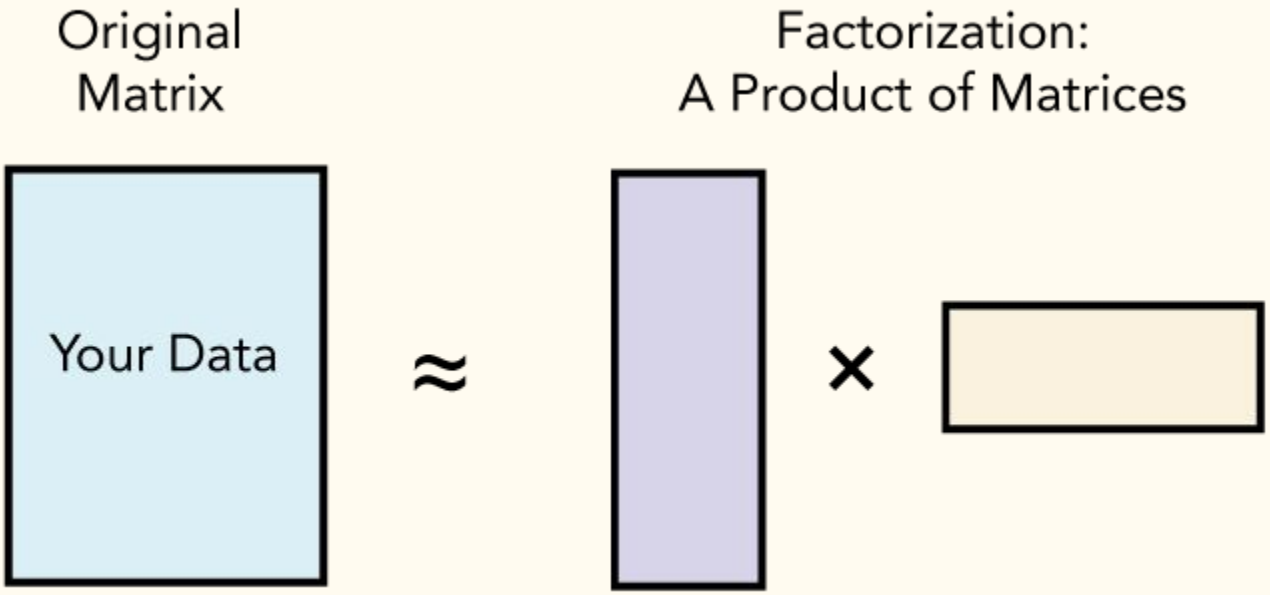
L101: Matrix Factorization

In a nutshell



Matrix factorization/completion you know?

In NLP?

- Word embeddings
- Topic models
- Information extraction
- FastText

Why complete the matrix?

Label	Features
1	f1, f2, f3, f4, f6
1	f3, f6
0	f1, f2, f5
0	f1, f2
?	f1, f3, f4
?	f2

Label	Label	f1	f2	f3	f4	f5	f6
1	1	1	1	1	1		1
1	1			1			1
0	0	1	1			1	
0	0	1	1				
	?	1		1	1		
	?		1				
		1	0	0	1	0	0
		0	1	0	0	1	0

Binary classification (transductive)
Semi-supervised
Multi-task

Matrix rank

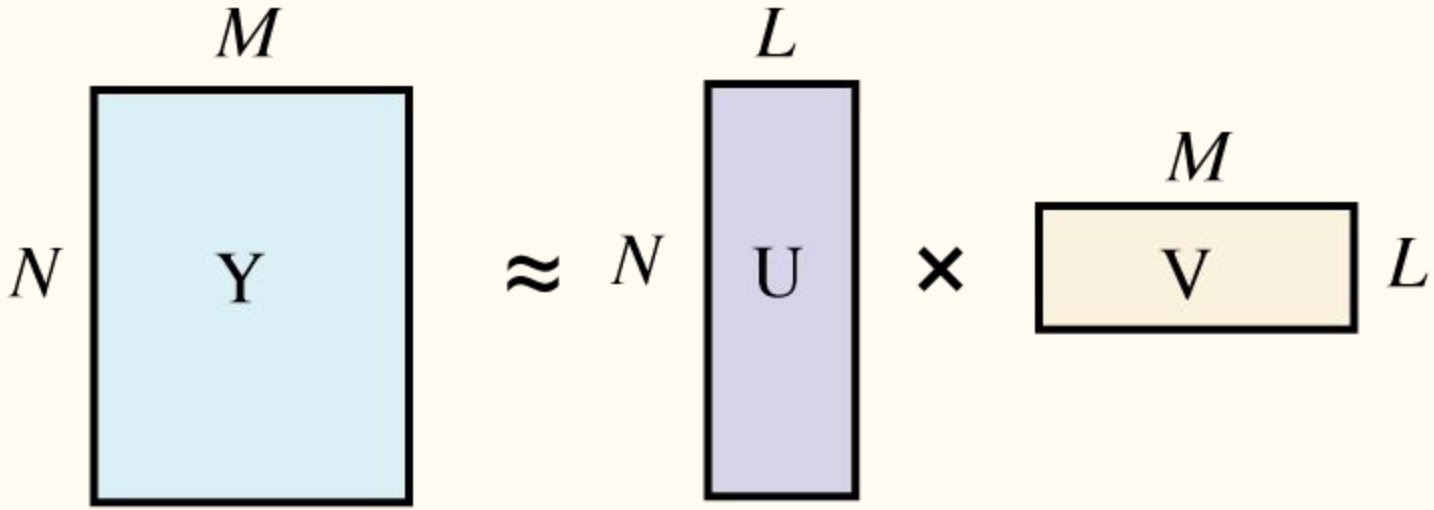
The maximum number of linearly independent columns/rows

For matrix : $\mathbf{Y} \in \mathfrak{R}^{\mathbf{N} \times \mathbf{M}}$

- if N=M=0 then rank($\mathbf{U}$) = 0
- else: max(rank($\mathbf{U}$))=min(N,M): full rank

$$rank \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 5 \\ 3 & 4 & 7 \\ 4 & 5 & 9 \end{pmatrix} = ?$$

Matrix completion via low rank factorization



Given $\mathbf{Y} \in \Re^{\mathbf{N}\times\mathbf{M}}$

Find $\mathbf{U} \in \Re^{\mathbf{N}\times\mathbf{L}}\mathbf{V} \in \Re^{\mathbf{M}\times\mathbf{L}}$, so that $\mathbf{Y} \approx \mathbf{U}\mathbf{V}^{\mathbf{T}}$

Low rank assumption: rank($\mathbf{Y}$)=L<<M,N

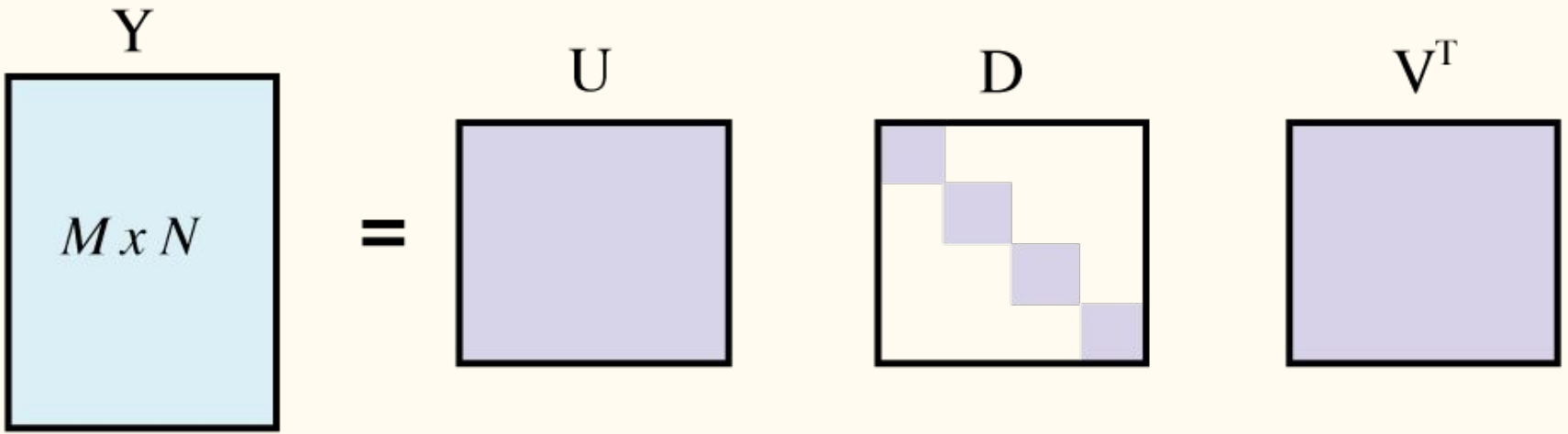
Why low rank?

Kind of odd:

- low-rank assumption usually does not hold
- reconstruction unlikely to be perfect
- if full-rank then perfect reconstruction is trivial: $\mathbf{Y}=\mathbf{YI}$

Key insight: original matrix exhibits redundancy and noise, low-rank reconstruction exploits the former to remove the latter

Singular Value Decomposition (SVD)

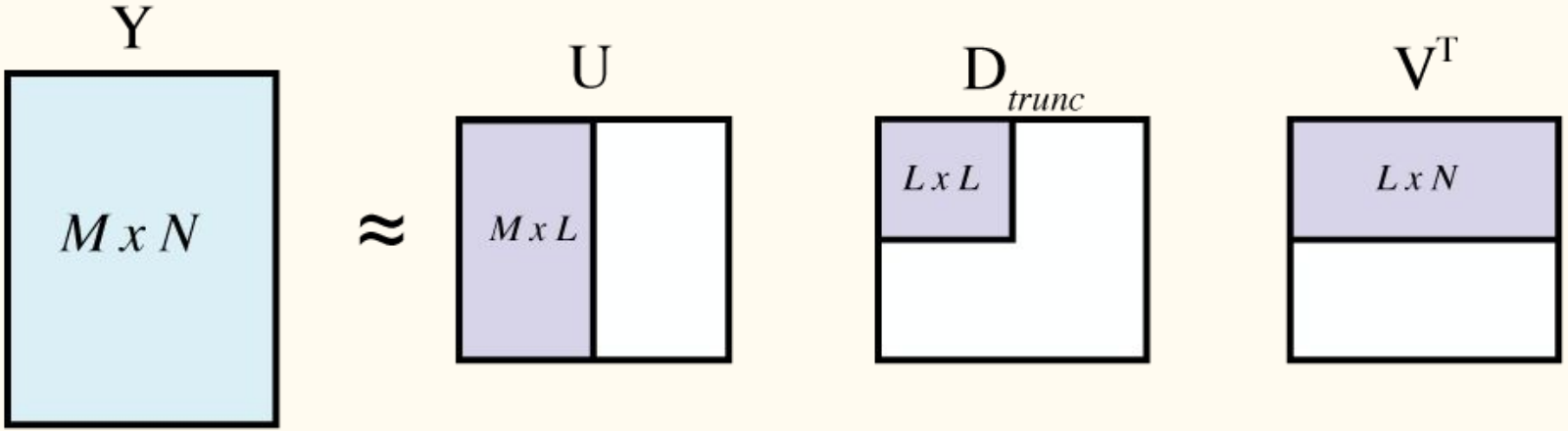


Given $\mathbf{Y} \in \Re^{\mathbf{N} \times \mathbf{M}}$

We can find orthogonal $\mathbf{U} \in \Re^{\mathbf{M} \times \mathbf{M}}, \mathbf{V}^{\mathbf{T}} \in \Re^{\mathbf{N} \times \mathbf{N}}, \mathbf{U}\mathbf{U}^{\mathbf{T}} = \mathbf{V}\mathbf{V}^{\mathbf{T}} = \mathbf{I}$

And diagonal $\mathbf{D} \in \Re^{\mathbf{M} \times \mathbf{N}} \geq \mathbf{0}$ such that $\mathbf{Y} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathbf{T}}$

Truncated Singular Value Decomposition



If we truncate **D** to its **L** largest values, then:

$$\hat{\mathbf{Y}} = \mathbf{U}\mathbf{D}_{\mathbf{trunc}}\mathbf{V}^{\mathbf{T}} = (\mathbf{U}\sqrt{\mathbf{D}_{\mathbf{trunc}}})(\sqrt{\mathbf{D}_{\mathbf{trunc}}}\mathbf{V}^{\mathbf{T}})$$

is the **rank-L** minimizer of the squared Frobenius norm:

$$\sum_{m=1}^{M}\sum_{n=1}^{N}||\hat{y}_{m,n} - y_{m,n}||^2$$

Truncated SVD

... finds the optimal solution for the chosen rank

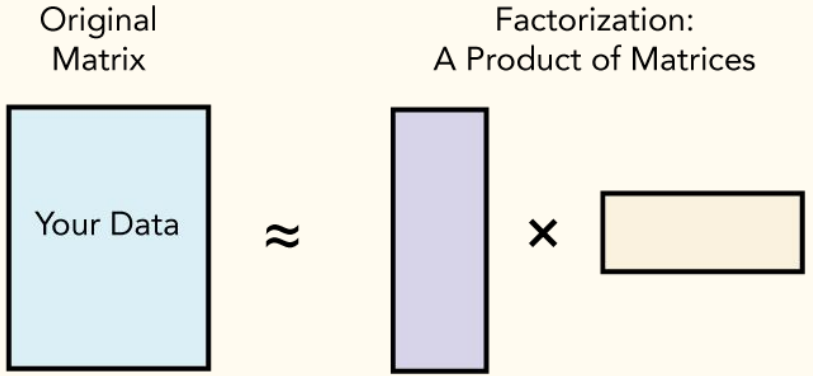
Why look further?

- SVD for large matrices is slow
- SVD for matrices with missing data is undefined
  - Can impute, but this biases the data
  - For many applications, 99% is missing (think Netflix movie recommendations)

Stochastic gradient descent (surprise!)

We have an objective to minimize:

$$\sum_{m=1}^{M} \sum_{n=1}^{N} ||\hat{y}_{m,n} - y_{m,n}||^2$$



Let's focus on the values we know **Ω**:

$$\mathbf{U}^\star, \mathbf{V}^\star = \arg\min_{\mathbf{U},\mathbf{V}} \sum_{m,n \in \Omega} (y_{m,n} - \mathbf{u}_m \cdot \mathbf{v}_n)^2$$

The gradient steps for each known value:

$$\mathbf{u}_m = \mathbf{u}_m + \alpha(y_{m,n} - \mathbf{u}_m \cdot \mathbf{v}_n)\mathbf{v}_n$$
$$\mathbf{v}_n = \mathbf{v}_n + \alpha(y_{m,n} - \mathbf{u}_m \cdot \mathbf{v}_n)\mathbf{u}_m$$

Word embeddings

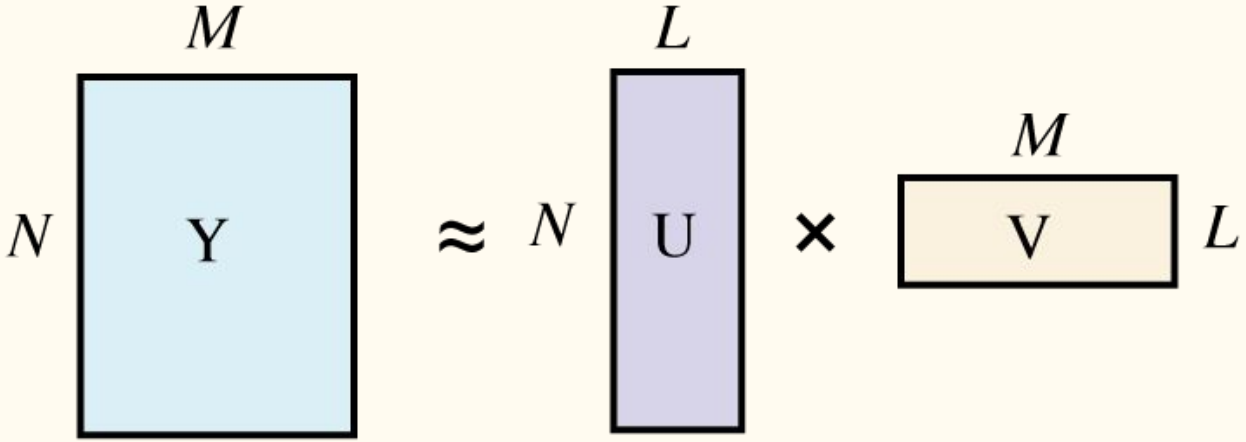
sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened
well suited to programming on the digital **computer**. In finding the optimal R-stage policy from
for the purpose of gathering data and **information** necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

Jurafsky and Martin (2019)

- SkipGram (Mikolov et al. 2013) MF implicitly
- GloVe (Socher et al. 2014), S-PPMI (Levy and Goldberg, 2014) MF explicitly

Non-negative matrix factorization

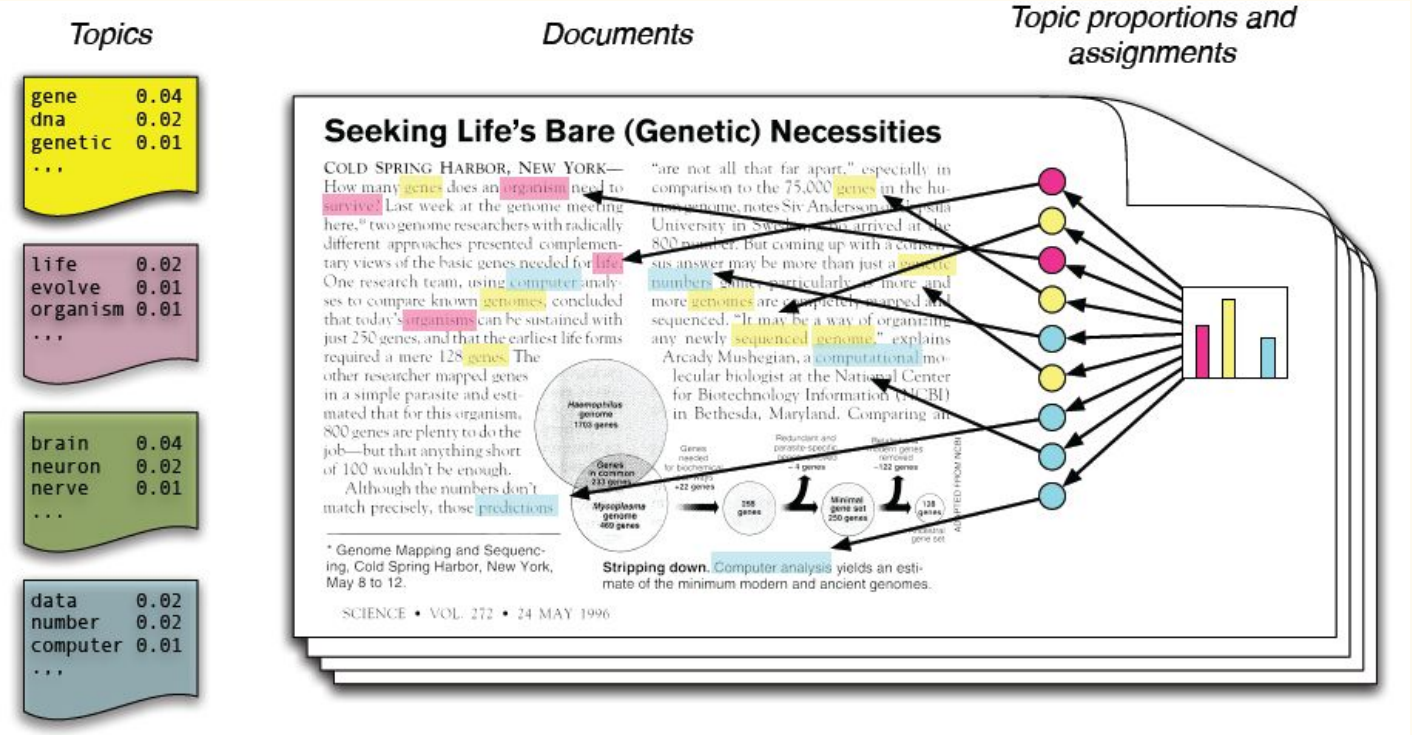


Given $\mathbf{Y} \in \Re^{\mathbf{N} \times \mathbf{M}}$

Find $\mathbf{U} \in \Re^{\mathbf{N} \times \mathbf{L}} \geq \mathbf{0}, \mathbf{V} \in \Re^{\mathbf{M} \times \mathbf{L}} \geq \mathbf{0}$, so that $\mathbf{Y} \approx \mathbf{U}\mathbf{V}^{\mathbf{T}}$

- NMF is essentially an additive mixture/soft clustering model
- Common algorithms are based on (constrained) alternating least squares

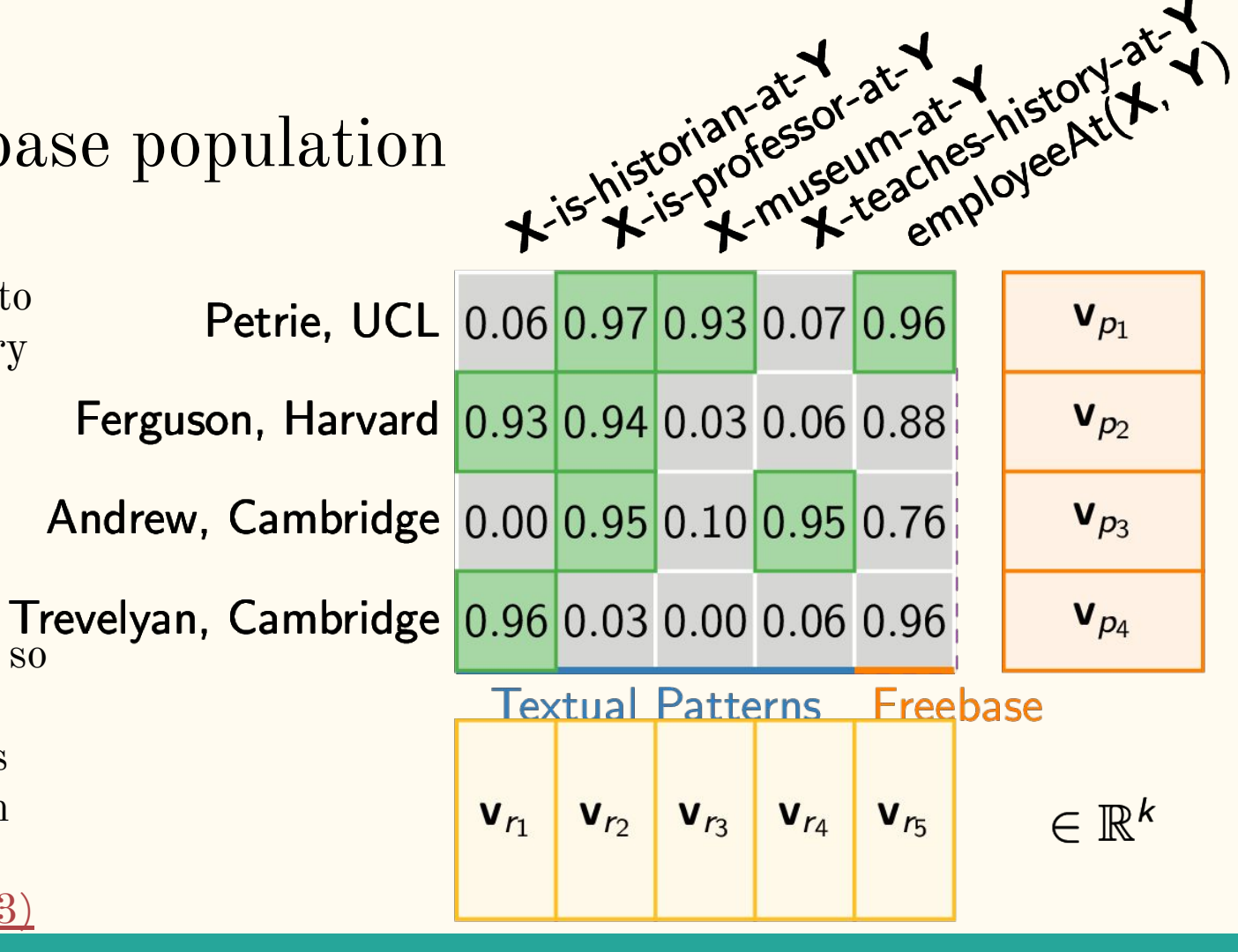
Topic models



Blei (2011)

Knowledge base population

- Sigmoid function to map reals to binary probabilities
- Combined distant supervision with representation learning
- No negative data, so just sampled negative instances from the unknown values
- Riedel et al. (2013)



Factorization of weight matrices

Remember logistic regression:

$$P(y = 1|x) = \sigma(\mathbf{w} \cdot \phi(\mathbf{x}))$$

What if we wanted to learn weights for feature interactions?

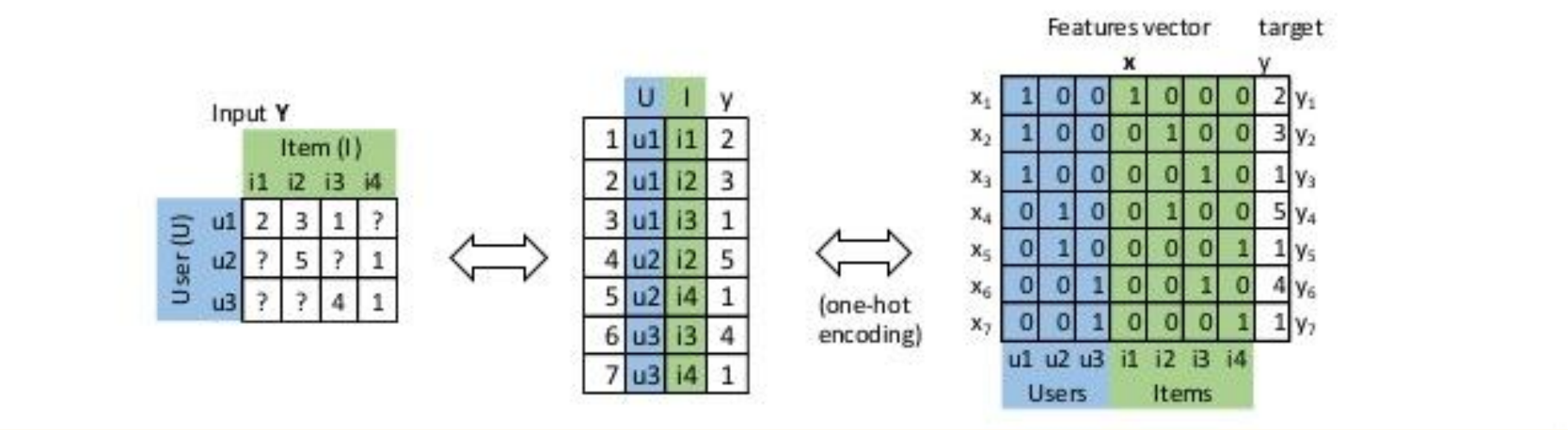
$$P(y = 1|x) = \sigma(\mathbf{w} \cdot \phi(\mathbf{x}) + \mathbf{W}(\phi(\mathbf{x})\phi(\mathbf{x})^{\mathbf{T}}))$$

Typically feature interaction observations will be sparse in the training data.
Instead of learning each weight in **W**, let's learn its low rank factorization:

$$P(y = 1|x) = \sigma(\mathbf{w} \cdot \phi(\mathbf{x}) + \mathbf{V} \cdot \mathbf{V}^{\mathbf{T}}(\phi(\mathbf{x})\phi(\mathbf{x})^{\mathbf{T}}))$$

Each vector of **V** is a feature embedding
Can be extended to high-order interactions by factorizing the feature weight tensor

Factorization Machines



Paweł Łagodziński

- Proposed by Rendle (2010)
- Can easily incorporate further features, meta-data
- Similar idea was employed for dependency parsing (Lei et al., 2014)

A different weight matrix factorization

Remember multiclass logistic regression:

$$P(y|x) = softmax(\mathbf{W} \cdot \phi(\mathbf{x})), \mathbf{W} \in \Re^{|\mathbf{Y}| \times |\phi(\mathbf{x})|}$$

For large number of labels with many sparse features, difficult to learn. Factorize!

$$P(y|x) = softmax((\mathbf{B} \cdot \mathbf{A}) \cdot \phi(\mathbf{x})), \mathbf{B} \in \Re^{|\mathbf{Y}| \times \mathbf{k}}, \mathbf{A} \in \Re^{\mathbf{k} \times |\phi(\mathbf{x})|}$$

A contains the feature embeddings and **B** maps them to labels

The feature embeddings can be initialized/fixed to word embeddings

FastText (Joulin et al., 2017) is the current go to baseline for text classification

Bibliography

The tutorial we gave at ACL 2015 from which a lot of the content was reused: http://mirror.aclweb.org/acl2015/tutorials-t5.html

- Tensors
- Collaborative Matrix Factorization

Nice tutorial on MF with code: http://nicolas-hug.com/blog/matrix_facto_1

Topic modelling and NMF: https://www.aclweb.org/anthology/D12-1087.pdf

Matrix Factorization is commonly used for model compression