BostonHousing

May 4, 2025

[3]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import seaborn as sns
```

[4]:
```
df = pd.read_csv('boston_housing.csv')
print("Dataset shape:", df.shape)
df.head()
```

```
Dataset shape: (506, 14)
```

[4]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

[5]:
```
# 3. Split features and target
X = df.drop('MEDV', axis=1)  # MEDV is the target column
y = df['MEDV']

# 4. Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

```
# 5. Normalize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[6]:
```
model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1)  # Linear activation for regression
])

model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.summary()
```

```
c:\Users\darsh\AppData\Local\Programs\Python\Python310\lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	896
dense_1 (Dense)	(None, 32)	2,080
dense_2 (Dense)	(None, 1)	33

Total params: 3,009 (11.75 KB)

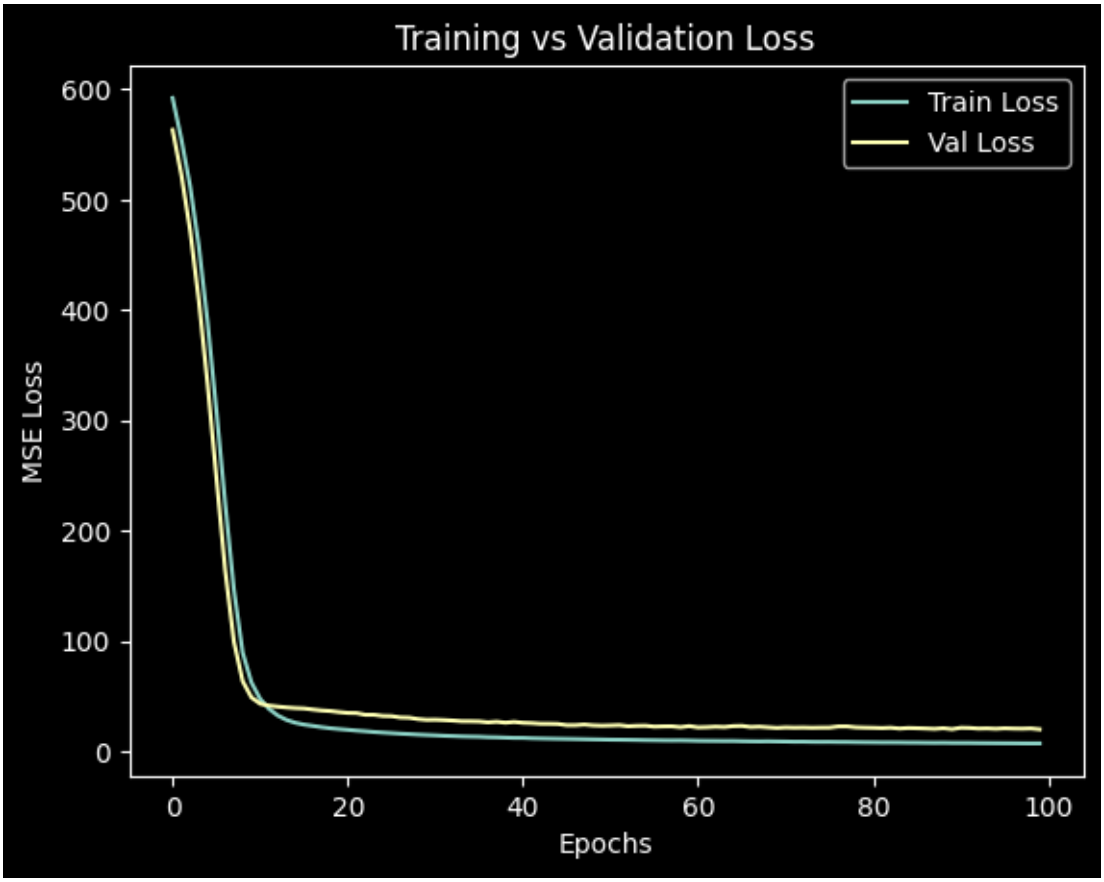
Trainable params: 3,009 (11.75 KB)

Non-trainable params: 0 (0.00 B)

[7]:
```
history = model.fit(X_train_scaled, y_train, epochs=100, validation_split=0.1,␣
↪verbose=0)
```

[8]:
```
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epochs')
plt.ylabel('MSE Loss')
```

```
plt.title('Training vs Validation Loss')
plt.legend()
plt.show()
```



[9]:
```
# 9. Predict on test data
y_pred = model.predict(X_test_scaled).flatten()

# 10. Show 10 actual vs predicted values
results_df = pd.DataFrame({'Actual': y_test.values, 'Predicted': y_pred})
results_df = results_df.reset_index(drop=True)
results_df.head(10)
```

```
4/4             0s 15ms/step
```

[9]:

	Actual	Predicted
0	23.6	26.828062
1	32.4	32.688301
2	13.6	16.475147
3	22.8	26.344967
4	16.1	15.890811

```
5    20.0  19.701498
6    17.8  16.424192
7    14.0  12.966073
8    19.6  24.885229
9    16.8  17.590311
```

[10]:
```python
# 11. Performance metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R² Score: {r2:.2f}")
```

```
Mean Absolute Error (MAE): 2.22
Mean Squared Error (MSE): 11.40
Root Mean Squared Error (RMSE): 3.38
R² Score: 0.84
```