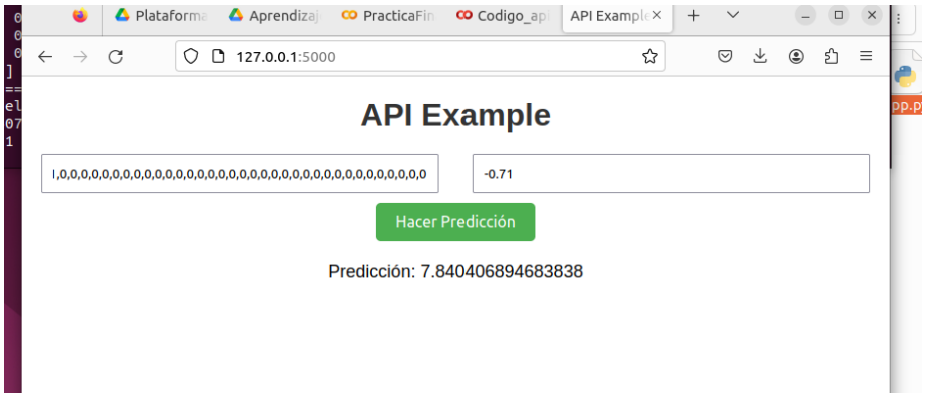
3.6. Ejecutar la aplicación y probar el modelo.

```
flask run
```

Ahora podrás ver tu aplicación en `http://localhost:5000` en tu navegador.



Probamos la api:



Como podemos obserbar todo funciona correctamente (siempre y cuando el formato de nuestros inputs tengan la forma correcta claro).

Link del repositorio de la api:
`https://github.com/Javifdz12/Aplicacion_prediccion.git`

4. Conclusión.

A lo largo de este proyecto, hemos entrenado 2 modelos, los cuales, debido a la falta de calidad de los datos, son demasiado imprecisos. No obstante teniendo en cuenta cómo eran los datos, bastante precisos han salido además de que no se produce overfitting ni nada.
Una vez creados, entrenados y validados, los hemos guardado y por último hemos implementado con éxito el que mejor funcionaba en una api.