CPSC 340: Machine Learning and Data Mining

Recommender Systems

Bonus slides

SVDfeature with SGD: the gory details

Objective: $\frac{1}{2}\sum_{(i,j)\in R} (\underbrace{\hat{y}_{ij} - y_{ij}}_{r_{ij}})^2$ with $\hat{y}_{ij} = \beta + \beta_i + \beta_j + w^T x_{ij} + (w^j)^T z_i$

Update based on random (i,j):

$$\underbrace{\begin{aligned} \beta &= \beta - \alpha r_{ij} \\ \beta_i &= \beta_i - \alpha r_{ij} \\ \beta_j &= \beta_j - \alpha r_{ij} \end{aligned}}_{\text{Updates are the same,}}$$

but 'β' is always update while β_i and β_j are only updated for the specific user + product

$w = w - \alpha r_{ij} x_{ij}$ ← Updated every time.

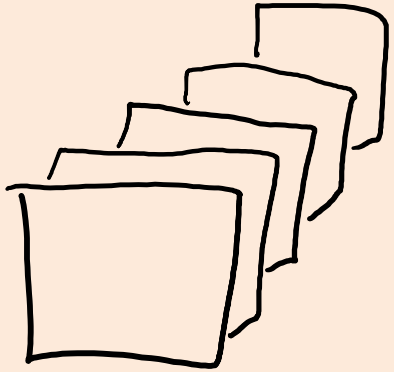
$$\left.\begin{aligned} z_i &= z_i - \alpha r_{ij} w^j \\ w^j &= w^j - \alpha r_{ij} z_i \end{aligned}\right\}$$

Updated for specific user and product.

(Adding regularization adds an extra term)

Tensor Factorization

- Tensors are higher-order generalizations of matrices:

Scalar $\alpha = \underset{1\times 1}{[\ ]}$ Vector $a = [\ ]_{d\times 1}$ Matrix $A = \underset{d\times d}{[\ ]}$ Tensor $A =$  $d \times d \times d$

- Generalization of matrix factorization is tensor factorization:

$$y_{ijm} \approx \sum_{c=1}^{k} w_{jc} z_{ic} v_{mc}$$

- Useful if there are other relevant variables:
  - Instead of ratings based on {user,movie}, ratings based {user,movie,group}.
  - Useful if you have groups of users, or if ratings change over time.

Field-Aware Matrix Factorization

- Field-aware factorization machines (FFMs):
  - Matrix factorization with multiple z_i or w_c for each example or part.
  - You choose which z_i or w_c to use based on the value of feature.

- Example from "click through rate" prediction:
  - E.g., predict whether "male" clicks on "nike" advertising on "espn" page.
  - A previous matrix factorization method for the 3 factors used:

$$w_{espn} w_{nike} + w_{espn} w_{male} + w_{nike} w_{male}$$

  - FFMs could use:

$$w_{espn}^{A} w_{nike}^{P} + w_{espn}^{G} w_{male}^{P} + w_{nike}^{G} w_{male}^{A}$$

    - wespnA is the factor we use when multiplying by a an advertiser's latent factor.
    - wespnG is the factor we use when multiplying by a group's latent factor.

- This approach has won some Kaggle competitions (link), and has shown to work well in production systems too (link).

Warm-Starting

- We've used data {X,y} to fit a model.
- We now have new training data and want to fit new and old data.

- Do we need to re-fit from scratch?

- This is the warm starting problem.
  - It's easier to warm start some models than others.

Easy Case: K-Nearest Neighbours and Counting

- K-nearest neighbours:
  - KNN just stores the training data, so just store the new data.

- Counting-based models:
  - Models that base predictions on frequencies of events.
  - E.g., naïve Bayes.
  - Just update the counts: $$p(\text{"vicodin"} \mid \text{"spam"}) = \frac{\text{count of \{vicodin, spam\} in } \underline{\text{new and old}} \text{ data}}{\text{count of "spam" in } \underline{\text{new and old}} \text{ data}}$$
  - Decision trees with fixed rules: just update counts at the leaves.

Medium Case: L2-Regularized Least Squares

- L2-regularized least squares is obtained from linear algebra:

$$w = (X^T X + \lambda I)^{-1}(X^T y)$$

  - Cost is $O(nd^2 + d^3)$ for 'n' training examples and 'd' features.
- Given one new point, we need to compute:
  - $X^T y$ with one row added, which costs $O(d)$.
  - Old $X^T X$ plus $x_i x_i^T$, which costs $O(d^2)$.
  - Solution of linear system, which costs $O(d^3)$.
  - So cost of adding 't' new data point is $O(td^3)$.

- With "matrix factorization updates", can reduce this to $O(td^2)$.
  - Cheaper than computing from scratch, particularly for large d.

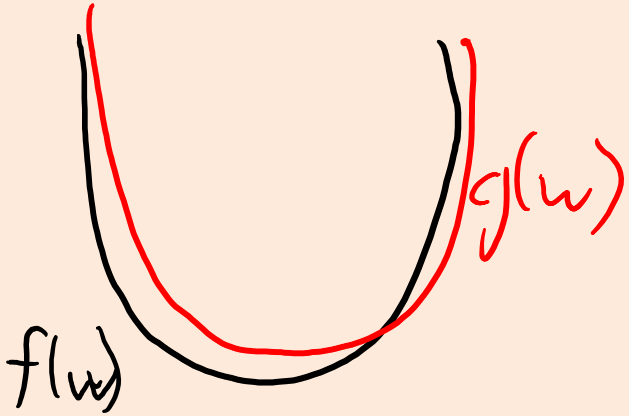
Medium Case: Logistic Regression

- We fit logistic regression by gradient descent on a convex function.

- With new data, convex function f(w) changes to new function g(w).

$$f(w) = \sum_{i=1}^{n} f_i(w) \qquad g(w) = \sum_{i=1}^{n+1} f_i(w)$$

- If we don't have much more data, 'f' and 'g' will be "close".
  - Start gradient descent on 'g' with minimizer of 'f'.
  - You can show that it requires fewer iterations.



Hard Cases: Non-Convex/Greedy Models

- For decision trees:
  - “Warm start”: continue splitting nodes that haven’t already been split.
  - “Cold start”: re-fit everything.

- Unlike previous cases, this won’t in general give same result as re-fitting:
  - New data points might lead to different splits higher up in the tree.

- Intermediate: usually do warm start but occasionally do a cold start.

- Similar heuristics/conclusions for other non-convex/greedy models:
  - K-means clustering.
  - Matrix factorization (though you can continue PCA algorithms).