

3/16

Recommendation

Neighborhood Method:

1) (user, user) measure

2) (item, item) measure

Pros: No training

Challenges: PPI rate differently for different reasons

shift over time

upfront price for user might be high

Feature Extraction - Content-Based

Similar to SVD where we decompose.

$$p^{(j)} = \underset{p}{\operatorname{argmin}} \frac{1}{\|m^{(j)}\|} \sum_{i \in m^{(j)}} (p^T Q^{(i)} - r_{ij})^2 + \underbrace{\lambda \|p\|^2}_{\text{Regularization}}$$

reg. parameter

We don't use SVD bc we have unknown values in our matrix

Can we provide prediction w/o train features: learn P & Q at the same time

Regularization for both term

Library we might use in midterm:

"SciKit-surprise" filtering \rightarrow result might be very large & very sparse

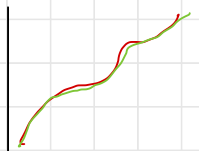
Linear Regression Challenge:

predict when alarm will go off using Linear Regression

Logistic Regression

① define a cost function to minimize the distance between $h(x_i)$, y_i .

when we have plot like this:



It's overfitting (i.e. memorize the model)

Should be:



Best fit

linear, but has randomness around it

② In terms of probability: maximize the probability observe the data $L(h) \approx P(F|h)$

These 2 above are the method we use i.e. maximize probability, or minimize loss function

Assumptions: $\vec{y} = h_{\beta}(x) + \epsilon$

① linear function

② noise

Least-Square (distance perspective)

$$\beta_{LS} = (X^T X)^{-1} X^T y$$

give the same result

Maximum Likelihood (probability approach)

$$\beta_{MLE} = \beta_{LS} = (X^T X)^{-1} X^T y$$

An Unbiased Estimator

$$E[\beta_{LS}] = \beta$$

Code:

```
1 # Linear regression by OLS
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Load data
6 data = np.loadtxt('data.txt')
7 X = data[:, 0]
8 y = data[:, 1]
9
10 # Least square estimate through first plot
11 beta = np.linalg.pinv(X.T @ X) @ X.T @ y
12
13 # plot = np.linspace(0, 10, 100)
14 plt.plot(plot, beta @ plot, 'r')
15 plt.plot(plot, y, 'b', 'o')
16 plt.show()
17
18 # Least square estimate through second plot
19 beta = np.linalg.pinv(X.T @ X) @ X.T @ y
20 X = np.array([np.ones(100), X.T]).T
21 beta = np.linalg.pinv(X.T @ X) @ X.T @ y
22
23 # plot = np.linspace(0, 10, 100)
24 plt.plot(plot, beta @ plot, 'r')
```

we want to get the constant
so not only using x , but the 1

plot quadratic

quat = $[1, a3, 5]$ \leftarrow 3 weights

