산업경영알고리즘 최종보고서

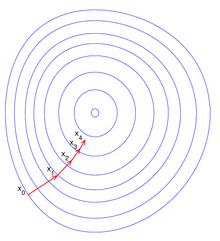
주제 : Gradient Descent 알고리즘으로 최적의 관계성 파악

2017100878 김익환

**문제정의**

**1. 경사 하강법(Gradient Descent)**

* 정의 : 경사 하강법(GD)은 여러 종류의 문제에서 최적의 해법을 찾을 수 있는 매우 일반적인 최적화 알고리즘이다.
* 경사 하강법의 기본 아이디어는 비용함수를 최소화하기 위해 반복해서 파라미터를 조정해 나가는 것이다.
* 해를 반복해 cost 함수가 최소화 되는 방향으로 파라미터를 개선하면서 문제를 풀어간다.
* 알고리즘을 실험하기 위한 문제 상황은 관계성 탐색 문제로 설정하였다.
* Q. 기계의 수 X대와 그때의 생산량 Y개가 존재할 때, 데이터간 관계성
* 입력 받는 데이터는 x 기계의 대수와 y 그에 따른 생산량을 지니는 문제가 된다. 두 값 모두 list 형태로 입력된다.
* Null값은 존재하지 않는다고 가정한다.
* 출력으로는 두 x, y값들의 선형 회귀(1차) 최적 기울기가 출력된다.

**알고리즘 설계 기본 원리**

경사하강법은 1차 근사값 발견용 최적화 알고리즘입니다.

함수의 기울기를 구하고 경사의 절댓값이 낮은 쪽으로 계속 이동시켜 극값에 이를 때까지 반복시키는 것이 주요 원리입니다.

최적화할 함수 f(x)에 대하여, 먼저 시작점 x0를 정합니다.

현재 xi가 주어진다면, 다음 단계의 점인 xi+1은 다음과 같이 계산된다.

Xi+1 = xi – ri f(xi)

Ri는 이동할 거리를 조절하는 매개변수이다.

이 알고리즘은 f와 ri의 선택에 따라 지역 최적해로 수렴시켜 줍니다.

**의사코드**

for epoch in range(무한반복):

        grad = 0

        error = 0

        for x\_val, y\_val in x\_data, y\_data:

# x, y데이터 한쌍씩 빼서 합산으로 한 턴의 데이터 출력

            temp\_grad = gradient(w, x\_val, y\_val)

            w = w - learning\_rate \* temp\_grad

# 기울기 갱신

            temp\_error = costfunction(w, x\_val, y\_val)

# cost함수의 예측의 에러값 계산

            error += temp\_error

            grad += temp\_grad

        if error < 0.0001:

# 종료조건

            return w

**파이썬코드**

def check\_input(x\_data, y\_data):

    """check user's input data

    Args:

        x\_data ([list]): data list

        y\_data ([list]): data list

    Returns:

        [boolean]: if num of each data is same, keep going

    """

    if len(x\_data) != len(y\_data):

        return False

    else:

        return True

def costfunction(w, x, y):

    """cost function

    Args:

        w ([double]): this turn 's gradient

        x ([double]): one of x data

        y ([double]): one of y data

    Returns:

        ([double]): cost function value

    """

    y\_pred = w \* x

    return (y\_pred - y) \*\* 2

def gradient(w, x, y):

    """ dericative of cost function about w

    Args:

        w ([double]): this turn 's gradient

        x ([double]): one of x data

        y ([double]): one of y data

    Returns:

        [double]: dericative of cost function about w

    """

    return 2 \* x \* (x \* w - y)

def solver(x\_data, y\_data, learning\_rate):

    """solver to find w

    Args:

        x\_list ([list]): base x data group

        y\_list ([list]): base x data group

    Returns:

        w [double]: optimized w value

    """

    w = 0

    # x\_data = [1.0, 2.0, 3.0]

    # y\_data = [2.0, 4.0, 6.0]

    if check\_input(x\_data, y\_data) == False:

        return "error"

    for epoch in range(1000):

        grad = 0

        error = 0

        for x\_val, y\_val in zip(x\_data, y\_data):

            temp\_grad = gradient(w, x\_val, y\_val)

            w = w - learning\_rate \* temp\_grad

            print('\tgrad : ', round(temp\_grad, 2))

            temp\_error = costfunction(w, x\_val, y\_val)

            error += temp\_error

            grad += temp\_grad

        print('progress : ', epoch, 'w = ', round(w, 4), 'error = ', round(error, 4))

        if error < 0.0001:

            print("done")

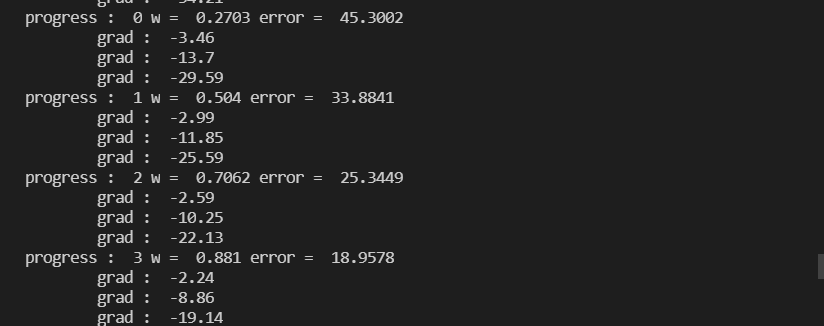
            return w

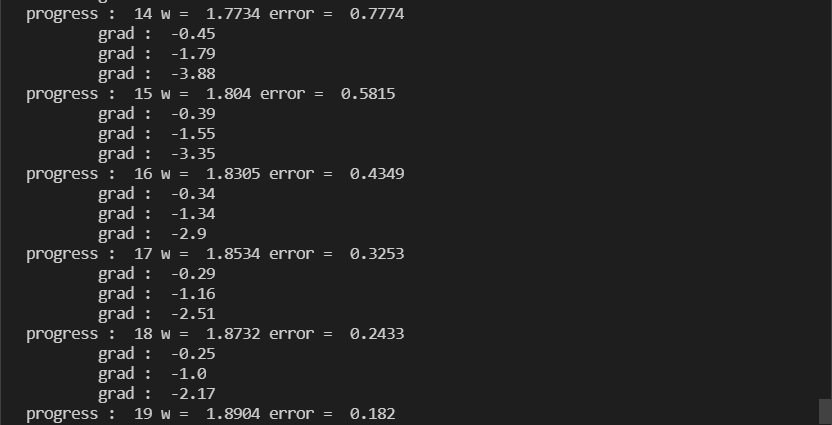
**시연영상**

**https://youtu.be/v9\_pZCAUaLo**

**실험결과**

* **Cost function을 통해서 나온 값이 계속적으로 반복되면서 현재 데이터 상태들에게**
* **가장 적합한 w으로 탐색되게 된다.**

****

****

**개선점**

* 현재는 데이터를 1차 식인 ax+b의 형태로 학습시키게 된다. 데이터의 형태가 다분히 복잡하면 예측과는 다른 근사값을 출력할 수 있다.
* 빠른 탐색을 위해서는 learning rate변수 값을 조절할 필요가 있다.
* 변수에 따라서 알고리즘이 최적의 값으로 수렴하기 위한 반복회수가 차이가 많이 나며, 이는 알고리즘 속도에 영향을 주기에 적절한 변수 선택이 필요하다.