COMPUTER SCIENCE
INDIANA UNIVERSITY
School of Informatics and Computing
Bloomington

Semi-supervised learning and matrix completion

Reminders/Comments

- Python speed: use built-in functions in numpy
  - e.g., numpy.dot and numpy.multiply
- Small comment: can equivalently use
  - diagonal matrix and standard matrix multiplication: C v
  - element-wise multiplication (Hadamard product) of vectors: c circa v
- Assignment due next week
- Thought questions due next week

Thought question

- What is the practical meaning of the weights in a RBF network? RBF is a non-parametric method, I think it is definitely not the importance of a feature to the target variable. Since we use a kernel trick to convert the original features.
  - Each feature now corresponds to a similarity
  - Weights could be interpreted as importance of that similarity value
  - Likely a similarity (center) is highly weighted because many data points are related to it; however, it might just be that to represent the function, that similarity needs to be highly weighted

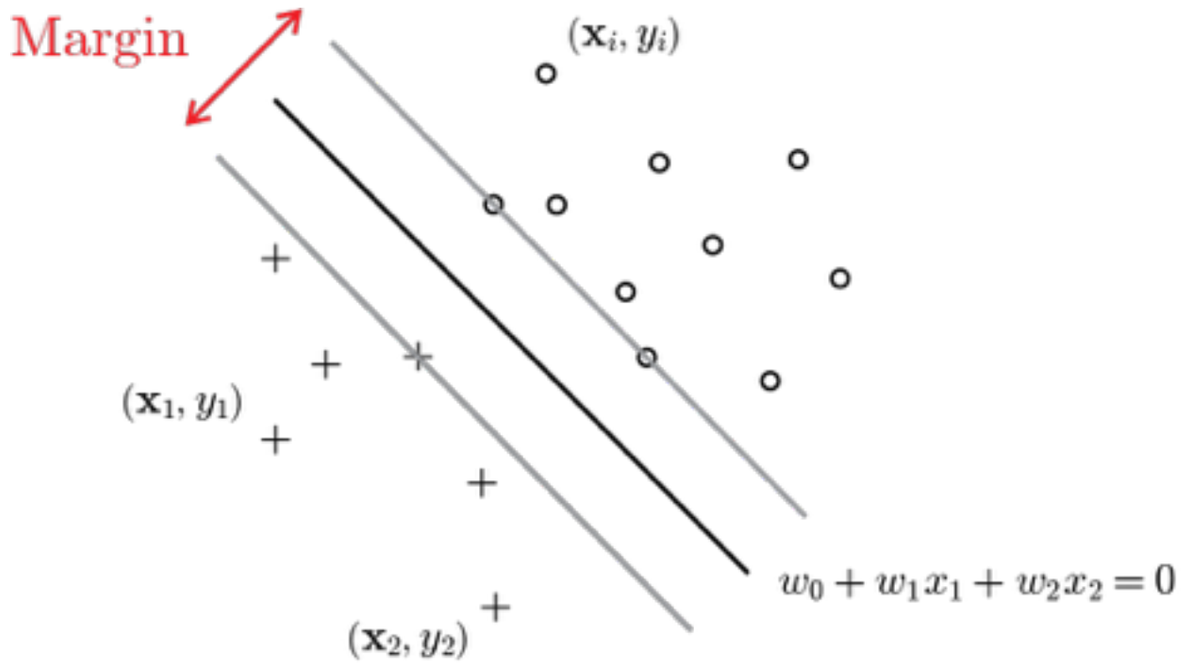
Thought question

- In the book, they explained that linear classifiers can find the relationship between input and output using linear function. Before this class, I studied about SVM (support vector machine). I think that SVM also divide data sets using linear functions. Is this the case?

Thought question

- Given that transfer functions have to be differentiable so we can compute gradients for gradient descent, are there times when a non-differentiable transfer is useful and preferred? and in such cases, how does one go about gradient-descent?
  - this is one of the reasons that differentiable functions are typically chosen for transfers
  - if the function has sub-gradients at a point, typical strategy (for neural networks and rectified linear, for example) is to simply pick one sub gradient, rather than carefully doing sub-gradient optimization
  - threshold “transfer” function for SVM

SVM clarifications



$$\mathbf{w}^T\mathbf{x}_i + w_0 > 0 \quad \Longrightarrow y_i = +1$$
$$\mathbf{w}^T\mathbf{x}_i + w_0 < 0 \quad \Longrightarrow y_i = -1$$

$$y_i(\mathbf{w}^T\mathbf{x}_i + w_0) > 0$$

$$i \in \{1, 2, \ldots, n\}$$

Idea: find $\mathbf{w}$ to maximize unsigned distance $d_i = \dfrac{y_i(\mathbf{w}^T\mathbf{x} + w_0)}{||\mathbf{w}||}$

$$(\mathbf{w}^*, w_0^*) = \arg\max_{\mathbf{w}, w_0} \left\{ \frac{1}{||\mathbf{w}||} \min_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \right) \right\}$$

Reformulating the Problem

$$(\mathbf{w}^*, w_0^*) = \arg\max_{\mathbf{w}, w_0} \left\{ \frac{1}{||\mathbf{w}||} \min_i \left(y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \right) \right\}$$

Scale $\mathbf{w}$ and w_0 such that $\min_i y_i(\mathbf{w}^\top \mathbf{x}_i + w_0) = 1$

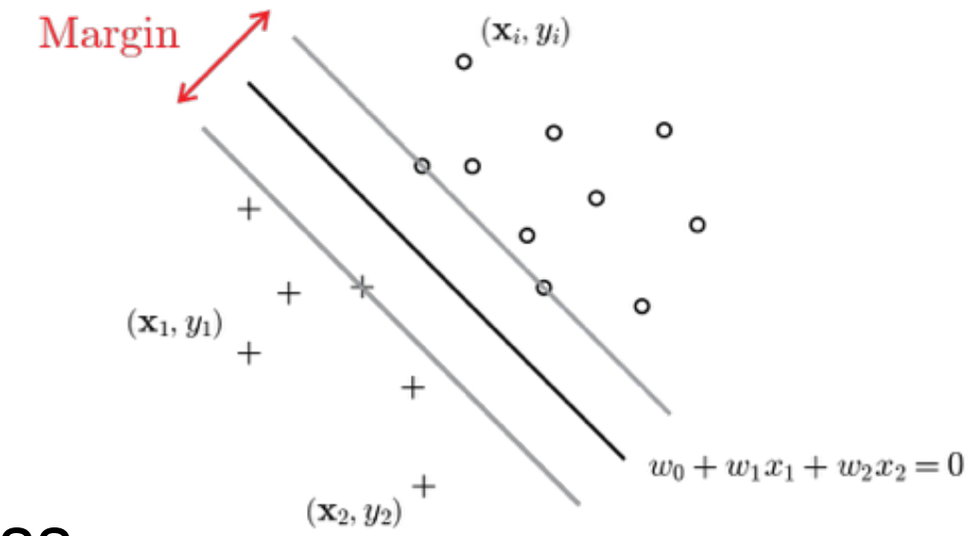
$$\mathbf{w} \leftarrow k \cdot \mathbf{w}$$
$$w_0 \leftarrow k \cdot w_0$$

$$(\mathbf{w}^*, w_0^*) = \arg\min_{\mathbf{w}} \{||\mathbf{w}||\}$$

Subject to:

$$y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1 \quad \forall i \in \{1, 2, \ldots, n\}$$

SVM clarifications



- Goal is to maximize margin, so use closest point to plane

- The resulting loss is called the hinge-loss

$$(1 - yf(x))_+$$

- "Transfer" corresponds to threshold function
  - if prediction greater than zero, predict positive
  - if prediction less than zero, predict negative

Representation learning

- Representation learning approaches can generally take advantage of this diversity of losses

- Neural networks: loss on last layer changes $L(\hat{\mathbf{y}}, \mathbf{y})$

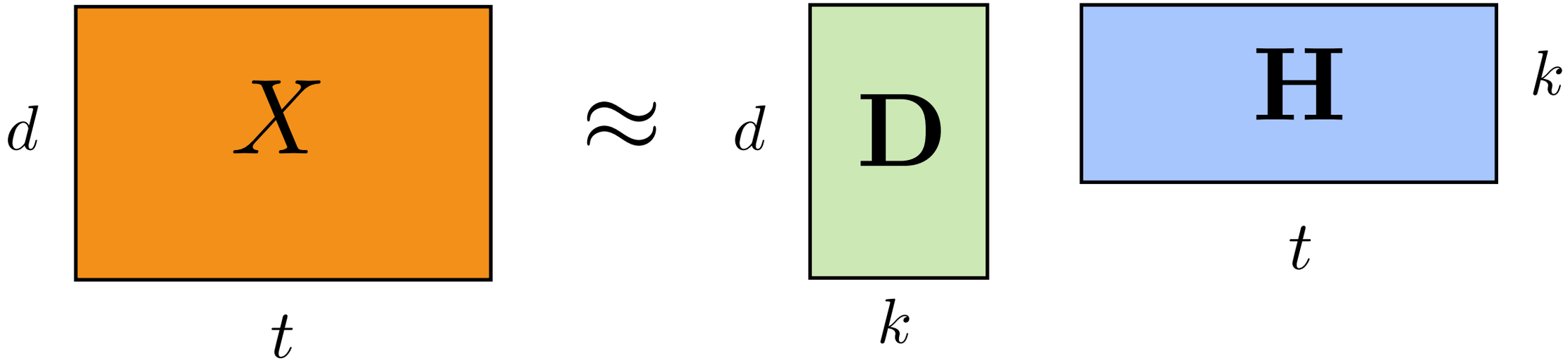
$$\hat{\mathbf{y}} = f_2(\mathbf{W}^{(2)} f_1(\mathbf{W}^{(1)}\mathbf{x}))$$

- Matrix factorization: loss on Y changes $L(\hat{\mathbf{y}}, \mathbf{y})$

$$\min_{\substack{D\in\mathbb{R}^{d\times k}\\ W\in\mathbb{R}^{m\times k}\\ H\in\mathbb{R}^{k\times t}}} \sum_{i=1}^{t} L_x(\mathbf{DH}_{:i}, \mathbf{X}_{:i}) + \sum_{i=1}^{t} L(\mathbf{WH}_{:i}, \mathbf{Y}_{:i}) + \alpha R(\mathbf{D}, \mathbf{W}, \mathbf{H})$$

$$\hat{\mathbf{y}} = f(\mathbf{Wh})$$

Unsupervised RFMs



If k < d, then we obtain dimensionality reduction (PCA)

Optimizing unsupervised RFMs

- Performed alternating minimization on

$$\min_{D \in \mathbb{R}^{d \times k}, H \in \mathbb{R}^{k \times t}} \sum_{i=1}^{t} L(\mathbf{DH}_{:i}, \mathbf{X}_{:i}) + \lambda R_D(\mathbf{D}) + \lambda R_H(\mathbf{H})$$

Initialize $\mathbf{D}, \mathbf{H}$ randomly

Until converged:

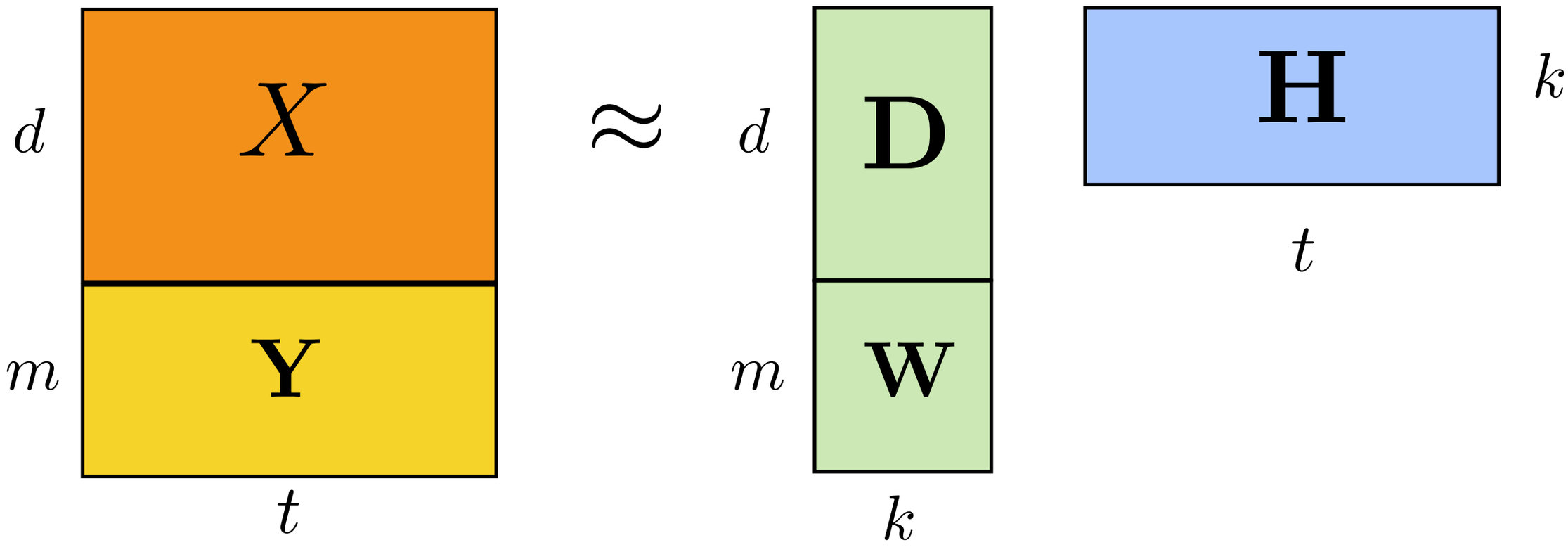
Fix $\mathbf{D}$, compute gradient w.r.t. $\mathbf{H}$:

$\mathbf{H} = \mathbf{H} - \alpha \nabla_H$

Fix $\mathbf{H}$, compute gradient w.r.t. $\mathbf{D}$:

$\mathbf{D} = \mathbf{D} - \alpha \nabla_D$

Supervised RFMs



As with generalized linear models, can use a nonlinear transfer (e.g., sigmoid)

Optimizing Supervised RFMs

- For supervised, perform alternating minimization on

$$\min_{\substack{D\in\mathbb{R}^{d\times k}\\ W\in\mathbb{R}^{m\times k}\\ H\in\mathbb{R}^{k\times t}}} \sum_{i=1}^{t} L_x(\mathbf{DH}_{:i}, \mathbf{X}_{:i}) + \sum_{i=1}^{t} L(\mathbf{WH}_{:i}, \mathbf{Y}_{:i}) + \alpha R(\mathbf{D}, \mathbf{W}, \mathbf{H})$$

Initialize $\mathbf{D}, \mathbf{W}, \mathbf{H}$ randomly

Until converged:

Fix $\mathbf{D}, \mathbf{W}$, compute gradient w.r.t. $\mathbf{H}$:

$\mathbf{H} = \mathbf{H} - \alpha\nabla_H$

Fix $\mathbf{H}$, compute gradient w.r.t. $\mathbf{D}, \mathbf{W}$:

$\mathbf{D} = \mathbf{D} - \alpha\nabla_D$

$\mathbf{W} = \mathbf{W} - \alpha\nabla_W$

Pros/cons

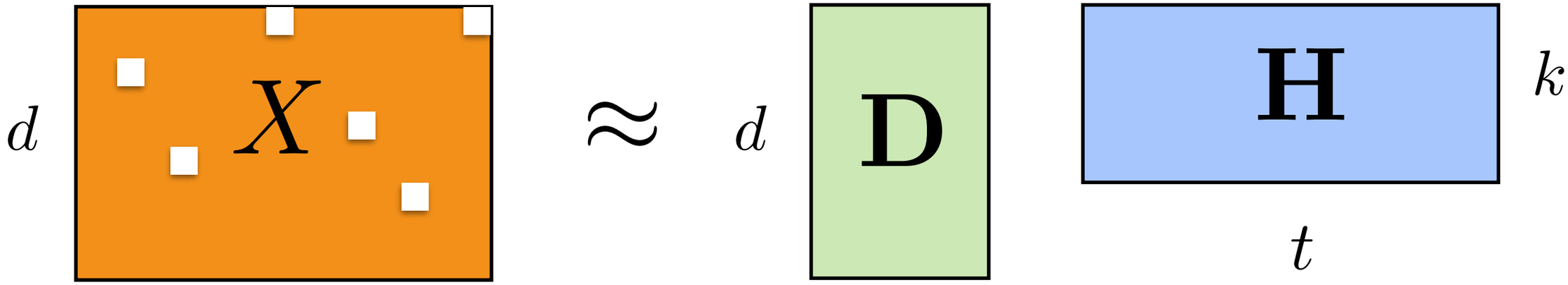
- Neural networks
  - ✓ demonstrably useful in practice
  - ✓ theoretical representability results
  - - can be difficult to optimize, due to non-convexity
  - - properties of solutions not well understood
  - - not natural for missing data

- Matrix factorization models
  - ✓ widely used for unsupervised learning
  - ✓ simple to optimize, with well understood solutions in many situation
  - ✓ amenable to missing data
  - - less well understood for supervised learning
  - - much fewer demonstration of utility

Missing data

- Missing features in both training and test
  - regression models have heuristics, like taking mean of feature
  - naive Bayes naturally handled missing features
  - strategies for neural networks similar to regression models
- Missing labels in training: called semi-supervised learning
- Factorization approaches naturally handle missing information
  - useful for missing features and semi-supervised learning
  - overall approach same as approach for missing features

Missing features for RFMs



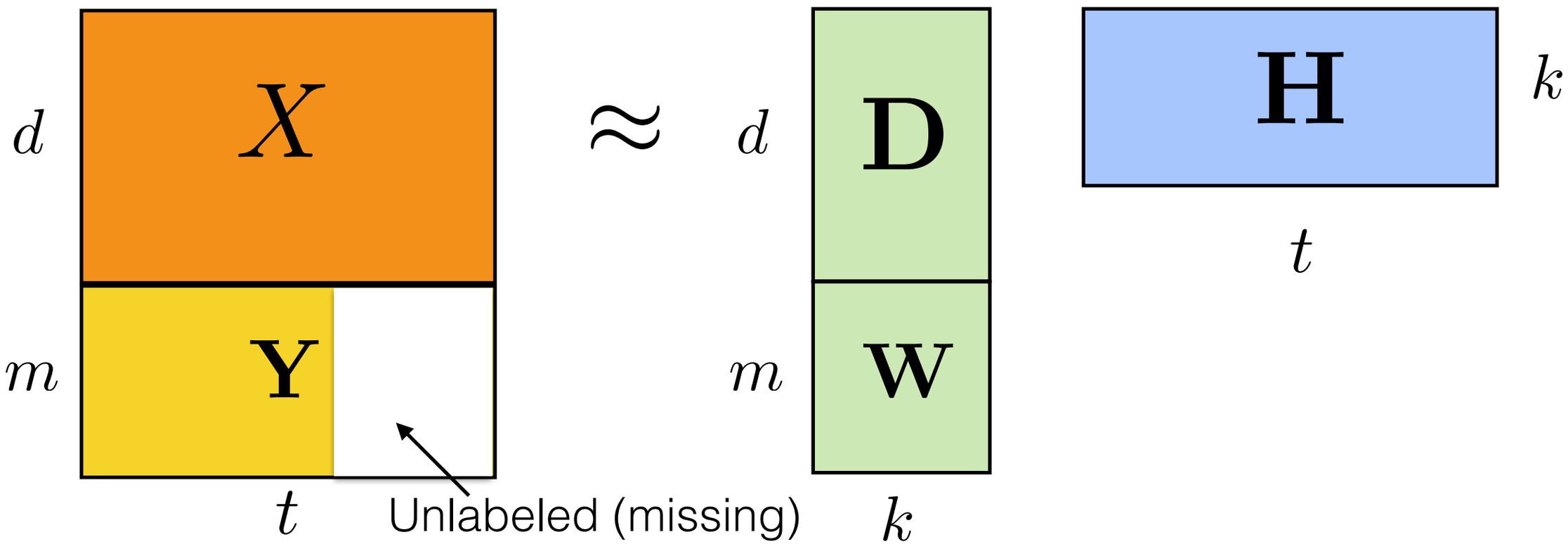
$$\min_{D \in \mathbb{R}^{d \times k}, H \in \mathbb{R}^{k \times t}} L_x(\mathbf{DH}, \mathbf{X}) + \alpha R(\mathbf{D}, \mathbf{H})$$

$$\text{e.g., } L_x(\mathbf{DH}, \mathbf{X}) = \|\mathbf{DH} - \mathbf{X}\|_F^2 = \sum_{i=1}^{d} \sum_{j=1}^{T} (\mathbf{D}_{i:}\mathbf{H}_{:j} - \mathbf{X}_{ij})^2$$

$$\rightarrow \sum_{j=1}^{T} \sum_{i:X_{ij} \text{ available}} (\mathbf{D}_{i:}\mathbf{H}_{:j} - \mathbf{X}_{ij})^2$$

Semi-supervised learning

- Semi-supervised learning: some of the data is labeled, much of it is unlabeled
  - e.g., some images are annotated with main subject, many are not
- How can we use this formulation for semi-supervised learning?



Optimizing Supervised RFMs

- For unsupervised, performed alternating minimization on

$$\min_{D\in\mathbb{R}^{d\times k},H\in\mathbb{R}^{k\times t}} \sum_{i=1}^{t} L(\mathbf{DH}_{:i},\mathbf{X}_{:i}) + \lambda R_D(\mathbf{D}) + \lambda R_H(\mathbf{H})$$

- For supervised, perform alternating minimization on

$$\min_{\substack{D\in\mathbb{R}^{d\times k}\\ W\in\mathbb{R}^{m\times k}\\ H\in\mathbb{R}^{k\times t}}} \sum_{i=1}^{t} L_x(\mathbf{DH}_{:i},\mathbf{X}_{:i}) + \sum_{i=1}^{t} L(\mathbf{WH}_{:i},\mathbf{Y}_{:i}) + \alpha R(\mathbf{D},\mathbf{W},\mathbf{H})$$

- For semi-supervised, perform alternating minimization on

$$\min_{\substack{D\in\mathbb{R}^{d\times k}\\ W\in\mathbb{R}^{m\times k}\\ H\in\mathbb{R}^{k\times t}}} \sum_{i=1}^{t} L_x(\mathbf{DH}_{:i},\mathbf{X}_{:i}) + \sum_{i:X_{:i}\ \text{labeled}} L(\mathbf{WH}_{:i},\mathbf{Y}_{:i}) + \alpha R(\mathbf{D},\mathbf{W},\mathbf{H})$$

Other strategies for semi-supervised learning

- There are many strategies
  - unfortunately still without much theoretical justification about if using unlabeled data helps classification
- State-of-the-art appears to be:
  - supervised dictionary learning (i.e., RFMs)
  - manifold regularization techniques: add regularizer that is computed from unsupervised data (LapSVM, LapRLSC)
- This has become less of a focus as we have obtained way more labeled data; but, in many situations, still have more unlabeled data than labeled data

Transductive learning

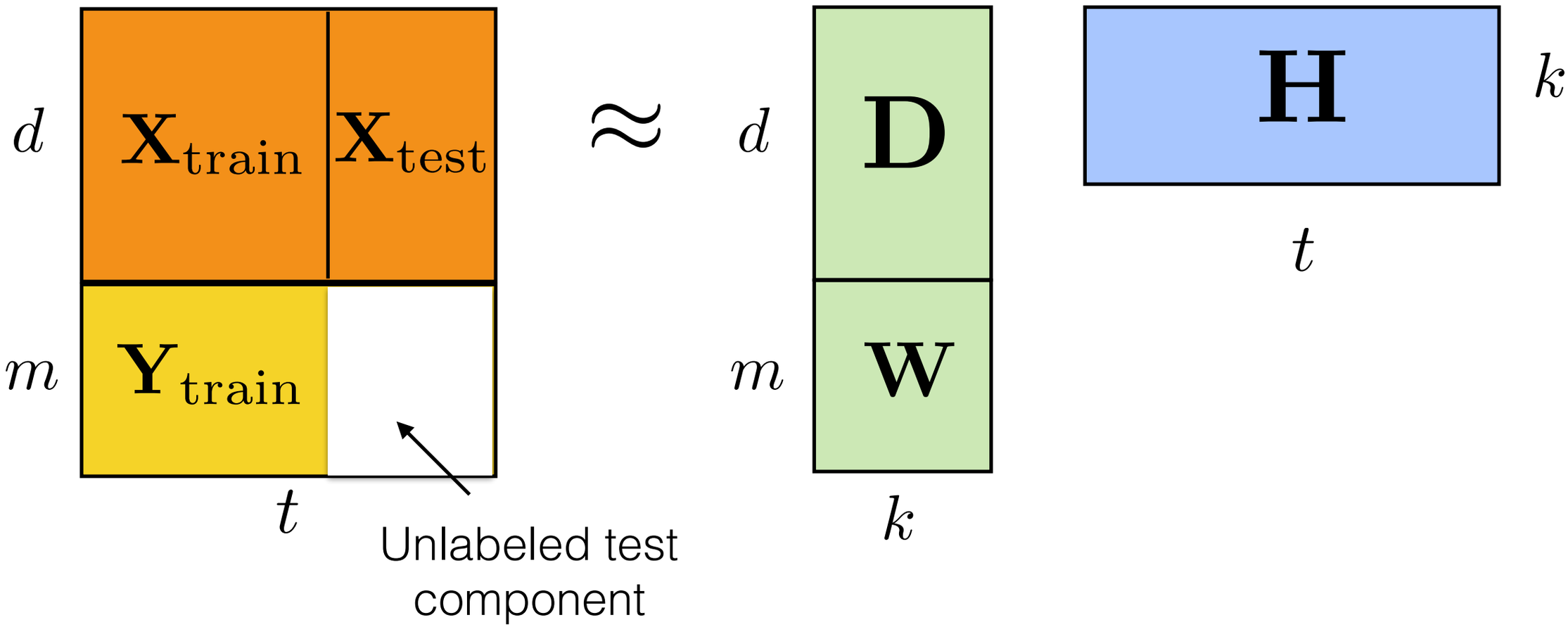
- Transductive learning uses features for test set to improve prediction performance (similar to semi-supervised learning)

- Induction (standard supervised learning): use training data to create general rule (prediction function) to apply to test cases

- Transduction: reason from both training and test cases to predict on test cases (i.e., learn model with both)
  - only training data has labels, but have access to unlabeled test cases

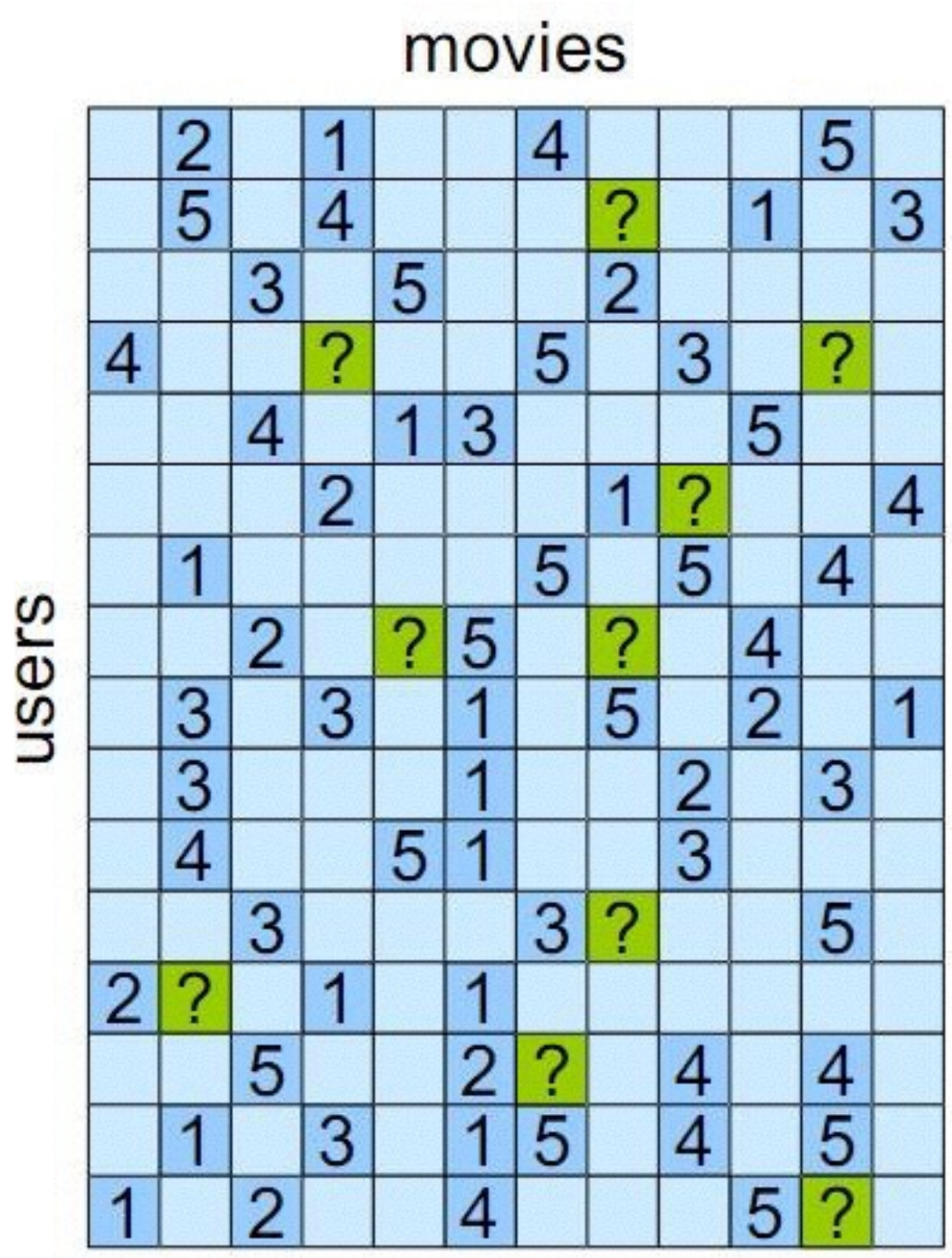
- Motivation: "When solving a problem of interest, do not solve a more general problem as an intermediate step. Try to get the answer that you really need but not a more general one."
  - “easier” to label test set specifically, rather than generalize further

Transductive learning



For non-RFM approaches, data matrix still looks as above, but do not necessarily use factorization

Matrix completion



Subspace (low-rank) form

$$\approx \quad n \quad \mathbf{D} \quad k \qquad \mathbf{H} \quad t$$

What might we need to add to our optimization?

Whiteboard

- Semi-supervised learning
- Matrix completion with the two forms:
  - subspace factorization
  - trace norm regularization