ToolsTechniques

December 15, 2020

1 Tools & Techniques: Neural Networks Using Keras

[1]:
```
import pandas as pd
import numpy as np
```

1.1 Read in Housing Data

- Data collected from Realtor.com.
- Houses sold during June 2020 in eastern Loudoun County, VA.
- For our demonstration, the data has already been cleaned.
- Mean Price: ~ $520,000

[2]:
```
df = pd.read_csv("tools_techniques.csv")
```

[3]:
```
df
```

[3]:

	Beds	SqFt	Built	Garage	FullBaths	HalfBaths	LotSqFt	AboveMeanPrice
0	3	2336	2004	2	2	1	2178	0
1	4	2106	2005	2	2	1	2178	0
2	3	1410	1999	1	2	0	3049	0
3	3	1769	1994	1	2	1	1742	0
4	4	2283	1999	2	3	1	2614	0
..	…	…	…	…	…	…	…	…
542	4	2780	1967	2	3	1	47480	1
543	4	3430	2013	0	4	1	43560	1
544	3	1346	1977	0	2	0	15682	0
545	5	3696	2002	2	3	1	11326	1
546	4	2491	1974	0	3	0	10019	1

[547 rows x 8 columns]

[4]:
```
dataset = df.values
```

[5]:
```
dataset
```

```
[5]: array([[    3,  2336,  2004, …,     1,  2178,     0],
        [    4,  2106,  2005, …,     1,  2178,     0],
        [    3,  1410,  1999, …,     0,  3049,     0],
        …,
        [    3,  1346,  1977, …,     0, 15682,     0],
        [    5,  3696,  2002, …,     1, 11326,     1],
        [    4,  2491,  1974, …,     0, 10019,     1]])
```

```
[6]: X = dataset[:, 0:7]
```

```
[7]: Y = dataset[:,7]
```

1.2 Normalize Data

```
[8]: from sklearn import preprocessing
```

```
[9]: min_max_scaler = preprocessing.MinMaxScaler()
X_scale = min_max_scaler.fit_transform(X)
X_scale
```

```
[9]: array([[0.4       , 0.16361181, 0.86885246, …, 0.2       , 1.        ,
         0.00172891],
        [0.6       , 0.14142389, 0.87704918, …, 0.2       , 1.        ,
         0.00172891],
        [0.4       , 0.0742813 , 0.82786885, …, 0.2       , 0.        ,
         0.00242031],
        …,
        [0.4       , 0.06810727, 0.64754098, …, 0.2       , 0.        ,
         0.01244845],
        [0.8       , 0.29480996, 0.85245902, …, 0.4       , 1.        ,
         0.00899064],
        [0.6       , 0.17856454, 0.62295082, …, 0.4       , 0.        ,
         0.00795313]])
```

1.3 Partition Data

```
[10]: from sklearn.model_selection import train_test_split
```

```
[11]: X_train, X_val_and_test, Y_train, Y_val_and_test = train_test_split(X_scale, Y,␣
 ↪test_size=0.3)
X_val, X_test, Y_val, Y_test = train_test_split(X_val_and_test, Y_val_and_test,␣
 ↪test_size=0.5)
print(X_train.shape, X_val.shape, X_test.shape, Y_train.shape, Y_val.shape,␣
 ↪Y_test.shape)
```

```
(382, 7) (82, 7) (83, 7) (382,) (82,) (83,)
```

1.4 Model Building

[12]:
```python
from keras.models import Sequential
from keras.layers import Dense
```

[13]:
```python
model = Sequential([
    Dense(16, activation='relu', input_shape=(7,)), #hidden layer  ***
    Dense(1, activation='sigmoid')]) #output layer (1 neuron)
```

[14]:
```python
model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[15]:
```python
hist = model.fit(X_train, Y_train,
          batch_size=16, epochs=100,
          validation_data=(X_val, Y_val))
```

```
Epoch 1/100
24/24 [==============================] - 0s 13ms/step - loss: 0.7133 - accuracy:
0.3691 - val_loss: 0.6971 - val_accuracy: 0.5244
Epoch 2/100
24/24 [==============================] - 0s 6ms/step - loss: 0.7079 - accuracy:
0.4005 - val_loss: 0.6923 - val_accuracy: 0.5366
Epoch 3/100
24/24 [==============================] - ETA: 0s - loss: 0.7044 - accuracy: 0.43
- 0s 19ms/step - loss: 0.7030 - accuracy: 0.4188 - val_loss: 0.6880 -
val_accuracy: 0.5488
Epoch 4/100
24/24 [==============================] - 0s 7ms/step - loss: 0.6982 - accuracy:
0.4634 - val_loss: 0.6840 - val_accuracy: 0.5976
Epoch 5/100
24/24 [==============================] - 0s 9ms/step - loss: 0.6939 - accuracy:
0.4869 - val_loss: 0.6802 - val_accuracy: 0.5976
Epoch 6/100
24/24 [==============================] - 0s 16ms/step - loss: 0.6898 - accuracy:
0.5026 - val_loss: 0.6765 - val_accuracy: 0.5854
Epoch 7/100
24/24 [==============================] - 0s 12ms/step - loss: 0.6860 - accuracy:
0.5340 - val_loss: 0.6728 - val_accuracy: 0.6341
Epoch 8/100
24/24 [==============================] - 0s 15ms/step - loss: 0.6820 - accuracy:
0.5785 - val_loss: 0.6691 - val_accuracy: 0.6585
Epoch 9/100
24/24 [==============================] - 0s 11ms/step - loss: 0.6783 - accuracy:
0.6047 - val_loss: 0.6655 - val_accuracy: 0.6707
```

```
Epoch 10/100
24/24 [==============================] - 0s 20ms/step - loss: 0.6744 - accuracy:
0.6230 - val_loss: 0.6617 - val_accuracy: 0.6707
Epoch 11/100
24/24 [==============================] - 0s 14ms/step - loss: 0.6705 - accuracy:
0.6309 - val_loss: 0.6577 - val_accuracy: 0.6707
Epoch 12/100
24/24 [==============================] - 0s 15ms/step - loss: 0.6661 - accuracy:
0.6466 - val_loss: 0.6532 - val_accuracy: 0.6951
Epoch 13/100
24/24 [==============================] - 0s 17ms/step - loss: 0.6612 - accuracy:
0.6649 - val_loss: 0.6479 - val_accuracy: 0.6951
Epoch 14/100
24/24 [==============================] - 0s 18ms/step - loss: 0.6550 - accuracy:
0.7094 - val_loss: 0.6416 - val_accuracy: 0.7561
Epoch 15/100
24/24 [==============================] - 0s 11ms/step - loss: 0.6479 - accuracy:
0.7408 - val_loss: 0.6342 - val_accuracy: 0.8415
Epoch 16/100
24/24 [==============================] - 0s 9ms/step - loss: 0.6407 - accuracy:
0.7827 - val_loss: 0.6269 - val_accuracy: 0.8659
Epoch 17/100
24/24 [==============================] - 0s 11ms/step - loss: 0.6341 - accuracy:
0.8115 - val_loss: 0.6204 - val_accuracy: 0.8902
Epoch 18/100
24/24 [==============================] - 1s 24ms/step - loss: 0.6281 - accuracy:
0.8272 - val_loss: 0.6128 - val_accuracy: 0.8902
Epoch 19/100
24/24 [==============================] - 0s 18ms/step - loss: 0.6211 - accuracy:
0.8194 - val_loss: 0.6046 - val_accuracy: 0.8780
Epoch 20/100
24/24 [==============================] - 0s 11ms/step - loss: 0.6149 - accuracy:
0.8168 - val_loss: 0.5982 - val_accuracy: 0.8780
Epoch 21/100
24/24 [==============================] - 0s 12ms/step - loss: 0.6096 - accuracy:
0.8220 - val_loss: 0.5930 - val_accuracy: 0.8780
Epoch 22/100
24/24 [==============================] - 0s 10ms/step - loss: 0.6047 - accuracy:
0.8220 - val_loss: 0.5883 - val_accuracy: 0.8780
Epoch 23/100
24/24 [==============================] - 0s 12ms/step - loss: 0.6002 - accuracy:
0.8168 - val_loss: 0.5835 - val_accuracy: 0.8902
Epoch 24/100
24/24 [==============================] - 0s 9ms/step - loss: 0.5958 - accuracy:
0.8194 - val_loss: 0.5786 - val_accuracy: 0.8902
Epoch 25/100
24/24 [==============================] - 0s 12ms/step - loss: 0.5914 - accuracy:
0.8272 - val_loss: 0.5740 - val_accuracy: 0.8902
```

```
Epoch 26/100
24/24 [==============================] - 0s 9ms/step - loss: 0.5871 - accuracy:
0.8325 - val_loss: 0.5694 - val_accuracy: 0.8902
Epoch 27/100
24/24 [==============================] - 0s 13ms/step - loss: 0.5827 - accuracy:
0.8298 - val_loss: 0.5646 - val_accuracy: 0.8902
Epoch 28/100
24/24 [==============================] - 0s 6ms/step - loss: 0.5784 - accuracy:
0.8351 - val_loss: 0.5601 - val_accuracy: 0.8902
Epoch 29/100
24/24 [==============================] - 0s 7ms/step - loss: 0.5741 - accuracy:
0.8272 - val_loss: 0.5551 - val_accuracy: 0.8902
Epoch 30/100
24/24 [==============================] - 0s 11ms/step - loss: 0.5697 - accuracy:
0.8351 - val_loss: 0.5503 - val_accuracy: 0.9024
Epoch 31/100
24/24 [==============================] - 0s 12ms/step - loss: 0.5653 - accuracy:
0.8325 - val_loss: 0.5456 - val_accuracy: 0.9024
Epoch 32/100
24/24 [==============================] - 0s 20ms/step - loss: 0.5609 - accuracy:
0.8403 - val_loss: 0.5408 - val_accuracy: 0.9024
Epoch 33/100
24/24 [==============================] - 0s 14ms/step - loss: 0.5566 - accuracy:
0.8377 - val_loss: 0.5362 - val_accuracy: 0.9024
Epoch 34/100
24/24 [==============================] - 0s 18ms/step - loss: 0.5523 - accuracy:
0.8429 - val_loss: 0.5314 - val_accuracy: 0.9024
Epoch 35/100
24/24 [==============================] - 0s 18ms/step - loss: 0.5480 - accuracy:
0.8482 - val_loss: 0.5268 - val_accuracy: 0.9024
Epoch 36/100
24/24 [==============================] - 0s 19ms/step - loss: 0.5435 - accuracy:
0.8429 - val_loss: 0.5224 - val_accuracy: 0.9024
Epoch 37/100
24/24 [==============================] - 0s 6ms/step - loss: 0.5393 - accuracy:
0.8455 - val_loss: 0.5176 - val_accuracy: 0.9024
Epoch 38/100
24/24 [==============================] - 0s 5ms/step - loss: 0.5348 - accuracy:
0.8482 - val_loss: 0.5129 - val_accuracy: 0.9024
Epoch 39/100
24/24 [==============================] - 0s 12ms/step - loss: 0.5302 - accuracy:
0.8482 - val_loss: 0.5085 - val_accuracy: 0.9146
Epoch 40/100
24/24 [==============================] - 0s 9ms/step - loss: 0.5260 - accuracy:
0.8534 - val_loss: 0.5036 - val_accuracy: 0.9146
Epoch 41/100
24/24 [==============================] - 0s 8ms/step - loss: 0.5217 - accuracy:
0.8560 - val_loss: 0.4987 - val_accuracy: 0.9146
```

```
Epoch 42/100
24/24 [==============================] - 0s 6ms/step - loss: 0.5173 - accuracy:
0.8534 - val_loss: 0.4943 - val_accuracy: 0.9146
Epoch 43/100
24/24 [==============================] - 0s 13ms/step - loss: 0.5130 - accuracy:
0.8586 - val_loss: 0.4895 - val_accuracy: 0.9146
Epoch 44/100
24/24 [==============================] - 0s 10ms/step - loss: 0.5086 - accuracy:
0.8534 - val_loss: 0.4848 - val_accuracy: 0.9146
Epoch 45/100
24/24 [==============================] - 1s 22ms/step - loss: 0.5043 - accuracy:
0.8665 - val_loss: 0.4810 - val_accuracy: 0.9268
Epoch 46/100
24/24 [==============================] - 0s 16ms/step - loss: 0.4998 - accuracy:
0.8639 - val_loss: 0.4761 - val_accuracy: 0.9268
Epoch 47/100
24/24 [==============================] - 0s 15ms/step - loss: 0.4957 - accuracy:
0.8665 - val_loss: 0.4711 - val_accuracy: 0.9268
Epoch 48/100
24/24 [==============================] - 0s 10ms/step - loss: 0.4914 - accuracy:
0.8665 - val_loss: 0.4655 - val_accuracy: 0.9268
Epoch 49/100
24/24 [==============================] - 0s 16ms/step - loss: 0.4874 - accuracy:
0.8639 - val_loss: 0.4615 - val_accuracy: 0.9268
Epoch 50/100
24/24 [==============================] - 0s 16ms/step - loss: 0.4831 - accuracy:
0.8691 - val_loss: 0.4571 - val_accuracy: 0.9390
Epoch 51/100
24/24 [==============================] - 0s 7ms/step - loss: 0.4791 - accuracy:
0.8691 - val_loss: 0.4523 - val_accuracy: 0.9390
Epoch 52/100
24/24 [==============================] - 0s 17ms/step - loss: 0.4749 - accuracy:
0.8743 - val_loss: 0.4477 - val_accuracy: 0.9390
Epoch 53/100
24/24 [==============================] - 0s 11ms/step - loss: 0.4706 - accuracy:
0.8717 - val_loss: 0.4440 - val_accuracy: 0.9390
Epoch 54/100
24/24 [==============================] - 0s 14ms/step - loss: 0.4670 - accuracy:
0.8717 - val_loss: 0.4394 - val_accuracy: 0.9390
Epoch 55/100
24/24 [==============================] - 0s 15ms/step - loss: 0.4628 - accuracy:
0.8770 - val_loss: 0.4355 - val_accuracy: 0.9390
Epoch 56/100
24/24 [==============================] - 0s 8ms/step - loss: 0.4590 - accuracy:
0.8717 - val_loss: 0.4307 - val_accuracy: 0.9390
Epoch 57/100
24/24 [==============================] - 0s 14ms/step - loss: 0.4548 - accuracy:
0.8848 - val_loss: 0.4273 - val_accuracy: 0.9268
```

```
Epoch 58/100
24/24 [==============================] - 0s 8ms/step - loss: 0.4513 - accuracy:
0.8770 - val_loss: 0.4234 - val_accuracy: 0.9268
Epoch 59/100
24/24 [==============================] - 0s 9ms/step - loss: 0.4475 - accuracy:
0.8743 - val_loss: 0.4190 - val_accuracy: 0.9268
Epoch 60/100
24/24 [==============================] - 0s 5ms/step - loss: 0.4436 - accuracy:
0.8743 - val_loss: 0.4140 - val_accuracy: 0.9268
Epoch 61/100
24/24 [==============================] - 0s 10ms/step - loss: 0.4402 - accuracy:
0.8770 - val_loss: 0.4095 - val_accuracy: 0.9268
Epoch 62/100
24/24 [==============================] - 0s 5ms/step - loss: 0.4365 - accuracy:
0.8796 - val_loss: 0.4059 - val_accuracy: 0.9268
Epoch 63/100
24/24 [==============================] - 0s 6ms/step - loss: 0.4327 - accuracy:
0.8796 - val_loss: 0.4029 - val_accuracy: 0.9268
Epoch 64/100
24/24 [==============================] - 0s 6ms/step - loss: 0.4294 - accuracy:
0.8796 - val_loss: 0.3992 - val_accuracy: 0.9268
Epoch 65/100
24/24 [==============================] - 0s 5ms/step - loss: 0.4260 - accuracy:
0.8796 - val_loss: 0.3951 - val_accuracy: 0.9268
Epoch 66/100
24/24 [==============================] - 0s 6ms/step - loss: 0.4226 - accuracy:
0.8796 - val_loss: 0.3900 - val_accuracy: 0.9268
Epoch 67/100
24/24 [==============================] - 0s 8ms/step - loss: 0.4192 - accuracy:
0.8822 - val_loss: 0.3867 - val_accuracy: 0.9268
Epoch 68/100
24/24 [==============================] - 0s 8ms/step - loss: 0.4161 - accuracy:
0.8770 - val_loss: 0.3829 - val_accuracy: 0.9268
Epoch 69/100
24/24 [==============================] - 0s 8ms/step - loss: 0.4128 - accuracy:
0.8796 - val_loss: 0.3796 - val_accuracy: 0.9268
Epoch 70/100
24/24 [==============================] - 0s 6ms/step - loss: 0.4097 - accuracy:
0.8743 - val_loss: 0.3761 - val_accuracy: 0.9268
Epoch 71/100
24/24 [==============================] - 0s 8ms/step - loss: 0.4065 - accuracy:
0.8770 - val_loss: 0.3730 - val_accuracy: 0.9268
Epoch 72/100
24/24 [==============================] - 0s 5ms/step - loss: 0.4033 - accuracy:
0.8796 - val_loss: 0.3690 - val_accuracy: 0.9268
Epoch 73/100
24/24 [==============================] - 0s 6ms/step - loss: 0.4006 - accuracy:
0.8822 - val_loss: 0.3658 - val_accuracy: 0.9268
```

```
Epoch 74/100
24/24 [==============================] - 0s 7ms/step - loss: 0.3974 - accuracy:
0.8796 - val_loss: 0.3622 - val_accuracy: 0.9268
Epoch 75/100
24/24 [==============================] - 0s 6ms/step - loss: 0.3945 - accuracy:
0.8848 - val_loss: 0.3585 - val_accuracy: 0.9268
Epoch 76/100
24/24 [==============================] - 0s 9ms/step - loss: 0.3918 - accuracy:
0.8848 - val_loss: 0.3557 - val_accuracy: 0.9268
Epoch 77/100
24/24 [==============================] - 0s 6ms/step - loss: 0.3890 - accuracy:
0.8796 - val_loss: 0.3524 - val_accuracy: 0.9268
Epoch 78/100
24/24 [==============================] - 0s 5ms/step - loss: 0.3862 - accuracy:
0.8822 - val_loss: 0.3506 - val_accuracy: 0.9268
Epoch 79/100
24/24 [==============================] - 0s 13ms/step - loss: 0.3837 - accuracy:
0.8848 - val_loss: 0.3473 - val_accuracy: 0.9268
Epoch 80/100
24/24 [==============================] - 0s 6ms/step - loss: 0.3809 - accuracy:
0.8822 - val_loss: 0.3438 - val_accuracy: 0.9268
Epoch 81/100
24/24 [==============================] - 0s 8ms/step - loss: 0.3786 - accuracy:
0.8848 - val_loss: 0.3407 - val_accuracy: 0.9268
Epoch 82/100
24/24 [==============================] - 0s 8ms/step - loss: 0.3760 - accuracy:
0.8822 - val_loss: 0.3383 - val_accuracy: 0.9268
Epoch 83/100
24/24 [==============================] - 0s 7ms/step - loss: 0.3734 - accuracy:
0.8796 - val_loss: 0.3357 - val_accuracy: 0.9268
Epoch 84/100
24/24 [==============================] - 0s 8ms/step - loss: 0.3711 - accuracy:
0.8822 - val_loss: 0.3334 - val_accuracy: 0.9268
Epoch 85/100
24/24 [==============================] - 0s 8ms/step - loss: 0.3689 - accuracy:
0.8796 - val_loss: 0.3302 - val_accuracy: 0.9268
Epoch 86/100
24/24 [==============================] - 0s 9ms/step - loss: 0.3663 - accuracy:
0.8848 - val_loss: 0.3273 - val_accuracy: 0.9268
Epoch 87/100
24/24 [==============================] - 0s 6ms/step - loss: 0.3643 - accuracy:
0.8796 - val_loss: 0.3245 - val_accuracy: 0.9268
Epoch 88/100
24/24 [==============================] - 0s 6ms/step - loss: 0.3620 - accuracy:
0.8848 - val_loss: 0.3227 - val_accuracy: 0.9268
Epoch 89/100
24/24 [==============================] - 0s 8ms/step - loss: 0.3597 - accuracy:
0.8848 - val_loss: 0.3212 - val_accuracy: 0.9268
```

```
Epoch 90/100
24/24 [==============================] - 0s 7ms/step - loss: 0.3576 - accuracy:
0.8822 - val_loss: 0.3176 - val_accuracy: 0.9268
Epoch 91/100
24/24 [==============================] - 0s 8ms/step - loss: 0.3558 - accuracy:
0.8901 - val_loss: 0.3154 - val_accuracy: 0.9268
Epoch 92/100
24/24 [==============================] - 0s 6ms/step - loss: 0.3534 - accuracy:
0.8874 - val_loss: 0.3127 - val_accuracy: 0.9268
Epoch 93/100
24/24 [==============================] - ETA: 0s - loss: 0.3579 - accuracy: 0.88
- 0s 8ms/step - loss: 0.3517 - accuracy: 0.8874 - val_loss: 0.3102 -
val_accuracy: 0.9268
Epoch 94/100
24/24 [==============================] - 0s 7ms/step - loss: 0.3497 - accuracy:
0.8874 - val_loss: 0.3093 - val_accuracy: 0.9268
Epoch 95/100
24/24 [==============================] - 0s 7ms/step - loss: 0.3477 - accuracy:
0.8874 - val_loss: 0.3068 - val_accuracy: 0.9268
Epoch 96/100
24/24 [==============================] - 0s 6ms/step - loss: 0.3458 - accuracy:
0.8927 - val_loss: 0.3051 - val_accuracy: 0.9268
Epoch 97/100
24/24 [==============================] - 0s 11ms/step - loss: 0.3444 - accuracy:
0.8848 - val_loss: 0.3021 - val_accuracy: 0.9268
Epoch 98/100
24/24 [==============================] - 0s 11ms/step - loss: 0.3420 - accuracy:
0.8874 - val_loss: 0.2990 - val_accuracy: 0.9268
Epoch 99/100
24/24 [==============================] - 0s 10ms/step - loss: 0.3401 - accuracy:
0.8874 - val_loss: 0.2960 - val_accuracy: 0.9268
Epoch 100/100
24/24 [==============================] - 0s 8ms/step - loss: 0.3389 - accuracy:
0.8874 - val_loss: 0.2946 - val_accuracy: 0.9268
```

You have a strong model if your test accuracy is between 80% and 95%. * Above 95%, you likely overfit the model. * Below 80%, the model didn't capture enough of the data's variability.

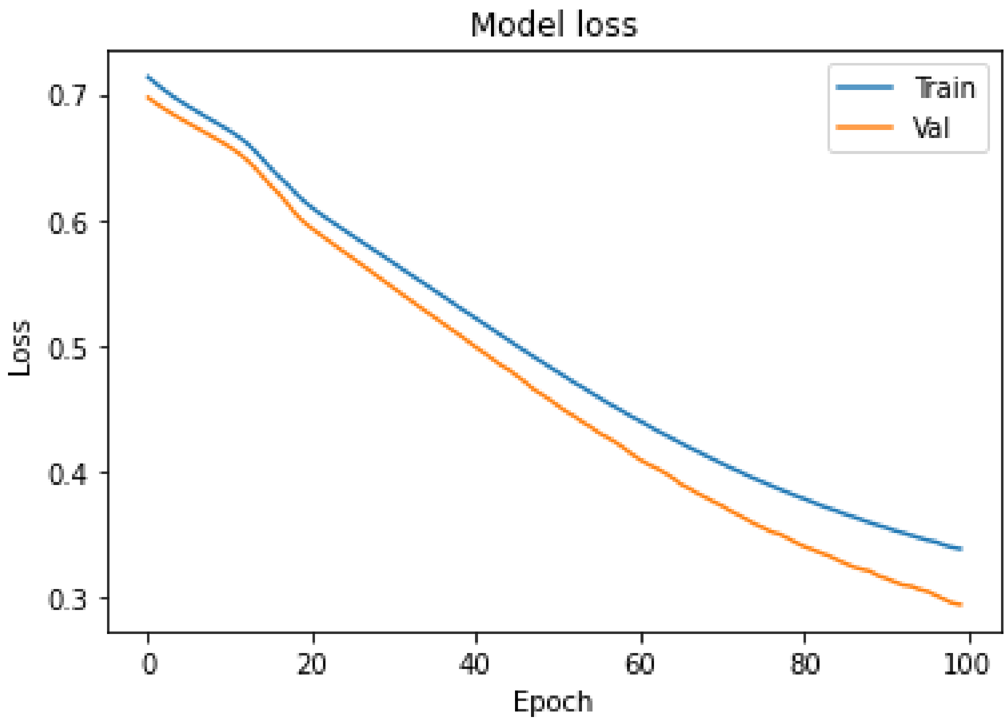
[16]:
```python
model.evaluate(X_test, Y_test)[1] #0 = loss, 1 = accuracy
```

```
3/3 [==============================] - 0s 3ms/step - loss: 0.3672 - accuracy:
0.8072
```

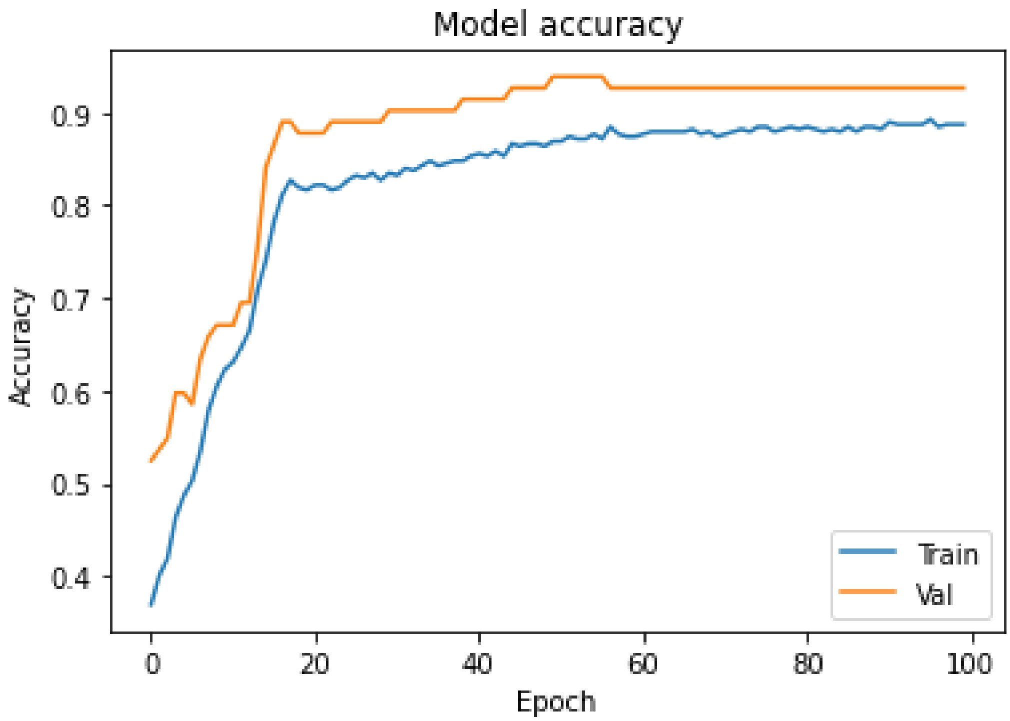
[16]: 0.8072289228439331

[17]:
```python
import matplotlib.pyplot as plt
```

[18]:
```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```



[19]:
```
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()
```



1.5 Making Predictions

Predict whether a house with the following characteristics is above or below the average price. * Bedrooms: 4 * Square Feet: 2,500 * Year Built: 2001 * Garage Spaces: 2 * Full Bathrooms: 3 * Half Bathrooms: 1 * Lot Size: 3,452 Square Feet

[20]:
```
x = [[4, 2500, 2001, 2, 3, 1, 3452]]
print(model.predict(x)[0])
```

```
[1.]
```

2 Summary

- Keras is an easy to use, well-known library for constructing neural networks in Python.
- It was designed with a user-friendly API with easy to decipher error messages.
- Neural Networks emulate human pattern-recognition skills to analyze large datasets (particularly nonlinear responses).

[ ]:
```
```