# gpa_years_experience_post

September 29, 2024

### 0.1 Linear Regression from scratch

The goal of this exercise is to implement the linear regression algorithm. The dataset is about predicting salary given gpa and years of experience. The steps to implement are as follows.

1. Read the data from a file (gpa_year_experience.csv)
2. Scale the attributes
3. Compute the error at each iteration and save the error values in vector
4. Plot the error vector as a curve in the end
5. Predict a new instance.
6. Compare with SGDRegressor
7. Create polynomial features and predict new instance

[1]:
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
# load data and show first 5 rows
data = pd.read_csv('https://raw.githubusercontent.com/thomouvic/SENG474/main/
↪data/gpa_years_experience.csv')
data.head()
```

[2]:

| | gpa | years_of_experience | salary |
|---|---|---|---|
| 0 | 70 | 1.0 | 50 |
| 1 | 80 | 2.0 | 55 |
| 2 | 65 | 2.0 | 45 |
| 3 | 70 | 2.5 | 60 |
| 4 | 65 | 2.7 | 58 |

[3]:
```
# prepare data, split columns into X and y
# X should be a numpy array of shape (m, n), use .values to convert from␣
↪dataframe to numpy array
# y should be a numpy array of shape (m,), use .values to convert from␣
↪dataframe to numpy array
```

```
X = data.drop('salary', axis=1).values
X.shape

y = data["salary"].values
y.shape
```

[3]: (25,)

[4]:
```
# extract m and n from X using X.shape[0] to get m and X.shape[1] to get n
m = X.shape[0]
n = X.shape[1]
m, n
```

[4]: (25, 2)

[5]:
```
# y should be a numpy array of shape (m, 1), use reshape(m, 1) to reshape y␣
↪from (m,) to (m, 1)
y = y.reshape(m, 1)
y.shape
```

[5]: (25, 1)

[6]:
```
# normalize X using min-max scaler (sklearn.preprocessing.MinMaxScaler)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler().fit(X)
X = scaler.fit_transform(X)
X
```

[6]:
```
array([[0.3125 , 0.     ],
       [0.625  , 0.125  ],
       [0.15625, 0.125  ],
       [0.3125 , 0.1875 ],
       [0.15625, 0.2125 ],
       [0.625  , 0.25   ],
       [0.9375 , 0.25   ],
       [1.     , 0.275  ],
       [0.     , 0.3125 ],
       [0.3125 , 0.3375 ],
       [0.5    , 0.375  ],
       [0.78125, 0.4375 ],
       [0.625  , 0.5    ],
       [0.     , 0.5625 ],
       [0.125  , 0.6    ],
       [0.     , 0.625  ],
       [0.84375, 0.625  ],
       [0.9375 , 0.6875 ],
```

```
       [0.46875, 0.75   ],
       [0.625  , 0.75   ],
       [0.46875, 0.8125 ],
       [0.3125 , 0.875  ],
       [0.625  , 0.9375 ],
       [0.9375 , 0.9625 ],
       [0.78125, 1.     ]])
```

[7]:
```python
# add dummy feature to X using scikit-learn dummy feature (sklearn.
↪preprocessing.add_dummy_feature)
from sklearn.preprocessing import add_dummy_feature
X = add_dummy_feature(X)
X
```

[7]:
```
array([[1.     , 0.3125 , 0.     ],
       [1.     , 0.625  , 0.125  ],
       [1.     , 0.15625, 0.125  ],
       [1.     , 0.3125 , 0.1875 ],
       [1.     , 0.15625, 0.2125 ],
       [1.     , 0.625  , 0.25   ],
       [1.     , 0.9375 , 0.25   ],
       [1.     , 1.     , 0.275  ],
       [1.     , 0.     , 0.3125 ],
       [1.     , 0.3125 , 0.3375 ],
       [1.     , 0.5    , 0.375  ],
       [1.     , 0.78125, 0.4375 ],
       [1.     , 0.625  , 0.5    ],
       [1.     , 0.     , 0.5625 ],
       [1.     , 0.125  , 0.6    ],
       [1.     , 0.     , 0.625  ],
       [1.     , 0.84375, 0.625  ],
       [1.     , 0.9375 , 0.6875 ],
       [1.     , 0.46875, 0.75   ],
       [1.     , 0.625  , 0.75   ],
       [1.     , 0.46875, 0.8125 ],
       [1.     , 0.3125 , 0.875  ],
       [1.     , 0.625  , 0.9375 ],
       [1.     , 0.9375 , 0.9625 ],
       [1.     , 0.78125, 1.     ]])
```

[8]:
```python
# print shapes of X and y
# X should be (m, n+1) and y should be (m, 1)
print("X shape:", X.shape)
print("u shape:", y.shape)
```

```
X shape: (25, 3)
u shape: (25, 1)
```

[9]:

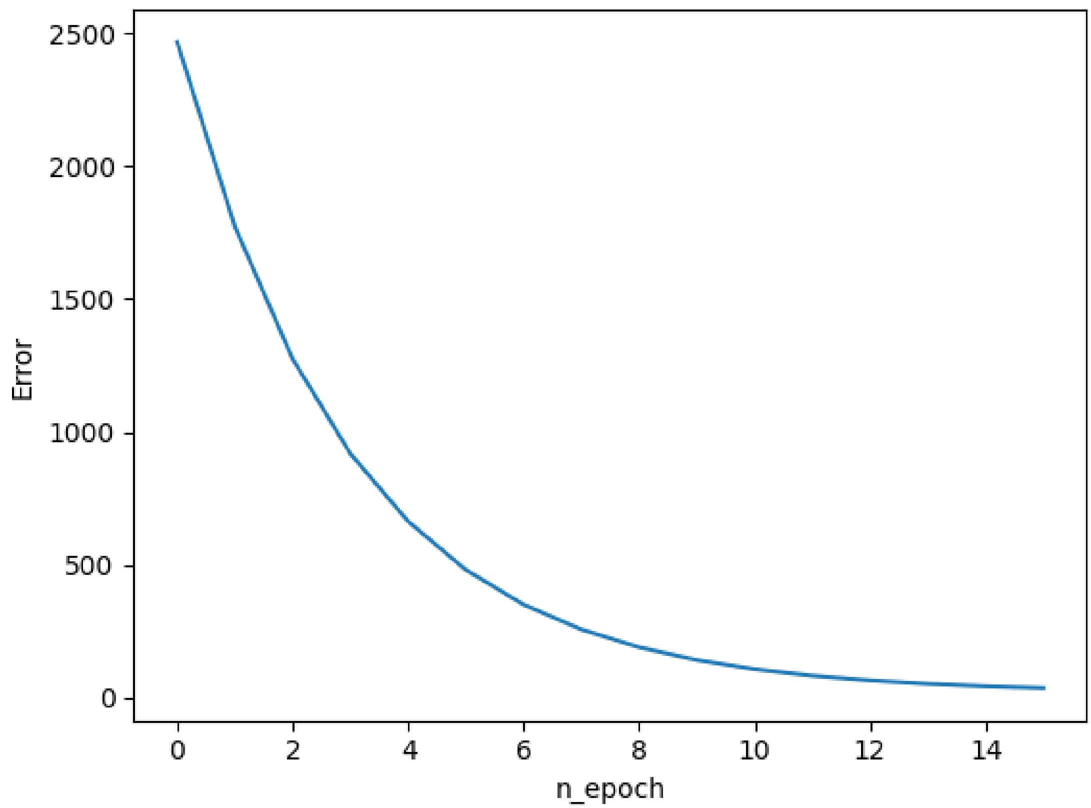
```
eta = 0.1 # learning rate
n_epochs = 15
np.random.seed(42) # set random seed to 42 for reproducibility

# create theta, of shape (n+1, 1) and initialize it to random values using np.
↪random.randn
theta = np.full(shape=[n+1, 1],
                fill_value=np.random.randn())

E = [] # list to store errors at each epoch
# compute error for initial theta and append to E
def compute_error(theta, X):
    error = np.sum((X @ theta), 1)
    error = error.reshape(len(error), 1)
    return sum((error-y) * (error-y))/(2 * m)

E.append(compute_error(theta, X))
# loop over n_epochs
# for each epoch: compute gradients, update theta, compute error, append error␣
↪to E
for i in range(0, n_epochs):
    theta -= (eta / m) * X.T @ (X @ theta - y)
    E.append(compute_error(theta, X))

# plot error vs epoch
plt.plot(E)
plt.xlabel("n_epoch")
plt.ylabel("Error")
plt.show()
# print final theta
print("Final theta:\n", theta)
```




```
Final theta:
 [[41.9484149 ]
 [20.93916703]
 [23.86242076]]
```

[10]:
```python
# let's predict the salary for a person who has gpa=70 and␣
 ↪years_of_experience=3.
# create a numpy array x of shape (1, 2) with these values
# scale features using the same scaler we used earlier
# insert dummy feature using dummy feature function
# Predict salary of x
x = np.array([[70,3]])
x = scaler.transform(x)
x = add_dummy_feature(x)
print(sum( x @ theta))
```

```
[54.45750979]
```

[11]:
```python
# Let's compare with scikit-learn's SGDRegressor
# use SGDRegressor from scikit-learn to fit the data
# use max_iter=1000, eta0=0.1, random_state=42
from sklearn.linear_model import SGDRegressor
```

```python
sgd_reg = SGDRegressor(alpha=0.1, random_state=42, max_iter=1000)
sgd_reg.fit(X, y)
```

[11]: SGDRegressor(alpha=0.1, random_state=42)

[12]:
```python
# predict salary of x using sgd
sgd_reg.predict(x)
```

[12]: array([62.504377])

[13]:
```python
# create polynomial features of degree 2 using scikit-learn PolynomialFeatures
# create X_poly using fit_transform
# create x_poly using transform
# fit the data using SGDRegressor
# predict salary of x using sgd
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
x_poly = poly.fit_transform(x)
sgd_poly = SGDRegressor()
sgd_poly.fit(X_poly, y)
sgd_poly.predict(x_poly)
```

[13]: array([59.48758531])