Lab - Gradient descent

Introduction

[PDF 파일 다운로드]()

이 번 랩은 우리가 강의를 통해 들은 Gradient descent을 활용하여 LinearRegression 모듈을 구현하는 것을 목표로 한다. 앞서 우리가 Normal equation lab을 수행하였듯이, LinearRegression 모듈의 구현을 위해서는 numpy와 Python OOP의 기본적인 개념이해가 필요하다. 실제 Tensorflow의 optimizer나 scikit-learn SGDRegressor의 가장 기본적인 형태로 이해하면 좋겠다.

backend.ai 설치

숙제를 제출하기 앞서, 레블업의 backend.ai를 여러분의 파이썬에 설치하셔야 합니다. 설치하는 과정은 매우 쉽습니다. 아래처럼 터미널 또는 cmd 창에서 입력을 하시면 됩니다.

```
pip install backend.ai-client
```

숙제 파일(lab_normal_equation.zip) 다운로드

먼저 해야 할 일은 숙제 파일을 다운로드 받는 것 입니다. 아래링크를 다운로드 하거나 Chrome 또는 익스플로러와 같은 웹 브라우저 주소창에 아래 주소를 입력합니다.

- 링크 lab_gradient_descent.zip
- https://s3.ap-northeast-2.amazonaws.com/teamlab-gachon/mooc_pic/lab_gradient_descent.zip

또는 Mac OS에서는 아래 명령을 쓰셔도 됩니다.

```
wget https://s3.ap-northeast-2.amazonaws.com/teamlab-gachon/mooc_pic/lab_gradient_
```

다운로드 된 `lab_gradient_descent.zip` 파일을 작업 폴더로 이동한 후 압축해제 후 작업하시길 바랍니

다. 압축해제 하면 폴더가 `linux_mac`과 `windows`로 나눠져 있습니다. 자신의 OS에 맞는 폴더로 이동해서 코드를 수정해 주시기 바랍니다.

linear_model.py 코드 구조

본 Lab은 LinearRegressionGD class를 만들기 위해 `linear_model.py` 라는 template 파일을 제공합니다. 앞서 normal equation lab가 모듈명은 동일하나 생성하는 클래스명이 `LinearRegressionGD`으로 차이가 납니다. 제공하는 template 파일은 아래와 같은 구조를 가지며 각 변수와 함수의 역할은 아래 테이블과 같습니다.

이름	종류	개요
fit	function	Linear regression 모델을 적합한다. Matrix X와 Vector Y가 입력 값으로 들어오면 Grdient descent를 활용하여, weight값을 찾는다.
predict	function	적합된 Linear regression 모델을 사용하여 입력된 Matrix X의 예측값을 반환한다. 이 때, 입력된 Matrix X는 별도의 전처리가 없는 상태로 입력되는 걸로 가정한다.
cost	function	Matrix X와 Hypothesis function을 활용해 Vector Y와 차이를 구한다.
hypothesis_function	function	Matrix X와 Weight vector를 활용하여 Y의 예측치를 구한다.
gradient	function	Matrix X와 Error(Y^hat - Y), 그리고 theta 값을 활용하여 gradient를 구한다.
coef	property	normal equation의 결과 생성된 계수 (theta, weight) 값들을 numpy array로 저장한다.
intercept	property	normal equation의 결과 생성된 절편 (theta_0, weight_0) 값을 scalar 로 저장한다.
weights_history	property	100번씩 Gradient update가 될 때 마다 weight 값을 list로 저장한다.
cost_history	property	100번씩 Gradient update가 될 때 마다 cost 값을 list로 저장한다.

```python
import numpy as np


class LinearRegressionGD(object):
    def __init__(self, fit_intercept=True, copy_X=True,
                 eta0=0.001, epochs=1000, weight_decay=0.9):
        self.fit_intercept = fit_intercept
        self.copy_X = copy_X
        self._eta0 = eta0
        self._epochs = epochs

        self._cost_history = []

        self._coef = None
        self._intercept = None
        self._new_X = None
        self._w_history = None
        self._weight_decay = weight_decay

    def cost(self, h, y):
        pass

    def hypothesis_function(self, X, theta):
        pass

    def gradient(self, X, y, theta):
        pass

    def fit(self, X, y):
        # Write your code

        for epoch in range(self._epochs):
            # 아래 코드를 반드시 활용할 것
            gradient = self.gradient(self._new_X, y, theta).flatten()

            # Write your code

            if epoch % 100 == 0:
                self._w_history.append(theta)
                cost = self.cost(
                    self.hypothesis_function(self._new_X, theta), y)
                self._cost_history.append(cost)
            self._eta0 = self._eta0 * self._weight_decay

        # Write your code


    def predict(self, X):
        pass

    @property
    def coef(self):
        return self._coef
```

```
    @property
    def intercept(self):
        return self._intercept

    @property
    def weights_history(self):
        return np.array(self._w_history)

    @property
    def cost_history(self):
        return self._cost_history
```

Mission

이번 랩의 목적은 Gradient descent를 직접구현해 보는 것이다. 물론 우리가 수업을 통해 Gradient descent를 어떻게 구현하는지 설명하였기 때문에 쉽게 느낄 수 도 있으나, 이번 랩은 한번에 여러개의 변수가 있을 때 손쉽게 graident를 구하는 문제를 풀 것이다.

fit 함수 만들기

fit 함수는 Linear regression 모델을 적합하는 함수 이다. 본 함수는 Matrix X와 Vector Y가 입력 값으로 들어오면 Gradient descent을 활용하여, weight값을 찾는다. 이 때, fit_intercept 설정에 따라 다른 방식으로 적합이 실행이 달라진다. 또한 fit을 할 때는 입력되는 X의 값은 반드시 새로운 변수(self._new_X)에 저장한 후 실행되어야 한다.

> fit_intercept가 True일 경우: Matrix X의 0번째 Column에 값이 1인 column vector를 추가한다.
> 단 fit_intercept가 False일 경우의 Matrix X의 변수는 2개 이상일 것이라고 가정한다.

Gradient descent를 할 때 몇 가지 고려해야 하는 변수가 있다.

- self._eta0 : Learning rate를 의미한다. 초기값은 0.001으로 설정한다.
- self._epochs : 전체 Training data를 몇 번 학습할 것인지를 결정한다. 초기값은 1000이다.
- self._weight_decay : 1번 Training data를 학습할 때 마다, Learning rate인 self._eta0을 weight_decay 비율로 줄인다. 초기값은 0.9이다.
- self._w_history : 전체 데이터를 100번 학습할 때마다 현재의 weight 값을 저장한다.
- self._cost_history : 전체 데이터를 100번 학습할 때마다 현재의 cost 값을 저장한다.

fit을 할 때는 아래의 Gradient descent의 공식을 활용한다.

$$w_n =: w_n - \alpha \frac{\partial J}{\partial w_n}$$

Gradient descent할 때 반드시 아래 코드를 사용해야 한다. 아래 코드에서 `self.gradient` 함수가 gradient를 구하는 함수이다. Gradient를 구하는 공식은 아래와 같다.

```
gradient = self.gradient(self._new_X, y, theta).flatten()
```

$$\frac{\partial J}{\partial w_n} = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}) \cdot x_n$$

학습을 할 때 매 100번 마다 weight값과 cost 값을 `self._w_history`, `self._cost_history`에 저장한다.

적합이 종료된 후 각 변수의 계수(coefficient 또는 weight값을 의미)는 self._coef와 self._intercept_coef에 저장된다. 이때 self._coef는 numpy array을 각 변수항의 weight값을 저장한 1차원 vector이며, self._intercept_coef는 상수항의 weight를 저장한 scalar(float) 이다. 입력되는 parameter와 리턴은 아래 정보를 참고한다.

```
Parameters
----------
X : numpy array, 2차원 matrix 형태로 [n_samples,n_features] 구조를 가진다
y : numpy array, 1차원 vector 형태로 [n_targets]의 구조를 가진다.

Returns
-------
self : 현재의 인스턴스가 리턴된다
```

predict 함수 만들기

적합된 Linear regression 모델을 사용하여 입력된 Matrix X의 예측값을 반환한다. 이 때, 입력된 Matrix X는 별도의 전처리가 없는 상태로 입력되는 걸로 가정한다.

> fit_intercept가 True일 경우: Matrix X의 0번째 Column에 값이 1인 column vector를추가한다.

predict 함수는 다음 수식을 사용한다.

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$$

입력되는 parameter와 리턴은 아래 정보를 참고한다.

```
Parameters
```

```
----------
X : numpy array, 2차원 matrix 형태로 [n_samples,n_features] 구조를 가진다

Returns
-------
y : numpy array, 예측된 값을 1차원 vector 형태로 [n_predicted_targets]의
    구조를 가진다.
```

cost 함수 만들기

본 함수는 예측치 y_hat과 실제 값 y 값간의 차이를 cost 함수를 구하는 공식으로 산출한다. 산출하는 공식은 아래와 같으며, 반드시 return을 포함하여 한줄로 함수를 만들어야 한다.

$$J = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)})^2$$

```
Parameters
----------
h : numpy array, 2차원 matrix 형태로 [-1, n_predicted_targets]의
    구조를 가진다.
y : numpy array, 실제 y 값을 1차원 vector 형태로 [n_real_values]의
    구조를 가진다.

Returns
-------
cost_value : float형태의 scalar 값을 반환함
```

hypothesis_function 함수 만들기

본 함수는 theta 값과 X의 값이 주어지면 예측치인 Y_hat을 반환하는 함수이다. 함수에서 사용되는 공식은 아래와 같으며, 리턴되는 값의 형태는 2차원 matrix 형태이다. 반드시 return을 포함하여 한줄로 함수를 만들어야 한다.

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$$

```
Parameters
----------
X : numpy array, 2차원 matrix 형태로 [n_samples,n_features] 구조를 가진다
theta : numpy array, 가중치 weight값을 1차원 vector로 입력한다.

Returns
-------
y_hat : numpy array, 예측된 값을 2차원 matrix 형태로 [-1, n_predicted_targets]의
```

```
        구조를 가진다.
```

gradient 함수 만들기

weight 값을 업데이트 하기 위한 Gradient를 생성하는 함수이다. 함수의 반환값은 Matrix 또는 Vector형태로 반환하면 되나, 이후에 처리를 고려하여 변환해야 한다.

$$\frac{\partial J}{\partial w_n} = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}) \cdot x_n$$

```
Parameters
----------
X : numpy array, 2차원 matrix 형태로 [n_samples,n_features] 구조를 가진다
y : numpy array, 실제 y 값을 1차원 vector 형태로 [n_real_values]의
        구조를 가진다.
theta : numpy array, 가중치 weight값을 1차원 vector로 입력한다.

Returns
-------
gradient : numpy array, 공식에 의해 산출된 gradient vector를 반환함.
   이때 gradient vector는 weight의 갯수 만큼 element를 가진다.
   반환되는 형태는 자유롭게 할 수 있으나 [-1, weight_gradient]나
   [weight_gradient]로 반환 후 다음 수식에서 사용하기 위한 처리가 필요하다.
```

구현 모듈 테스트하기

여러분이 만들 모듈이 잘 작동하는지 테스트 하기 위해서 이제 `ipynb` 파일을 만들어 쥬피터 노트북에서 실행해보겠습니다.

모듈 호출하기

먼저 우리가 사용할 모듈을 호출한다. 기본적으로 pandas와 numpy를 호출한다. `linear_model` 모듈을 호출하기 위해서는 `import linear_model` 명령을 사용한다. 단 여기서 `linear_model.py` 파일이 자주 바꾸니 `imp` 모듈을 활용하여, 파일이 바뀌고 다시 저장될 때 `reload` 할 수 있도록 하여야 한다.

```
import pandas as pd
import numpy as np

import linear_model
import imp
imp.reload(linear_model)
```

데이터 불러하기

첫 번째 데이터는 one variable의 형태를 `test.py` 파일이다. 파일을 pandas로 호출하면 아래와 같은 구조를 확인할 수 있다.

```
df = pd.read_csv("./train.csv")
df.head()
```

	x	y
0	24	21.549452
1	50	47.464463
2	15	17.218656
3	38	36.586398
4	87	87.288984

numpy data 구성하기

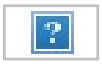
pandas로 부터 필요한 데이터를 구성한다.

```
X = df["x"].values.reshape(-1,1)
y = df["y"].values
```

Model 생성하기

LinearRegression 클래스를 사용해서 모델을 생성한다.

```
lr = linear_model.LinearRegressionGD(eta0=0.0001, epochs=500000, weight_decay=1)
lr.fit(X, y)
lr.intercept # -0.12015404827836433
lr.coef # array([ 1.00077823])
lr.predict(X)[:10]
array([ 23.89852337,  49.91875724,  14.89151934,  37.90941853,
        86.9475516 ,  35.90786208,  11.88918466,  80.94288225,
        24.8993016 ,   4.88373708])
import matplotlib.pyplot as plt
plt.plot(lr.cost_history[1000:2000])
plt.show()
```



Model Validation

구현한 LinearRegression 클래스와 scikit-learn의 linear regression 클래스로 만든 결과물을 비교한다.

```
from sklearn import linear_model
sk_lr = linear_model.LinearRegression(normalize=False)
sk_lr.fit(X, y)
sk_lr.intercept_  # -0.1201555318131966
sk_lr.coef_ # array([ 1.00077825])
np.isclose(lr.coef, sk_lr.coef_) #True
lr.predict(X_test)[:5]
# array([ 23.88223775,  50.25495698,  14.75321955,  38.08293272,  87.78536512])
```

Multiple variable

동일한 과정으로 다변수일 경우에도 작동할 수 있는지 확인한다.

```
df = pd.read_csv("./mlr09.csv")
df.head()
```

	height_in_feet	weight_in_pounds	successful_field_goals	percent_
0	6.8	225	0.442	0.672
1	6.3	180	0.435	0.797
2	6.4	190	0.456	0.761
3	6.2	180	0.416	0.651
4	6.9	205	0.449	0.900

```
y = df["average_points_scored"].values
X = df.iloc[:,:-1].values

mu_X = np.mean(X, axis=0)
```

```
std_X = np.std(X, axis=0)

rescaled_X = (X - mu_X) / std_X # RESCALED
```

Fitting 실행하기

```
lr.fit(rescaled_X, y)
lr.coef # array([-1.67758077,  0.28337482,  2.68587788,  1.12815751])
lr.intercept # 11.790740740731859

sk_lr.fit(rescaled_X, y)
sk_lr.coef_ # array([-1.67779283,  0.28359762,  2.68586629,  1.12816882])
sk_lr.intercept_ # 11.790740740740736
```

숙제 template 파일 제출하기 (윈도우의 경우)

1. `windows` + `r` 를 누르고 cmd 입력 후 확인을 클릭합니다.
2. 작업을 수행한 폴더로 이동 합니다.
3. 밑에 명령어를 cmd창에 입력합니다.

```
install.bat
submit.bat [YOUR_HASH_KEY]
```

숙제 template 파일 제출하기 (Mac or Linux)

1. 터미널을 구동합니다.
2. 작업을 수행한 디렉토리로로 이동 합니다.
3. 밑에 bash창을 입력합니다.

```
bash install.sh
bash submit.sh [YOUR_HASH_KEY]
```

> backend.ai 서비스의 업데이트에 의해 실행전 반드시 `bash install.sh` 또는 `install.bat` 수행을 바랍니다.

Next Work

고생했습니다. 현대 머신러닝에 있어 가장 핵심이 되는 알고리즘중 하나인 Gradient descent를 구현해보았

습니다. Noraml equation과 달리 진짜 프로그래밍을 하는 기분도 많이 들고 Handling 하기 상당히 어려우셨을 거 같습니다.
하지만 이제 시작입니다. 다음시간에는 좀 더 복잡한 Linear regression 구현 코드를 작성해보겠습니다!!

> **Human knowledge belongs to the world** - from movie 'Password' -