2019.07.08

House price prediction

Youngtaek Hong, PhD

By using regression

회귀 (regression)

- 앞의 두 예제는 분류 문제classification 였습니다.
- 개별적인 레이블 대신에 연속적인 값을 예측하는 회귀(regression)입니다.
- 예를 들어 기상 데이터가 주어졌을 때 내일 기온을 예측
- 회귀와 로지스틱 회귀 알고리즘을 혼돈하지 마세요. 로지스틱 회귀는 회귀 알고리즘이 아니라 분류 알고리즘입니다.

Boston House Price

- 1970년 중반 보스턴 외곽 지역의 범죄율, 지방세율 등의 데이터가 주어졌을 때 주택 가격의 중간 값을 예측해 보겠습니다.
- 여기서 사용할 데이터셋은 이전 두 개의 예제와 다릅니다. 데이터 포인트가 506개로 비교적 개수가 적고 404개는 훈련 샘플로 102개는 테스트 샘플로 나누어져 있습니다.
- 입력 데이터에 있는 각 특성(예를 들어 범죄율)은 스케일이 서로 다릅니다. 어떤 값은 0과 1 사이의 비율을 나타내고 어떤 것은 1과 12 사이의 값을 가지거나 1과 100 사이의 값을 가집니다.

Loading the dataset

```
from keras.datasets import boston_housing

(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()
```

```
Using TensorFlow backend.


Downloading data from https://s3.amazonaws.com/keras-datasets/boston_housing.npz
57344/57026 [==============================] - 0s 0us/step
```

Check the data

```
# take a look at the data

print(f'Training data : {train_data.shape}')
print(f'Test data : {test_data.shape}')
print(f'Training sample : {train_data[0]}')
print(f'Training target sample : {train_targets[0]}')
```

```
Training data : (404, 13)
Test data : (102, 13)
Training sample : [  1.23247   0.        8.14      0.        0.538     6.142    91.7
   3.9769    4.      307.       21.      396.9      18.72   ]
Training target sample : 15.2
```

Preparing the data

We are going to do a feature normalization . Feature normalizaion is when you subtract the mean of the feature from each feature and divide each result by the standard deviation.

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std

# Note that the quantities used for normalizing the test data are computed using the
# training data. You should never use in your workflow any quantity computed on the
# test data, even for something as simple as data normalization.
```

Building the network

```
from keras import models
from keras import layers

def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu', input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))

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

K-fold Validation

```
k = 4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []

for i in range(k):
    print(f'Processing fold # {i}')
    val_data = train_data[i * num_val_samples: (i+1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i+1) * num_val_samples]

    partial_train_data = np.concatenate(
                         [train_data[:i * num_val_samples],
                         train_data[(i+1) * num_val_samples:]],
                         axis=0)
    partial_train_targets = np.concatenate(
                         [train_targets[:i * num_val_samples],
                         train_targets[(i+1)*num_val_samples:]],
                         axis=0)
    model = build_model()
    model.fit(partial_train_data,
              partial_train_targets,
              epochs=num_epochs,
              batch_size=1,
              verbose=0)
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)
```

Evaluation

```
print(f'all_scores : {all_scores}')
print(f'mean all scores : {np.mean(all_scores)}')
```

```
all_scores : [2.0419103780595385, 2.047238885766209, 2.907045543783962, 2.3685315398886653]
mean all scores : 2.341181586874594
```

Increase epochs to 80

```
model = build_model()
model.fit(train_data, train_targets, epochs=80, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

```
102/102 [==============================] - 0s 628us/step
```