NYU FRE 7773 - Week 12

Machine Learning in Financial Engineering

Jacopo Tagliabue

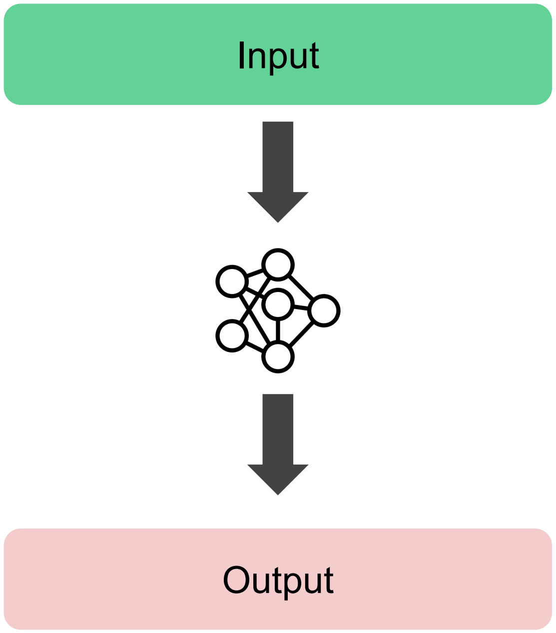
RecSys 101 (again!)

Machine Learning in Financial Engineering

Jacopo Tagliabue

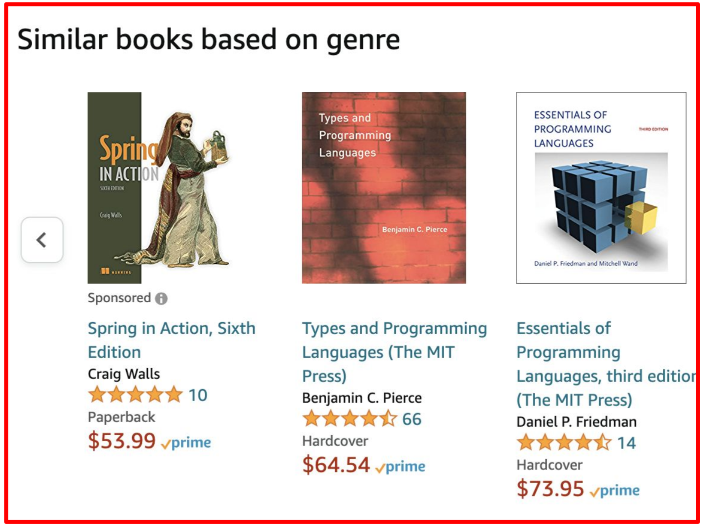
RecSys by use case

- RS can be understood easily by use case and input-output:



RecSys by use case

- RS can be understood easily by use case and input-output:
    - Item as input, item as output: similar vs complementary

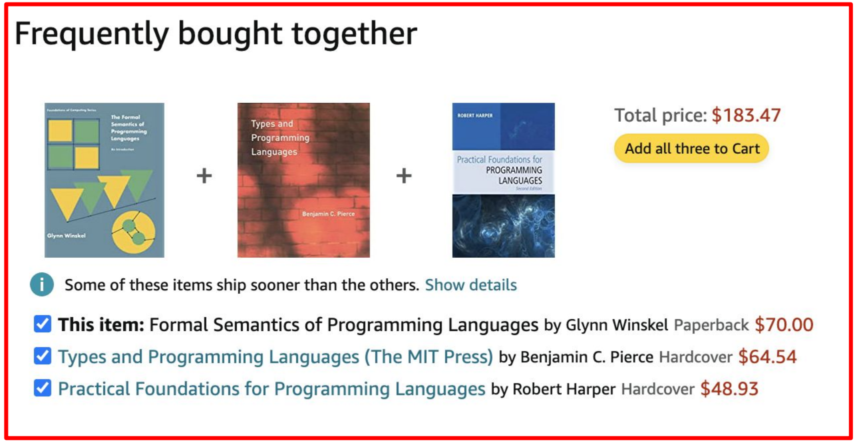


Input

RecSys by use case

- RS can be understood easily by use case and input-output:
  - Item as input, item as output: similar vs complementary

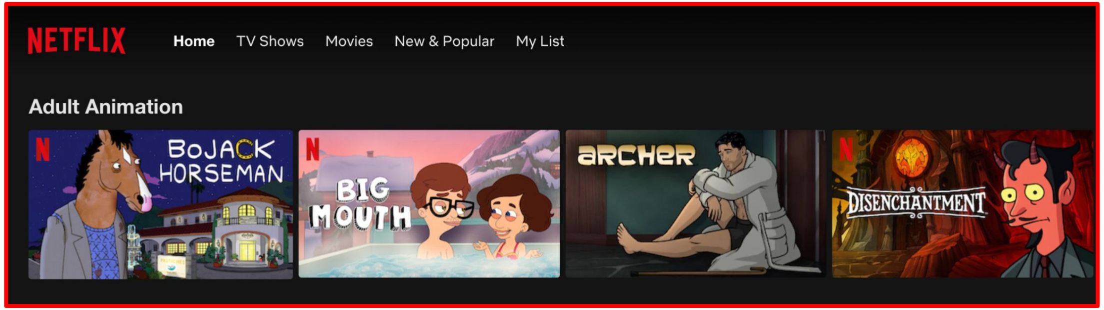
Input



RecSys by use case

- RS can be understood easily by use case and input-output:
  - Item as input, item as output: similar vs complementary
  - User as input, item as output: “for you”

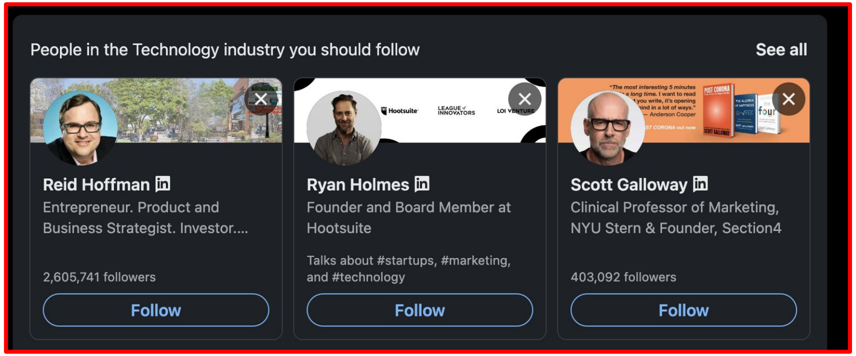
Input



RecSys by use case

- RS can be understood easily by use case and input-output:
    - Item as input, item as output: similar vs complementary
    - User as input, item as output: “for you”
    - User as input, user as output: people you may know

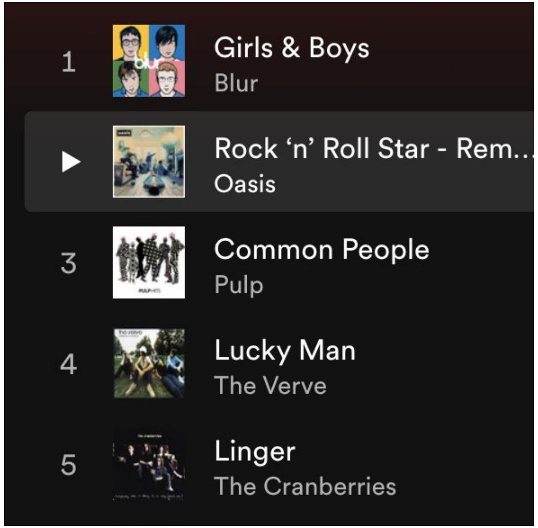
Input



RecSys by use case

- RS can be understood easily by use case and input-output:
  - Item as input, item as output: similar vs complementary
  - User as input, item as output: “for you”
  - User as input, user as output: people you may know
  - Session as input, item as output: what are you doing next?

Input



RecSys by use case

- RS can be understood easily by use case and input-output:
  - Item as input, item as output: similar vs complementary
  - User as input, item as output: “for you”
  - User as input, user as output: people you may know
  - Session as input, item as output: what are you doing next?
  - Item as input, user as output: who should we sell this to?

Input

New Fantastic SaaS Product!

RecSys by use case (with refs!)

- RS can be understood easily by use case and input-output:
  - Item as input, item as output: similar vs complementary
  - User as input, item as output: “for you”
  - User as input, user as output: people you may know
  - Session as input, item as output: what are you doing next?
  - Item as input, user as output: who should we sell this to?

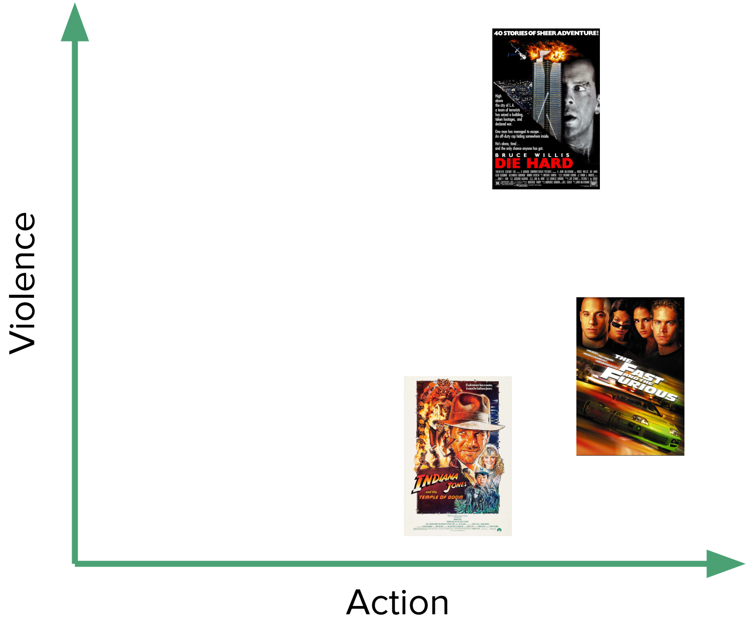
Key intuition: if you like **X**, you like things similar to **X** as well!

What does “similar” mean?

Similarity and representation

Intuition: similarity is “closeness” in a “proper” space

- Let’s map movies along two dimensions:
    - How much action is there?
    - How much violence is there?
- Observation #1: items as vectors
    - Indiana Jones: [3, 1]
    - Fast and Furious: [5, 2]
    - Die Hard: [4, 4]
- Observation #2: similar movies are close in the space
    - Back to RecSys: if you like Indiana Jones, you’re more likely to like FF then Die Hard



Intuition: similarity is “closeness” in a “**proper**” space

- If the space does not *represent* the underlying concepts well, we are in trouble!
- Machines understand vectors, but not all vectorizations define an appropriate space in this sense.
- For example, let’s consider one-hot encoding:
    - Is “cat” more similar to “dog” than “snake”?
    - Note: *how big is the vector with 1000 animals? And how sparse*?

Dog

1	**0**	**0**

Snake

0	**1**	**0**

Cat

0	**0**	**1**

Intuition: similarity is “**closeness**” in a “proper” space

- If the space *represents* the underlying concepts well, items *close* in the space will be similar, items far apart are not so similar.
- While there are many different ways to characterize “close”, cosine distance (or dot product on scaled vectors) is the most common.
- **Corollary**: “similarity inference” is “just” nearest neighbor search in the vector space

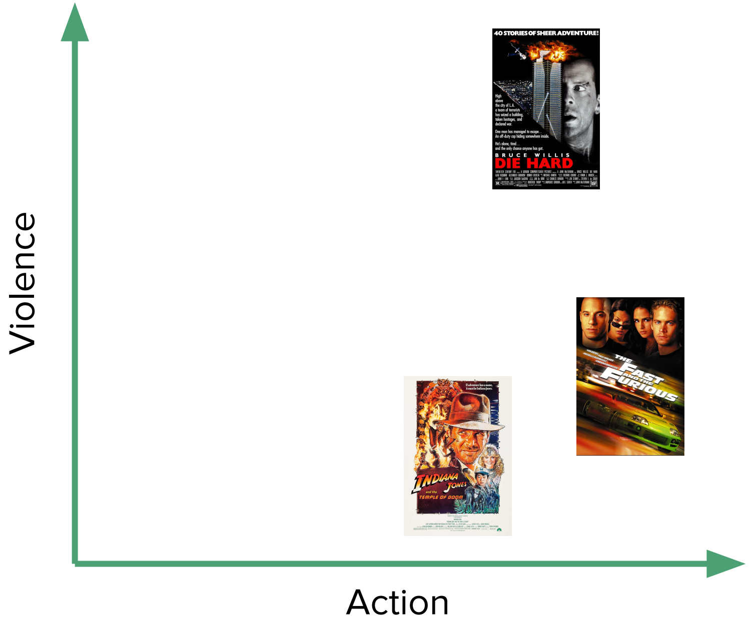
$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\boxed{\sum_{i=1}^{N} v_i w_i}}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Dot Product

Intuition: a (basic) recSys is like a GPS navigator

Consider a movie recommendation systems (user-item case)

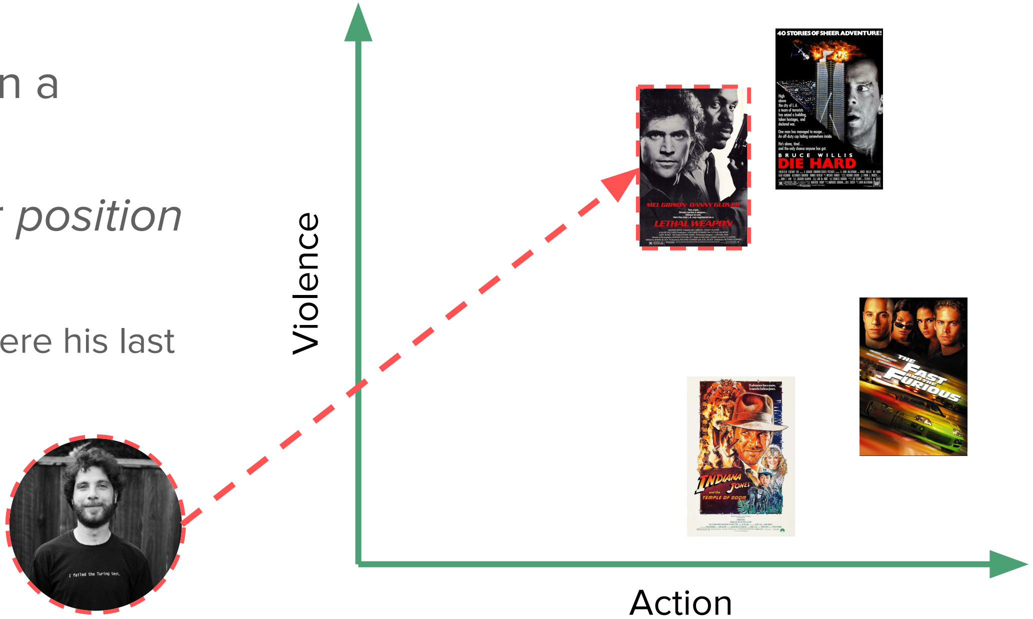
- **Step 1**: represent movies in a suitable space



Intuition: a (basic) recSys is like a GPS navigator

Consider a movie recommendation systems (user-item case)

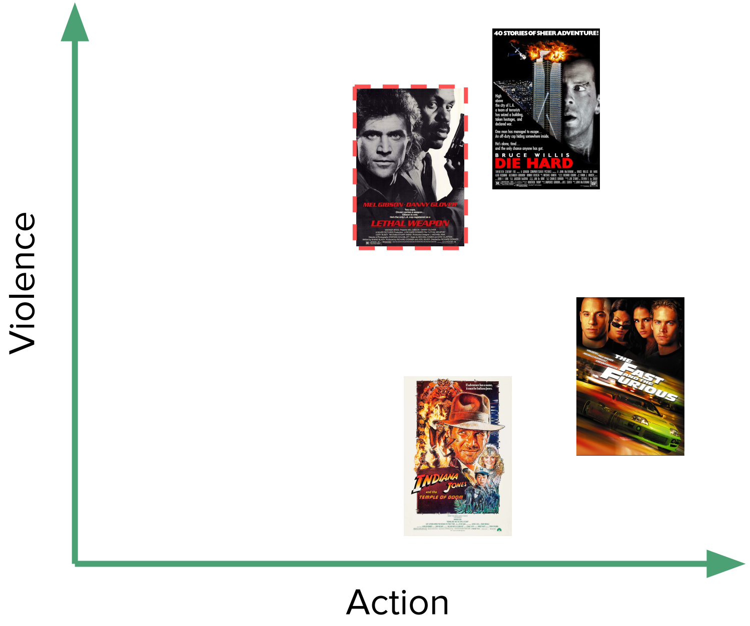
- **Step 1**: represent movies in a suitable space
- **Step 2**: represent the user *position* in the space
  - For example, Jacopo is “where his last movie is”



Intuition: a (basic) recSys is like a GPS navigator

Consider a movie recommendation systems (user-item case)

- **Step 1**: represent movies in a suitable space
- **Step 2**: represent the user *position* in the space
- **Step 3**: recommend the closest K items (KNN search) to the user!
  - Recommendation: Die Hard!



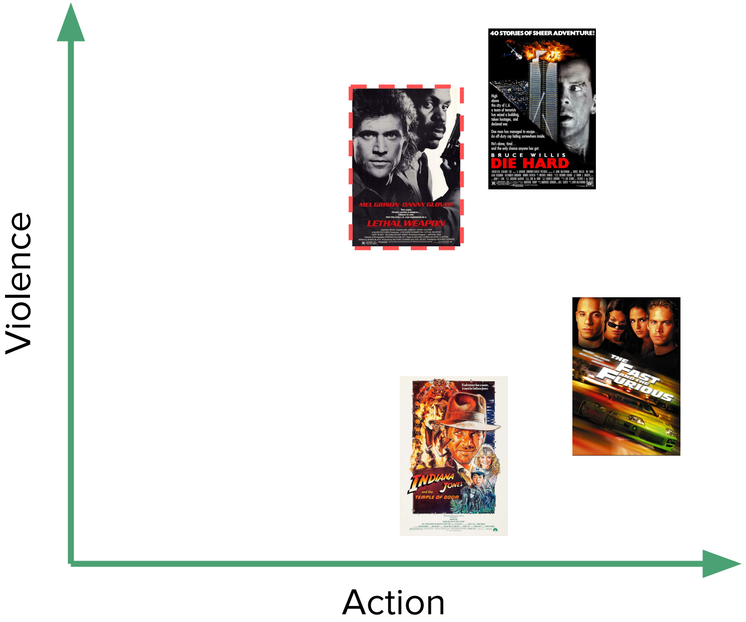
Intuition: a (basic) recSys is like a GPS navigator

Consider a real life example:

- **Jacopo** goes on vacation in Maui, Hawaii
  - Does Jacopo like surfing?
- **Ethan** goes on vacation in Boulder, Colorado
  - Does Ethan like climbing?
- **Intuition**: by knowing the position of the users in the space (in this case, Earth), we can tell a lot about their preferences!

A huge part in our success when building recSys boils down to the quality of the representation in our space.

How do we map users and items to vectors, then?



Representations and data sources

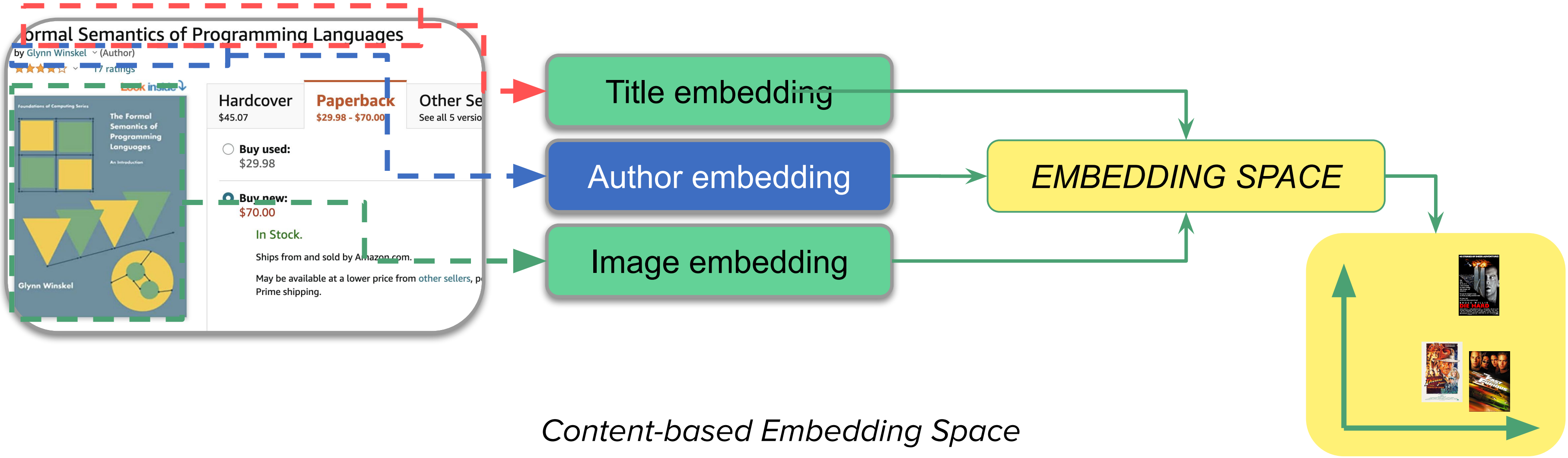
- **GOAL**: learning a good representation space!
  - A “good” space is a space where items that are indeed similar are close, and items that are far from each other are unrelated.
- While we could ask humans to rate *action* vs *violence* vs *comedy* … for all movies on Netflix, that is impractical:
  - A ton of manual work (imagine doing this for all the books on Amazon!)
  - Unclear where to stop: should we have a dimension for actors as well? What about movie length? What about cost of production? Etc.
- We typically distinguish between **content-based** and **behavioral-based** representations (of course, hybrid are also possible!): e.g. for Netflix
  - Content: analyze the title, script, images from the movie, genre etc. - i.e. *what do we know about this item in our catalog*?
  - Behavioral: analyze the behavior of users wrt the items - **if users 1 like items A and B, and then likes also C, can we suppose C is similar to A and B?**

Representations and data sources

- **Content-based** representations require only the “catalog” of our target entities: their quality depends on the ability to turn images, text and categories into “good” vectors.
  - Sometimes the meta-data are not good (for example, movies are mis-categorized!)
  - Sometimes, the vectorization is not very good (for example, our language model does not work well in the target language!)
- **Behavioral-based** representations require real-world data from “users”, e.g.:
  - Purchase data from Amazon
  - Streaming data from Netflix
  - Playlist data from Spotify
  - etc.
- **Note**: a huge lesson of the last 20 years in RecSys is that behavioral-based representations are surprisingly useful in producing good representations (i.e. there is a lot of signals in people behavior!).
  - Q: when a behavioral strategy won’t be helpful (we discussed it in class!)?

Representations and data sources

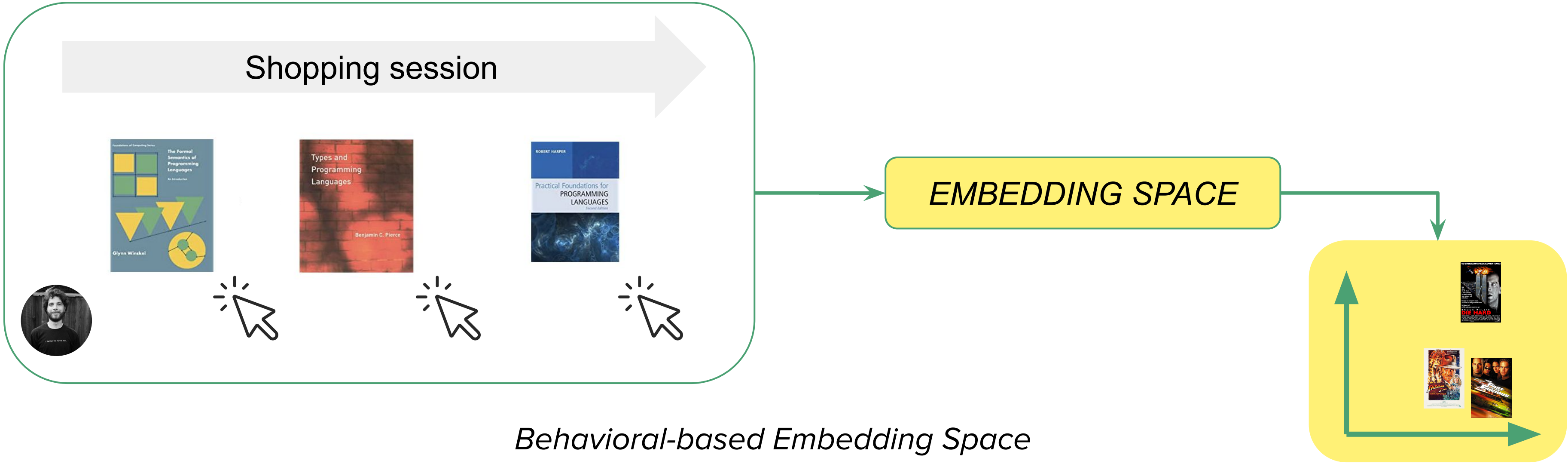
- We call the process of mapping high-cardinality entities (users, items, words etc.) to a low-dimensional space “**embedding**”:
  - Remember the one-hot encoding, sparse vectors? Embeddings are small (100s dimensions for thousands of items) and dense!



Content-based Embedding Space

Representations and data sources

- We call the process of mapping high-cardinality entities (users, items, words etc.) to a low-dimensional space "**embedding**":
    - Remember the one-hot encoding, sparse vectors? Embeddings are small (100s dimensions for thousands of items) and dense!



Behavioral-based Embedding Space

Word2Vec, Song2Vec, Everything2Vec

The NLP Analogy: similar things appear often together

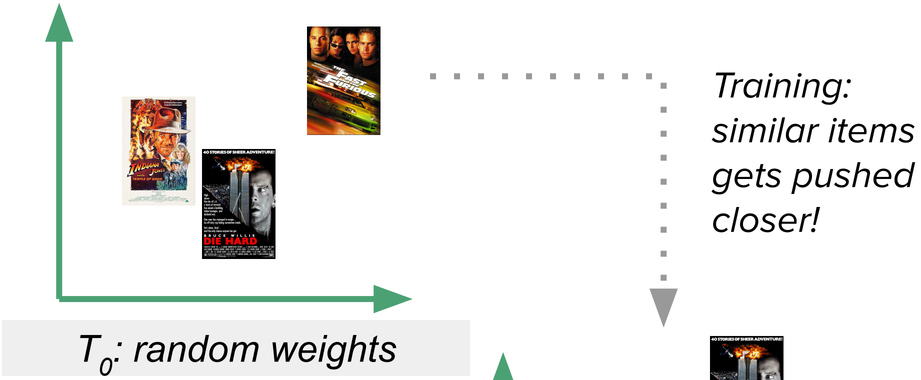
- **Distributional hypothesis**: “words that appear in similar contexts have similar meanings”
  - Example 1: if two books are often viewed in the same shopping session, they are probably similar!
  - Example 2: if two songs are often after each other in playlists, they are probably similar!

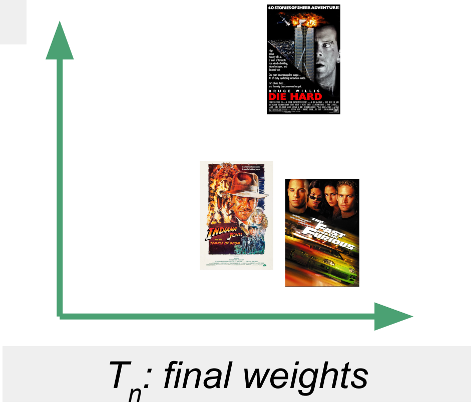
A recipe for learning word embeddings (“word2vec”)

- Outline of the general argument:
  - We need to learn vectors for words to make a “good” space
  - Words which are similar tend to appear in the same sentence
  - If we use vectors as weights for a classifier that tells when two words are likely to appear together, we can learn vectors that encode similarity and therefore produce a good space!
- In other words, the distributional hypothesis gives us a proxy measure of similarity: embeddings that are good in the distributional settings SHOULD therefore be good representations for word similarity.

A recipe for learning word embeddings (“word2vec”)

- Outline of the general argument:
  - We need to learn vectors for words to make a “good” space
  - Words which are similar tend to appear in the same sentence
  - If we use vectors as weights for a classifier that tells when two words are likely to appear together, we can learn vectors that encode similarity and therefore produce a good space!
- In other words, the distributional hypothesis gives us a proxy measure of similarity: embeddings that are good in the distributional settings SHOULD therefore be good representations for word similarity.





BONUS POINT: this actually works in most successful applications of deep learning - if you find an appropriate prediction task, you can learn good representations for anything!

A recipe for learning word embeddings ("word2vec")

Corpus

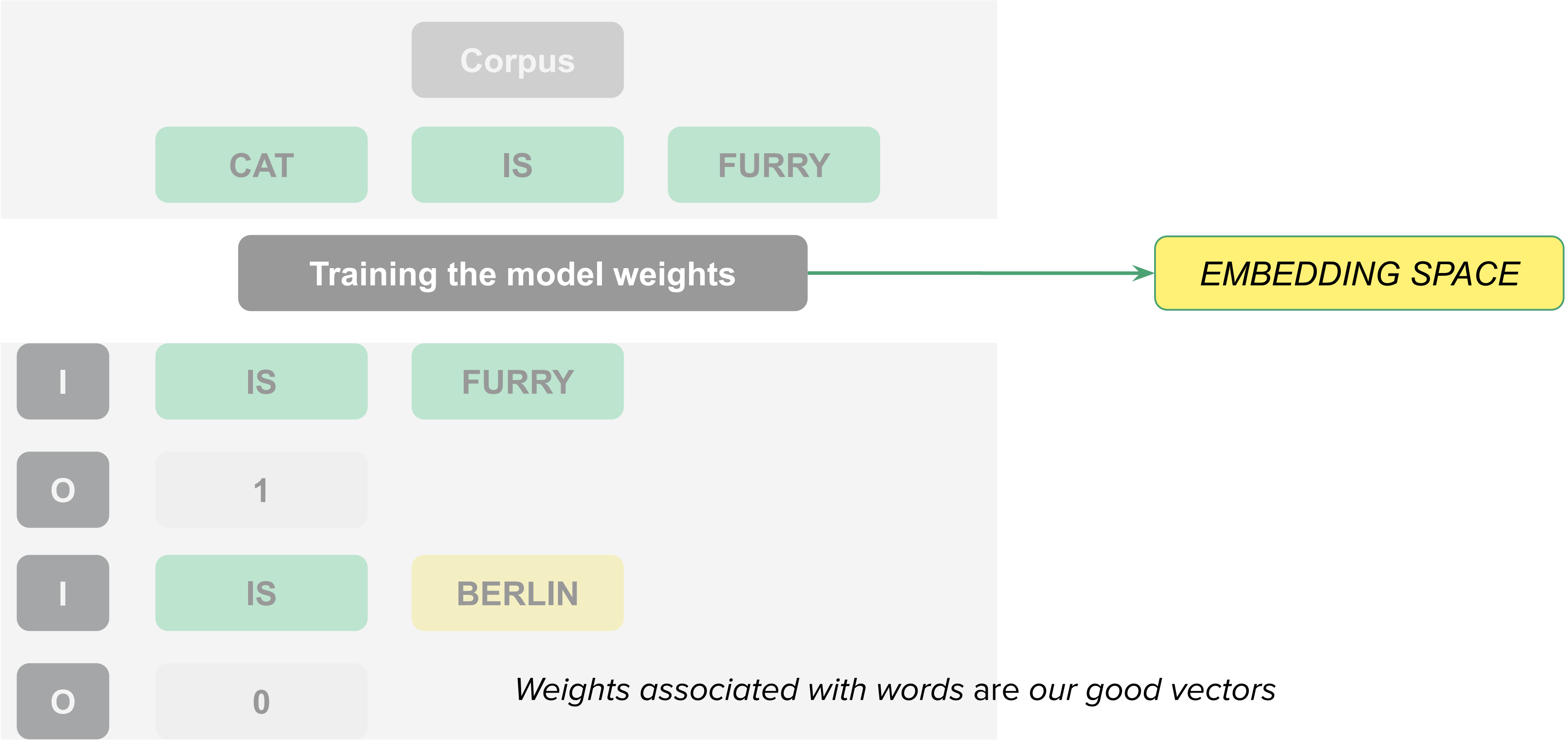
CAT	IS	FURRY

Training the model weights

I	IS	FURRY
O	1	
I	IS	BERLIN
O	0	

Turn sequential data into a prediction problem

A recipe for learning word embeddings (“word2vec”)



Weights associated with words are our good vectors

Word vectors in a *prediction* task

- CORPUS: “The furry cat is on the mat”
- WINDOW LENGTH: 2
- TARGET: “cat”
- INPUT PREPARATION, positive and negative samples

Target	Context	Label
cat	furry	1
cat	the	1
cat	is	1
cat	on	1

Target	Context	Label
cat	Berlin	0
cat	Jacopo	0
cat	ciao	0
cat	table	0

Word vectors in a *prediction* task

- CORPUS: “The furry cat is on the mat”
- WINDOW LENGTH: 2
- TARGET: “cat”
- INPUT PREPARATION, positive and $\boxed{\text{negative samples (a=0.75)}}$

$$\boxed{P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}}$$

Target	Context	Label
cat	furry	1
cat	the	1
cat	is	1
cat	on	1

Target	Context	Label
cat	Berlin	0
cat	Jacopo	0
cat	ciao	0
cat	table	0

Word vectors in a *prediction* task

- We have turned a word prediction problem into a binary classification problem
  - Is the context word likely to appear next to the target word?
- Let’s define our learning objective:
  - We want to maximize the similarity of (t,c) drawn from the positive examples
  - We want to minimize the similarity of (t,c) drawn from the negative examples

$$L(\theta) = \sum_{(t,c)\in +} \log P(+|t,c) + \sum_{(t,c)\in -} \log P(-|t,c)$$


CHAPTER

6 **Vector Semantics and Embeddings**

荃者所以在鱼，得鱼而忘荃 Nets are for fish;
Once you get the fish, you can forget the net.
言者所以在意，得意而忘言 Words are for meaning;
Once you get the meaning, you can forget the words
庄子(Zhuangzi), Chapter 26

Word vectors in a *prediction* task

- Let’s define our learning objective:
  - We want to maximize the similarity of (t,c) drawn from the positive examples
  - We want to minimize the similarity of (t,c) drawn from the negative examples

$$= \log \underbrace{\sigma}_{\text{Sigmoid}}(\underbrace{c \cdot t}_{\text{Dot product}}) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t)$$

$$= \log \frac{1}{1+e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i \cdot t}}$$

Word vectors in a *prediction* task

- **Remember**: we maximize the dot product of the word with the context words, and minimize the dot products of the word with the negative sampled words!
- Training procedure:
  - Random initialization of vectors for N words in the vocabulary.
  - At each step, move embeddings of related words closer in the vector space, and push others further away (using gradient descent).

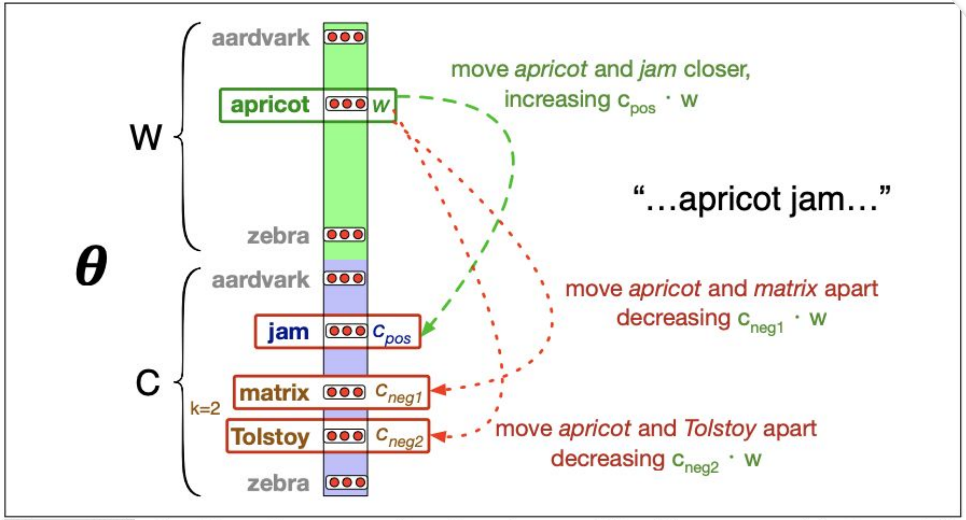
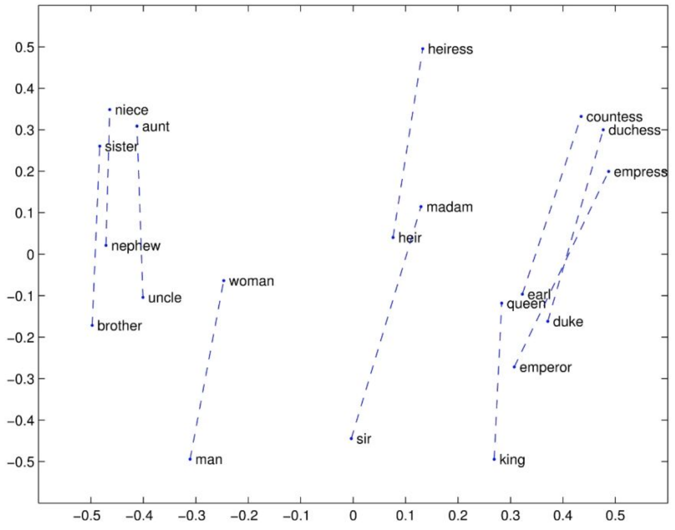


Figure 6.14 Intuition of one step of gradient descent. The skip-gram model tries to shift embeddings so the target embeddings (here for *apricot*) are closer to (have a higher dot product with) context embeddings for nearby words (here *jam*) and further from (lower dot product with) context embeddings for noise words that don't occur nearby (here *Tolstoy* and *matrix*).

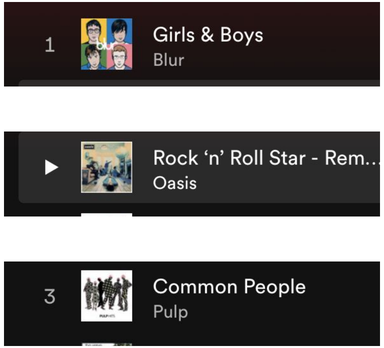
Is this a "good" space?

- word2vec tends to capture well similarity between words and some <u>analogical relations</u> - **without any human labels / intervention!**
- Once you have a well-trained embedding space, the offsets between vector embeddings can be used to solve analogies such as: "man : king = women : ?" (*queen*)
  - This is possible since the result of vector('king') - vector('man') + vector('woman') is a vector close to vector('queen').



From NLP, back to RecSys

Remember: the same intuition about "words in a sentence" can be applied whenever we have meaningful sequences of target items (e.g. playlist, shopping sessions etc.)



CAT

IS

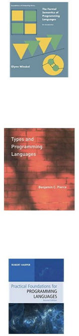
FURRY

Song2Vec

CAT

IS

FURRY



Book2Vec

Coding time!