Data Science Assignment 3

Nike Marie Pulow – Henri Paul Heyden
stu239549 – stu240825

1 Linear Regression

1.1 Regression parameter calculation

We get:

$$\begin{aligned}
\hat{\beta} &= (X^T X)^{-1} X^T y \\
&= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 8 & 6 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 1 & 8 \\ 1 & 6 \\ 1 & 4 \end{bmatrix} \right)^{-1} \cdot \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 8 & 6 & 4 \end{bmatrix} \cdot \begin{bmatrix} 8 \\ 2 \\ 2 \\ 4 \end{bmatrix} \right) \\
&= \begin{bmatrix} 4 & 20 \\ 20 & 120 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 16 \\ 60 \end{bmatrix} \\
&= \begin{bmatrix} 1.5 & -0.25 \\ -0.25 & 0.05 \end{bmatrix} \cdot \begin{bmatrix} 16 \\ 60 \end{bmatrix} \\
&= \begin{bmatrix} 9 \\ -1 \end{bmatrix}
\end{aligned}$$

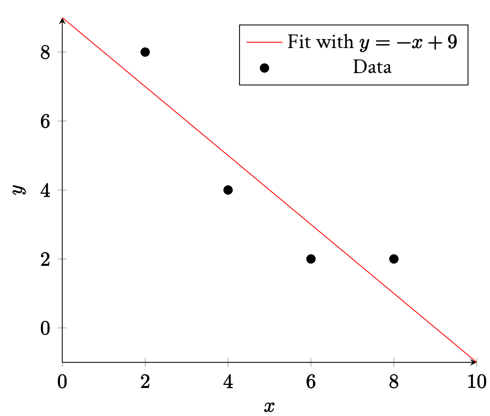
1.2 Error evaluation

For the individual errors ε, we know $y = X\hat{\beta} + \varepsilon$, we conclude $\varepsilon = y - X\hat{\beta}$ and get values:

$$\varepsilon = \begin{bmatrix} 8 - 9 + 2 \\ 2 - 9 + 8 \\ 2 - 9 + 6 \\ 4 - 9 + 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$

As we know that the SSE is $\sum_{i=1}^{n} (\beta \cdot x_i - y_i)^2$, we can also write $\sum_{i=1}^{n} {\varepsilon_i}^2$ which is 4.

1.3 Plot



MF-Homework

November 30, 2024

1 Matrix Factorization

In this task you are supposed to (manually) implement the matrix factorization variant you learned in the Data Cleaning chapter using the `numpy` library.

[108]:
```
import numpy as np
```

We continue the scenario from the tutorials.

Assume that you have a ginormous database D of three users and three movies and ratings provided by some users to some movies, which we represent as a matrix, where the entry D_{ij} represents the rating user i gave to movie j. Since not all users have rated movies, and the rating ranges from 1 to 5, we encode missing ratings as 0.

[109]:
```
# missing values encoded as 0
D = [
     [3,1,0],
     [1,0,3],
     [0,3,5],
    ]
D = np.array(D)

N = len(D)
M = len
```

First, randomly initialize the two factors E and A for $f = 2$ latent features. For evaluating the correctness of your results from the tutorial, you may *additionally* provide hard-coded inital factors as they have been provided in the tutorial.

[110]:
```
# number of latent features
f = 2

E = np.random.rand(len(D), f)
A = np.random.rand(f, len(D[0]))

print(E)
print(A)
```

```
[[0.17090442 0.67335086]
 [0.1718979  0.93930908]
```

```
 [0.82030736 0.85955506]]
[[0.09847075 0.69015436 0.19238301]
 [0.19110697 0.25447107 0.70284467]]
```

Implement a function that takes the data matrix D, the inital factors E, A, the number of epochs (iterations), the learning rate η, and performs the factorization of D. Use a default number of 5000 for the epochs and 0.001 for η.

Updates to E and A are applied immediately. $\tilde{D}$ is updated after an entry from D was completely dealt with. Update ordered by latent features and E before A.

[111]:
```
def train(D: np.ndarray, E: np.ndarray, A: np.ndarray, learning_rate: float = 0.
↪001, epochs: int = 5000) -> list[float]:
    temp = []
    for _ in range(epochs):
        Dt = np.matmul(E, A)

        # calculate the sse-gradient
        d_sse = 2*(D - Dt)

        # calculate the Summed-Squared-Error to visualize an error-curve later.
        sse = 0
        for i in range(len(D)):
            for j in range(len(D)):
                if D[i][j] != 0:
                    sse += (D[i][j] - Dt[i][j])**2
                    # parameter updates on every latent factor as well a with␣
↪epoch size 1
                    for k in range(f):
                        # hmm shouldn't there be a way to update all parameters␣
↪by directly doing matrix multiplication?
                        E[i][k] = E[i][k] + learning_rate * d_sse[i][j] *␣
↪A[k][j]
                        A[k][j] = A[k][j] + learning_rate * d_sse[i][j] *␣
↪E[i][k]
        #print(sse)
        temp.append(sse)
    return temp
```

Now test your matrix factorization for the parameters sepcified above.

[112]:
```
# "DEA" lmao
errors = train(D, E, A)
print(np.matmul(E, A))

import matplotlib.pyplot as plt

plt.title("Gradient decent matrix factorization")
plt.plot(errors)
```

```
plt.xlabel("epochs")
plt.ylabel("summed square error")
plt.legend(["error with learning rate 0.001"])
plt.show()
```

```
[[2.9992587  1.00006906 6.42077004]
 [1.0021269  1.39268882 2.99914167]
 [1.37862299 2.99988207 5.00021907]]
```