Day 4 - Solve a Regression Problem using Multi-layer Perceptron

Tags

Implement Multi-layer Perceptron for California Housing Dataset

Author: **Chandan Kumar**

`enchandan.com`

- Implement Multi-layer Perceptron for California Housing Dataset
  - Load the dataset
  - Split the dataset into training and test set
  - Take out validation dataset
  - Scale
  - Load `tensorflow`
  - Build MLP Neural Network Model
  - Compile the Model
  - Train the Model
  - Evaluate the Model
  - Predict using Model

Load the dataset

```
from sklearn.datasets import fetch_california_housing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

housing = fetch_california_housing()
```

```
print(housing.feature_names)
# features: 8
print(housing.data.shape)
# target
print(housing.target.shape)

>>>
['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude',
'Longitude']
(20640, 8)
(20640,)
```

Split the dataset into training and test set

- test set data must be untouched and to be only used for testing the performance of the model after training

```
X_train_full, X_test, y_train_full, y_test = train_test_split(housing.data, housing.target, random_state=42) # by default split: 25%

print(X_train_full.shape)
print(y_train_full.shape)
print(X_test.shape)
print(y_test.shape)

>>>
(15480, 8)
(15480,)
(5160, 8)
(5160,)
```

Take out validation dataset

- Remaining dataset (after taking out test dataset) should be split again into train and validation dataset
- validation dataset is used while training to measure the performance of the model

```
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full, y_train_full, random_state=42)

print(X_train.shape)
print(y_train.shape)
print(X_valid.shape)
print(y_valid.shape)

>>>
(11610, 8)
(11610,)
(3870, 8)
(3870,)
```

Scale

- features of the training dataset should be at similar scale
- we have learnt in previous sections that similar scale helps gradient descent algorithm to converge faster

```
# Values before transformation
X_train[0]

>>>
array([ 3.52140000e+00,  1.50000000e+01,  3.04994451e+00,  1.10654828e+00,
        1.44700000e+03,  1.60599334e+00,  3.76300000e+01, -1.22430000e+02])
```

- Using `StandardScaler()` from `sklearn.preprocessing`

```
scalar = StandardScaler()

X_train = scalar.fit_transform(X_train)

X_valid = scalar.transform(X_valid)
X_test = scalar.transform(X_test)
```

- `fit_transform()`: `fit().transform()`
  - `fit()` calculates z-score to be used during scaling
  - `transform()` applies scaling to all features

```
# transformed values
X_train[0]

>>>
array([-0.19397883, -1.07781319, -0.94338545,  0.01485314,  0.02073335,
       -0.57291624,  0.92926047, -1.42215523])
```

Load `tensorflow`

```
import tensorflow as tf
from tensorflow import keras
```

Build MLP Neural Network Model

- Neural network with,
  - one Input layer with 30 neurons
  - one Output layer with 1 neuron (1 neuron as the output is a single value - sales_price)

```
model = keras.models.Sequential([
  keras.layers.Dense(30, activation='relu', input_shape=X_train.shape[1:]),
  keras.layers.Dense(1)
])
```

- no activation function needed as a single value to be predicted for regression problem. Use the output of Output layer directly as prediction

```
model.summary()

>>>
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 30)                270       
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 31        
=================================================================
Total params: 301
Trainable params: 301
Non-trainable params: 0
```

Compile the Model

- To decide while training which optimizer to use, which loss function to use etc

```
model.compile(optimizer='sgd', loss='mean_squared_error')
```

Train the Model

```
history = model.fit(X_train, y_train, epochs=30, validation_data=(X_valid, y_valid))

>>>
Epoch 1/30
```

```
363/363 [==============================] - 1s 2ms/step - loss: 0.7468 - val_loss: 2.9
443
Epoch 2/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4523 - val_loss: 9.3
193
Epoch 3/30
363/363 [==============================] - 0s 1ms/step - loss: 0.5181 - val_loss: 10.
3993
Epoch 4/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4771 - val_loss: 1.4
660
Epoch 5/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4100 - val_loss: 0.5
284
Epoch 6/30
363/363 [==============================] - 1s 2ms/step - loss: 0.3958 - val_loss: 0.4
369
.
.
.
Epoch 30/30
363/363 [==============================] - 0s 1ms/step - loss: 0.3510 - val_loss: 0.4
280
```

Evaluate the Model

```
mse_test = model.evaluate(X_test, y_test)
print(mse_test)

>>>
0.34973791241645813
```

Predict using Model

```
X_new = X_test[:3]
y_pred = model.predict(X_new)
y_pred

>>>
array([[0.69302183],
       [1.7205677 ],
       [4.09157   ]], dtype=float32)
```