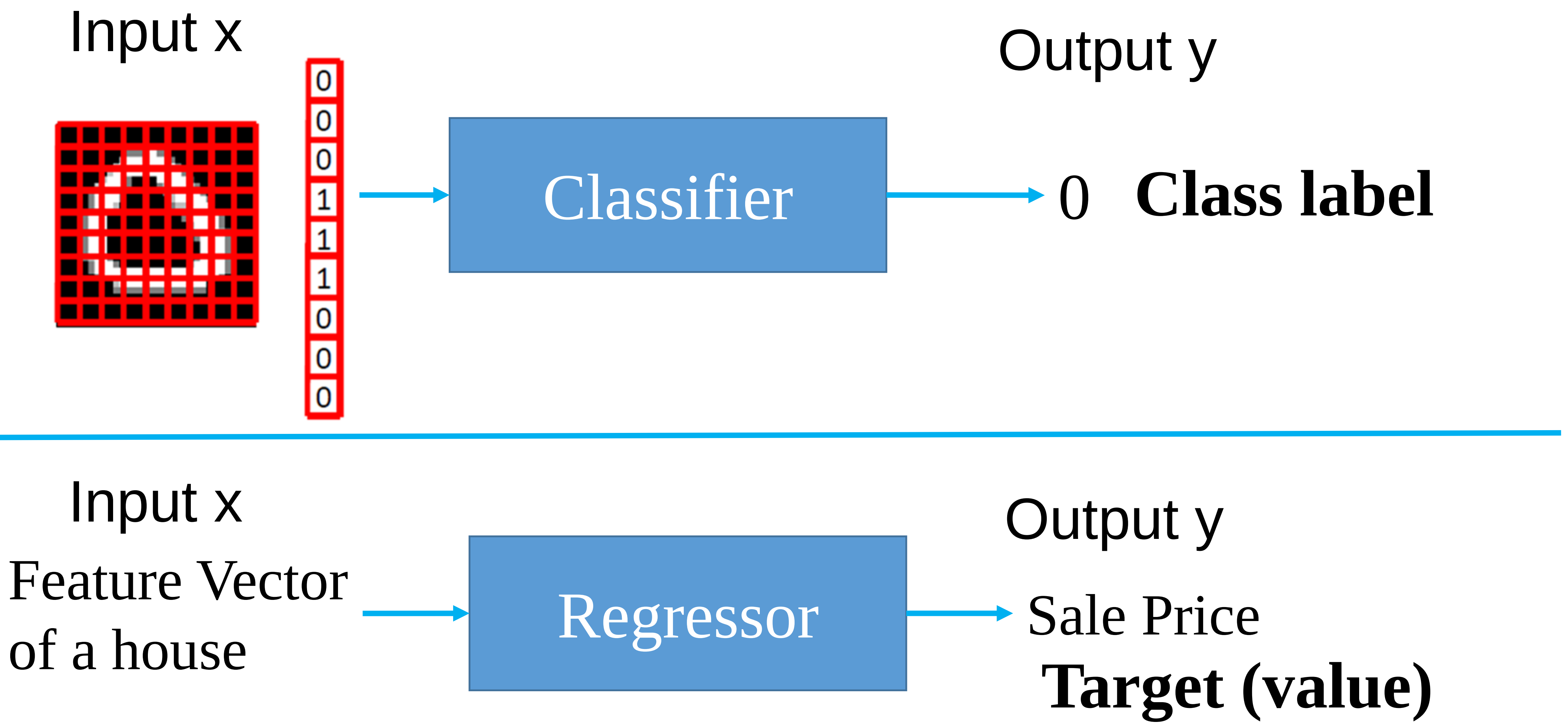
K-Nearest Neighbor (KNN) Classifier and Regressor

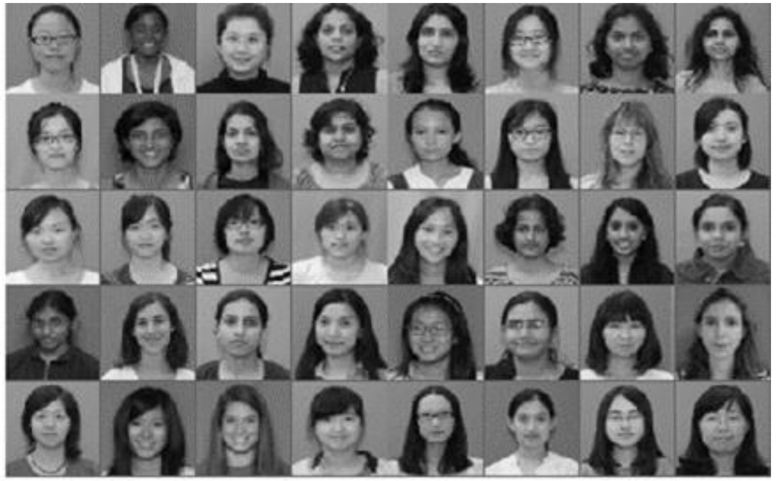
Liang Liang

Supervised Learning: classification and regression



Binary Classification

- Data points are from two classes. A data point only belongs to one class.



Classifier

label of the data point x

$y = 0$ male

$y = 1$ female

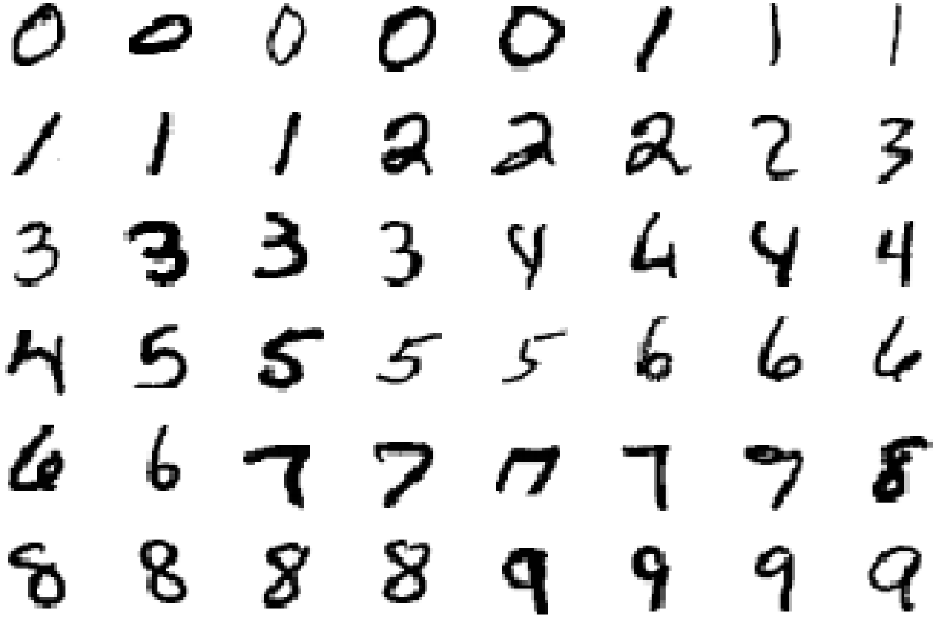
or

$y = -1$ male

$y = 1$ female

Multiclass Classification

- Data points are from many classes.
- A data point only belongs to one class.



Classifier

label of the data point x

label = ?

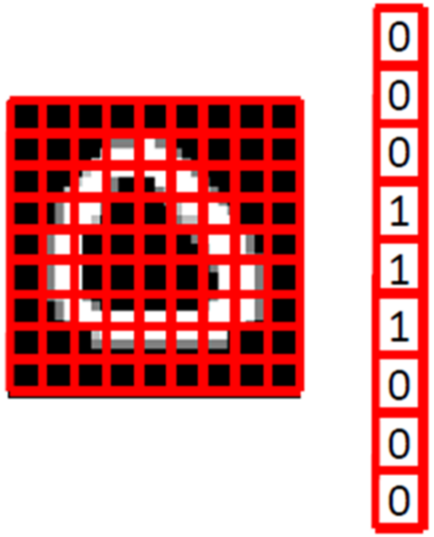
10 possible labels:

(0, 1, 2, 3, 4, 5 ,6, 7, 8, 9)

Multiclass Classification

$$y = g(x)$$

$$x = \begin{pmatrix} x_{(1)} \\ x_{(2)} \\ \cdots \\ x_{(M)} \end{pmatrix} \longrightarrow \boxed{\text{Classifier}} \longrightarrow y$$



y is the *class label* of the data point x

$y = 0$

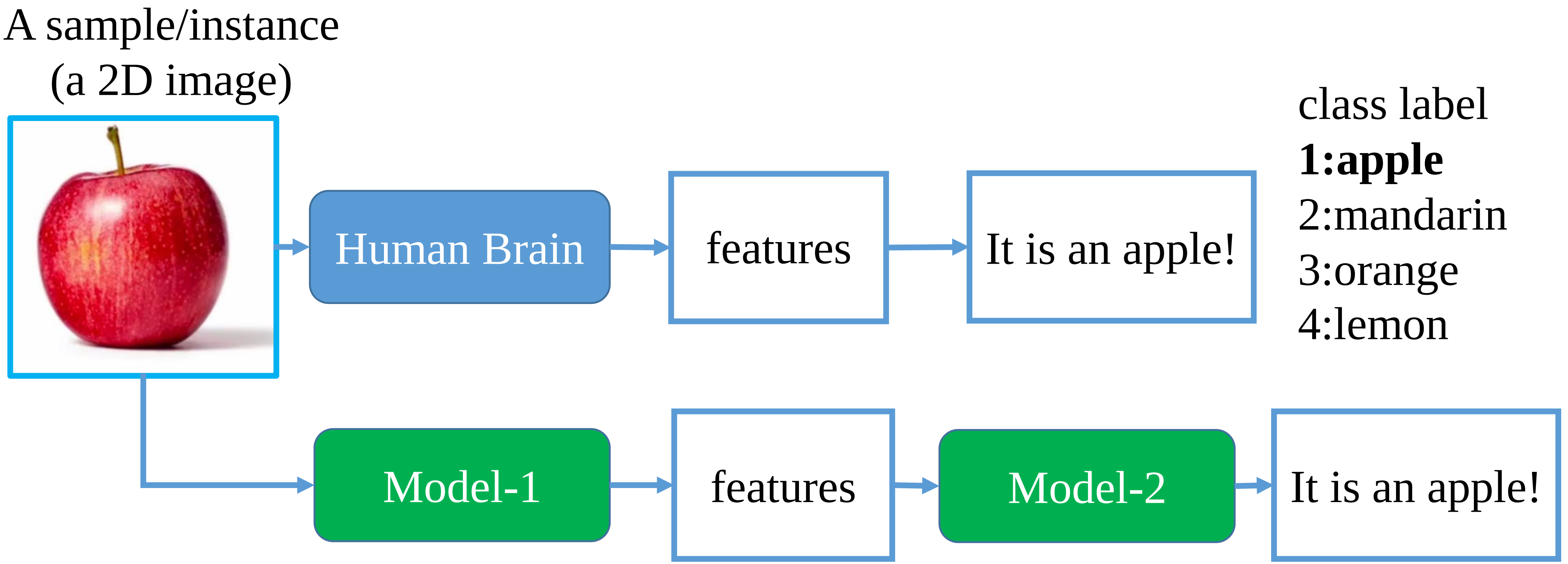
A Classification Task

- A simple task: classify a fruit into four classes/categories {1:apple, 2:mandarin, 3:orange, 4:lemon}, note: class-3 contains oranges that are not mandarin oranges

A sample/instance

perform classification

class/fruit label
1:apple
2:mandarin
3:orange
4:lemon



It is not easy to develop Model-1 for feature extraction

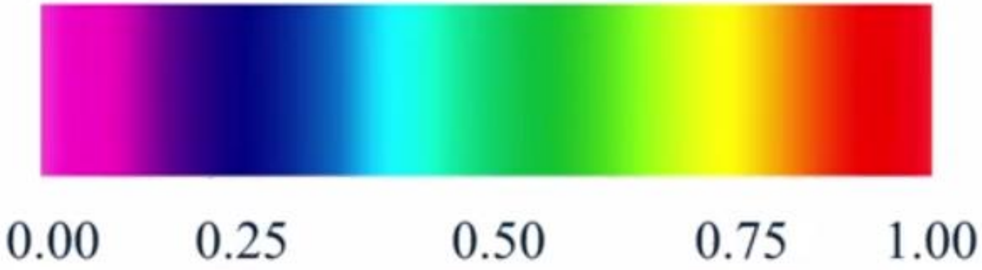
It is relatively easy to develop Model-2 for classification, given the features of the sample.

Now, let's develop Model-2 for classification.

1:apple
2:mandarin
3:orange
4:lemon

The feature vector of a fruit sample: [width, height, color_score]

color_score is a number (0~1) to describe the color



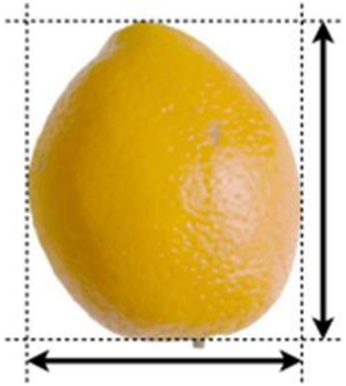
Color category	color_score
Red	0.85 - 1.00
Orange	0.75 - 0.85
Yellow	0.65 - 0.75
Green	0.45 - 0.65

The Fruit Dataset

A bucket of fruits

The fruit dataset was created by Dr. Iain Murray at the University of Edinburgh. He bought a few dozen oranges, lemons and apples, and recorded their features in a table.

4 classes: {1:apple, 2:mandarin, 3:orange, 4:lemon}



The fruit dataset (a table)

Each row contains the information of a fruit sample/instance

fruit label	fruit_name	subtype	mass (g)	width (cm)	height (cm)	color_score
1	apple	granny_smith	192	8.4	7.3	0.55
4	lemon	spanish_belsan	194	7.2	10.3	0.70

http://usapple.org/the-industry/apple-varieties/

In total, there are 59 fruit samples (i.e. 59 rows) in the table

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79
5	2	mandarin	mandarin	80	5.8	4.3	0.77
6	2	mandarin	mandarin	80	5.9	4.3	0.81
7	2	mandarin	mandarin	76	5.8	4.0	0.81
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn	172	7.4	7.0	0.89
10	1	apple	braeburn	166	6.9	7.3	0.93

4 classes: {1:apple, 2:mandarin, 3:orange, 4:lemon}

Select 3 features: width, height, color_score

```
fruits = pd.read_table('fruit_data_with_colors.txt')
```

```
fruits
```

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60

```
fruits.shape
```

```
(59, 7)
```

```
features = fruits.columns[-3:].tolist()
```

```
features
```

```
['width', 'height', 'color_score']
```

Split data (59) into a training set (80%, 47) and a testing set (20%, 12)

```
1  X = fruits[features]
2  Y = fruits['fruit_label']
3  X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2
```

```
1  X_train.shape
```

```
(47, 3)
```

X_train contains the features of the 47 training samples
Each row of X_train is a feature vector of a training sample.

```
1  Y_train.shape
```

```
(47,)
```

Y_train contains the class/fruit labels of the 47 training samples
Each element of Y_train is a class label of a training sample.

```
1  X_test.shape
```

```
(12, 3)
```

X_test contains the features of the 12 testing samples
Each row of X_test is a feature vector of a testing sample.

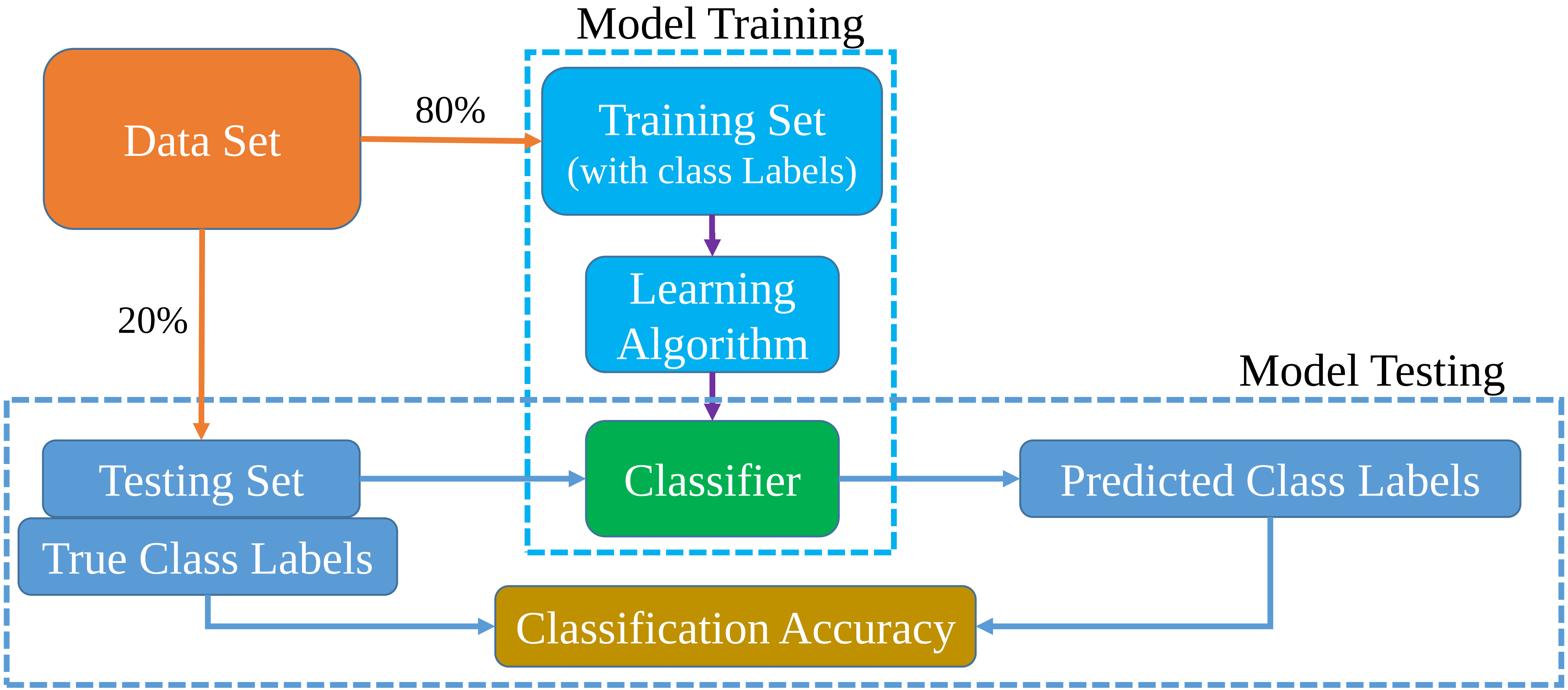
```
1  Y_test.shape
```

```
(12,)
```

Y_test contains the class/fruit labels of the 12 testing samples
Each element of Y_test is a class label of a testing sample.

The flowchart of a classification study

- Classification is a subcategory of supervised learning where the goal is to predict the class labels of new samples.



Let's build and train a KNN classifier using sk-learn

Build a KNN classifier, name it knn

```
from sklearn.neighbors import KNeighborsClassifier
```

```
# instance of the classifier
knn = KNeighborsClassifier(n_neighbors = 5)
```

K=5

Train the KNN classifier (fit the model to the data)

```
knn.fit(X_train, Y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

L2-based distance

Model training is to let **knn *memorize*** all of the training samples (features and labels), and build a tree for K-nearest neighbor search.

Use the trained KNN classifier to classify a sample in testing set

Select a sample in the testing set

```
sample_test = X_test.iloc[0,:]
sample_test
```

```
width          9.60
height         9.20
color_score    0.74
Name: 26, dtype: float64
```

We know the true label of this sample

```
label_true = Y_test.iloc[0]
print('The true label is', label_true, ':', fruit_lable_to_name[label_true])
```

```
The true label is 3 : orange
```

Use knn to Predict the label of this sample

```
label_predicted = knn.predict([sample_test])
print('The label predicted by knn is', label_predicted[0], ':', fruit_lable_to_name[label_predicte
if label_predicted[0] == label_true:
    print('Classification is accurate for this testing sample')
else:a
    print('Classification is wrong for this testing sample')
```

```
The label predicted by knn is 3 : orange
Classification is accurate for this testing sample
```

Use the trained KNN classifier to classify a new, previously unseen sample that is not in the training set nor in the testing set

the Feature Vector of a new sample

```
sample_new = [6.0, 4.0, 0.8]
label_predicted = knn.predict([sample_new])
print('The label predicted by knn is', label_predicted[0], ':', fruit_lable_to_name[label_predicte
```

```
The label predicted by knn is 2 : mandarin
```

Evaluate the Performance of the KNN Classifier (K=5)

- $\text{Classification Accuracy} = \dfrac{\text{the number of correctly classified samples}}{\text{total number of samples}}$

- Training Accuracy: accuracy on training set (80% of the data)

```
knn.score(X_train, Y_train)
```

```
0.8723404255319149
```

Testing Accuracy: accuracy on testing set (20% of the data)

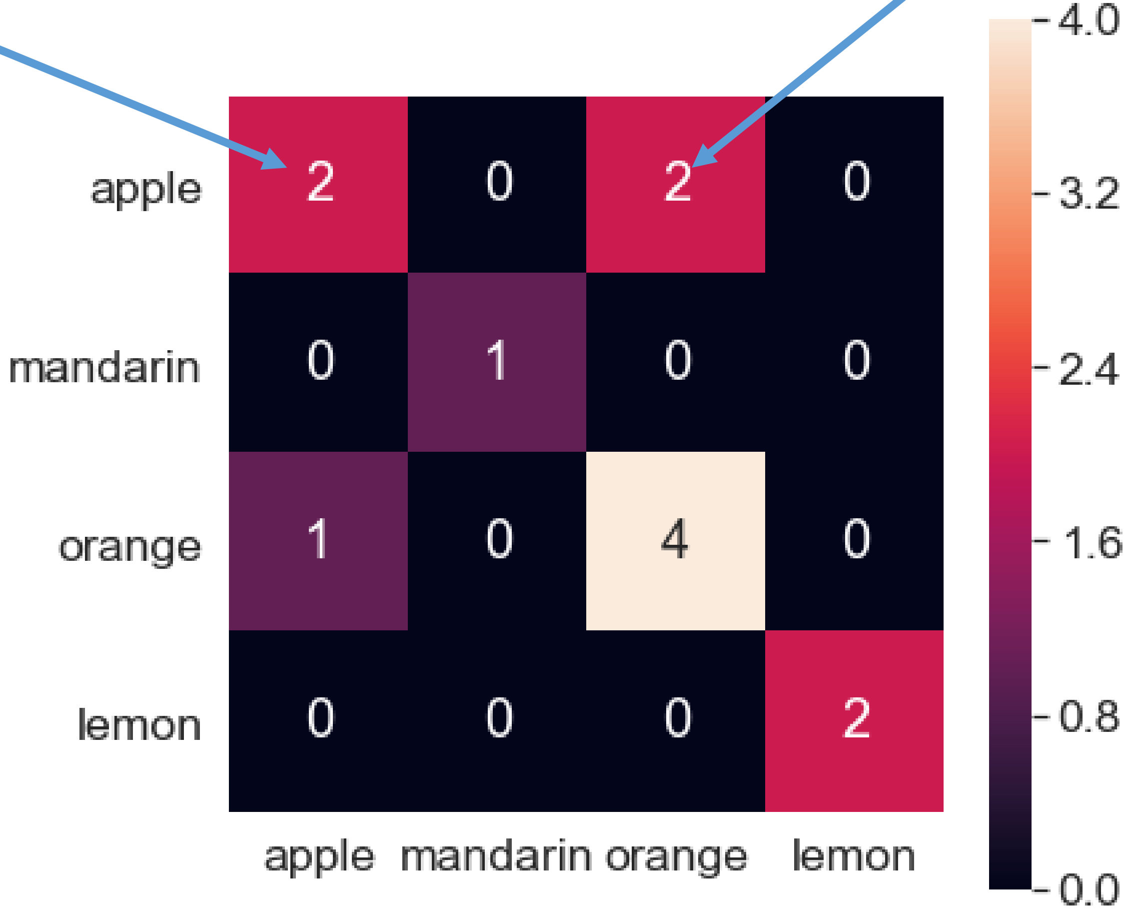
```
knn.score(X_test, y_test)
```

```
0.75
```

Use confusion matrix to visualize the classification result on the testing set

2 apples are classified as apples

2 apples are classified as oranges



The off diagonal numbers show the wrong classifications

the sum of the numbers in the matrix is the total number of testing samples

The diagonal numbers show the correct classifications

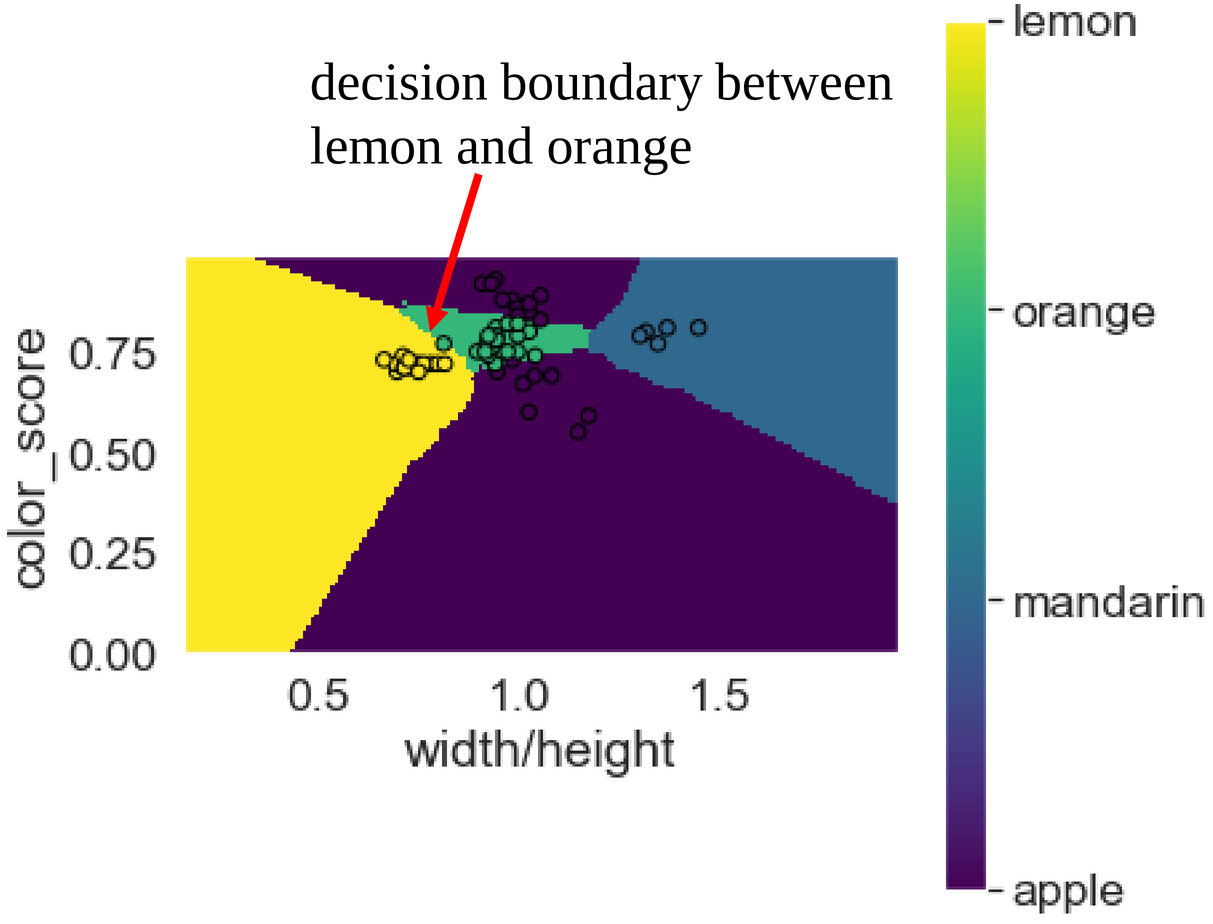
KNN_fruit_classification.ipynb

Plot the **Decision Boundary** to Visualize the Classification Result

A point on the plot represents a sample (a feature vector), which may be in training set or the testing set or unobserved yet.

Roughly speaking, to get the decision boundary plot, we use the KNN classifier to predict the class label of every point on the plot.

In fact, we do not need to check every point: we only need to predict the class labels of the points on a dense grid, and interpolate the result.



KNN can be used for classification and regression

- **For classification**, the output from a KNN classifier is a discrete value (class label), which is done by majority vote among the K-nearest neighbors (training samples) of the input $\boldsymbol{x}$
- **For regression**, the output from a KNN regressor is a continues value (target value)
- **For regression**, the average target value of the *K*-nearest neighbors will be the predicted target value of the input $\boldsymbol{x}$

Assume K=3 and training samples $\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}$ are the (K=3) nearest neighbors of $\boldsymbol{x}$, the target values are $\boldsymbol{y_1}, \boldsymbol{y_2}, \boldsymbol{y_3}$

Then, the predicted target value $\tilde{y}$ of $\boldsymbol{x}$ is $(\boldsymbol{y_1} + \boldsymbol{y_2} + \boldsymbol{y_3})/3$

Boston Housing Dataset

The Housing dataset, which contains information about houses in the different districts of Boston collected by D. Harrison and D.L. Rubinfeld in 1978.

The dataset is a large table that has 506 samples (rows) and 14 columns

Each row contains information/attributes of a region in Boston

y: target

x: input (13 attributes)

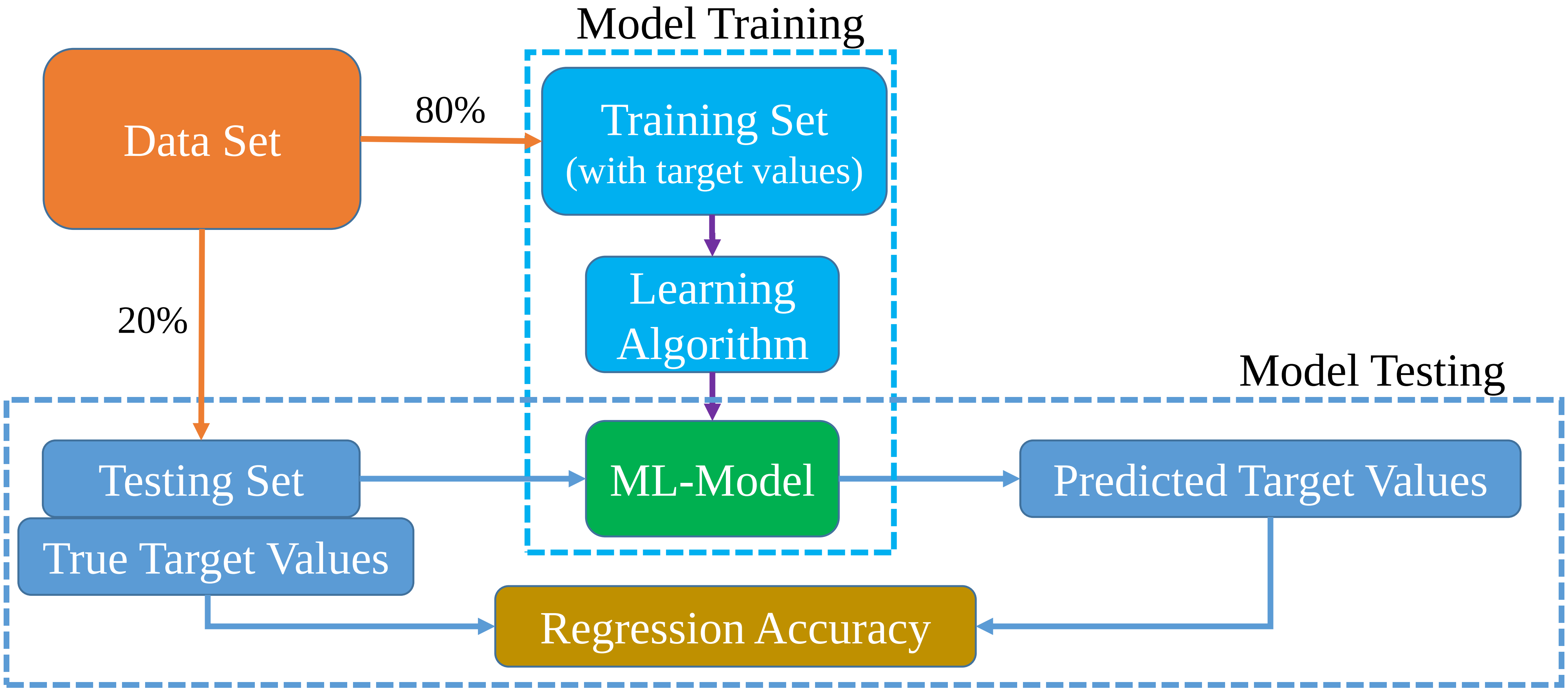
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	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	222.0	18.7	396.90	5.33	36.2

MEDV: Median value of owner-occupied homes in $1000s

The attributes in the dataset

- 1. **CRIM** per capita crime rate by town
- 2. ZN proportion of residential land zoned for lots over 25,000 sq.ft.
- 3. INDUS proportion of non-retail business acres per town
- 4. CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- 5. NOX nitric oxides concentration (parts per 10 million)
- 6. RM average number of rooms per dwelling
- 7. AGE proportion of owner-occupied units built prior to 1940
- 8. DIS weighted distances to five Boston employment centres
- 9. RAD index of accessibility to radial highways
- 10. TAX full-value property-tax rate per $10,000
- 11. PTRATIO pupil-teacher ratio by town
- 12. B 1000(Bk - 0.63)^2
- 13. LSTAT % lower status of the population
- 14. **MEDV** Median value of owner-occupied homes in $1000s

Regression y=f(x)



Model Testing: apply the model to the testing dataset

The testing set contains K input-output pairs: $\{(x_k, y_k), k = 1, \dots, K\}$
$\hat{y}_k$ is the output from the machine learning model, given the input x_k

mean squared error (MSE)

$$MSE = \frac{1}{K}\sum_{k=1}^{K}(y_k - \hat{y}_k)^2$$

mean absolute error (MAE)

$$MAE = \frac{1}{K}\sum_{k=1}^{K}|y_k - \hat{y}_k|$$

Mean absolute percentage error (MAPE)

$$MAPE = \frac{1}{K}\sum_{k=1}^{K}\left|\frac{y_k - \hat{y}_k}{y_k}\right| \times 100\%$$

KNN_Regression.ipynb

```python
knn = KNeighborsRegressor(n_neighbors = 1)
scaler=MinMaxScaler()
#normalize the features
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)
# train the KNN classifier
knn.fit(X_train_s, Y_train)
# test the KNN classifier
Y_test_pred = knn.predict(X_test_s)
# Calculate errors
MSE = np.mean((Y_test - Y_test_pred)**2)
MAE = np.mean(np.abs(Y_test - Y_test_pred))
MAPE =  np.mean(np.abs(Y_test - Y_test_pred)/Y_test)
print('MSE=', MSE)
print('MAE=', MAE)
print('MAPE=', MAPE)
```

```
MSE= 28.68029411764706
MAE= 3.265686274509848
MAPE= 0.1546596701024964
```