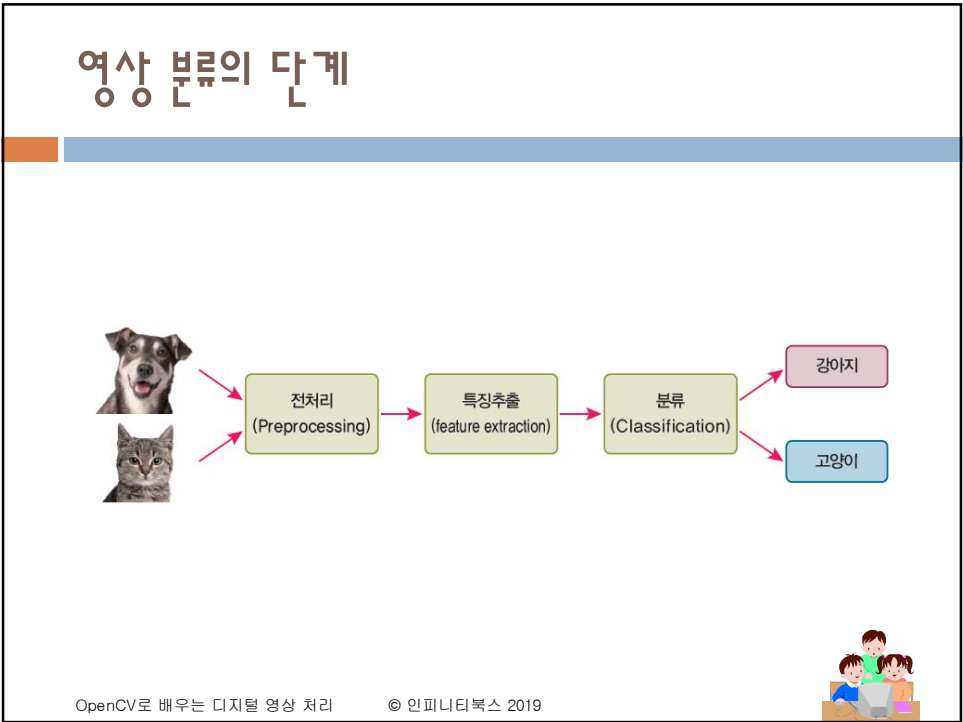


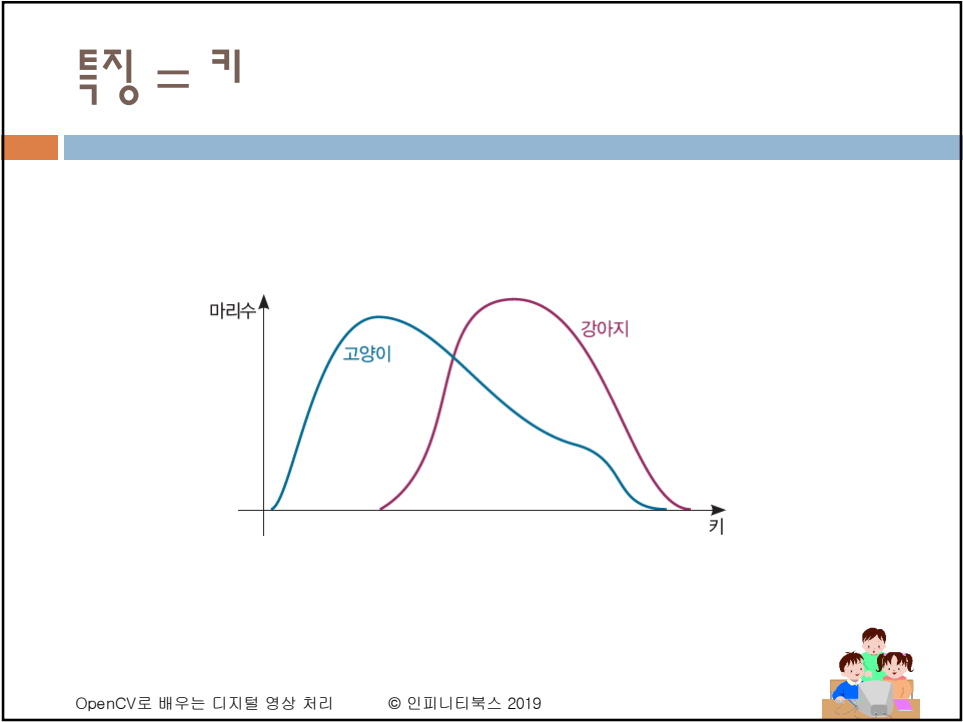
1



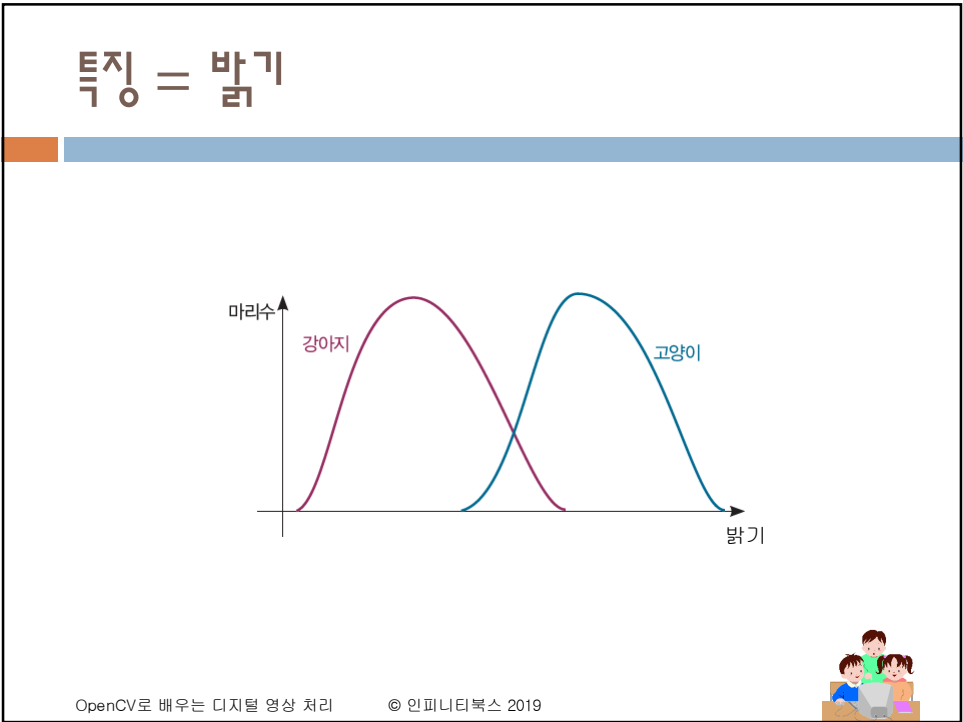
2



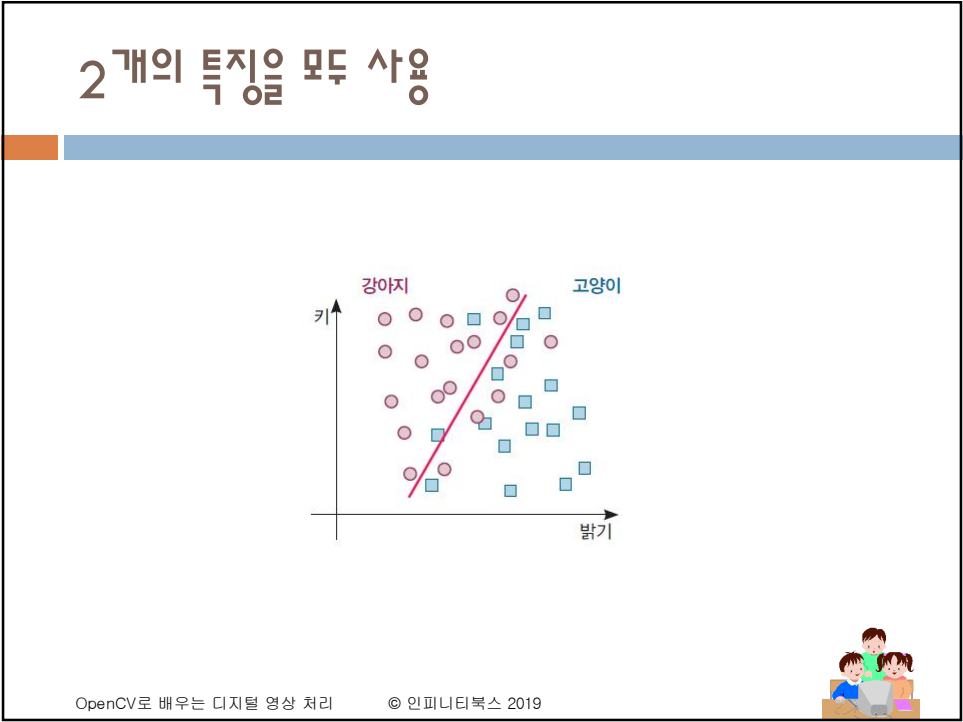
3



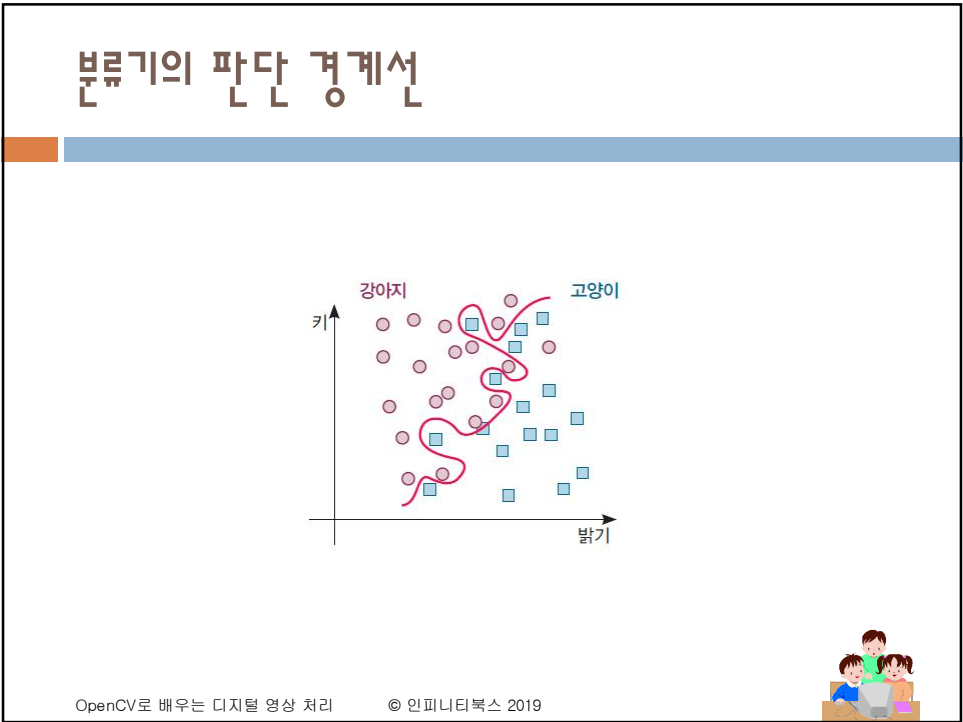
4



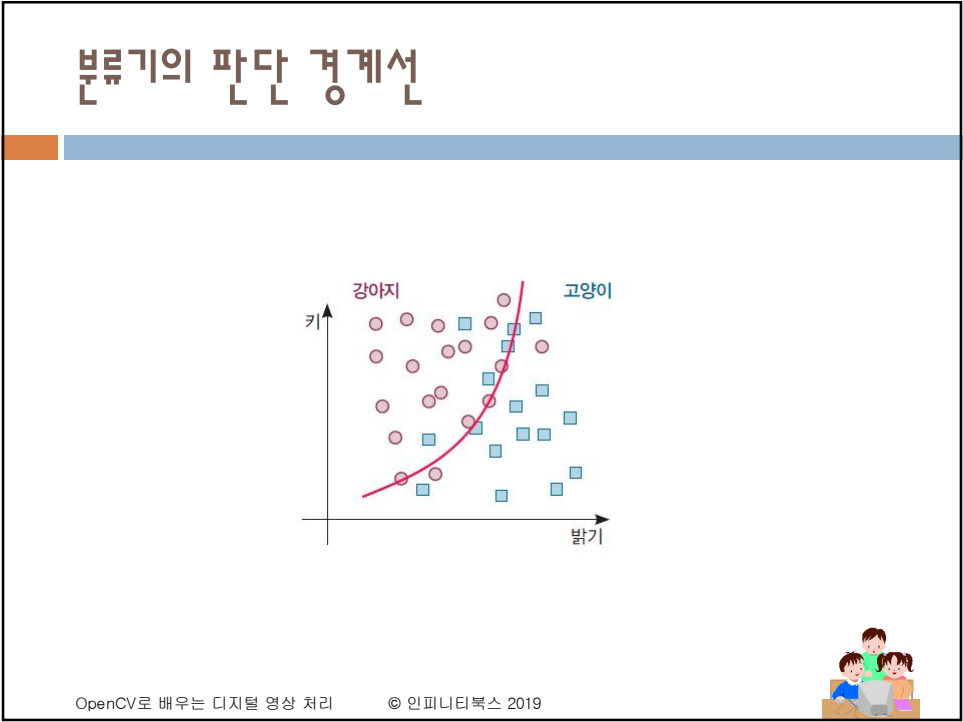
5



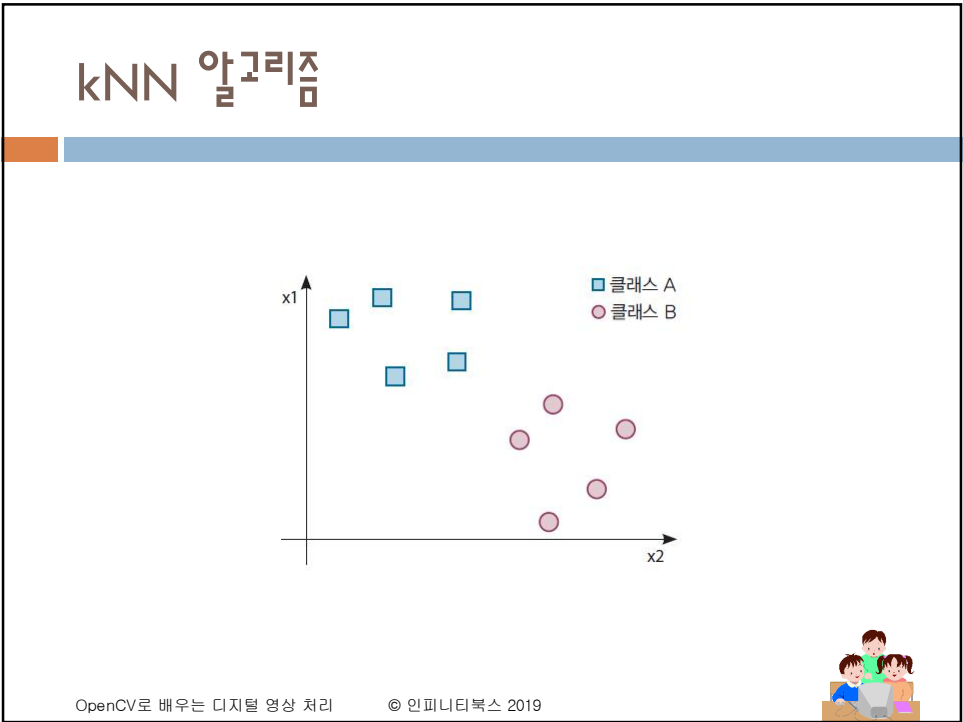
6



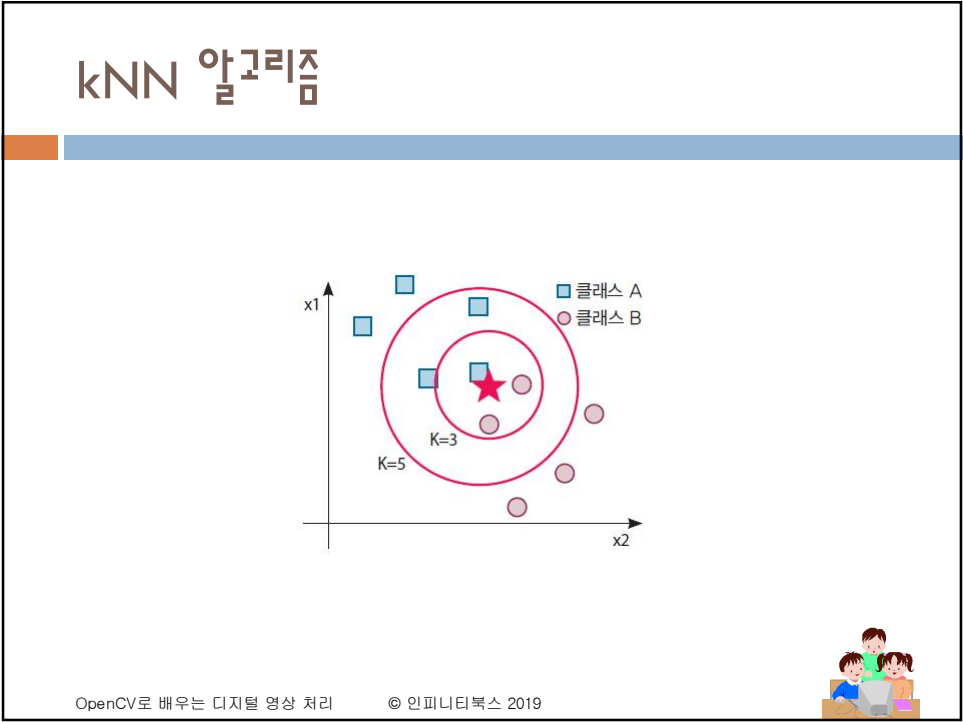
7



8



9



10

## kNN 알고리즘의 장점과 단점

- 특징 공간에 있는 모든 데이터에 대한 정보가 필요하다. 왜냐하면 가장 가까운 이웃을 찾기 위해 새로운 데이터에서 모든 기존 데이터까지의 거리를 확인해야 하기 때문이다. 데이터와 클래스가 많이 있다면, 많은 메모리 공간과 계산 시간이 필요하다.
- 어떤 종류의 학습이나 준비 시간이 필요 없다.

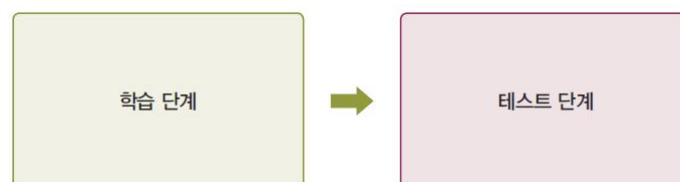
OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019



11

## OpenCV에서 kNN



OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019



12

## 학습 단계

- `Ptr <ml::KNearest> knn = ml::KNearest::create();`
- `Ptr <ml::TrainData> trainData = ml::TrainData::create(train_features, ROW_SAMPLE, labels);`
- `knn->train (trainData);`

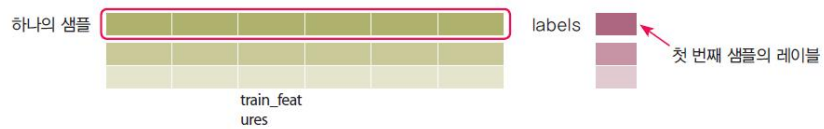
OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019



13

## 학습 단계



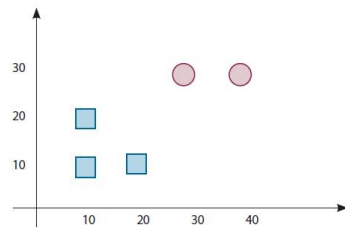
OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019



14

## 학습 단계



train\_features

10	10
10	20
20	10
30	30
40	30

1
1
1
2
2

OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019



15

## 테스트 단계

- Mat predictedLabels;
- knn->findNearest(sample, K, predictedLabels);

OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019



16



```

int main()
{
    Mat train_features(5, 2, CV_32FC1);
    Mat labels(5, 1, CV_32FC1);

    // 점의 좌표를 train_features에 입력한다.
    train_features.at<float>(0, 0) = 10, train_features.at<float>(0, 1) = 10;
    train_features.at<float>(1, 0) = 10, train_features.at<float>(1, 1) = 20;
    train_features.at<float>(2, 0) = 20, train_features.at<float>(2, 1) = 10;
    train_features.at<float>(3, 0) = 30, train_features.at<float>(3, 1) = 30;
    train_features.at<float>(4, 0) = 40, train_features.at<float>(4, 1) = 30;

    // 원하는 레이블을 labels에 입력한다.
    labels.at<float>(0, 0) = 1;
    labels.at<float>(1, 0) = 1;
    labels.at<float>(2, 0) = 1;
    labels.at<float>(3, 0) = 2;
    labels.at<float>(4, 0) = 2;

```

OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019



17

```

// 학습 과정
Ptr<ml::KNearest> knn = ml::KNearest::create();
Ptr<ml::TrainData> trainData = ml::TrainData::create(train_features,
ROW_SAMPLE, labels);
knn->train(trainData);

// 테스트 과정
Mat sample(1, 2, CV_32FC1);
Mat predictedLabels;

// 테스트 데이터를 입력한다.
sample.at<float>(0, 0) = 28, sample.at<float>(0, 1) = 28;
knn->findNearest(sample, 2, predictedLabels);

float prediction = predictedLabels.at<float>(0, 0);
cout << "테스트 샘플의 라벨 = " << prediction << endl;
return 0;
}

```


OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019




18

# 실행 결과




OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019




# kNN을 이용한 숫자 인식



OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019



```

int main()
{
    Mat img;
    img = imread("d:/digits.png", IMREAD_GRAYSCALE);
    namedWindow("original", WINDOW_AUTOSIZE);
    imshow("original", img);
    waitKey(0);

    Mat train_features(5000, 400, CV_32FC1);
    Mat labels(5000, 1, CV_32FC1);

    // 각 숫자 영상을 행 벡터로 만들어서 train_feature에 저장한다.
    for (int r = 0; r < 500; r++) {
        for (int c = 0; c < 100; c++) {
            int i = 0;
            for (int y = 0; y < 20; y++) {
                for (int x = 0; x < 20; x++) {
                    train_features.at<float>(r * 100 + c,
i++) = img.at<uchar>(r * 20 + y, c * 20 + x);
                }
            }
        }
    }
}

```

21

```

// 각 숫자 영상에 대한 레이블을 저장한다.
for (int i = 0; i < 5000; i++) {
    labels.at<float>(i, 0) = (i / 500);
}

// 학습 과정
Ptr<ml::KNearest> knn = ml::KNearest::create();
Ptr<ml::TrainData> trainData = ml::TrainData::create(train_features,
ROW_SAMPLE, labels);
knn->train(trainData);

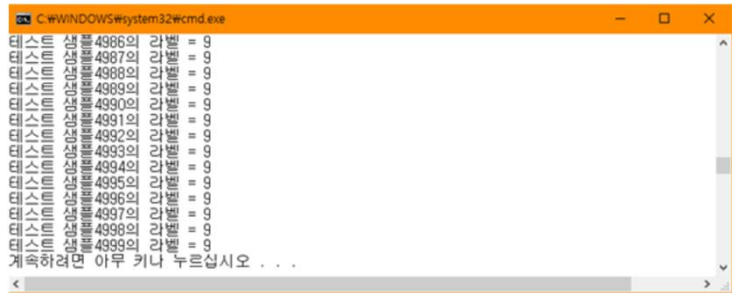
// 테스트 과정
Mat predictedLabels;
for (int i = 0; i < 5000; i++) {
    Mat test = train_features.row(i);
    knn->findNearest(test, 3, predictedLabels);
    float prediction = predictedLabels.at<float>(0);
    cout << "테스트 샘플" << i << "의 라벨 = " << prediction << '\n';
}
}

```

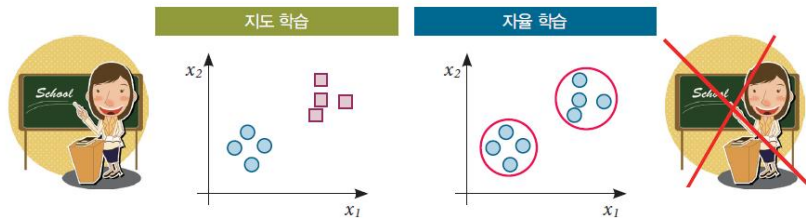


22

# 실행결과



# K-means 알고리즘



## K-means 알고리즘

- K-means 알고리즘은 주어진  $n$ 개의 관측값을  $k$ 개의 클러스터로 분할하는 알고리즘으로, 관측값들은 거리가 최소인 클러스터로 분류된다.
- K-means 알고리즘은 자율 학습의 일종으로, 레이블이 달려 있지 않은 입력 데이터에 레이블을 붙여주는 역할을 수행한다.

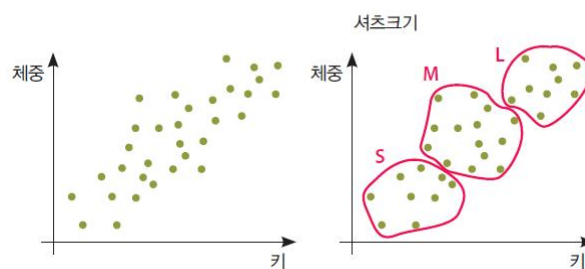
OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019



25

## K-means 알고리즘의 예



OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019



26

# K-means 알고리즘

Algorithm 13.1

입력값

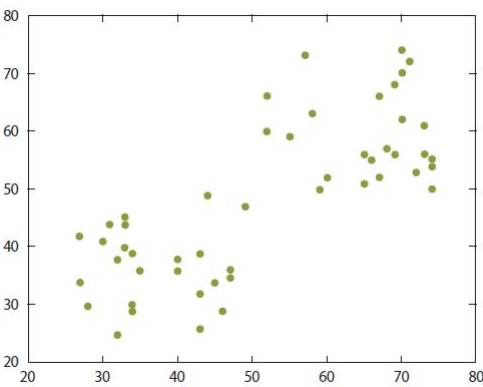
- ①  $k$ : 클러스터 수
  - ②  $D$ :  $n$ 개의 데이터를 포함하는 집합
- 출력값:  $k$ 개의 클러스터

알고리즘

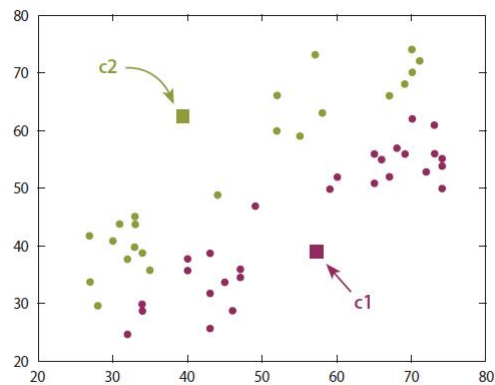
- ① 집합  $D$ 에서  $k$ 개의 데이터를 임의로 추출하고, 이 데이터들을 각 클러스터의 중심(centroid)으로 설정한다(초기값 설정).
- ② 집합  $D$ 의 각 데이터에 대해  $k$ 개의 클러스터 중심과의 거리를 계산하고, 각 데이터가 어느 중심점(centroid)과 가장 유사도가 높은지 알아낸다. 그리고 그렇게 찾아낸 중심점으로 각 데이터들을 할당한다.
- ③ 클러스터의 중심점을 다시 계산한다. 즉, 2에서 재할당된 클러스터들을 기준으로 중심점을 다시 계산한다.
- ④ 각 데이터의 소속 클러스터가 바뀌지 않을 때까지 ②, ③ 과정을 반복한다.



# K-means 알고리즘



## K-means 알고리즘



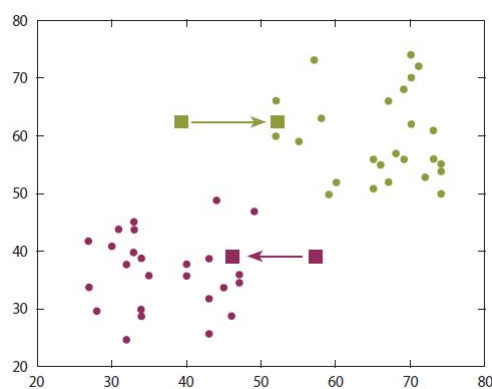
OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019



29

## K-means 알고리즘



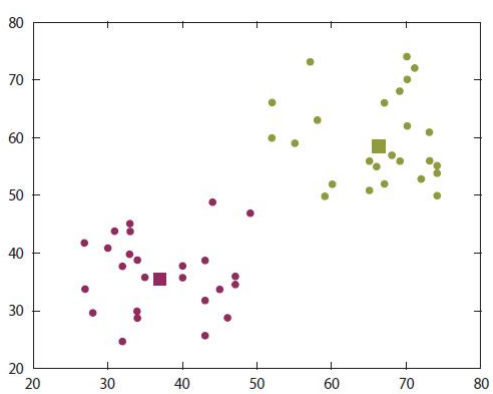
OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019




30

# K-means 알고리즘



OpenCV로 배우는 디지털 영상 처리

© 인피니티박스 2019




# OpenCV 함수

```
double kmeans(InputArray samples, int K, InputOutputArray labels,
              TermCriteria criteria, int attempts, int flags,
              OutputArray centers=noArray())
```

매개 변수	설명
samples	샘플 데이터 행렬. 행렬의 행에 샘플이 저장되어 있다.
K	우리가 원하는 클러스터의 개수
labels	각 클러스터의 레이블이 저장될 행렬
criteria	알고리즘을 종료하는 조건을 기술한다.
attempts	알고리즘이 수행되는 횟수
flags	알고리즘 초기화 플래그. KMEANS_PP_CENTERS는 Arthur and Vassilvitskii이 주장한 클러스터 초기화 방법을 의미한다. KMEANS_RANDOM_CENTERS은 랜덤으로 클러스터의 중심점을 잡는 것이다.
centers	클러스터의 중심이 저장된 출력 행렬

OpenCV로 배우는 디지털 영상 처리

© 인피니티박스 2019





```

int main()
{
    Mat samples(50, 2, CV_32F);

    for (int y = 0; y < samples.rows; y++) {
        samples.at<float>(y, 0) = (rand() % 255);
        samples.at<float>(y, 1) = (rand() % 255);
    }
    Mat dst(256, 256, CV_8UC3);

    for (int y = 0; y < samples.rows; y++) {
        float x1 = samples.at<float>(y, 0);
        float x2 = samples.at<float>(y, 1);
        circle(dst, Point(x1, x2), 3, Scalar(255, 0, 0));
    }
    imshow("dst", dst);
}

```

OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019



33

```

    Mat result;
    Mat labels(50, 1, CV_8UC1);

    Mat centers;
    result = Mat::zeros(Size(256, 256), CV_8UC3);
    kmeans(samples, 2, labels, TermCriteria(CV_TERMCRIT_ITER |
CV_TERMCRIT_EPS, 10000, 0.0001),
        3, KMEANS_PP_CENTERS, centers);

    for (int y = 0; y < samples.rows; y++) {
        float x1 = samples.at<float>(y, 0);
        float x2 = samples.at<float>(y, 1);
        int cluster_idx = labels.at<int>(y, 0);
        if (cluster_idx == 0)
            circle(result, Point(x1, x2), 3, Scalar(255, 0, 0));
        else
            circle(result, Point(x1, x2), 3, Scalar(255, 255, 0));
    }
    imshow("result", result);
    waitKey(0);
    return(0);
}

```

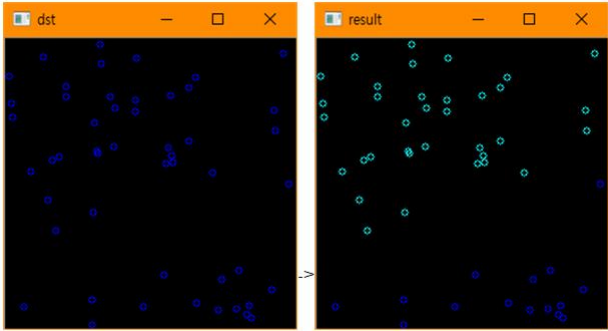
OpenCV로 배우는 디지털 영상 처리

© 인피니티북스 2019




34

# 실행결과



OpenCV로 배우는 디지털 영상 처리      © 인피니티북스 2019



# Q & A



OpenCV로 배우는 디지털 영상 처리      © 인피니티북스 2019

