3장 *Getting started with neural networks*

“기회와 준비가 만났을 때 … ”

6. Predicting house prices: a regression example

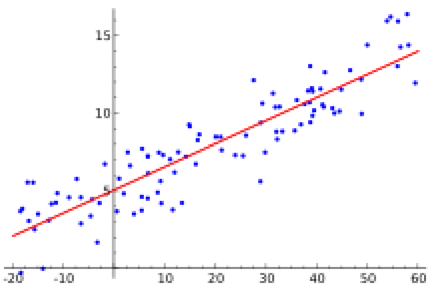
<-> Classification

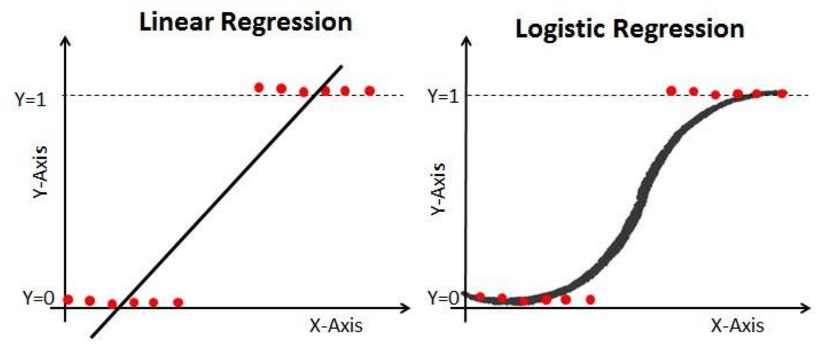
회귀 모델

- The two previous examples - predict a single discrete label of an input data point.
- Another common type of machine-learning problem is *regression* - predicting a continuous value: temperature tomorrow, time, or price

NOTE Don't confuse *regression* and the algorithm *logistic regression*. Confusingly, logistic regression isn't a regression algorithm—it's a classification algorithm.

regression 이라고해서 다 regression은 아니고
Linear나 Logistic이 붙으면 Classification이다!





6. Predicting house prices: a regression example

3.6.1 The Boston Housing Price dataset

- predict the median price of homes in a given Boston suburb with the crime rate in the mid-1970s
- only 506 samples - 404 training samples and 102 test samples.
- And each *feature* in the input data (for example, the crime rate, the local property tax rate) has a different scale. For instance, some values are proportions, which take values between 0 and 1; others take values between 1 and 12, others between 0 and 100, and so on.
- **Listing 3.24 Loading the Boston housing**

```
from keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets)
        =boston_housing.load_data()
```

- Let's look at the data:

```
>>> train_data.shape
(404, 13) # 13 features
>>> test_data.shape
(102, 13)
```

6. Predicting house prices: a regression example

3.6.1 The Boston Housing Price dataset

- As you can see, you have 404 training samples and 102 test samples, each with 13 numerical features - capita crime rate, average number of rooms per dwelling, accessibility to highways, and so on.

- The targets are the median values of owner-occupied homes, in thousands of dollars:

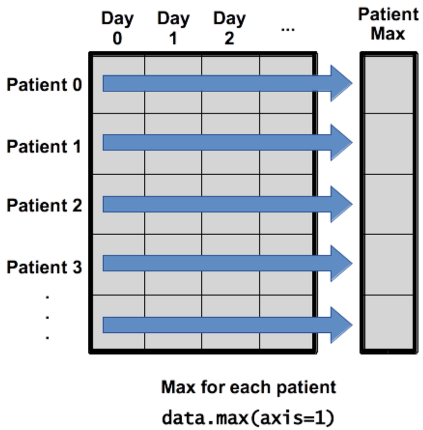
```
>>> train_targets
[15.2, 42.3, 50. ...  19.4, 19.4, 29.1]# 404 in $(*1000)
```

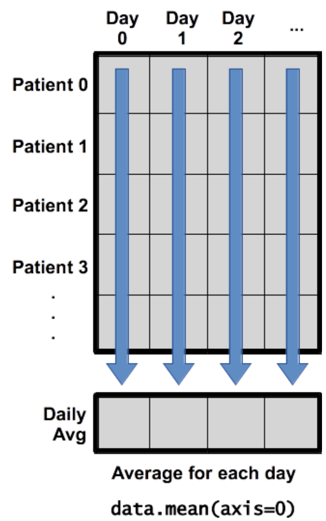
- The prices are typically between $10,000 and $50,000

6. Predicting house prices: a regression example

3.6.2 Preparing the data

- feature-wise normalization: feature is centered around 0 and has a unit standard deviation.

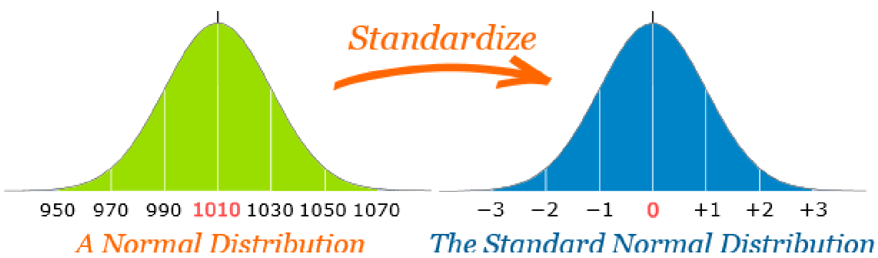


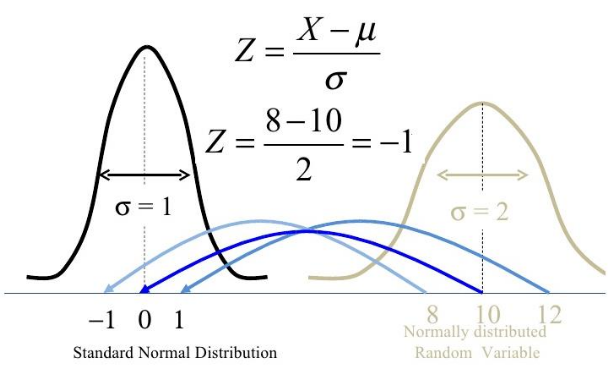


Listing 3.25 Normalizing the data

```
mean = train_data.mean(axis=0)#(404,13)
train_data -= mean
std = train_data.std(axis=0) #(102,13)
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.





6. Predicting house prices: a regression example

3.6.3 Building your network

- small samples - small network with two hidden layers, each with 64 units.
- less training data - worse overfitting, a small network is one way to mitigate overfitting.

Listing 3.26 Model definition

```
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
```

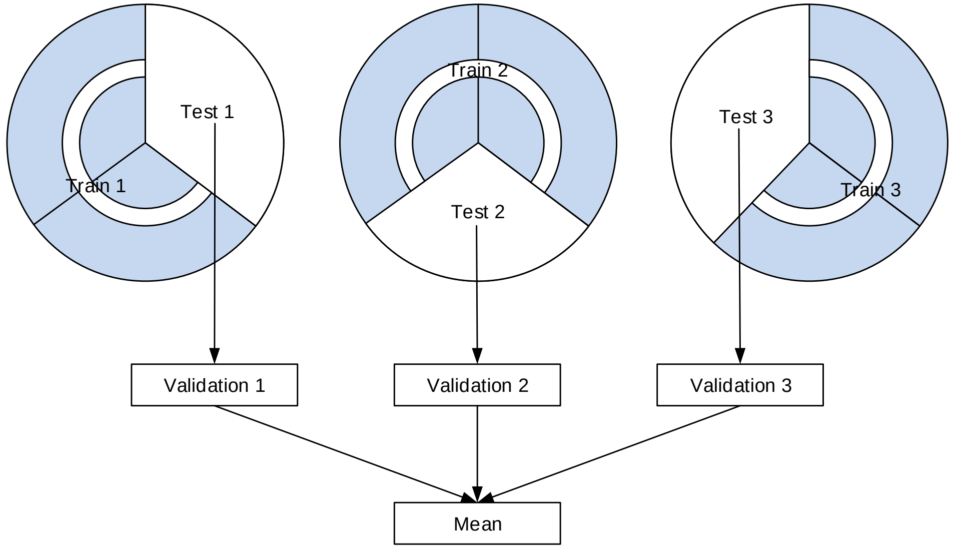
- output - a single unit and no activation (it will be a linear layer) for scalar regression to predict a single continuous value, free to learn to predict values in any range
- `sigmoid` activation function - predict values between 0 and 1
- *mean absolute error* (MAE) loss function for regression problems - 0.5 → off by $500 on average.

3.6.4 Validating your approach using K-fold validation

매우 중요(Validation하는 방법)

- very small validation set (for instance, about 100 examples) - high *variance* with regard to the validation split
- *K-fold* cross-validation (see figure) - splitting the available data into K partitions (typically K = 4 or 5), instantiating K identical models, and training each one on K – 1 partitions while evaluating on the remaining partition.
- The validation score for the model - average of the K validation scores

3-fold cross-validation



3.6.4 Validating your approach using K-fold validation

Listing 3.27 K-fold validation

```
import numpy as np
k = 4
num_val_samples = len(train_data) // k  # 나눗셈의 몫
num_epochs = 100
all_scores = []
for i in range(k):  # i = 0,1,2,3
   print('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()
   history = model.fit(partial_train_data, partial_train_targets,
        validation_data=(val_data, val_targets), epochs=num_epochs,
        batch_size=1, verbose=0)
   mae_history = history.history['val_mean_absolute_error']
   all_mae_histories.append(mae_history)
```

6. Predicting house prices: a regression example

3.6.4 Validating your approach using K-fold validation

- Running this with `num_epochs = 100` yields the following results:

```
>>> all_scores
[2.588258957792037, 3.1289568449719116, 3.1856116051248984, 3.0763342615401386]
>>> np.mean(all_scores)
2.9947904173572462
```

- different validation scores, from 2.6 to 3.2.
- The average (3.0) is a much more reliable metric K-fold cross-validation
- $3,000 on average - significant considering with $10,000 to $50,000.

6. Predicting house prices: a regression example

3.6.4 Validating your approach using K-fold validation

- Let's try training the network a bit longer: 500 epochs. To keep a record of how well the model does at each epoch, you'll modify the training loop to save the per-epoch validation score log.

Listing 3.28 Saving the validation logs

```
num_epochs = 500
all_mae_histories = []
for i in range(k):
    print('처리중인 폴드 #', 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()
    history = model.fit(partial_train_data, partial_train_targets,
                        validation_data=(val_data, val_targets),
                        epochs=num_epochs, batch_size=1, verbose=0)
    mae_history = history.history['val_mean_absolute_error']
    all_mae_histories.append(mae_history)
```

6. Predicting house prices: a regression example

3.6.4 Validating your approach using K-fold validation

- the average of the per-epoch MAE scores for all folds.

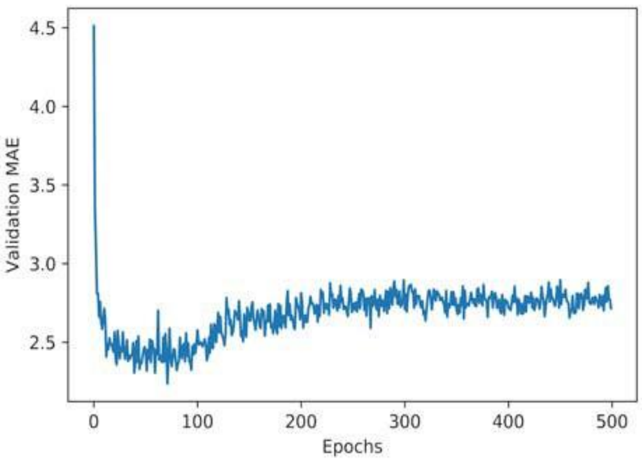
Listing 3.29 Building the history of successive

```
average_mae_history = [np.mean([x[i] for x in all_mae_histories])
    for i in range(num_epochs)]
```

Let's plot this; see figure 3.12.

Listing 3.30 Plotting validation scores

```
import matplotlib.pyplot as plt
plt.plot(range(1,len(average_mae_history)+1),
    average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



이 다음에 그래프 나오는 ppt 한장이 짤림

Figure 3.12 Validation MAE by epoch

6. Predicting house prices: a regression example

- validation MAE stops improving significantly after 80 epochs.
- adjust the size of the hidden layers, and then look at its performance on the test data.

Listing 3.32 Training the final model

```
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)
```

- Here's the final result:

```
>>> test_mae_score 2.5532484335057877
```

- You're still off by about $2,550

6. Predicting house prices: a regression example

3.6.5 Wrapping up

- Here's what you should take away from this example:
- Mean squared error (MSE) is a loss function commonly used for regression.
- The concept of accuracy doesn't apply for regression. A common regression metric is mean absolute error (MAE). output 노드의 개수는 1개, activation 함수 따로 없음
- When features in the input data have values in different ranges, each feature should be scaled independently as a preprocessing step.
- When there is little data available, using K-fold validation is a great way to reliably evaluate a model.
- When little training data is available, it's preferable to use a small network with few hidden layers (typically only 1 or 2), in order to avoid severe overfitting.

Chapter summary

- binary classification, multiclass classification, and scalar regression
- preprocess raw data before feeding it into a neural network.
- features with different ranges, scale each feature independently
- As training progresses, neural networks eventually begin to overfit on never-before-seen data.
- If you have small training data, use a small network with only one or two hidden layers, to avoid severe overfitting.
- If your data is divided into many categories, you may cause information bottlenecks if you make the intermediate layers too small.
- Regression uses different loss functions and different evaluation metrics than classification.
- When you're working with little data, K-fold validation can help reliably evaluate your model.