W4995 Applied Machine Learning

Introduction to Recommender Systems

05/01/19

Nicolas Hug

Recommender Systems: examples

- product recommendation: books, movies
- friends recommendation
- news feed
- cars, credit cards
- Amazon, Netflix, Facebook, Youtube, Twitter...
- Ultimate goal: help the user to make decisions (and make money)
- It's vaaaaaast topic!

What we'll cover

1. Taxonomy: Collaborative Filtering, Content-based, Knowledge-based
2. Collaborative Filtering:
   - Neighborhood methods
   - Matrix factorization
3. Examples with Surprise
4. Recommendation with neural networks

Different kinds of RS (1)

Content-based recommendation

- Leverage information about items **content** and target user history
- Based on the item *profiles* (metadata)
- e.g. for movie: director, actors, genre, ...
- Looks a lot like traditional classification with text-based (extracted) features
- Tend to recommend items that are similar to those liked in the past

Oh you just bought a fridge? I'm sure you'll love these 5 other million fridges

Different kinds of RS (2)

Collaborative Filtering

- Leverage *social* information (not just info about the target user)
- typically based on past user-item interactions
  - **explicit** feedback: Bob likes *The Advengers*: ★★★★☆
  - **implicit** feedback: Bob has watched *The Advengers* -- Bob has visited a web page related to *The Advengers*

Typically:

- neighborhood methods: recommend me items liked by my peers
- or fancier models like matrix factorization

Different kinds of RS (3)

Knowledge based

- Leverage **user requirements**
- Used for cars, loans, real estate
- Very task specific

Different kinds of RS (4)

In practice, frontier isn't sharp

Models are always **hybrid** (e.g.: neural nets)

We'll talk about cornerstones of Collaborative Filtering:

- neighborhood methods (k-NN)
- matrix factorization

And also a bit of neural network recommendation

Collaborative Filtering

The rating prediction problem

$$\begin{array}{cl} \begin{pmatrix} ? & 2 & ? & 3 & 1 \\ 1 & 5 & 1 & 4 & ? \\ ? & 4 & ? & ? & ? \\ 2 & 3 & ? & 5 & 1 \\ 2 & ? & 4 & ? & 3 \\ ? & 1 & 4 & 5 & ? \end{pmatrix} & \begin{array}{c} \text{Alice} \\ \text{Bob} \\ \text{Charlie} \\ \text{Daniel} \\ \text{Eric} \\ \text{Frank} \end{array} \end{array}$$

Rows are users, columns are items

Fill the gaps!

rating prediction != classification or regression

$$\begin{pmatrix} \checkmark & ? & \checkmark & ? & ? \\ ? & ? & ? & ? & \checkmark \\ \checkmark & ? & \checkmark & \checkmark & ? \\ ? & ? & \checkmark & ? & ? \\ ? & \checkmark & ? & \checkmark & ? \\ \checkmark & ? & ? & ? & \checkmark \end{pmatrix} \neq \begin{pmatrix} \checkmark & \checkmark & \checkmark & \checkmark & \checkmark \\ \checkmark & \checkmark & \checkmark & \checkmark & \checkmark \\ \checkmark & \checkmark & \checkmark & \checkmark & \checkmark \\ \checkmark & \checkmark & \checkmark & \checkmark & \checkmark \\ \checkmark & \checkmark & \checkmark & \checkmark & ? \\ \checkmark & \checkmark & \checkmark & \checkmark & ? \end{pmatrix}$$

9 / 38

Neighborhood methods

- We have a history of past ratings
- **We need to predict Alice's rating for Titanic**

1. Find the users that have the same tastes as Alice, using the rating history
2. Average their rating for Titanic

That's it!

How to find similar users? use a **similarity metric**

Similarity computation

$$\begin{array}{c l} \begin{pmatrix} ? & 2 & ? & 2 & 5 & ? & 4 \\ 1 & 1 & ? & 3 & 5 & ? & ? \\ & & & \vdots & & & \\ 4 & 5 & ? & ? & 1 & 4 & 2 \end{pmatrix} & \begin{matrix} \text{Alice} \\ \text{Bob} \\ \vdots \\ \text{Zoe} \end{matrix} \end{array}$$

- $\text{sim}(u, v) =$ number of common rated items
- $\text{sim}(u, v) =$ average absolute difference between ratings (it's actually a distance)
- $\text{sim}(u, v) =$ cosine angle between u and v
- $\text{sim}(u, v) =$ Pearson correlation coefficient between u and v
- ...

11 / 38

Training and prediction

- Training: pre-compute the `n_users * n_users` similarity matrix
- Predicting: weighted average of the neighbors ratings

$$\hat{r_{ui}} = \frac{\sum_{v \in k\mathrm{NN}(u)} \mathrm{sim}(u, v) \times r_{vi}}{\sum_{v \in k\mathrm{NN}(u)} \mathrm{sim}(u, v)}$$

Same, with code

```
def predict_rating(u, i):
    """Return estimated rating of user u for item i."""
    # rating_history is a list of tuples (user, item, rating)

    # Retrieve users having rated i
    neighbors = [(sim[u, v], r_vj)
                 for (v, j, r_vj) in rating_history if (i == j)]
    # Sort them by similarity with u
    neighbors.sort(key=lambda tple: tple[0], reversed=True)
    # Compute weighted average of the k-NN's ratings
    num = sum(sim_uv * r_vi for (sim_uv, r_vi) in neighbors[:k])
    denum = sum(sim_uv for (sim_uv, _) in neighbors[:k])

    return num / denum
```

There are lots of variants

- Normalize the ratings
- Remove bias (some users are mean)
- Use a fancier aggregation
- Discount similarities (give them more or less confidence)
- **Use item-item similarity instead**
- Or use both kinds of similarities!
- Cluster users and/or items
- Learn the similarities
- Blah blah blah...

Matrix Factorization

Made some people rich (Netflix Prize: improve Netflix RMSE by 10%)

Model the ratings in an insightful way

Still a cornerstone of modern RS

Takes its root in dimensionality reduction and **SVD**

15 / 38

PCA refresher (1)

- Here are 400 greyscale images (64 x 64):

- Put them in a 400 x 4096 matrix X:

$$X = \begin{pmatrix} \text{—} & \text{Face 1} & \text{—} \\ \text{—} & \text{Face 2} & \text{—} \\ & \vdots & \\ \text{—} & \text{Face 400} & \text{—} \end{pmatrix}$$

PCA refresher (2)

PCA will *reveal* 400 of these eigen faces

These eigenfaces can build back all of the original faces

$$\begin{aligned}\text{Face 1} = &\alpha_1 \cdot \text{Creepy guy \#1}\\ &+\alpha_2 \cdot \text{Creepy guy \#2}\\ &+\cdots\\ &+\alpha_{400} \cdot \text{Creepy guy \#400}\end{aligned}$$

PCA also gives you the α_i.

You actually don't need all the 400 eigenfaces to have a good approximation:

PCA on a rating matrix? Sure!

Assume all ratings are known

$$X = \begin{pmatrix} \text{---} & \text{Face 1} & \text{---} \\ \text{---} & \text{Face 2} & \text{---} \\ & \vdots & \\ \text{---} & \text{Face 400} & \text{---} \end{pmatrix} \quad R = \begin{pmatrix} \text{---} & \text{Alice} & \text{---} \\ \text{---} & \text{Bob} & \text{---} \\ & \vdots & \\ \text{---} & \text{Zoe} & \text{---} \end{pmatrix}$$

Exact same thing! We just have ratings instead of pixels. PCA will reveal typical users.

$$\text{Alice} = 10\%\ \text{Action fan} + 10\%\ \text{Comedy fan} + 50\%\ \text{Romance fan} + \cdots$$
$$\text{Bob} = 50\%\ \text{Action fan} + 20\%\ \text{Comedy fan} + 10\%\ \text{Romance fan} + \cdots$$

PCA on a rating matrix? Sure!

Assume all ratings are known. Transpose the matrix

$$X = \begin{pmatrix} — & \text{Face 1} & — \\ — & \text{Face 2} & — \\ & \vdots & \\ — & \text{Face 400} & — \end{pmatrix} \qquad R^T = \begin{pmatrix} — & \text{Titanic} & — \\ — & \text{Toy Story} & — \\ & \vdots & \\ — & \text{Fargo} & — \end{pmatrix}$$

Exact same thing! PCA will reveal typical movies.

$$\begin{aligned} \text{Titanic} &= 20\%\ \text{Action} + 0\%\ \text{Comedy} + 70\%\ \text{Romance} + \cdots \\ \text{Toy Story} &= 30\%\ \text{Action} + 60\%\ \text{Comedy} + 0\%\ \text{Romance} + \cdots \end{aligned}$$

Note: in practice, the factors semantic is not clearly defined.

SVD is PCA^2

- PCA on R gives you the typical users U
- PCA on R^T gives you the typical movies M
- SVD gives you both in one shot!

$$R = M\Sigma U^T$$

Σ is diagonal, it's *just* a scaler.

$$\boxed{R = MU^T}$$

This is our matrix factorization!

The model of SVD

$$\boxed{R = MU^T}$$

$$\begin{pmatrix} & & \\ & r_{ui} & \\ & & \end{pmatrix} = \begin{pmatrix} & & \\ — & p_u & — \\ & & \end{pmatrix} \begin{pmatrix} & | & \\ & q_i & \\ & | & \end{pmatrix}$$

$$r_{ui} = p_u \cdot q_i$$

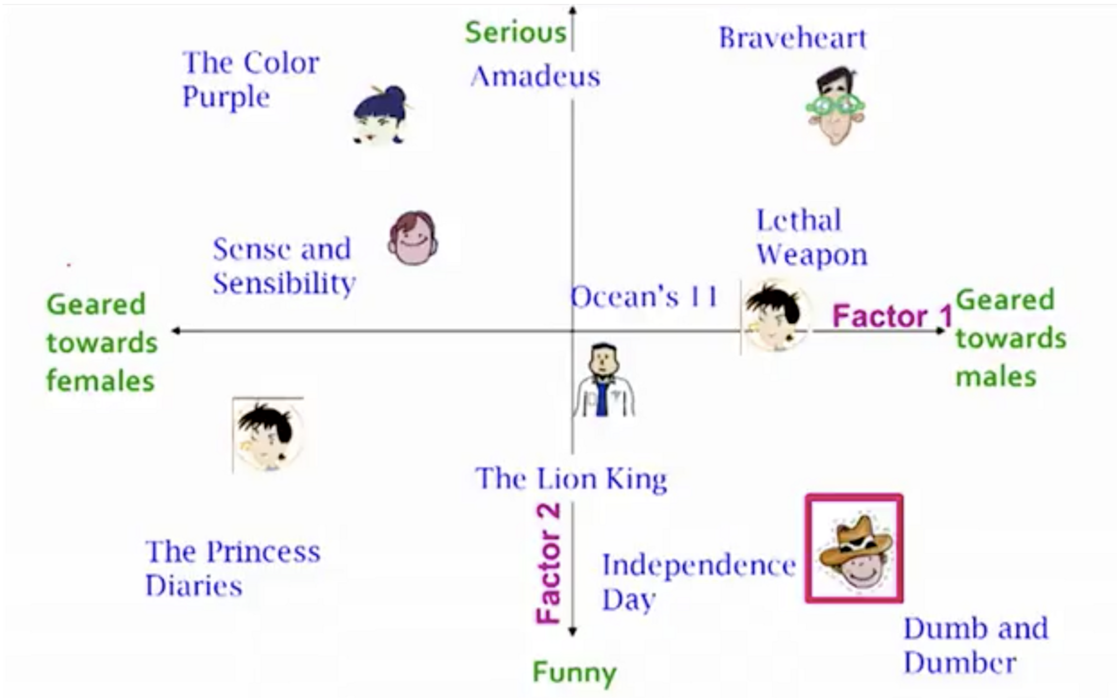
$$r_{ui} = \sum_{c \in \text{concepts}} \text{affinity of } u \text{ for } c \times \text{affinity of } i \text{ for } c$$

$$\text{Titanic} = 20\% \text{ Action} + 0\% \text{ Comedy} + 70\% \text{ Romance} + \cdots$$

$$\text{Alice} = 15\% \text{ Action} + 0\% \text{ Comedy} + 80\% \text{ Romance} + \cdots$$

22 / 38

Projection in a latent space



From Stanford [CS246]

23 / 38

So how to compute M and U?

We assumed that R had no missing entry! But it's actually very sparse

- SVD of a dense matrix is easy
- SVD of a sparse matrix is easy
- But we don't want to treat the missing entries as zero! (too biased)

Alternate option: find the p_us and the q_is that minimize the reconstruction error

$$\sum_{r_{ui} \in R} (r_{ui} - p_u^t \cdot q_i)^2$$

(With some orthogonality constraints, that we ignore)

'SVD' of a rating matrix: optimization

F = number of factors

Find $p_u \in \mathbb{R}^F$ and $q_i \in \mathbb{R}^F$ for all users and item that minimize:

$$f(p, q) = \sum_{r_{ui} \in \text{R\_train}} (r_{ui} - p_u \cdot q_i)^2$$

Classical sum of squared errors! 2 main techniques:

- Stochastic Gradient Descent
- Alternating Least Squares
  - Fix the p_u and optimize the q_i, then fix the q_i and optimize the p_u. Repeat.

Optimization with SGD

```
def compute_SVD():
    '''Fit pu and qi to known ratings by SGD'''
    p = np.random.normal(size=(n_users, n_factors))
    q = np.random.normal(size=(n_items, n_factors))
    for iter in range(n_max_iter):
        for u, i, r_ui in rating_history:
            err = r_ui - np.dot(p[u], q[i])
            p[u] = p[u] + learning_rate * err * q[i]
            q[i] = q[i] + learning_rate * err * p[u]

def predict_rating(u, i):
    return np.dot(p[u], q[i])
```

Some last details

Unbias the ratings, add regularization: you get "*SVD*":

$$\min_{p_u, q_i, b_u, b_i} \sum_{r_{ui} \in R} \begin{pmatrix} \left[r_{ui} - (\mu + b_u + b_i + p_u^T q_i)\right]^2 \\ +\lambda \left(||p_u||^2 + ||q_i||^2 + b_u^2 + q_i^2\right) \end{pmatrix}$$

Biases (or *baselines*) b_u model the tendency of some users to give high/low ratings. Same for items

Pretty far from the traditional Linear Algebra SVD

But very good at predicting ratings

Has been extended in zillions of different forms

Surprise

https://surprise.readthedocs.io/

```
pip install scikit-surprise
```

or

```
conda install -c conda-forge scikit-surprise
```

Python lib for explicit ratings prediction

```python
from surprise import SVD
from surprise import Dataset
from surprise import accuracy
from surprise.model_selection import train_test_split

data = Dataset.load_builtin('ml-100k')

trainset, testset = train_test_split(data, test_size=.25)

algo = SVD(n_factors=100, n_epochs=20, verbose=True)

algo.fit(trainset)
predictions = algo.test(testset)

accuracy.rmse(predictions, verbose=True)
```

```
RMSE: 0.9376
```

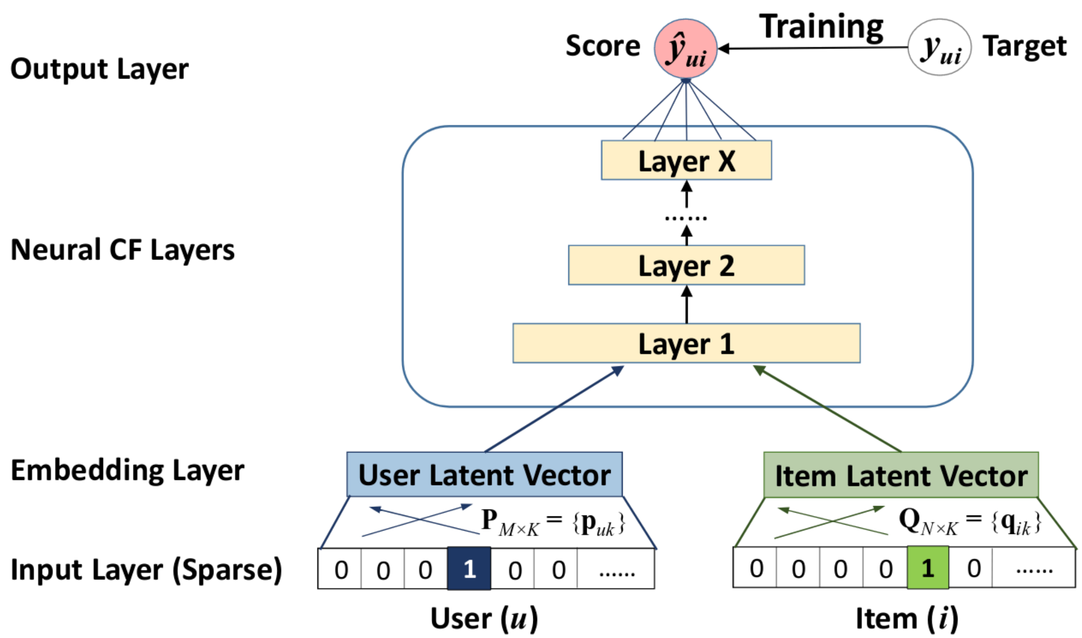
```python
from surprise import KNNBasic
from surprise import Dataset
from surprise.model_selection import cross_validate

data = Dataset.load_builtin('ml-100k')

sim_options = {'name': 'cosine',  # 'pearson', 'pearson_baseline', 'm
               'user_based': False}  # compute sim between items
algo = KNNBasic(k=40, sim_options=sim_options)

cross_validate(algo, data, cv=5, verbose=True)
```

```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.0276  1.0193  1.0186  1.0318  1.0357  1.0266  0.0068
MAE (testset)     0.8130  0.8069  0.8070  0.8151  0.8175  0.8119  0.0043
Fit time          1.57    1.64    1.55    1.68    1.72    1.63    0.06
Test time         3.58    3.31    3.39    3.58    3.64    3.50    0.13
```

Surprise

- Custom CV iterators like in `scikit-learn`
- `GridSearchCV`, `RandomizedSearchCV`
- Other prediction algorithms (MF, Neighborhood-based, baselines, etc...)
- You can also write your own

Neural recommendations



From [He & al, Neural Collaborative Filtering]()

32 / 38

Embed whatever you want (1)

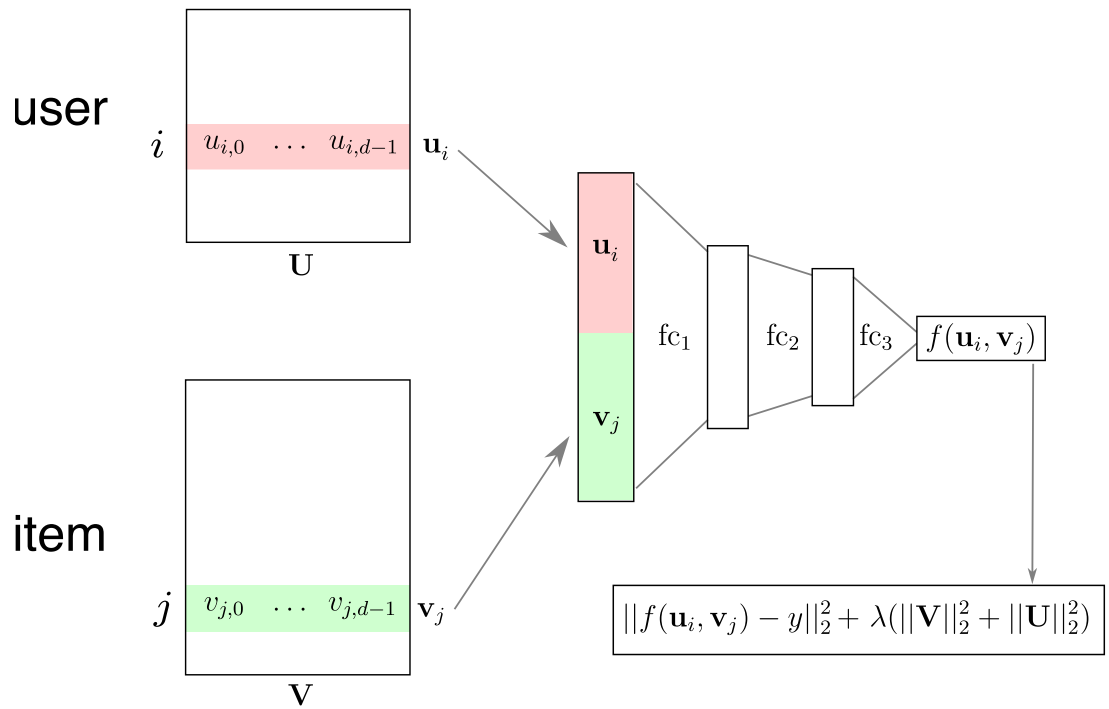


Image from Olivier Grisel

Embed whatever you want (2)

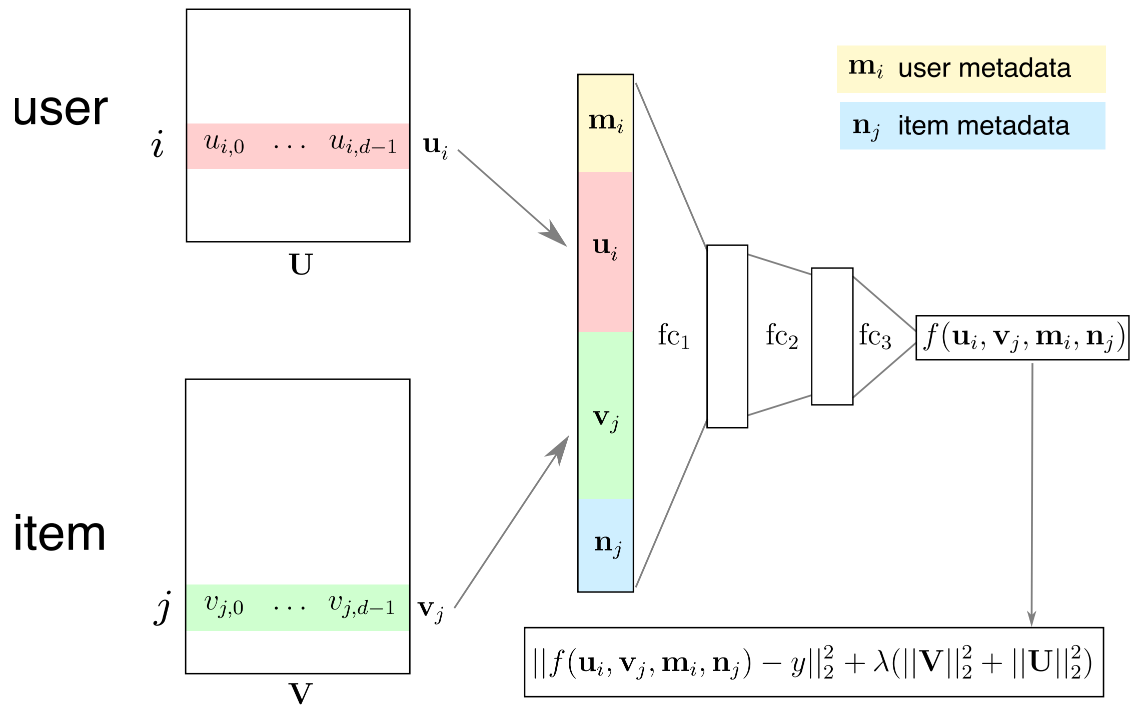
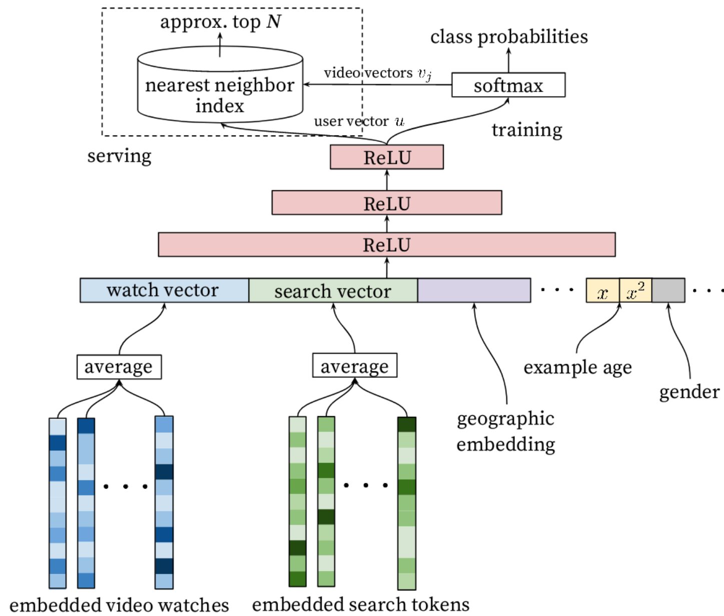


Image from [Olivier Grisel]

34 / 38

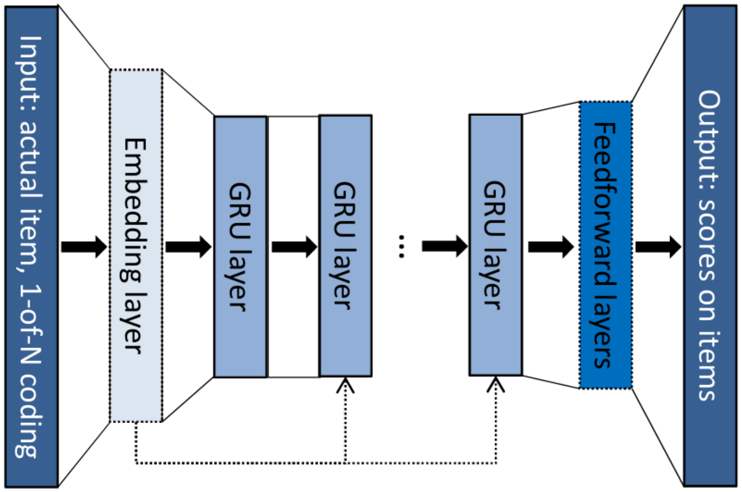
Embed whatever you want (3)



From [Deep Neural Networks for YouTube Recommendations]()

35 / 38

Recommendation as sequence prediciton with RNNs



From [Session-based Recommendations with RNNs]

Spotlight

Pytorch-based neural recommendation library

https://github.com/maciejkula/spotlight

```
conda install -c maciejkula -c pytorch spotlight=0.1.5
```

Also check out Microsoft RS repo

Collection of notebooks with many different techniques

https://github.com/Microsoft/Recommenders

37 / 38

Questions ?