chap 15 머신러닝

머신러닝과 OpenCv

머신러닝 개요 1

- 머신러닝이란 주어진 데이터를 분석하여 규칙성, 패턴등을 찾고, 이를 이용하여 의미있는 정보를 추출하는 과정을 말함.

1. 학습(train) 또는 훈련

- 데이터로 부터 규칙을 찾아내는 과정

2. 모델(model)

- 학습에 의해 결정된 규칙

3. 예측(predict) 또는 추론

- 새로운 데이터를 학습된 모델에 입력으로 전달하고 결과를 판단하는 과정

4. 레이블

- 훈련 데이터에 대한 정답에 해당하는 내용
- 머신러닝은 크게 지도학습과 비지도 학습으로 구분

지도학습

- 정답을 알고 있는 데이터를 이용하여 학습을 진행

비지도학습

- 정답이 없이 규칙을 찾아내는 방식

머신러닝 개요 2

- 영상 데이터는 픽셀로 구성되어 있지만, 해당 픽셀값을 그대로 머신 러닝 입력으로 사용하는것은 그다지 흔치 않음 -> 영상의 픽셀값은 조명, 객체의 이동 및 회전들 민감하게 변화
- 그래서 대부분 머신러닝에서 특징정보를 추출하여 입력으로 전달
- 이처럼 영상데이터를 사용하는 지도학습에서는 먼저 다수의 훈련 영상에서 *특징 벡터* 를 추출함 -> 이를 이용하여 머신러닝 알고리즘을 학습
- 예측과정에서도 입력영상으로부터 특징 벡터를 추출 -> 해당 특징 벡터를 학습모델입력으로 전달하면 입력영상이 어떤 영상인지에 대한 예측 결과를 얻을 수 있음

지도학습

- 지도학습은 주로 회귀 또는 분류에 사용된다

회귀 : 연속된 수치 값 예측 (예 : 키-몸무게 상관관계 학습, 새로운 키 입력에 대한 몸무게 예측)

분류 : 이산적인 값을 결과로 출력하는 머신러닝 (입력이 들어오면 어떤 결과인지 분류)

비지도 학습

- 훈련 데이터의 정답에 대한 정보 없이 오로지 데이터 자체만을 이용하는 학습 방식
- 무작위로 입력을 전달하고, 전체 입력을 두 개의 그룹으로 나누도록 학습시키는 방식
- 이 경우 분리된 두개의 그룹이 무엇을 의미하는지는 알 수 없음 -> 단지 두 그룹에서 서로 구분되는 특징을 이용하여 분리하는 작업만 수행 -> 군집화에 사용됨

머신러닝 개요 3

- 많은 머신 러닝 알고리즘이 지도학습을 이용한 영상 분류 문제에 사용.
- 학습된 분류 모딜에 얼마나 제대로 동작하는지를 확인해야 하는 경우가 있기에 영상 데이터 전체를 학습에 사용하지 않음. -> 일부는 성능 측정을 위한 테스트 용도로 사용한다. -> 오버피팅 : 전체를 학습하면 치우쳐서 학습하게 된다.
- 많은 머신러닝 알고리즘 종류에 따라서 파라미터에 의해 성능이 달라지기도함. -> 최적의 파라미터를 찾는것이 또 하나의 해결 문제.
- 해당 경우 훈련 데이터를 k개의 부분 집합으로 분할
- 학습과 검증을 반복하면서 최적의 파라미터를 찾을 수 있음

K-폴드 교차 검증

- 훈련 데이터를 k개의 부분 집합으로 분할하여 학습과 검증을 반복하는 작업
- 예 : 8000개의 훈련영상을 800개씩 10개의 부분집합으로 분할하여 이중 9개로 학습하고 1개로 성능을 검증함.
- 그리고 이를 바꿔가면서 여러번 학습과 검증을 수행하면서 다양한 파라미터에 대한 가장 좋은 파라미터를 찾을 수 있음

훈련데이터의 잡음 혹은 이상치

- 훈련 데이터에 포함된 잡음 또는 이상치의 영향을 고려해야함.
- 많은 머신러닝 분류 알고리즘이 훈련 데이터를 효과적으로 구분하는 경계면을 찾으려 함.
- 이때 잘못된 정보가 섞여있다면 경계면을 어떻게 설정하는것이 좋은지 모호해질수 있음
- 훈련 데이터에서는 100% 동작할 수 있지만, 오버피팅이 되었다면 실제로는 정확도가 오히려 떨어 질 수 있다.

OpenCv 머신러닝 클래스

- ml모듈에 주로 포함되어 있음
- 모듈의 StatModel클래스가 있고, predict(), train()함수를 갖고 있어서 이를 재정의 함으로 사용

1. train()

- 인자로 훈련 데이터가 어떻게 저장되어 있는지 명시해 주어야 함.
- ROW_SAMPLE이 많은 경우 사용. (행단위저장) , COL_SAMPLE로 열단위 저장 데이터도 훈련시킬수 있음.
- 해당 ㅎ마수는 가상함수로 이를 상속받은 객체의 알고리즘의 맞게 사용

2. predict()

- 순수 가상함수로서 자신만의 알고리즘을 이용한 예측을 수행하도록 이를 상속받은 클래스가 재정의 해야 함
- 일부 알고리즘은 고유의 예측함수를 제공하기도 함

3. OpenCV 머신러닝 클래스와 이름들

- ANN_MLP - 인공신경망, 다층퍼셉트론, 여러개의 은닉층을 포함한 신경망을 학습시킬수 있고, 결과 예측가능
- DTrees : 이진의사결정트리. 부스팅알고리즘, 랜덤트리 알고리즘의 부모역할
- Boost : 부스팅 알고리즘, 다수의 약한 분류기에 적절한 가중치를 부여하여 성능이 좋은 분류기를 만듬
- Rtrees : 랜덤트리 or 랜덤포레스트 알고리즘. 다수의 트리로 예측하고 결과를 취합하여 분류 또는 회귀 수행
- EM : 기댓값최대화, 가우시안 혼합모델을 이용한 군집화 알고리즘
- KNearest : 최근접이웃알고리즘, 샘플과 인접한 k개의 훈련데이터를 찾고, 가장 많은 개수에 해당하는 클래스를 지정
- LogisticRegression : 로지스특 회귀, 이진 분류 알고리즘
- NormalBayesClassifier : 정규 베이즈 분류기.
- SVM : 서포트 백터머신 , 두 클래스의 데이터를 가장 여유있게 분리하는 초평면을 구함.
- SVMSDG : 통계적 그래디언트 하향 방법

최근접 이웃

- k최근접 이웃 알고리즘은 분류 또는 회귀에 사용되는 지도 학습 알고리즘의 하나
- KNN알고리즘을 분류에 사용할 경우, 특징 공간에서 테스트 데이터와 가장 가까운 K개의 훈련 데이터를 찾음
- K개의 훈련 데이터중에서 가장 많은 클래스를 테스트 데이터의 클래스로 지정
- KNN알고리즘을 회귀에 사용할 경우 테스트 데이터에 인접한 k개의 훈련데이터 평균을 테스트 데이터 값으로 설정

KNN알고리즘에 의한 점분류

1. 가장 간단한 분류

- 새로 점이 들어온다면, 가장 가까이 있는 훈련 데이터 점을 찾아 해당 훈련 데이터 클래스와 같게 설정. -> 즉 최근접 이웃으로 지정
- 예시) 새로 녹색점이 들어왔을때, 가장 가까이 있는 점이 빨간색이라면, 해당 클래스와 같게 녹색점을 빨간색 클래스로 지정
- NN기법 (인접)

2. 자역을 보는 분류

- 녹색점에서 가장 가까운 점은 빨간색 점이지만, 해당 지점에 파란색 점이 더 많기에 녹색점을 파란색 클래스로 지정하는 것이 합리적 -> KNN기법 (K개가 인접)

KNearest 클래스 사용하기

- knn알고리즘에서 k값을 1로 설정하면 최근접 이웃알고리즘(nn) -> 보통은 1보다 크게 설정

- k값에 따라 분류 및 회귀 결과가 달라질 수 있음.
- 보통 k값이 커질수록 잡음 또는 이상치 데이터의 영향이 감소.
- 그러나 k값이 어느정도 이상 커지면 오히려 성능이 떨어짐. -> 왜냐하면 k값이 커질수록 전체 data를 많이 보기 때문
- opencvd에는 knn이 구현되어 있음. (KNearest클래스에 구현)
- 기본으로 k값이 10으로 설정. setDefaultK()로 변경가능
- 기본적으로 분류를 위한 용도로 생성. 회귀에 쓰고 싶으면 setIsClassfier()에 false지정이 필요.

- KNearest 객체를 생성하고 속성을 설정한 후 train()함수로 학습 진행.
- 그러나 실제적인 학습이 아닌 데이터 저장만 이루어짐. 예측에서 시간이 더 오래걸리기 때문.
- 예측을 수행할때 predic()을 사용할 수 있지만, findNearest()가 더 많은 정보를 반환하기에 유용

```
train = []
label = []
k_value = 1

def on_k_changed(pos): # trackbar값이 바뀌면 update해주는 역할
    global k_value

    k_value = pos
    if k_value < 1:
        k_value = 1
    trainAndDsiplay()

def addPoint(x,y,c):
    train.append([x,y])
    label.append([c])

def trainAndDisplay():
    train_array = np.array(train).astype(np.float32)
    label_array = np.array(label)
    knn.train(train_array, cv.ml.ROW_SAMPLE, label_array)
    # 데이터를 append했기에 ROW형태로 저장

    for j in range(img.shape[0]):
        for i in range(img.shape[1]):
            # 한 픽셀씩 접근
            sample = np.array([[i,j]]).astype(np.float32)

            ret, res, _, _, = knn.findNearest(sample, k_value)
            response = int(res[0,0])
            # 클래스 판별해서 색으로 칠하기

            if response == 0:
                img[j,i] = (128,128,255)
            elif response == 1:
                img[j,i] = (128,255,128)
```

```
            elif respones == 2:
                img[j,i] = (255,128,128)

    # 라벨에 따라 원그리기.
    for i in range(len(train)):
        x, y = train[i]
        l = label[i][0]

        if l == 0:
            cv.circle(img, (x,y), 5, (0,0,128), -1, cv.LINE_AA)
        elif l == 1:
            cv.circle(img, (x,y), 5, (0,128,0), -1, cv.LINE_AA)
        elif l == 2:
            cv.circle(img, (x,y), 5, (128,0,0), -1, cv.LINE_AA)
    cv.imshow()

img = np.zeros((500,500, 3), np.uint8)

knn = cv.ml.KNearest_create()

cv.namedWindow('knn')
cv.createTrackbar('k_value', 'knn', k_value, 5, on_k_changed)

NUM = 30
rn = np.zeros((NUM, 2), np.int32)

cv.randn(rn,0,50)
for i in range(NUM):
    addPoint(rn[i,0] + 150, rn[i,1] + 150, 0)
    # (150,150) 좌표를 중심으로 30개의 가우시안분포점 생성. 0번 클래스

cv.randn(rn,0,50)
for i in range(NUM):
    addPoint(rn[i,0] + 350, rn[i,1] + 150, 1)
    # (350,150) 좌표를 중심으로 30개의 가우시안분포점 생성. 1번 클래스

cv.randn(rn,0,50)
for i in range(NUM):
    addPoint(rn[i,0] + 250, rn[i,1] + 400, 2)
    # (350,400) 좌표를 중심으로 30개의 가우시안분포점 생성. 2번 클래스

trainAndDisplay()
cv.imshow()
```

- k값을 트랙바에서 조정하면서 분류 결과를 볼 수 있음

KNN을 이용한 필기체 숫자 인식

- opencv에서 제공하는 필기체 숫자 data로 훈련 시키기.
- 각 숫자는 20 * 20 픽셀 크기로 적혀 있고, 전체 크기는 2000 *1000

- 숫자 영상을 부분으로 추출하여 훈련 데이터를 생성해야 함.
- 해당 예제에서는 특징백터를 추출하지 않고, 픽셀값 자체를 입력으로 사용.
- 20*20 훈련 데이터를 먼저 일렬로 늘어놓아 1 * 400형태의 행렬로 만듬.
- 이후 모든 데이터의 과정을 반복하면 5000 * 400 형태의 데이터로 가공 가능
- 즉 20*20을 한줄로 늘리면 숫자 훈련 데이터 하나가 400개의 값으로 표현됨.
- 이를 모든 훈련 데이터에 반복해서 쌓는다.

훈련시키기

- 이후 영상을 전달할때 숫자의 값 또한 함께 전달해야 함.
- 행 넘버로 숫자 값을 전달함.

필기체 숫자 인식 예제

```
def on_mouse(event, x, y, flags, _): # mouse 입력에 따라서 이미지에 선을 그리는 함수
    global oIdx, oIdy

    if event == cv.EVENT_LBUTTONDOWN:
        oIdx, oIdy = x, y
    elif event == cv.EVENT_LBUTTONUP:
        oIdx, oIdy = -1, -1
    elif event == cv.EVENT_MOUSEMOVE:
        if flags & cv.EVENT_FLAG_LBUTTON:
            cv.line(img, (oIdx, oIdy), (x,y), (255,255,255), 40,
cv.LINE_AA)
            oIdx, oIdy = x,y
            cv.imshow()

digit = cv.imread()

h, w = digits.shape[:2]

cells = [np.hsplit(row, w//20) for row in np.vsplit(digits, h//20)]
# 먼저 20 * 20으로 쪼갠다.

cells = np.array(cells)

train_images = cells.reshape(-1, 400).astype(np.float32)
# 1*400으로 하나씩 펴주는 작업
train_labels = np.repeat(np.arrange(10), len(train_images) /10)


# train하는 과정

knn = cv.ml.KNearest_create()
knn.train(train_images, cv.ml.ROW_SAMPLE, train_labels)
```

```
# test

# 일단 400 * 400 빈화면 만들기
img = np.zeros((400,400), np.uint8)


while True:
    c = cv.waitkey()

    if c == 27:
        break
    elif c == ord(' '):
        img_resize = cv.resize(img, (20,20), interpoloation=cv.INTER_AREA)
        img_flatten = img_resize.reshape(-1,400).astype(np.float32)
        # 입력한 이미지를 1 * 400으로 펴줌.

        _ret, res, _, _ = knn.findNearest(img_flatten, 3)
        #data를 넣어주고.
        print(int(res[0,0]))
        # 숫자가 무엇인지 출력

        img.fill(0)
        cv.imshow()
```

- on_mouse()함수는 마우스 왼쪽을 클릭한 상태로 움직이면 해당 위치에 40픽셀 두깨의 글씨를 쓸 수 있음
- 해당 입력을 확인하고, 사용자가 (' ')를 입력할때마다 글씨를 인식하여 결과를 콘솔창에 출력