AI 실습

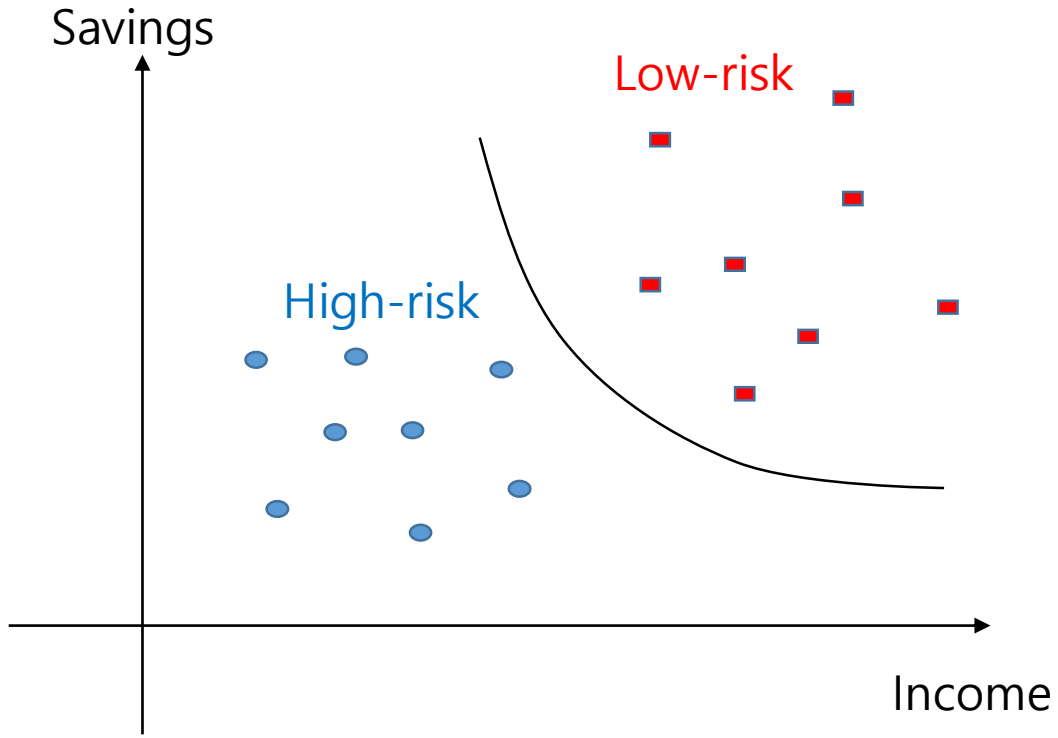
Contents

Classification(인식)

- Classification
  - Ex) Credit scoring
    - Low-risk and high-risk customers from their incomes and savings

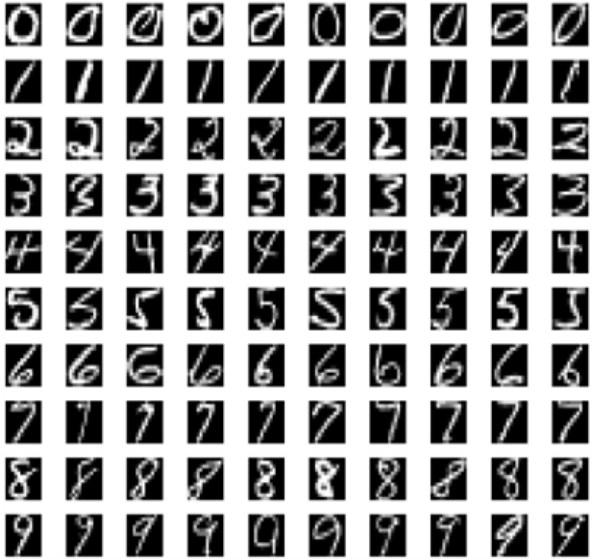
  - Class membership
  - Input Features (입력 특징)

  - Other examples
    - Handwritten digit recognition
    - Speech Recognition
    - Fraud detection



2.1. *k*-Nearest Neighbors Algorithm의 개요

- Handwritten Digit Example

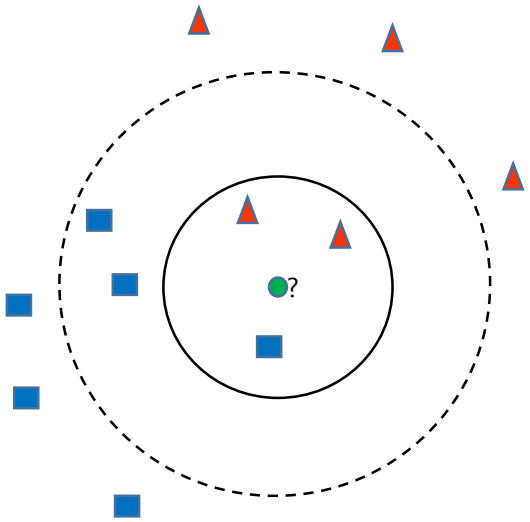


- Do As Your Neighbor Does
  - 학습 데이터 집합 $D = \{\boldsymbol{x}^n c^n\},\ n = 1,2,\dots,N$
  - 새로운 입력 $\boldsymbol{x}^*$에 대한 class $\mathrm{c}(\boldsymbol{x}^*)$ =?
  - 학습 데이터 집합에서 가장 가까운 입력을 찾아 그 class를 이용하여 결정

- k-NN classifier 의 예시

  Green Circle(The test sample) should be classified either to the first class(blue squares) or to the second class(red triangles).

  - If *k* = *3* , there are 2 triangles vs. only 1 square inside the inner circle.
  - If *k* = *5* , there are 3 squares vs. 2 triangles inside the outer circle.

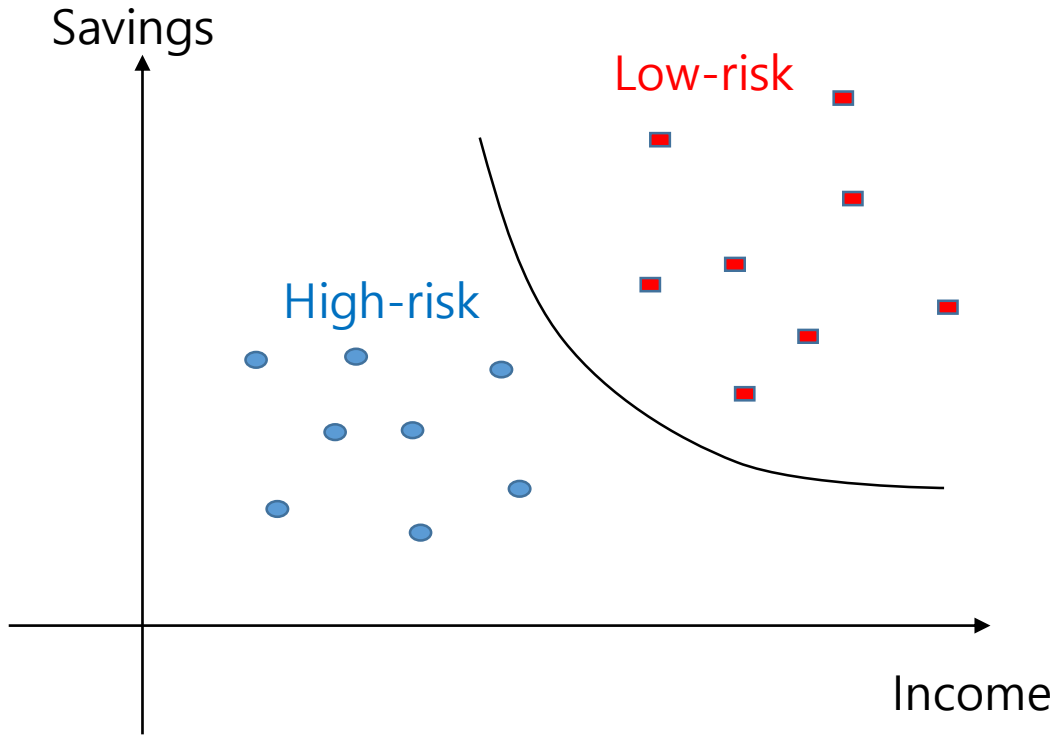


2.2. *k*-Nearest Neighbors Algorithm에 의한 분류

- 분류(Classification)
  - Ex) Credit scoring
    - Low-risk and high-risk customers from their incomes and savings

  - Class membership
  - Input Features (입력 특징)

  - Other examples
    - Handwritten digit recognition
    - Speech Recognition
    - Fraud detection



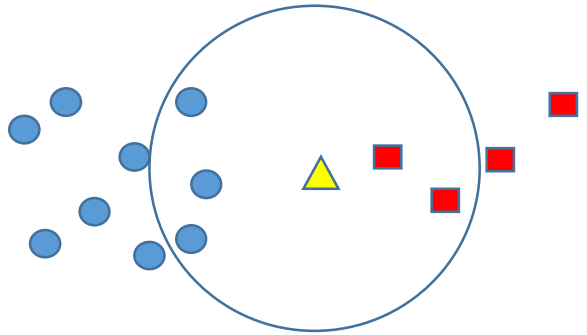
k-Nearest Neighbors Algorithm

- 학습이 없이 예제에 기반한 알고리즘
  - 학습 데이터 집합 $D = \{\boldsymbol{x}^n c^n\},\ n = 1,2,\ldots,N$
  - 새로운 입력 $\boldsymbol{x}^*$에 대한 class $c(\boldsymbol{x}^*) = ?$
  - 학습 데이터 집합에서 가장 가까운 입력을 찾아 그 class를 이용하여 결정

- 거리가 가까운 입력 k를 찾아내면, 그 입력에 해당하는 class 정보의 다수 투표로 새로운 입력 $\boldsymbol{x}^*$에 대한 class $c(\boldsymbol{x}^*)$를 결정

- 거리 측정 기준? Euclidean distance, Minkowski Metric

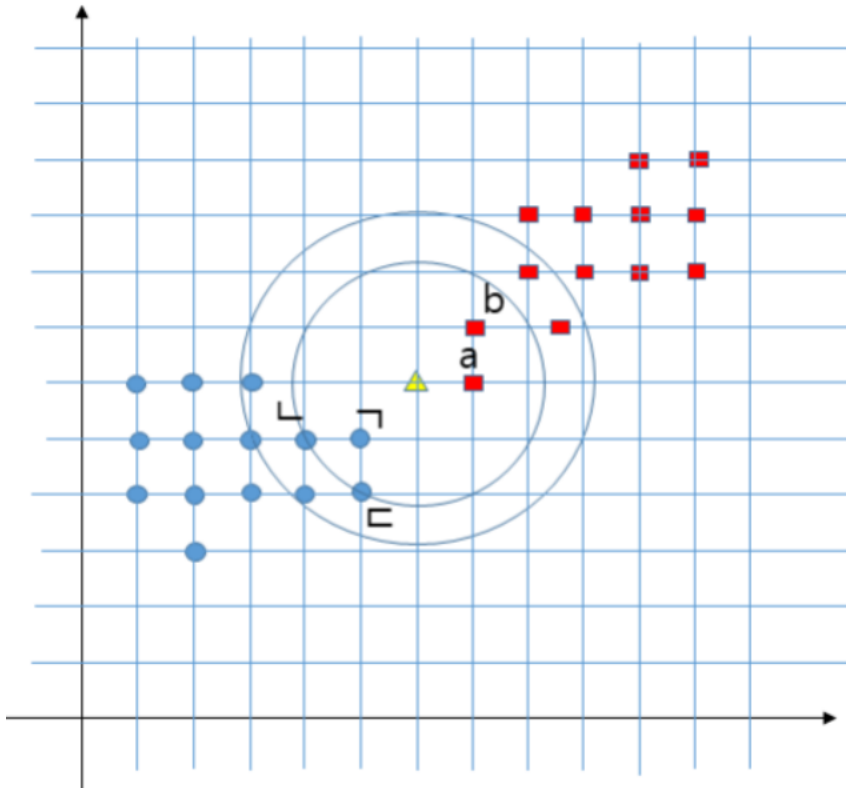
- 분포가 치우쳐 있을 때 성능이 심각하게 저하 됨.



- 거리 정보 d를 이용하면 더 정확한 결정을 할 수 있음

예제 2.2-1

아래 그림과 같이 두 클래스(사각형 혹은 원)에 속하는 데이터가 2차원 공간에 분포하고 있는 경우, 입력이 삼각형으로 주어지면 k-NN에서 k=5로 두고서 이 삼각형은 사각형과 원 중 어느 클래스에 속하는지 구하라? 투표에 거리에 반비례하는 가중치를 적용한 경우와 다수 투표만을 적용한 경우의 결과를 비교하여 보아라.



k-NN Algorithm

k-NN 알고리즘

① 학습패턴 $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$ 를 저장한다.

② 새로운 입력 $\boldsymbol{x}^*$이 주어지면 입력과 학습패턴 사이의 거리 d를 계산한다.

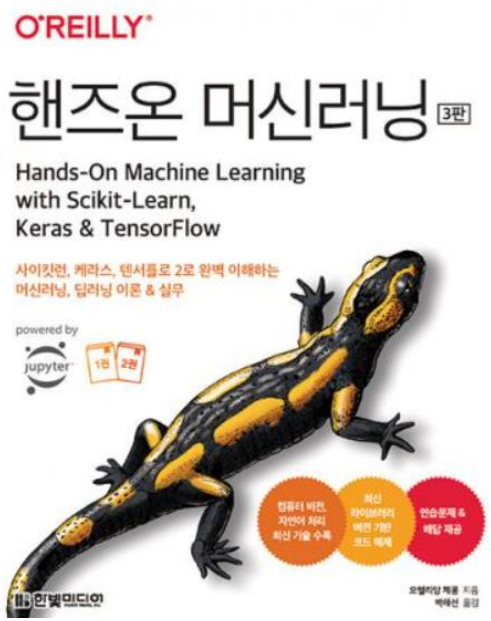
③ 거리가 가장 가까운 이웃 k개를 선정한다.

$$(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), (\boldsymbol{x}_3, y_3), \ldots, (\boldsymbol{x}_k, y_k)$$

④ 분류의 문제이면 다수투표를 수행하고, 회귀의 문제이면 평균을 수행한다.

⑤ 다수 투표 혹은 평균의 과정에 가중치를 거리의 역수 $1/d$로 적용할 수 있다.

k-NN Algorithm 실습



knn-3장-3rdEd.py

3장

MNIST 데이터셋

```
from sklearn.datasets import fetch_openml
mnist=fetch_openml('mnist_784',as_frame=False)

X,y=mnist.data,mnist.target
X.shape
y.shape
X[0]
y[0]
```

DESCR: 데이터 셋 설명
data: 입력데이터 2D numpy 배열
target: Label

```
import matplotlib.pyplot as plt

def plot_digit(image_data):
    image=image_data.reshape(28,28)
    plt.imshow(image, cmap='binary')
    plt.axis("off")

some_digit=X[0]
plot_digit(some_digit)
plt.show()
y[0]

X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

sklearn.neighbors.KNeighborsClassifier

class sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs*) [source]

Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

Parameters:

n_neighbors : *int, default=5*

Number of neighbors to use by default for `kneighbors` queries.

weights : *{'uniform', 'distance'} or callable, default='uniform'*

weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'*

Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

leaf_size : *int, default=30*

Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

p : *int, default=2*

Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

metric : *str or callable, default='minkowski'*

the distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric. See the documentation of `DistanceMetric` for a list of available metrics. If metric is "precomputed", X is assumed to be a distance matrix and must be square during fit. X may be a sparse graph, in which case only "nonzero" elements may be considered neighbors.

Methods

fit(X, y)	Fit the k-nearest neighbors classifier from the training dataset.
get_params([deep])	Get parameters for this estimator.
kneighbors ([X, n_neighbors, return_distance])	Finds the K-neighbors of a point.
kneighbors_graph ([X, n_neighbors, mode])	Computes the (weighted) graph of k-Neighbors for points in X
predict(X)	Predict the class labels for the provided data.
predict_proba(X)	Return probability estimates for the test data X.
score(X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
set_params(**params)	Set the parameters of this estimator.

fit(*X*, *y*) [source]

```
#k-NN classifier
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

y_train_5 = (y_train=='5')
y_test_5 = (y_test == '5')

knn_clf=KNeighborsClassifier()
knn_clf.fit(X_train, y_train_5)
knn_clf.predict([some_digit])
```

8.4. Evaluation of Classifiers(인식기의 평가)

- **"Accuracy(정확도)" or "Recognition Ratio(인식률)"**

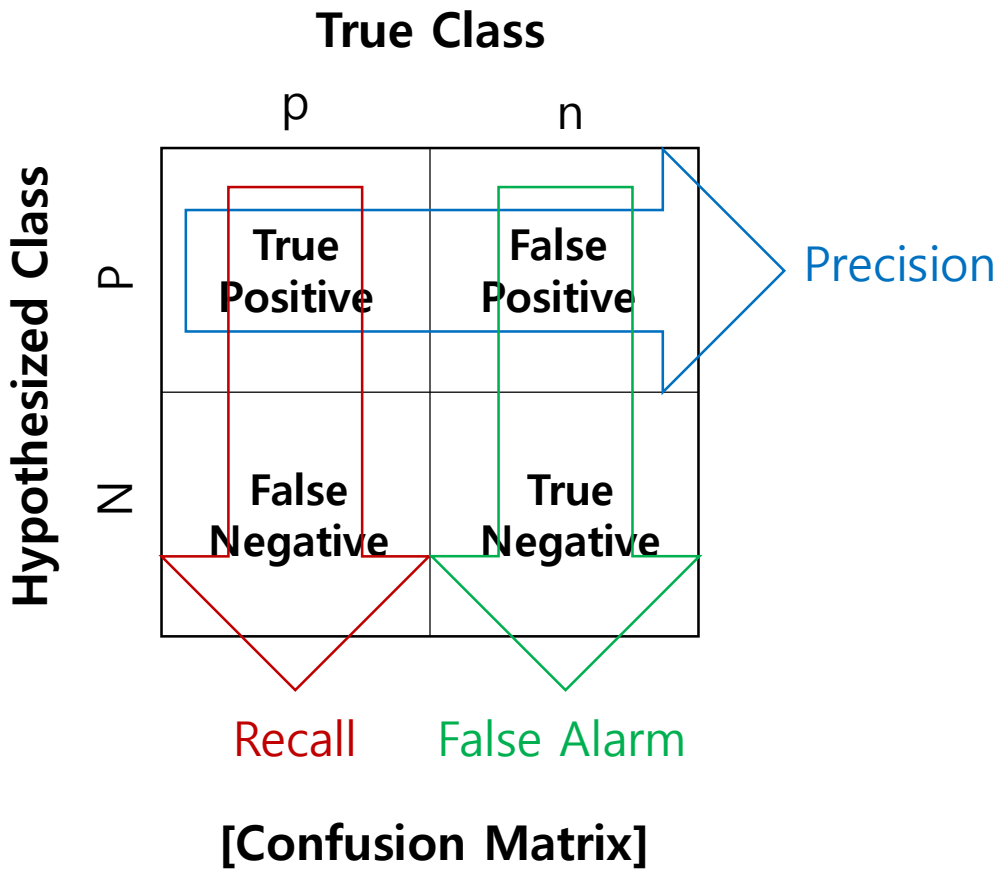
$$Accuracy = \frac{No.(Correctly\ Classified\ Samples)}{No.(Presented\ Samples)} \quad (8.4.1)$$

- **"Precision(정밀도) and Recall(상기율)"**
  - TP : True Positive
  - FP : False Positive
  - TN : True Negative
  - FN : False Negative Positive

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$



[Confusion Matrix]

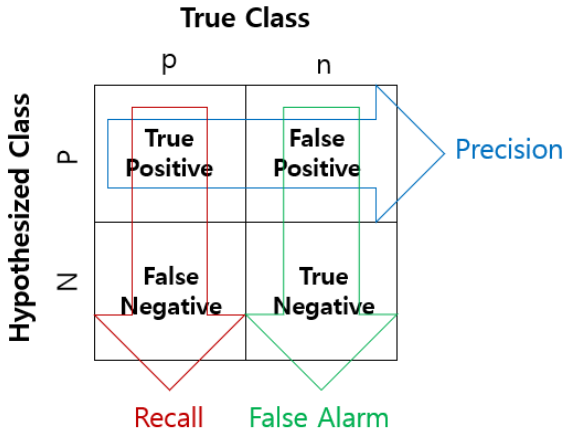
- A precision score of 1.0 for a class C: every item labeled as belonging to class C (P) does indeed belong to class C (but says nothing about FN the number of items from class C that were not labeled correctly).

- A recall of 1.0: every item from class C (p) was labeled as belonging to class C (but says nothing about FP how many other items were incorrectly also labeled as belonging to class C)

- An inverse relationship between precision and recall, where it is possible to increase one at the cost of reducing the other. (FP와 FN이 서로 경쟁관계)

$$Precision = \frac{TP}{TP + FP}$$

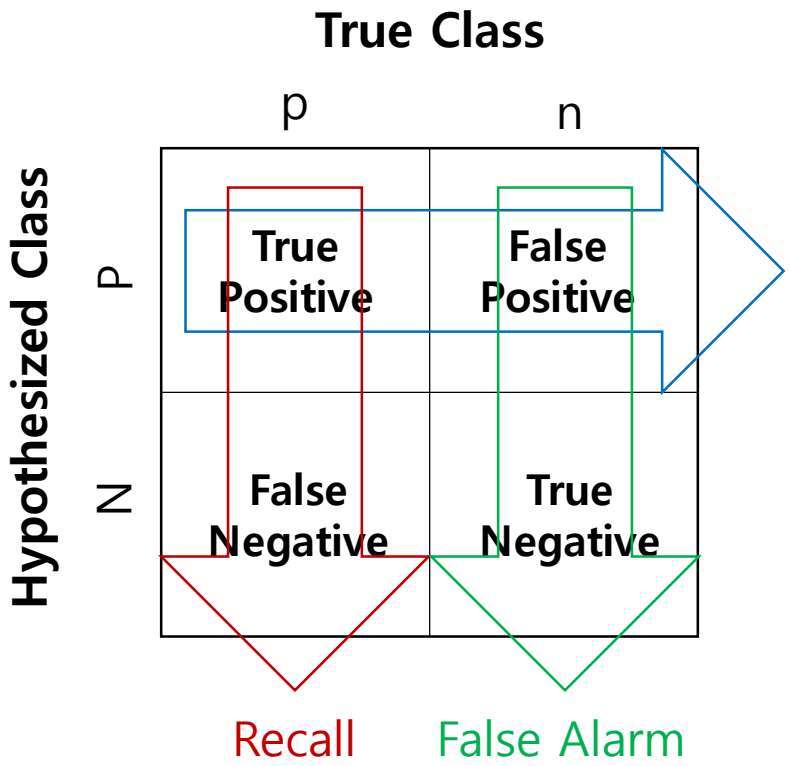
$$Recall = \frac{TP}{TP + FN}$$

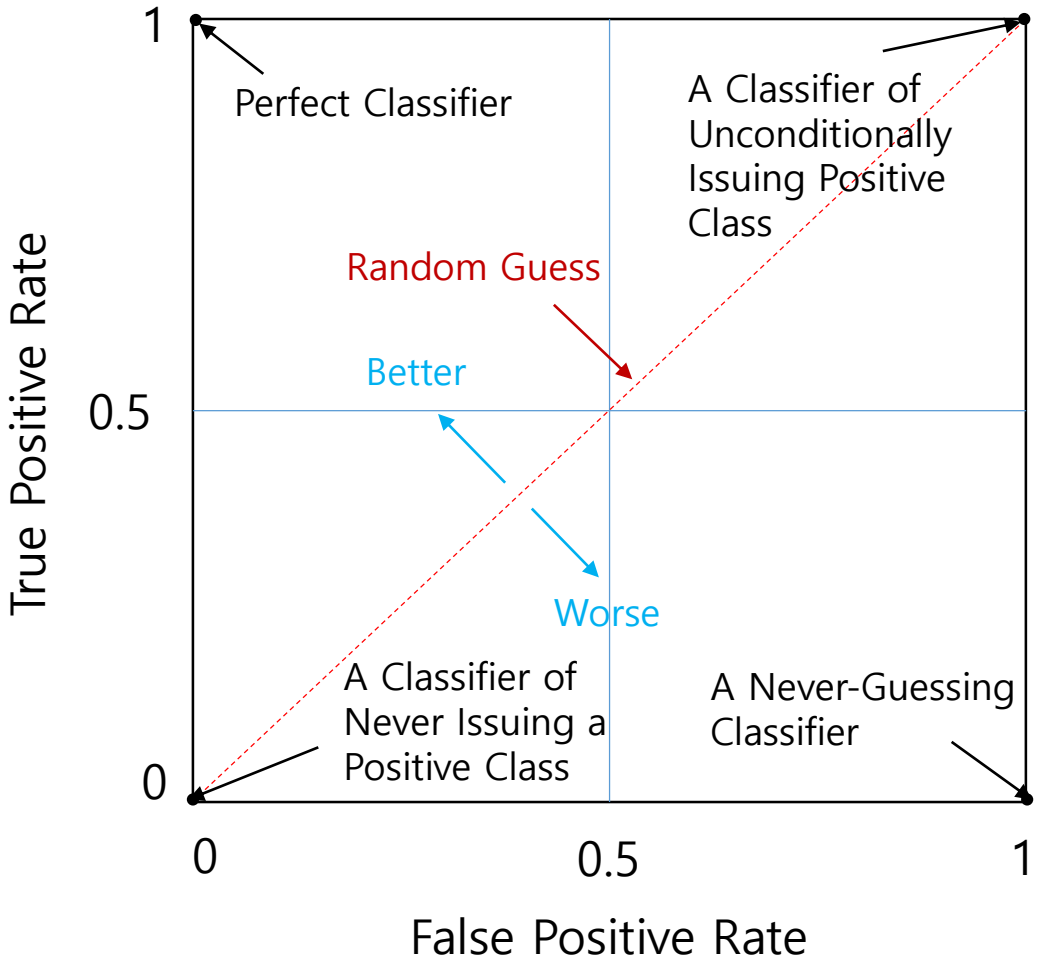


$$F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

- **ROC(Receiver Operating Characteristic)**
  - a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.
  - plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

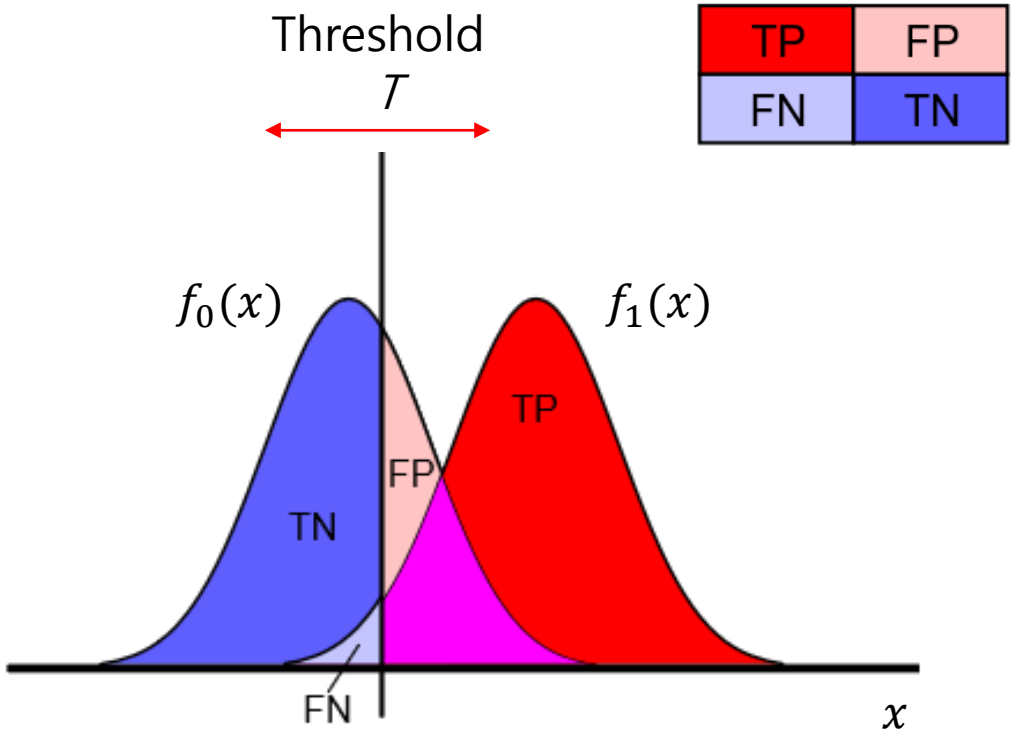
$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN}$$

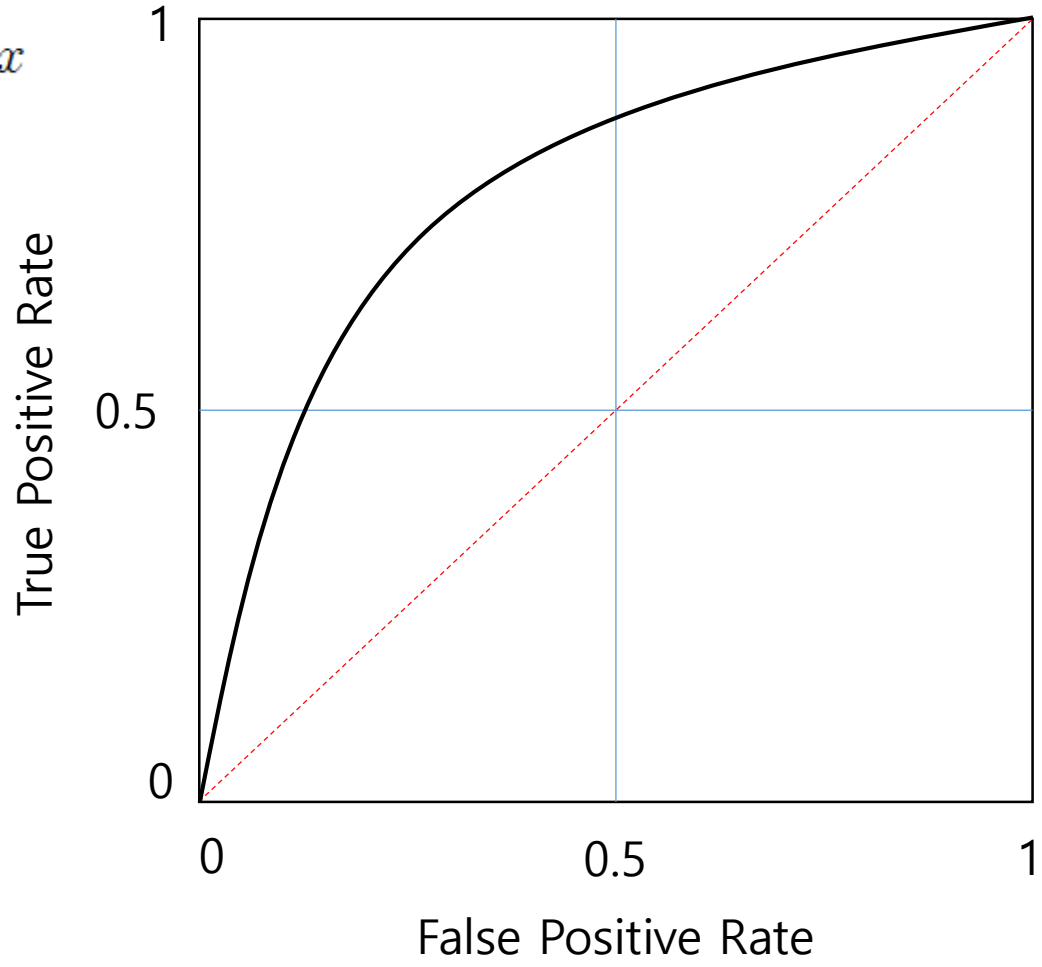




- **Curves in ROC Space : Useful for Imbalanced Data!... AUC(Area Under Curve)**
  - Given a threshold T, the instance is classified as "positive" if $X > T$, and "negative" otherwise.
  - X follows a probability density $f_1(x)$ if the instance actually belongs to class "positive", and $f_0(x)$ if otherwise.

$$TPR(T) = \int_T^{\infty} f_1(x)dx \qquad FPR(T) = \int_T^{\infty} f_0(x)dx$$



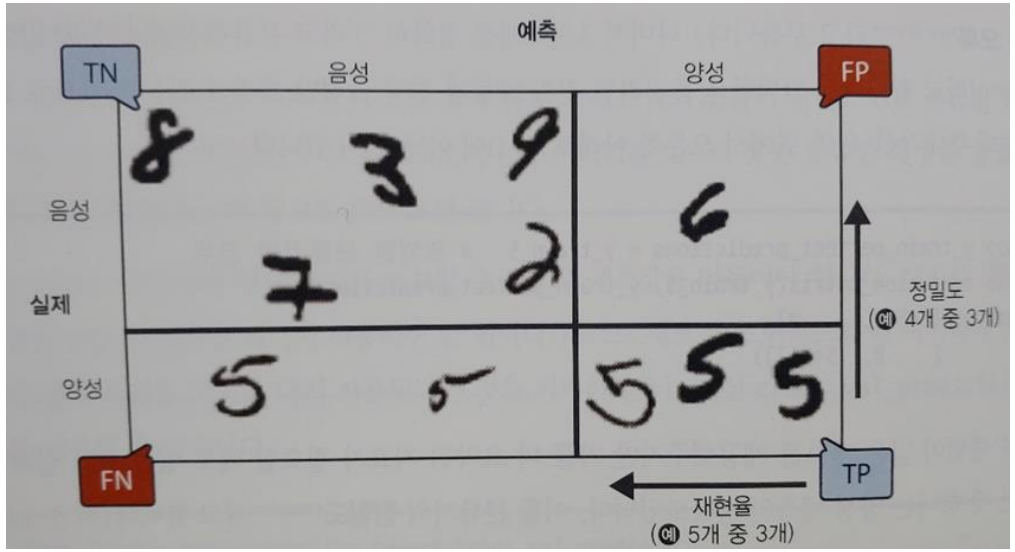


예제 8.4-1

혼동행렬이 아래와 같이 주어진 경우에 정확도, 정밀도와 상기율, TPR, FPR을 계산하여라. 그리고, (FPR,TPR)을 ROC 공간상에 점으로 표시하라.

TP=80	FP=10
FN=20	TN=5

K겹 교차검증(k-fold cross-validation)




```
from sklearn.model_selection import cross_val_score
cross_val_score(knn_clf, X_train, y_train_5, cv=3, scoring="accuracy")

from sklearn.model_selection import cross_val_predict
y_train_pred=cross_val_predict(knn_clf, X_train, y_train_5, cv=3)

from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_5,y_train_pred)

y_train_perfect_predictions = y_train_5
confusion_matrix(y_train_5,y_train_perfect_predictions)

from sklearn.metrics import precision_score, recall_score
precision_score(y_train_5, y_train_pred)
recall_score(y_train_5, y_train_pred)

from sklearn.metrics import f1_score
f1_score(y_train_5, y_train_pred)
```

ROC 와 AUC관련 기능은 KNeighborsClassifier에서 제공되지 않음

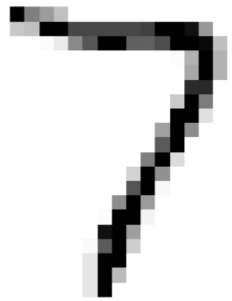
```
knn_clf=KNeighborsClassifier(weights="distance") #distance를 적용해보자
knn_clf.fit(X_train, y_train_5)
y_train_pred=knn_clf.predict(X_train)
confusion_matrix(y_train_5, y_train_pred)
precision_score(y_train_5, y_train_pred)
recall_score(y_train_5, y_train_pred)
f1_score(y_train_5, y_train_pred)
```

```
knn_clf=KNeighborsClassifier(weights="uniform") #uniform을 적용해보자
knn_clf.fit(X_train, y_train_5)
y_train_pred=knn_clf.predict(X_train)
confusion_matrix(y_train_5, y_train_pred)
precision_score(y_train_5, y_train_pred)
recall_score(y_train_5, y_train_pred)
f1_score(y_train_5, y_train_pred)
```

```
#Multi-Label

y_train_large=(y_train >= '7')
y_train_odd = (y_train.astype('int8') %2 ==1)
y_multilabel=np.c_[y_train_large, y_train_odd]

knn_clf=KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)

knn_clf.predict([some_digit])

y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
f1_score(y_multilabel, y_train_knn_pred, average="macro") #모든 label의 가중치가 같다고 가정
```

```
>>> y_train_large=(y_train >= '7')
>>> y_train_odd = (y_train.astype('int8') %2 ==1)
>>> y_multilabel=np.c_[y_train_large, y_train_odd]
>>> knn_clf=KNeighborsClassifier()
>>> knn_clf.fit(X_train, y_multilabel)
KNeighborsClassifier()
>>> knn_clf.predict([some_digit])
array([[False,  True]])
>>> y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
>>> f1_score(y_multilabel, y_train_knn_pred, average="macro")
0.976410265560605
```

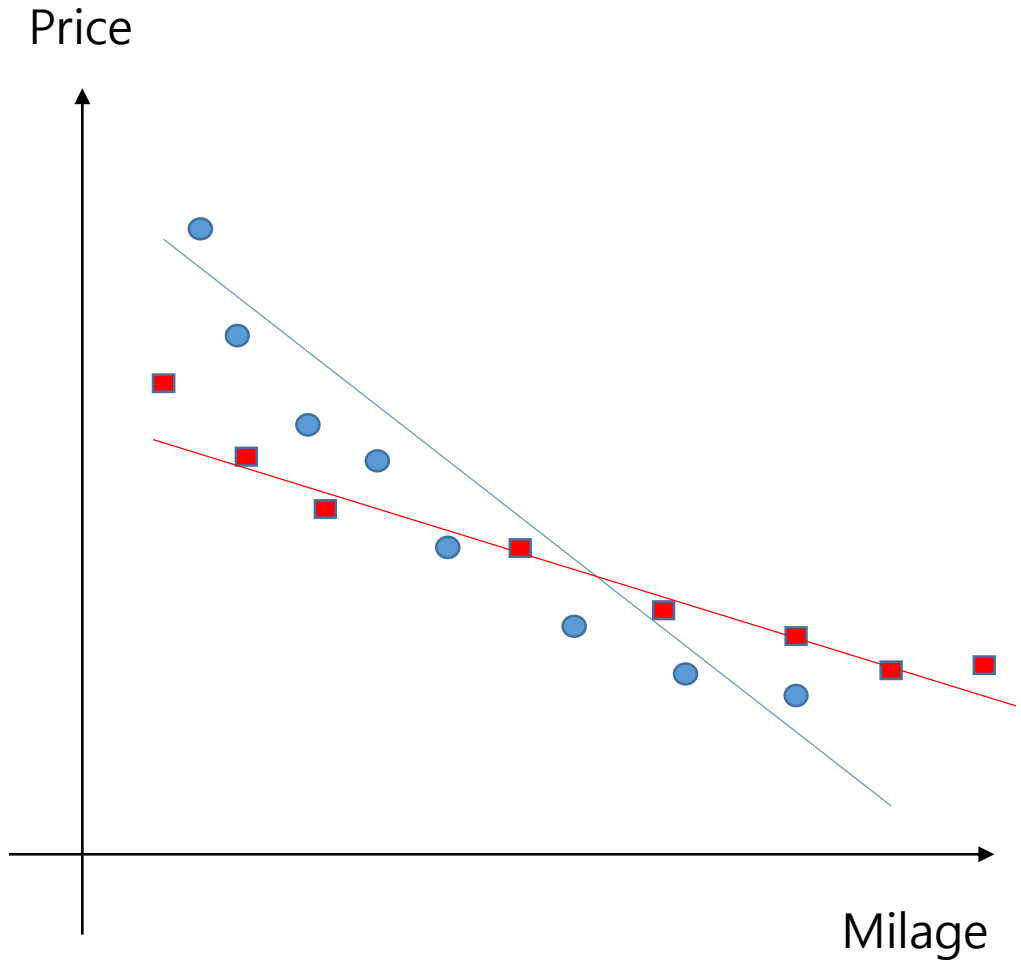
```
#3.7 다중출력분류: 이미지에서 잡음 제거
np.random.seed(42)
noise=np.random.randint(0,100, (len(X_train),784))
X_train_mod=X_train+noise
noise=np.random.randint(0,100, (len(X_test),784))
X_test_mod=X_test+noise
y_train_mod=X_train
y_test_mod=X_test


knn_clf=KNeighborsClassifier()
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit=knn_clf.predict([X_test_mod[0]])
plot_digit(clean_digit)
plt.show()
plot_digit(X_test_mod[0])
plt.show()
```



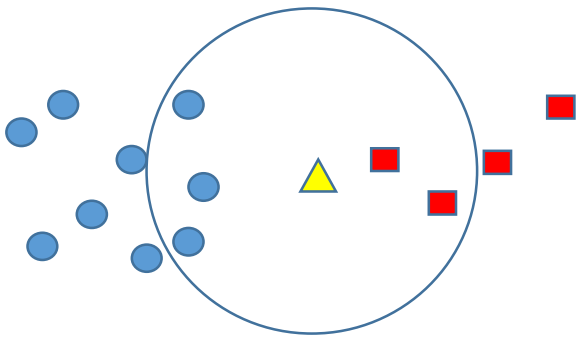
출력 수=픽셀 개수
레이블: 0~255 사이 값

2.3. *k*-Nearest Neighbors Algorithm에 의한 회귀

- 회귀(Regression)
  - Ex) The price of a used car
  - 출력: 연속(회귀) vs. 이산(discrete) (분류)
  - Other examples
    - Stock price prediction
    - Oil price prediction
    - Water level prediction



- 입력은 특징값들로 구성
- 출력은 입력에 해당하는 성질값으로 k 근접 이웃에 해당하는 성질값들의 평균으로 주어짐
- 거리 측정 기준? Euclidean distance, Minkowski Metric
- 분포가 치우쳐 있을 때 성능이 심각하게 저하 됨.
- 거리 정보 d를 이용하면 더 정확한 결정을 할 수 있음



예제 2.3-1

k-NN으로 회귀 문제를 해결하고자 한다. k=3로 두고서 새로운 입력에 가장 가까운 세 벡터에 해당하는 출력 값이 y_1, y_2, y_3이고 이들의 거리는 각각 d_1, d_2, d_3이다. 평균에 의해 출력을 구하는 경우와 거리에 반비례하는 가중치를 적용한 경우(가중치 평균)의 출력 값 계산을 하여라. 그리고, 가중치 평균의 적용 시 가중치의 합은 1이 됨을 증명하여라.

k-NN 실습

knn-3장-p.131.py

```
import numpy as np

#http://bit.ly/perch_data에서 복사
perch_length = np.array([8.4, 13.7, 15.0, 16.2, 17.4, 18.0, 18.7, 19.0, 19.6, 20.0, 21.0,
       21.0, 21.0, 21.3, 22.0, 22.0, 22.0, 22.0, 22.0, 22.5, 22.5, 22.7,
       23.0, 23.5, 24.0, 24.0, 24.6, 25.0, 25.6, 26.5, 27.3, 27.5, 27.5,
       27.5, 28.0, 28.7, 30.0, 32.8, 34.5, 35.0, 36.5, 36.0, 37.0, 37.0,
       39.0, 39.0, 39.0, 40.0, 40.0, 40.0, 40.0, 42.0, 43.0, 43.0, 43.5,
       44.0])
perch_weight = np.array([5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0, 110.0,
       115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0, 130.0,
       150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0, 197.0,
       218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0, 320.0, 514.0,
       556.0, 840.0, 685.0, 700.0, 700.0, 690.0, 900.0, 650.0, 820.0,
       850.0, 900.0, 1015.0, 820.0, 1100.0, 1000.0, 1100.0, 1000.0,
       1000.0])

from sklearn.model_selection import train_test_split

train_input, test_input, train_target, test_target=train_test_split(perch_length,perch_weight, random_state=42)

#train data와 test data를 2차원 배열로 바꾸기
train_input=train_input.reshape(-1,1)
test_input=test_input.reshape(-1,1)

from sklearn.neighbors import KNeighborsRegressor

knr=KNeighborsRegressor(n_neighbors=3)
#length로 weight 예측
knr.fit(train_input, train_target)

#length=50인 weight 예측
print(knr.predict([[50]]))

import matplotlib.pyplot as plt
#이웃 구하기
distances, indexes=knr.kneighbors([[50]])

#train data의 scatter plot
plt.scatter(train_input, train_target)
plt.scatter(train_input[indexes], train_target[indexes], marker='D')

plt.scatter(50, knr.predict([[50]]), marker='^')
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```