CM3-Computer Science

What is data engineering

Data engineering is the development, implementation, and maintenance of systems and processes that take in raw data and produce high-quality, consistent information that supports downstream use cases, such as analysis and machine learning. Data engineering is the intersection of security, data management, DataOps, data architecture, orchestration, and software engineering. A data engineer manages the data engineering lifecycle, beginning with getting data from source systems and ending with serving data for use cases, such as analysis or machine learning.

-Fundamentals of Data Engineering

Our Sessions

Lecture	Big Idea
1. What is the essence of a tree?	Approach a problem at the right level of abstraction
2. Functions can travel first class too.	Idempotent functions and Functional Programming
3. Do not change things to make them faster.	Unchangedness and Immutability of data
4. Your state of mind is not my state of mind.	Modeling the state of objects with Object-Oriented Programming
5. If it looks and quacks like a duck, it is a duck.	Polymorphism and well defined interfaces.
6. If they cant eat bread, let them eat trees.	How the choice of the right data structure leads to performant algorithms.
7. Any problem can be solved by adding one extra layer of indirection	Separate the specification of an algorithm from its implementation
8. You may not use the bathroom unless I give you the key!	Concurrency and Parallelism in programs
9. Floating down a lazy river.	Lazy Evaluation and Streams
10. Deforming Mona Lisa with a new tongue.	Languages for stratified design
11. It’s turtles all the way down!	Languages and the machines which run them
12. Feed me Seymour, and I’ll remember you!	GPUs enable the learning instead of the writing of programs
13. Hey Markov! Whats the next word?	Large Language Models generate the next token

Vectors, Embedding, and Recommendations

Vectors

Vectors in Computer Science

$$\vec{v} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

```
v = Vector([1, 2])

len(v) # returns 2
v[0] # returns 1
v[1] # returns 2
```

We will call this a 2D vector, or a 2D **column** vector. The numbers might represent anything, for example, the properties of a house: 1 (in units of 1000) square feet and 2 bedrooms.

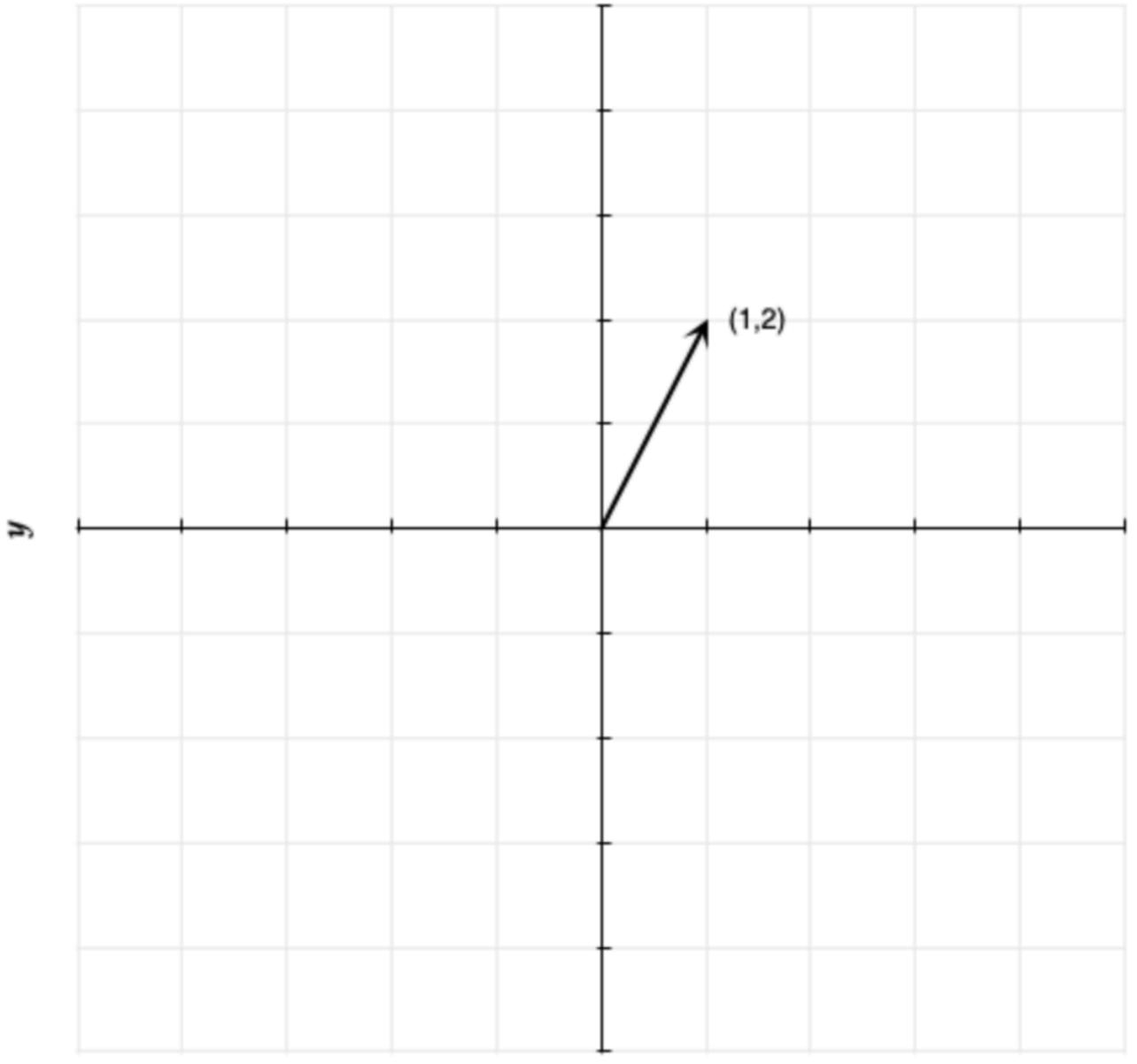
For us, a vector is an ordered list of numbers. We used this to define, last time, a `Vector` class which has a length and *components*

```
class Vector:

    def __init__(self, lst):
        self.storage = lst.copy()

    def __len__(self):
        return len(self.storage)

    def __getitem__(self, i):
        return self.storage[i]
```



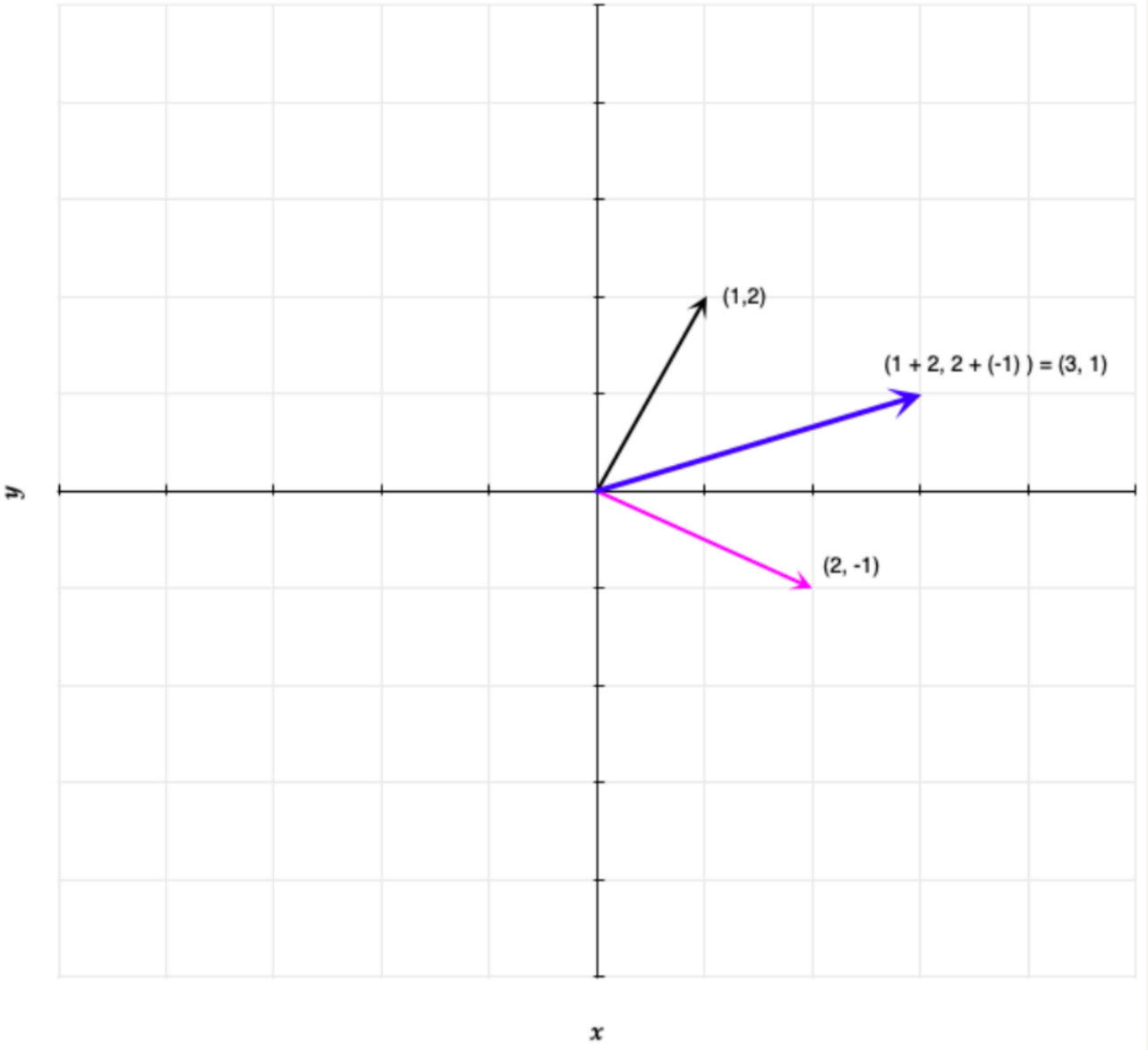
Geometric Interpretation of a vector

Vectors are arrows pointing in space.

For $\vec{v} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$, imagine moving 1 step to the right (x-axis) and 2 steps up (y-axis) from the origin.

Every pair of numbers corresponds to one arrow, and every such arrow from the origin, corresponds to a unique pair of numbers.

Thus, a vector has a visual *length* and *direction* in space.



Vector addition

We can add vectors by adding the individual components using the geometric interpretation from previous. Here we first move one step right and 2 steps up, and then 2 steps right and one step *down* (-1). The net is 3 steps right and 1 step up. This can be seen as:

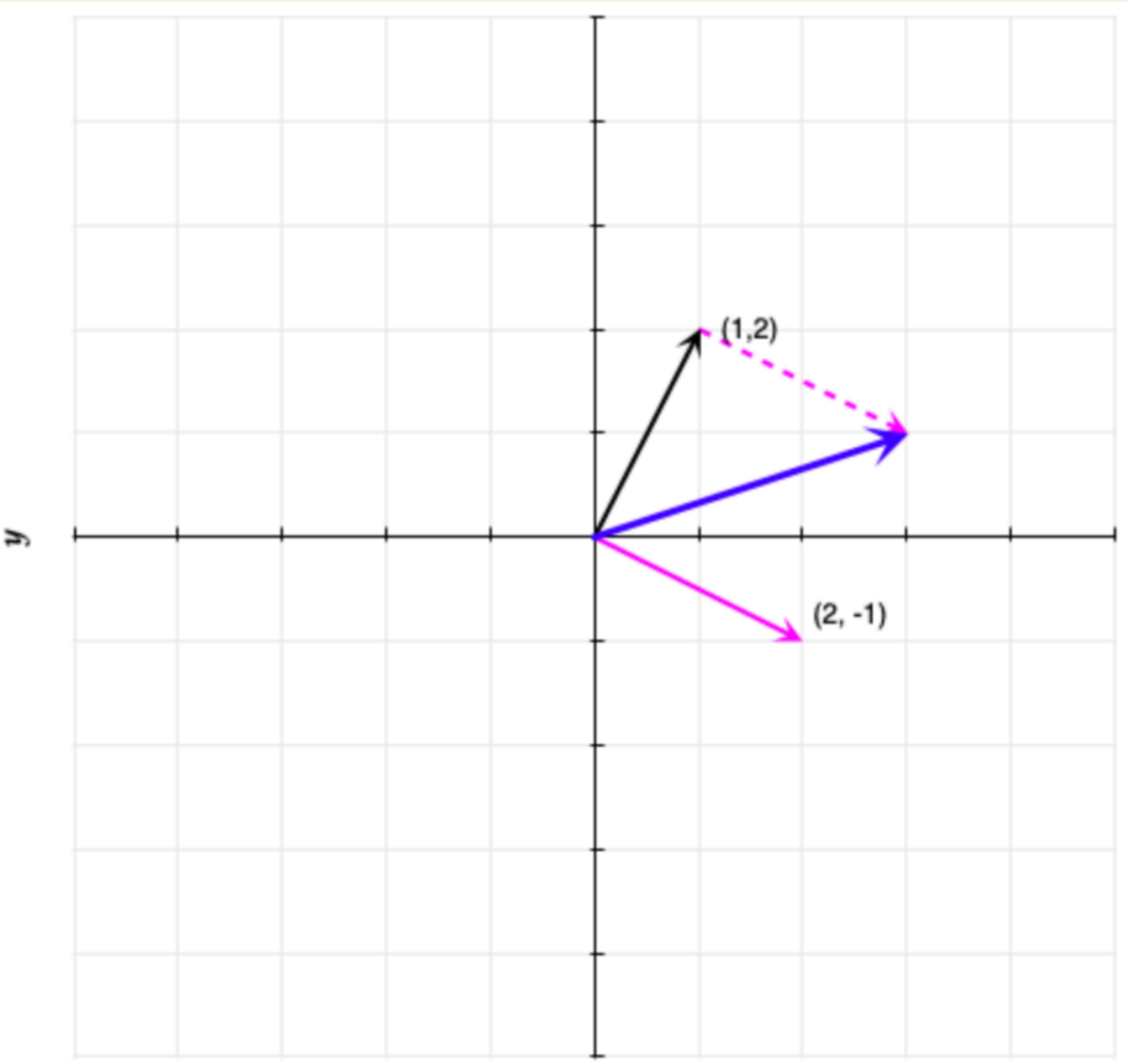
$$\vec{u} = \vec{v} + \vec{w} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} 2 \\ -1 \end{pmatrix} = \begin{pmatrix} 1+2 \\ 2+(-1) \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$$

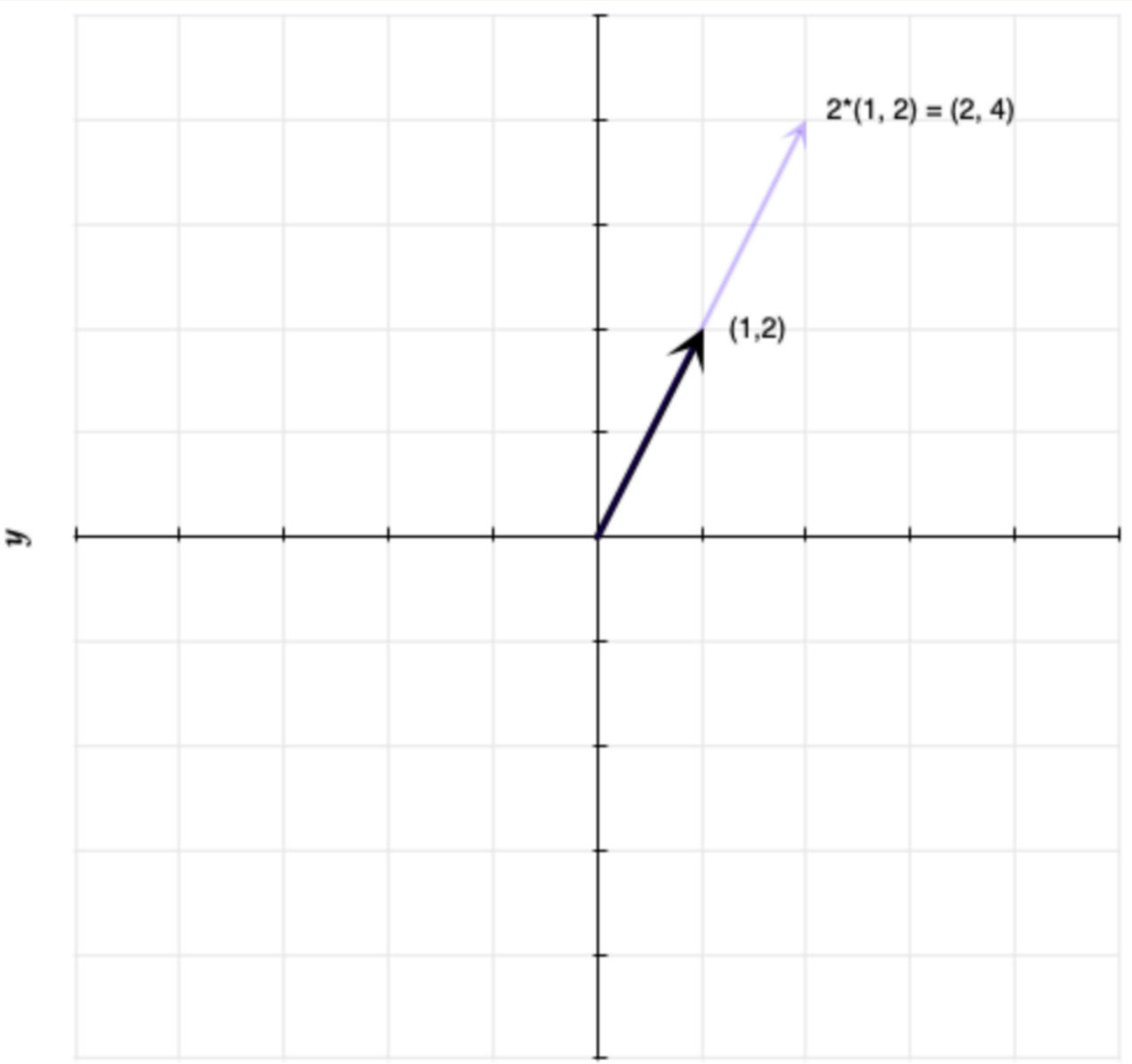
We are adding the x components in the column vectors to get the x component of the new column vector $\vec{u}$. We do the same for the y components.

Geometrically, you can think of sliding the tail of the second vector $\vec{w}$ over to the head of the first one ($\vec{v}$), and then joining the tail of the first vector with the head of the second vector to create the new "sum" vector $\vec{u}$.

Since our vectors have the notion of a length and direction, you can think of walking a certain length and direction as suggested by $\vec{v}$, and then after having reached its tip, walking a length and direction suggested by $\vec{w}$. That would give you the *sum* of all the walking you need to do...

Subtraction then would be to reverse the second vector but do the same kind of walking. Or in the column vector representation, subtract the components...





Scalar multiplication

This is the idea of multiplying a vector by a number, $\lambda\vec{v}$ for some number λ, so called because it *scales* the vector by whatever number you multiplied the vector by.

Here

$$\vec{u} = 2\vec{v} = 2 \times \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \times 1 \\ 2 \times 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}.$$

The geometric interpretation is to walk twice as long in the same direction.
Thecorresponding mathematical idea is to multiply each component by the same *scalar* so that we *scale* the vector.

Dot Products

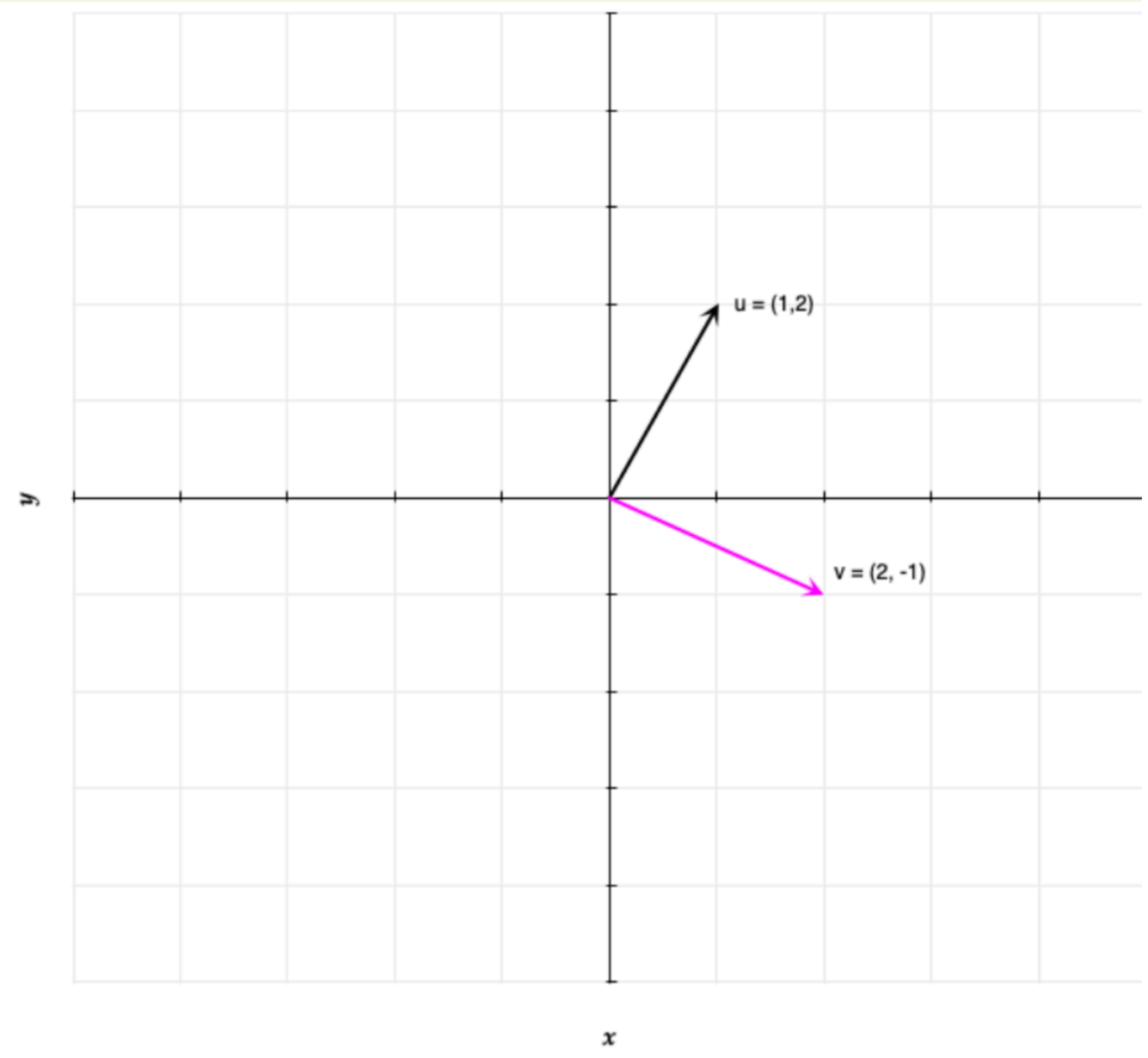
The dot product of two vectors $\vec{u}$ and $\vec{v}$ is defined as the sum of the products of the vectors components:

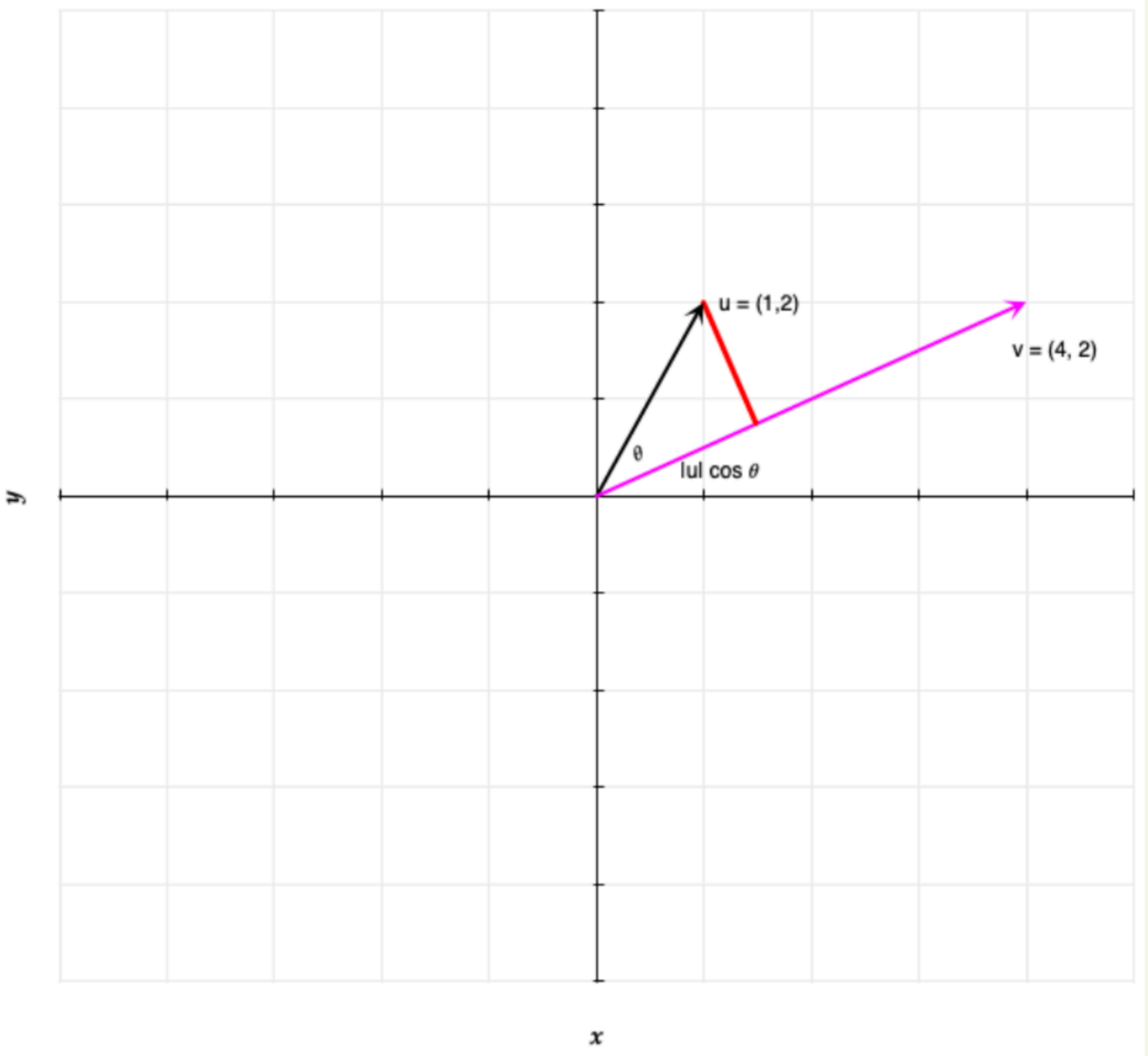
$$\vec{u} \cdot \vec{v} = \begin{pmatrix} u_x \\ u_y \end{pmatrix} \cdot \begin{pmatrix} v_x \\ v_y \end{pmatrix} = u_x v_x + u_y v_y$$

In this case, we have:

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ -1 \end{pmatrix} = 1 \times 2 + 2 \times -1 = 0$$

These vectors are *perpendicular* or *orthogonal*: their dot product is 0.





You might recall a definition from high school:

The dot product of two vectors $\vec{u}$ and $\vec{v}$ is defined as the product of $\vec{v}$ with the projection of $\vec{u}$ onto $\vec{v}$

$$\vec{u} \cdot \vec{v} = |\vec{v}| \times |\vec{u}| \, cos\theta,$$

where $|\vec{v}|$ is the length of $\vec{v}$ and θ the angle between $\vec{u}$ and $\vec{v}$.

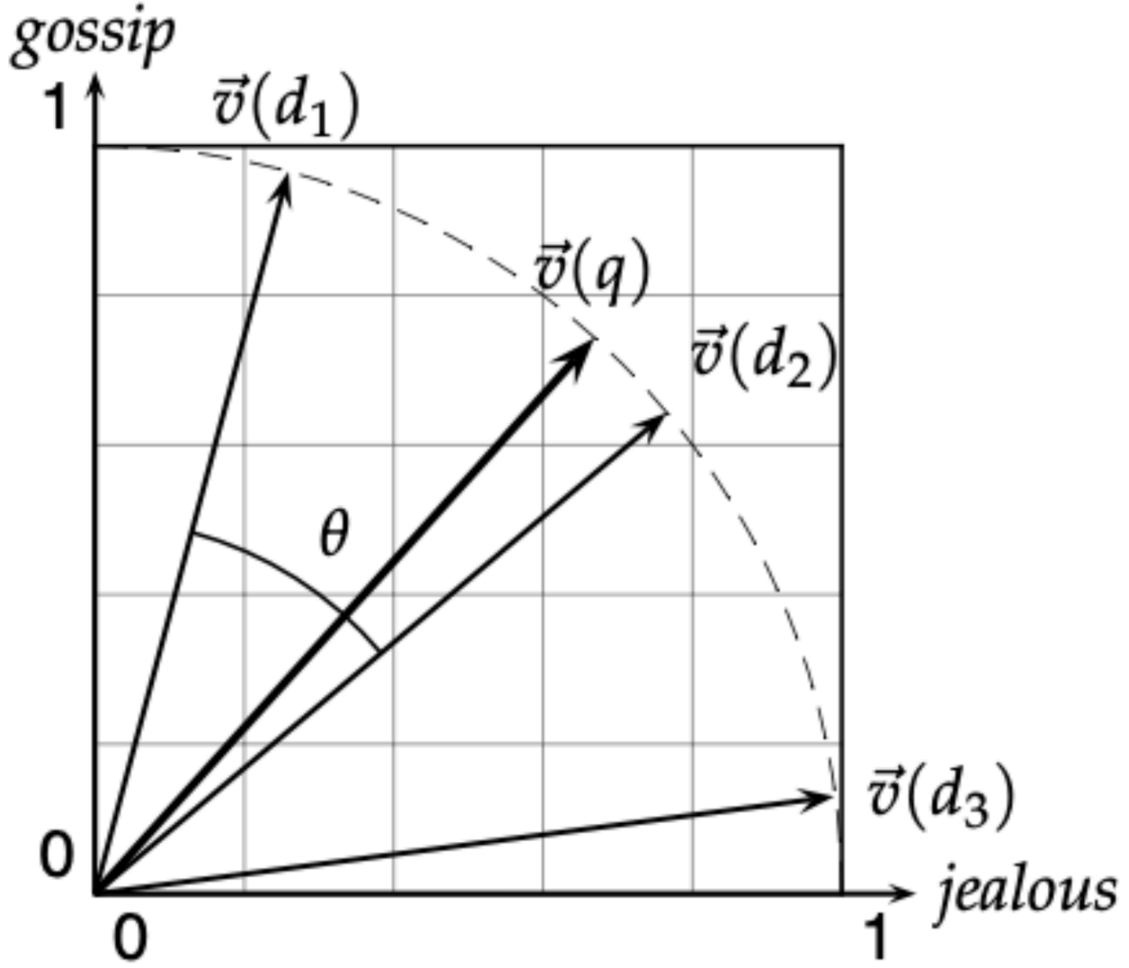
These definitions are identical. Imagine a new x-axis aligned with the pink vector. Then the product becomes simply $u_x v_x$ and is equivalent to the previous definition. You can also prove this (with more effort) in the general case.

Cosine Similarity

The cosine similarity is simply the dot product of two vectors divided by their magnitudes.

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{\left|\vec{V}(d_1)\right| \left|\vec{V}(d_2)\right|'}$$

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6



cosine similarity:

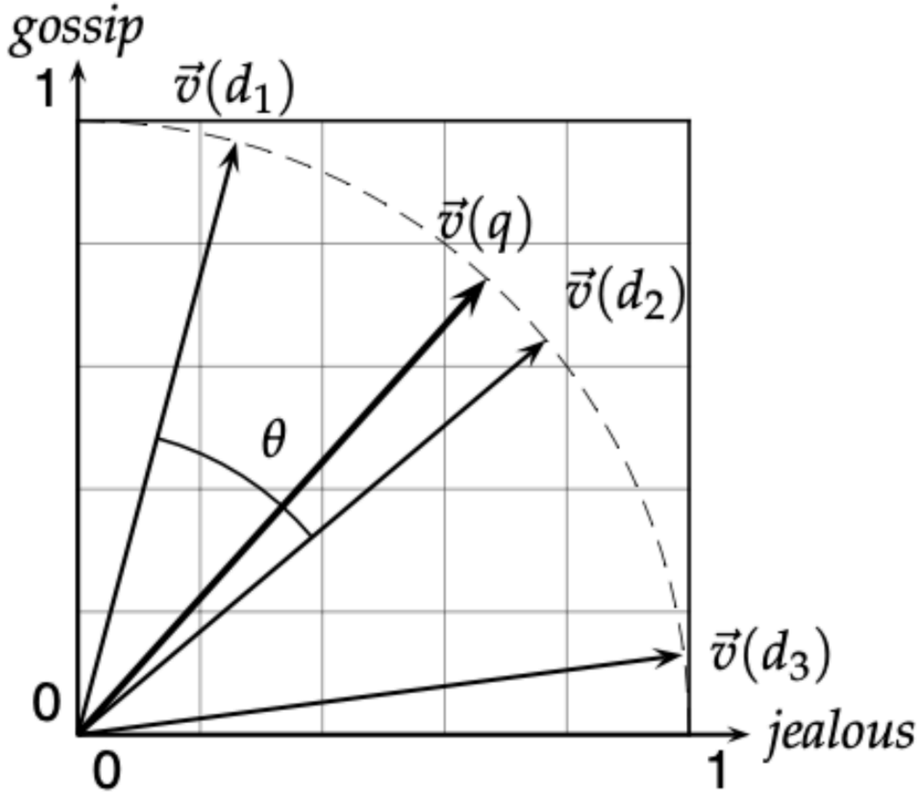
term	SaS	PaP	WH
affection	0.996	0.993	0.847
jealous	0.087	0.120	0.466
gossip	0.017	0	0.254

It can be used to make queries:

$$\text{score}(q,d) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|}$$

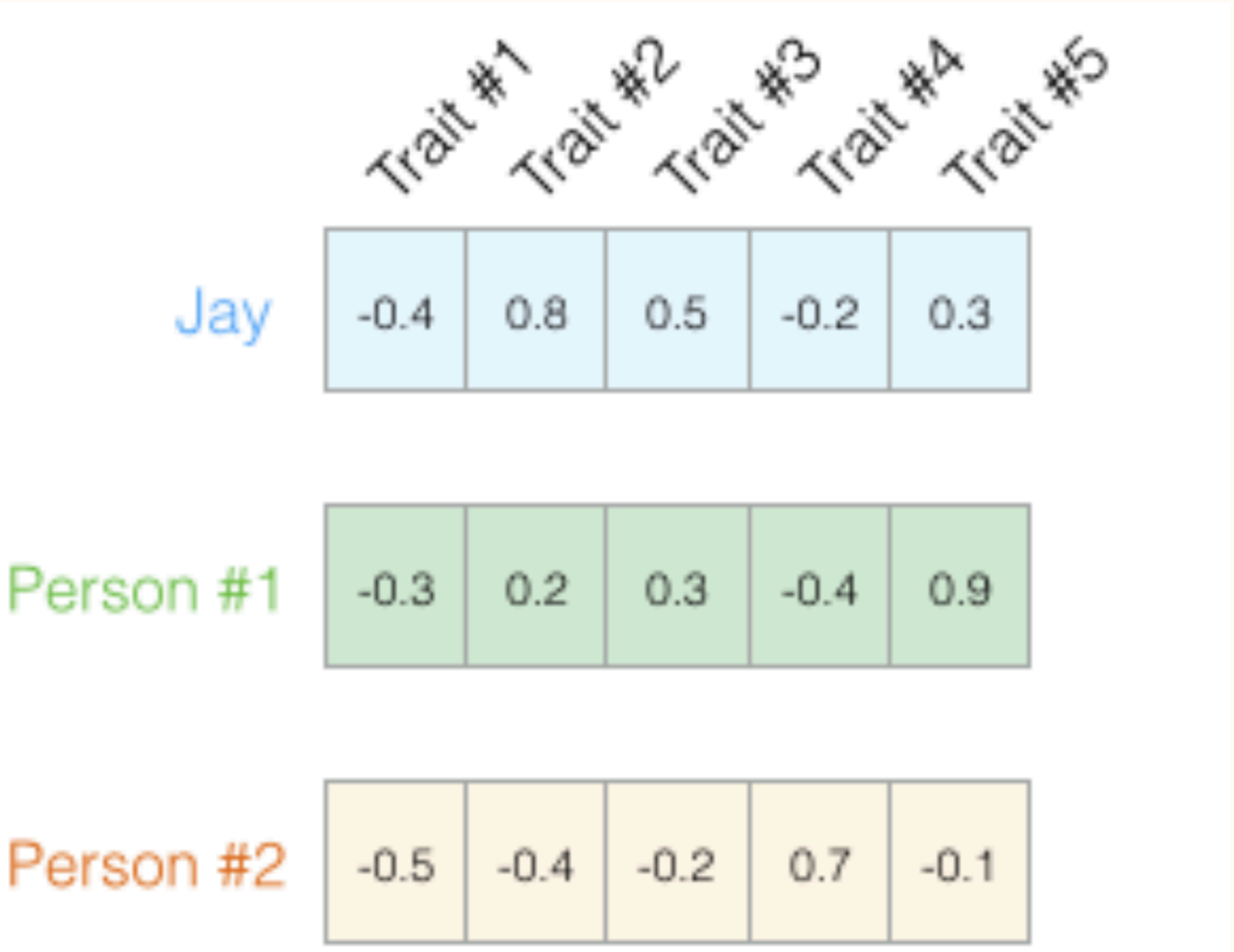
Consider the query $q = \texttt{jealous gossip}$. This query turns into the unit vector $\vec{v}(q) = (0, 0.707, 0.707)$. Now assign to each document d a score equal to the dot product $\vec{v}(q) \cdot \vec{v}(d)$. Sort.

cosine of the angle between two unit vectors

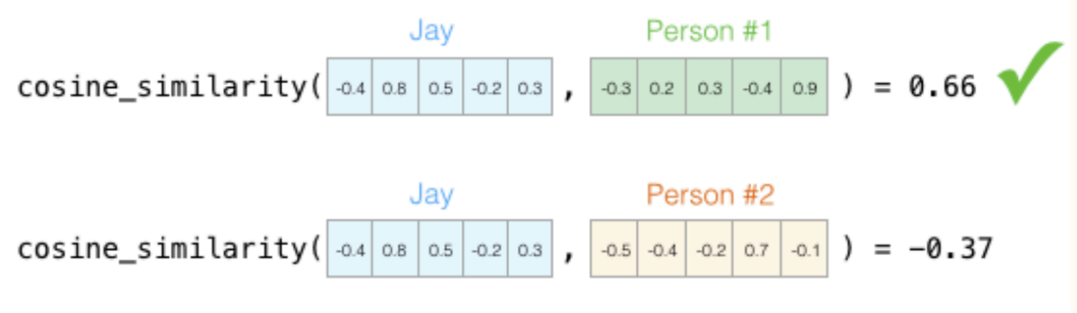


Embeddings

The Basic Idea



- Observe a bunch of people
- **Infer** Personality traits from them
- Vector of traits is called an **Embedding**
- Who is more similar? Jay and who?
- Use Cosine Similarity of the vectors



Categorical Data

Example:

Rossmann Kaggle Competition. Rossmann is a 3000 store European Drug Store Chain. The idea is to predict sales 6 weeks in advance.

Consider `store_id` as an example. This is a **categorical** predictor, i.e. values come from a finite set.

We usually **one-hot encode** this: a single store is a length 3000 bit-vector with one bit flipped on.

What is the problem with this?

- The 3000 stores have commonalities, but the one-hot encoding does not represent this
- Indeed the dot-product (cosine similarity) of any-2 1-hot bitmaps must be 0
- Would be useful to learn a lower-dimensional **embedding** for the purpose of sales prediction.
- These store "personalities" could then be used in other models (different from the model used to learn the embedding) for sales prediction
- The embedding can be also used for other **tasks**, such as employee turnover prediction

BOW

Treat language as a bag of words. Define a vocabulary V which consists of all the words.

Now encode language using the index of the word in the vocabulary.

For example the sentence "The cow jumped over the moon" will become a V sized array with a 0 or 1 at each position corresponding to the index of the word.

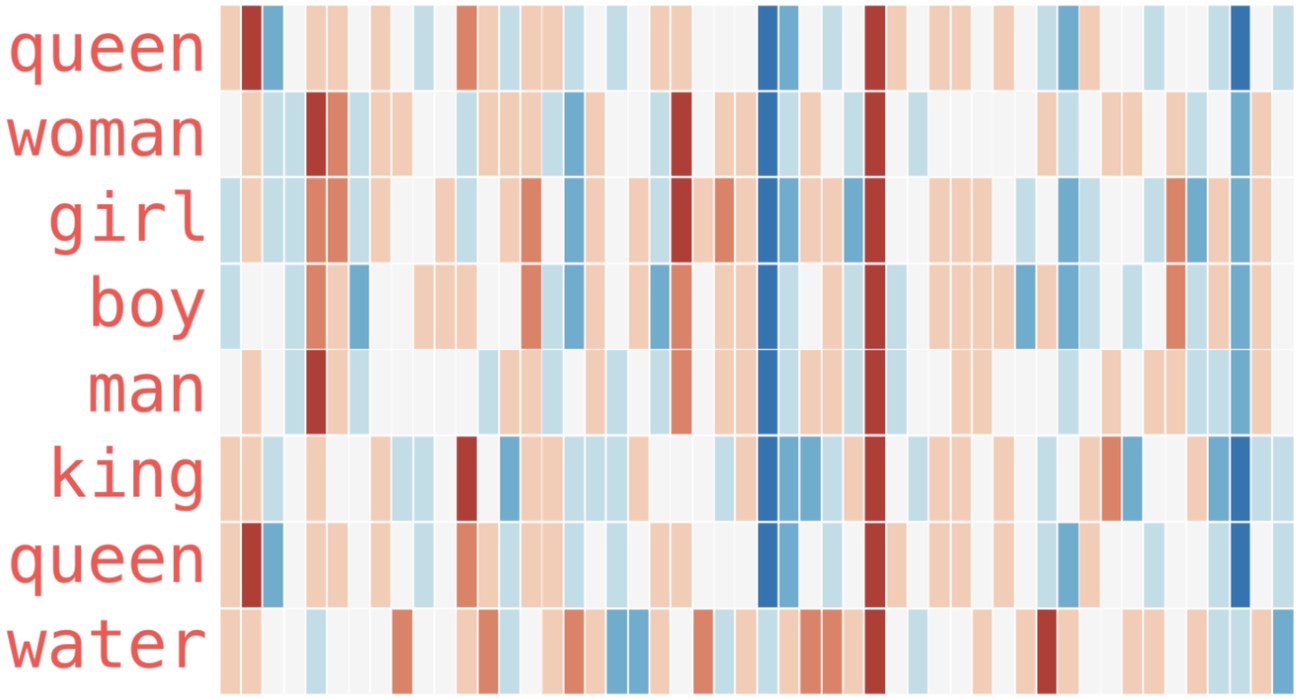
So the previous sentence might become `[0,0, 1, 0,....0, 1, ..., 1, 1, 0, 0,...1, 0,0..,0]` where the 1's correspond to the index of the word.

A variation on this is to use tf-idf where the fact that "the" comes twice is important, but

Word Embeddings

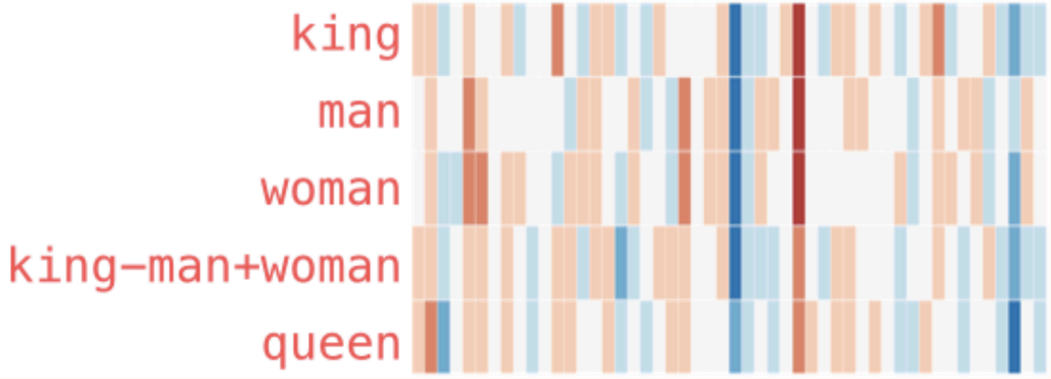
- The Vocabulary V of a corpus (large swath of text) can have 10,000 and maybe more words
- a 1-hot encoding is huge, moreover, similarities between words cannot be established
- we map words to a smaller dimensional latent space of size L by considering some downstream task to train on
- we hope that the embeddings learnt are useful for other tasks.

Obligatory example



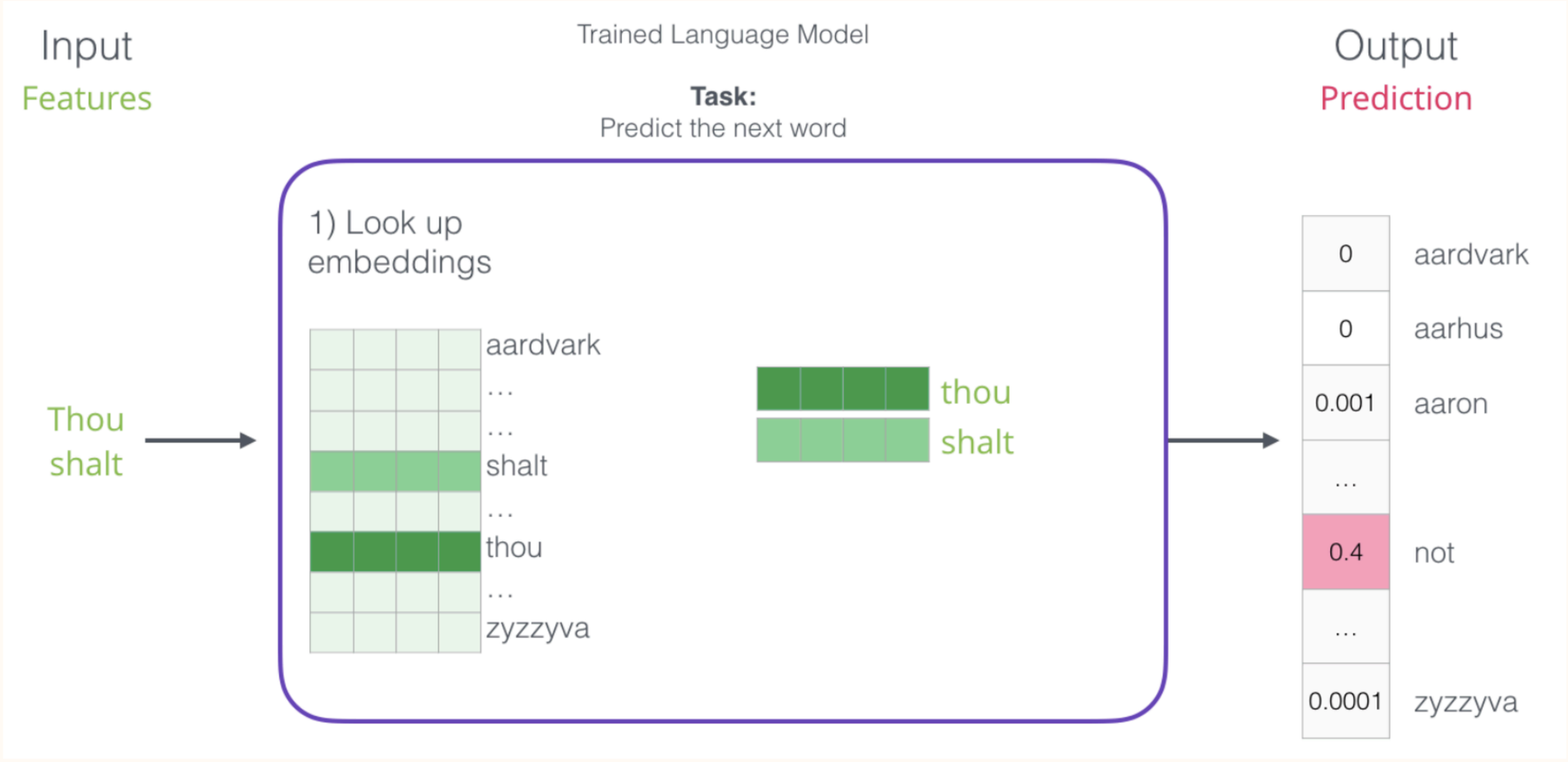
See man->boy as woman->girl, similarities of king and queen, for eg.
These are lower dimensional GloVe embedding vectors

king − man + woman ~= queen



How do we train word embeddings?

We need to choose a downstream task. We could choose **Language Modeling**: predict the next word. We'll start with random "weights" for the embeddings and other parameters and run SGD. A trained model+embeddings would look like this:



Why is language modeling a great task?

How do we set up a training set? All the labels are provided for us!!! This is **self-supervised** learning!

Thou shalt not make a machine in the likeness of a human mind

Sliding window across running text

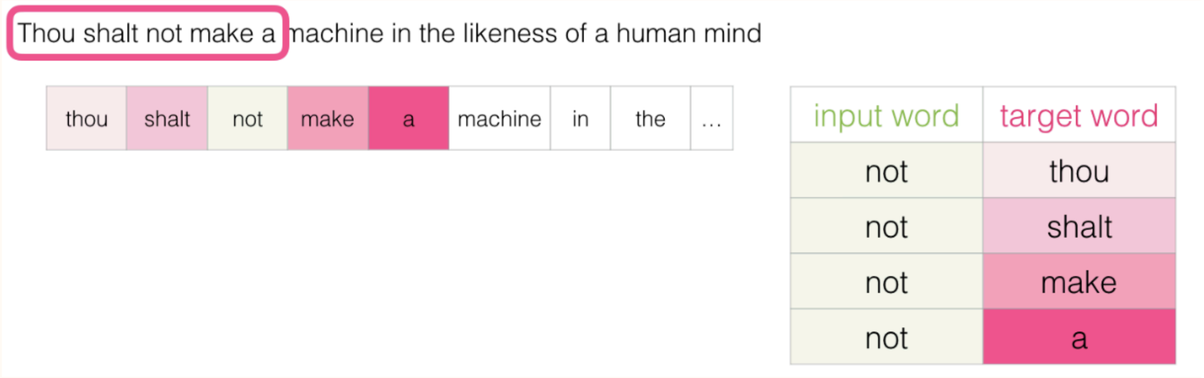
thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	

Dataset

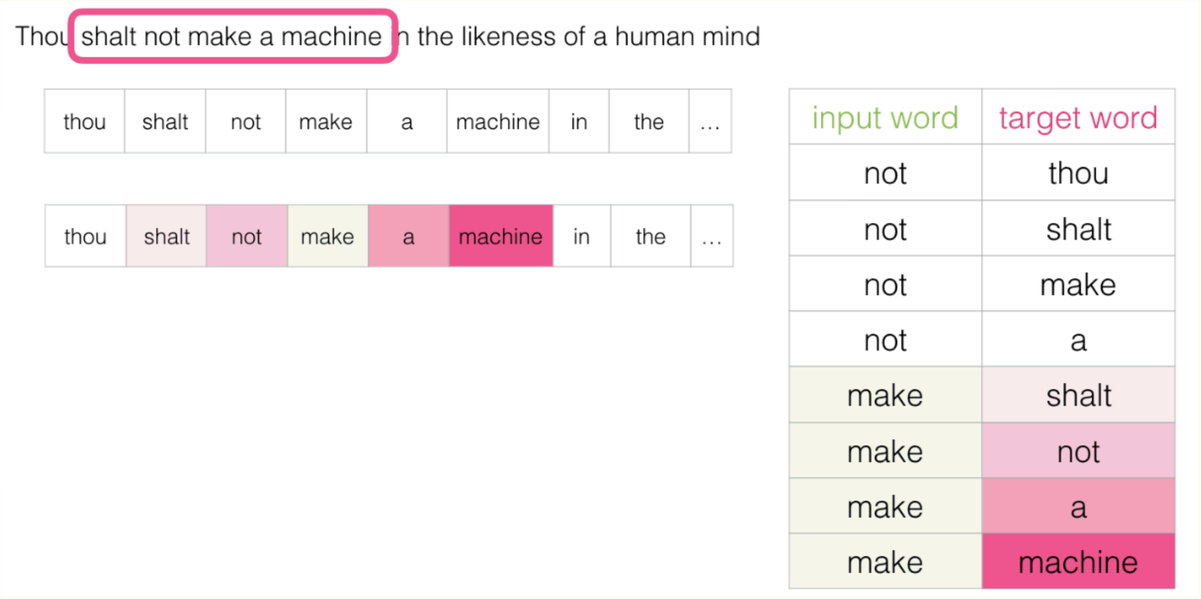
input 1	input 2	output
thou	shalt	not
shalt	not	make
not	make	a
make	a	machine
a	machine	in

SKIP-GRAM: Predict Surrounding Words

Why not look both ways? This leads to the Skip-Gram and CBOW architectures.. Choose a window size (here 4) and construct a dataset by sliding a window across.



input word	target word
not	thou
not	shalt
not	make
not	a

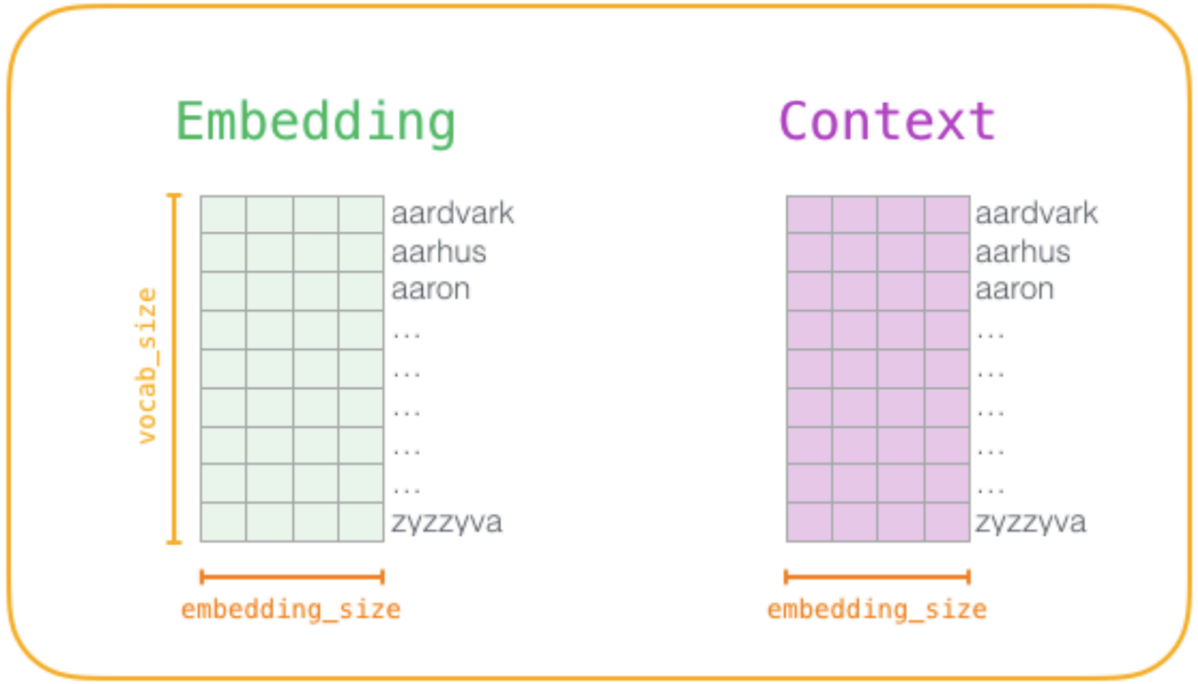


input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

We assume that, Naive Bayes style, the joint probability of all **context** words $\{w_o\}$ in a window conditioned on the central word w_c is the product of the individual conditional probabilities:

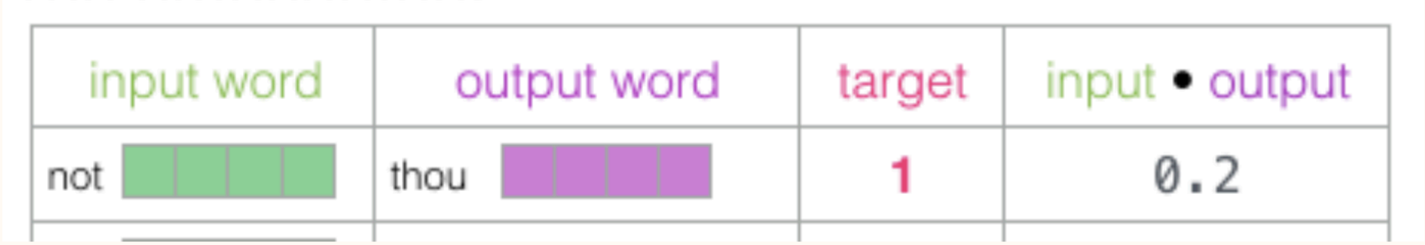
$$P(\{w_o\} \mid w_c) = \prod_{i \in window} p(w_{oi} \mid w_c)$$

Now assume that each word w_i is represented as 2 embeddings, an **input** embedding $\boldsymbol{v}_i$ when it is a central word and a context embedding $\boldsymbol{u}_i$ when it is in the surrounding window.



The probability of an output word, givem a central word, is assumed to be given by a softmax of the dot product of the embeddings

$$\mathbb{P}(w_o \mid w_c) = \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\boldsymbol{u}_i^\top \boldsymbol{v}_c)},$$



Then, assuming a text sequence of length T and window size m , the likelihood function is:

$$\mathcal{L} = \prod_{t=1}^{T} \prod_{-m \leq j \leq m,\ j \neq 0} \mathbb{P}(w^{(t+j)} \mid w^{(t)}).$$

We'll use the Negative Log Likelihood as loss (NLL).

Usage of word2vec

- the pre-trained word2vec and other embeddings (such as GloVe) are used everywhere in NLP today
- the ideas have been used elsewhere as well. AirBnB and Anghami model sequences of listings and songs using word2vec like techniques
- Alibaba and Facebook use word2vec and graph embeddings for recommendations and social network analysis.
- Coveo creates product2vec a way of making embeddings using product browsing history
  and using it for intention prediction and recommendations.

Recommendation Systems

What are we trying to predict?

We are a website like Yelp which has users rate restaurants and write reviews about them. Or like Amazon, rating products.

Consider a restaurant or item m. There are many such restaurants in a city. Now consider a person, a "user" of our recommendations system. Lets denote her by the letter u. We wish to create a model for Y_{um}, the rating a user u will give reataurant m.

	movieId	27	49	57	72	79	89	92	99	143	179	180	197	402	417	505
userId	14	3.0	5.0	1.0	3.0	4.0	4.0	5.0	2.0	5.0	5.0	4.0	5.0	5.0	2.0	5.0
	29	5.0	5.0	5.0	4.0	5.0	4.0	4.0	5.0	4.0	4.0	5.0	5.0	3.0	4.0	5.0
	72	4.0	5.0	5.0	4.0	5.0	3.0	4.5	5.0	4.5	5.0	5.0	5.0	4.5	5.0	4.0
	211	5.0	4.0	4.0	3.0	5.0	3.0	4.0	4.5	4.0		3.0	3.0	5.0	3.0	
	212	2.5		2.0	5.0		4.0	2.5		5.0	5.0	3.0	3.0	4.0	3.0	2.0
	293	3.0		4.0	4.0	4.0	3.0		3.0	4.0	4.0	4.5	4.0	4.5	4.0	
	310	3.0	3.0	5.0	4.5	5.0	4.5	2.0	4.5	4.0	3.0	4.5	4.5	4.0	3.0	4.0
	379	5.0	5.0	5.0	4.0		4.0	5.0	4.0	4.0	4.0		3.0	5.0	4.0	4.0
	451	4.0	5.0	4.0	5.0	4.0	4.0	5.0	5.0	4.0	4.0	4.0	4.0	2.0	3.5	5.0
	467	3.0	3.5	3.0	2.5			3.0	3.5	3.5	3.0	3.5	3.0	3.0	4.0	4.0
	508	5.0	5.0	4.0	3.0	5.0	2.0	4.0	4.0	5.0	5.0	5.0	3.0	4.5	3.0	4.5
	546		5.0	2.0	3.0	5.0		5.0	5.0		2.5	2.0	3.5	3.5	3.5	5.0
	563	1.0	5.0	3.0	5.0	4.0	5.0	5.0		2.0	5.0	5.0	3.0	3.0	4.0	5.0
	579	4.5	4.5	3.5	3.0	4.0	4.5	4.0	4.0	4.0	4.0	3.5	3.0	4.5	4.0	4.5
	623		5.0	3.0	3.0		3.0	5.0		5.0	5.0	5.0	5.0	2.0	5.0	4.0

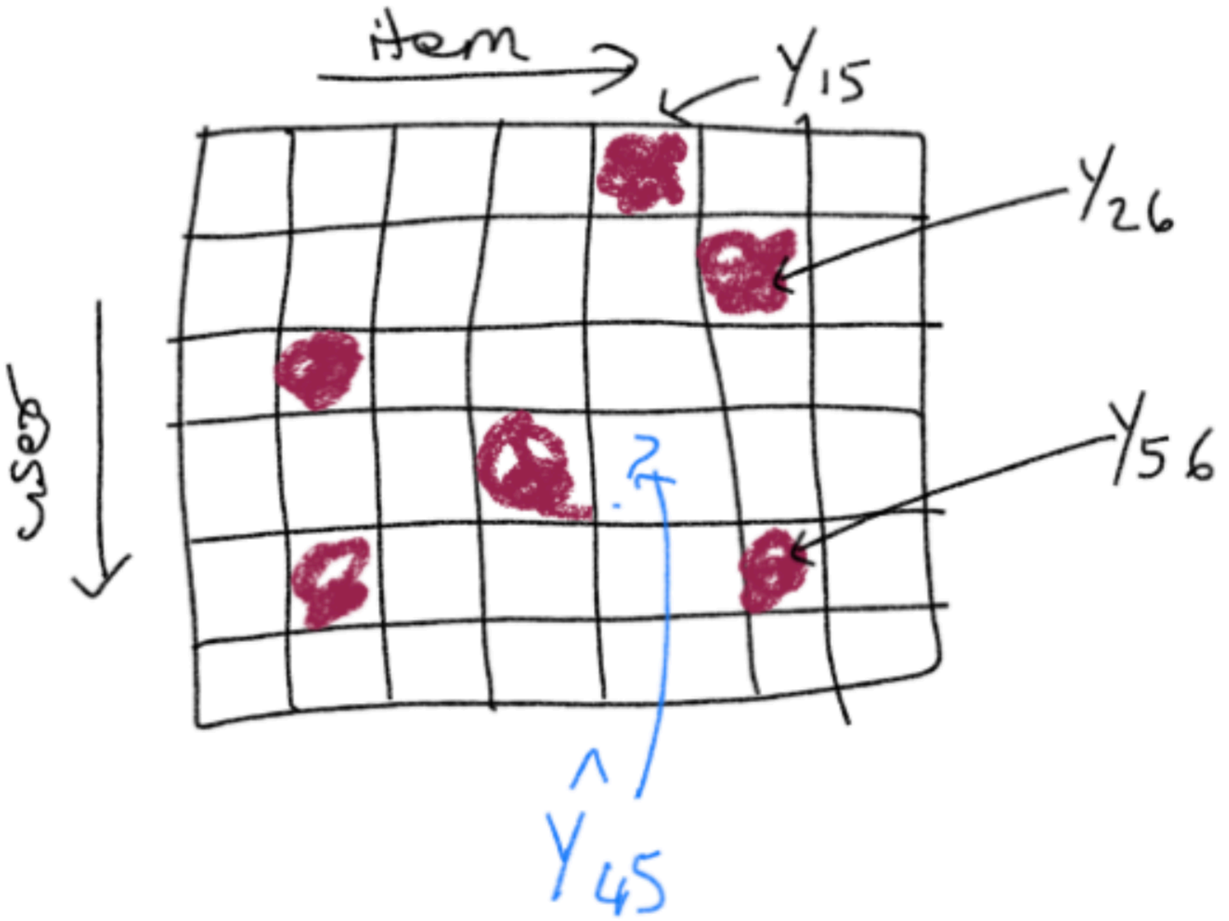
Making estimates

We wish to create a model for Y_{um}, the rating a user u will give reataurant m.

You can think of this as completing a matrix. As you might imagine, this is a very sparse matrix.

The maroon ratings we have. The blue ratings we dont, and the best we can do is to make some *estimate* of them. Very few users rate, and even those who do dont rate that much.

That is, we are after $\hat{Y}_{um}$ for an unknown rating Y_{um}, like $\hat{Y}_{45}$. And we'll train by figuring $\hat{Y}_{um}$ for a known rating Y_{um}, like $\hat{Y}_{15}$, comparing it to Y_{15}.



The Central Dogma of collaborative Filtering

Similar users will rate similar restaurants(items) similarly

What are Latent Factors?

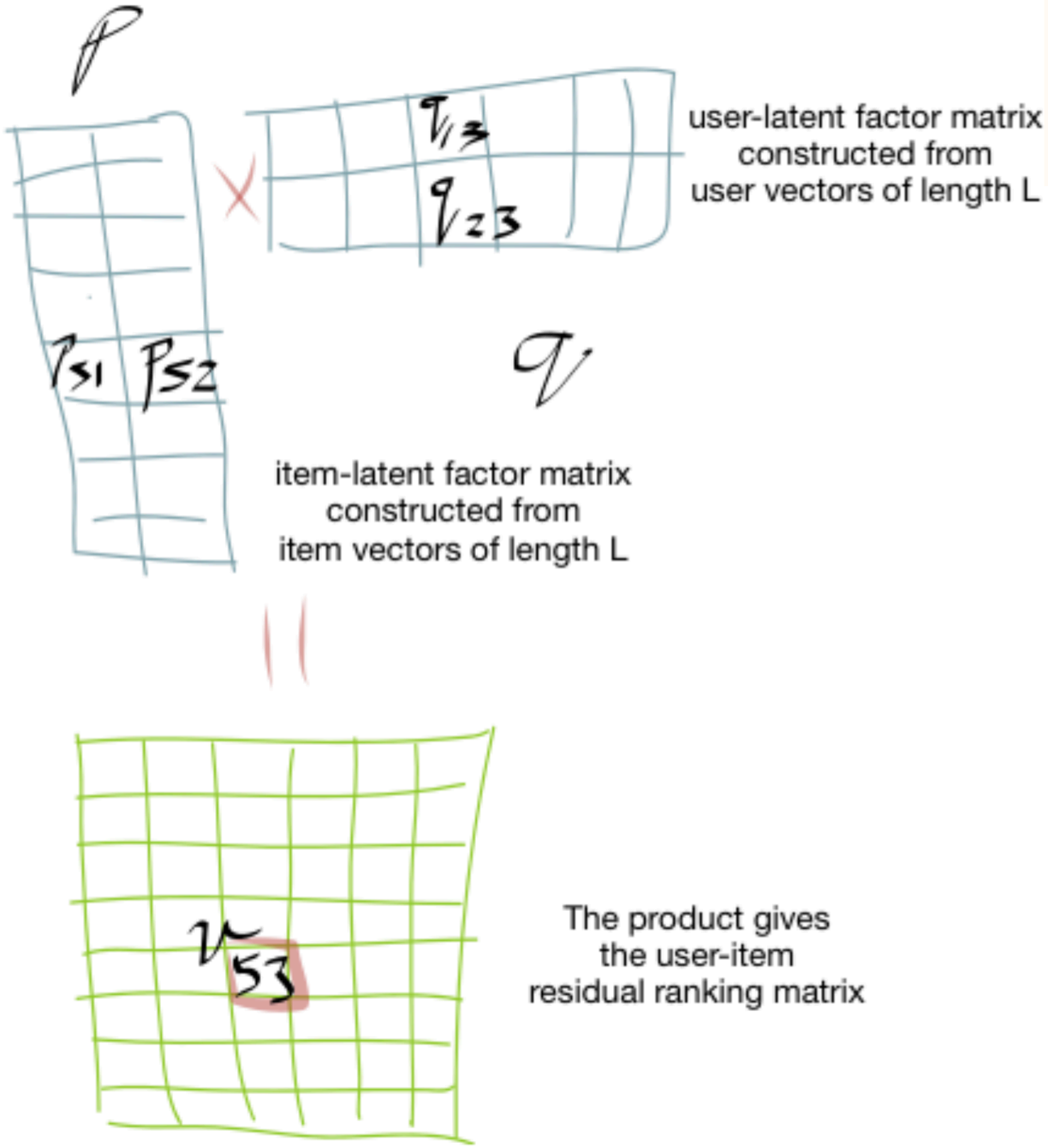
- We can think of latent factors as properties of restaurants (e.g., spiciness of food or price) that users have a positive or negative preference for.
- We do not observe these factors or the users' preferences directly, but we assume that they affect how users tend to rate restaurants. For example, if a restaurant serves a lot of spicy food and a user dislikes spicy food, then the restaurant would have a high "spiciness" factor, and the user would have a strongly negative preference, resulting in a prediction of a low rating.
- Latent factors thus give us an intuitive way to specify a generative model the obeys the central dogma, while retaining a certain explanatory globality, rather than simply a notion that one user is like another.

A latent factor is an embedding!

Let us associate with each item m a vector $\bar{q}_m$ of length L, where L is the number of latent factors in the model.

Similarly, lets associate with each user a vector $\bar{p}_u$ of length L.

Then we write as our model: $r_{um} = \bar{q}_m^T \cdot \bar{p}_u$.



Let us associate with each item m a vector $\bar{q}_m$ of length L, where L is the number of latent factors in the model.

Similarly, lets associate with each user a vector $\bar{p}_u$ of length L.

Then we write as our model: $r_{um} = \bar{q}_m^T \cdot \bar{p}_u$.

Latent Factor Matrix Multiplication

$$r_{um} = \bar{q}_m^T \cdot \bar{p}_u.$$

As matrices:

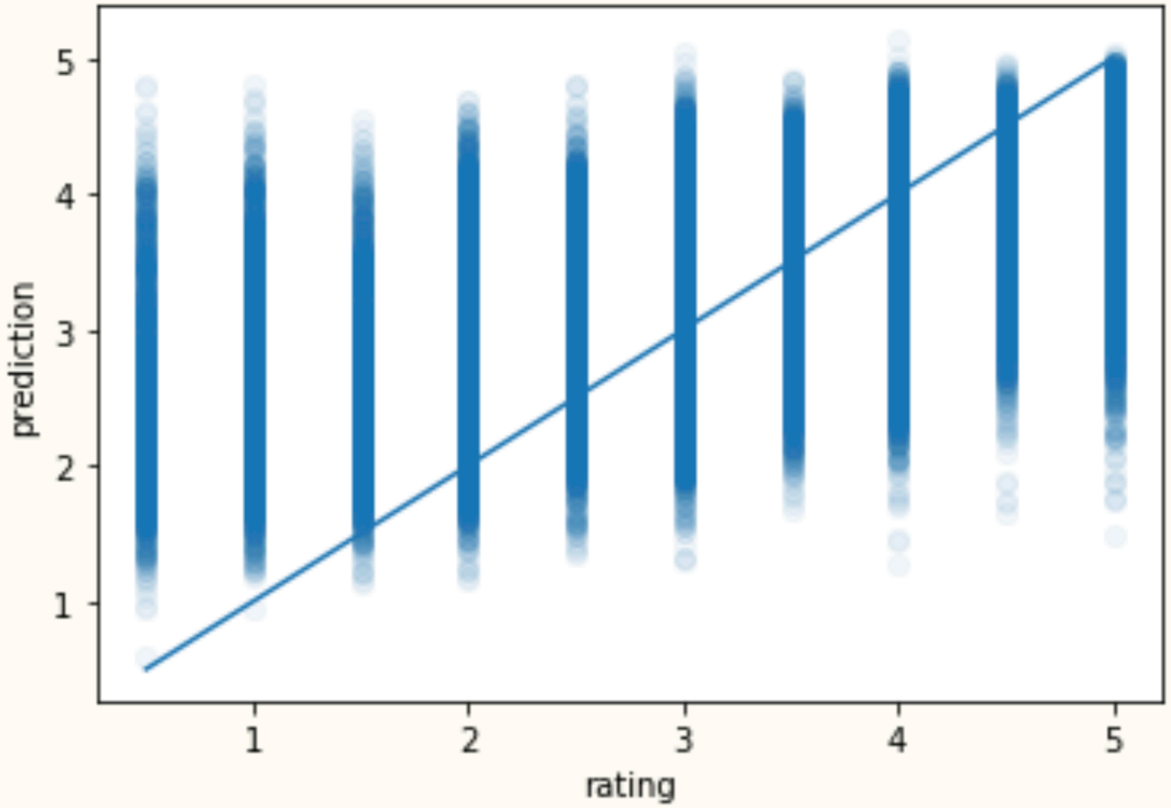
$$(M \times L) \cdot (L \times U) \rightarrow (M \times U)$$

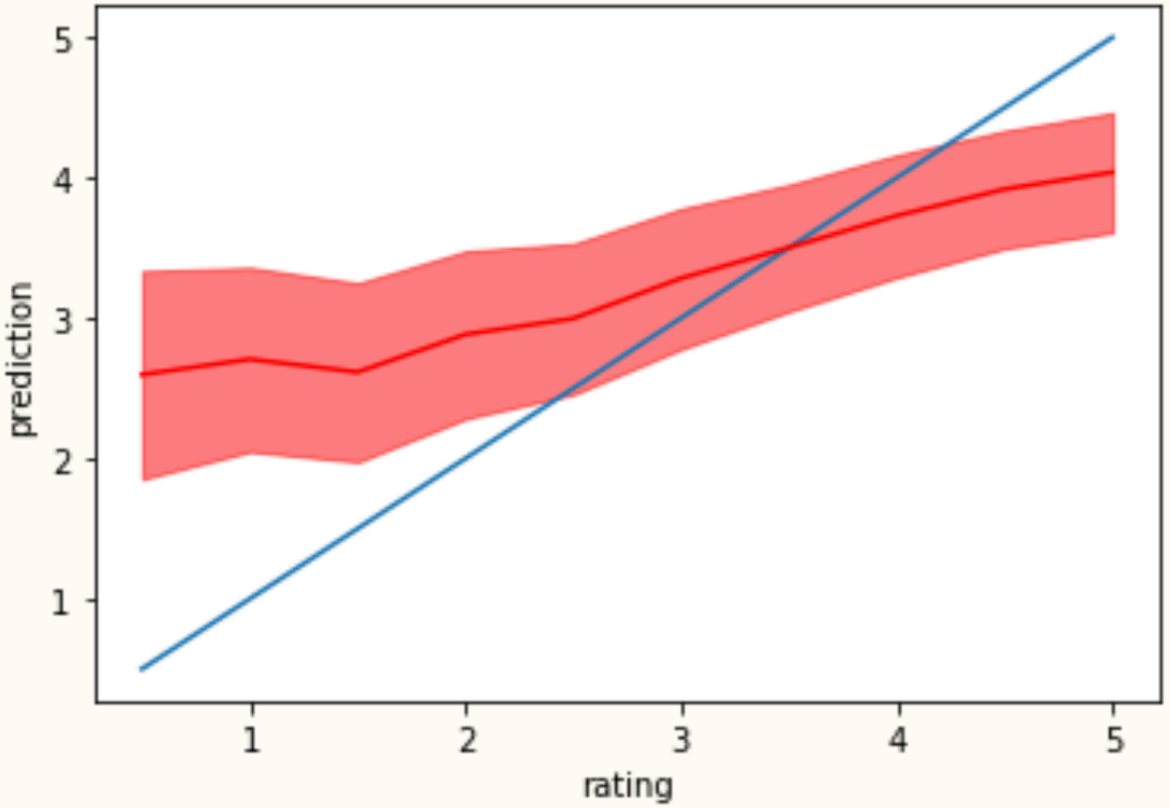
So we factorize the $M \times U$ ratings matrix into two matrices, one M times L, the number of latent factors, and the other $U \times L$.

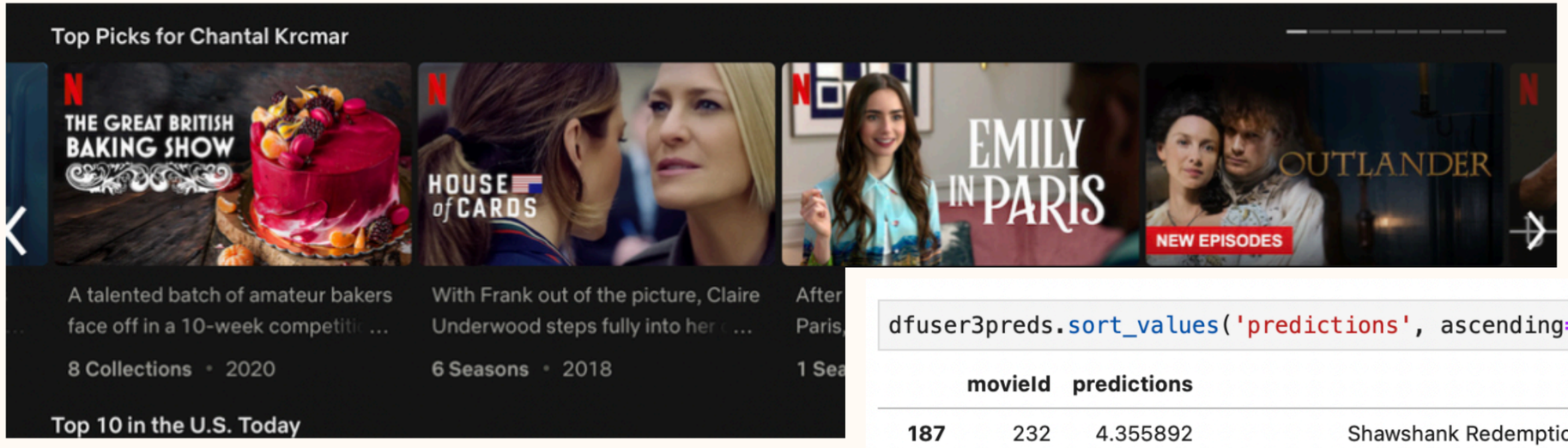
Here, for $L = 2$,

$$r_{53} = \bar{q}_3^T \cdot \bar{p}_5 = q_{31}p_{51} + q_{32}p_{52}$$

Predictions vs Ratings








```
dfuser3preds.sort_values('predictions', ascending=False).head(5)
```

	movieId	predictions	movie	userId
187	232	4.355892	Shawshank Redemption, The (1994)	3
888	1104	4.354324	Grand Day Out with Wallace and Gromit, A (1989)	3
2179	2395	4.306676	Lawrence of Arabia (1962)	3
4045	4261	4.298936	Paths of Glory (1957)	3
2211	2427	4.295087	Streetcar Named Desire, A (1951)	3

```
dfuser3.sort_values('predictions', ascending=False).head(5)
```

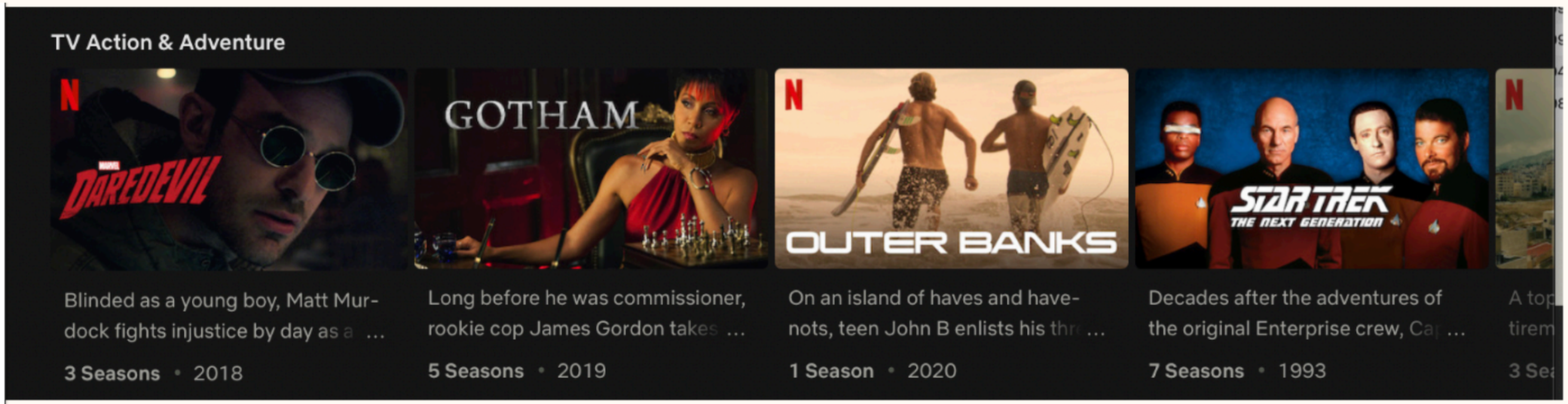
	userId	movieId	rating	timestamp	movie	predictions
498	3	442	3.0	986848685	In the Mood For Love (Fa yeung nin wa) (2000)	4.291507
458	3	192	2.0	945078528	Fight Club (1999)	4.225124
382	3	74	4.0	945079729	Goodfellas (1990)	4.214609
358	3	337	5.0	964538571	Notorious (1946)	4.196904
379	3	350	2.0	945078967	Brazil (1985)	4.188709

Predictions by genre

Top 5 war movies for user 3:

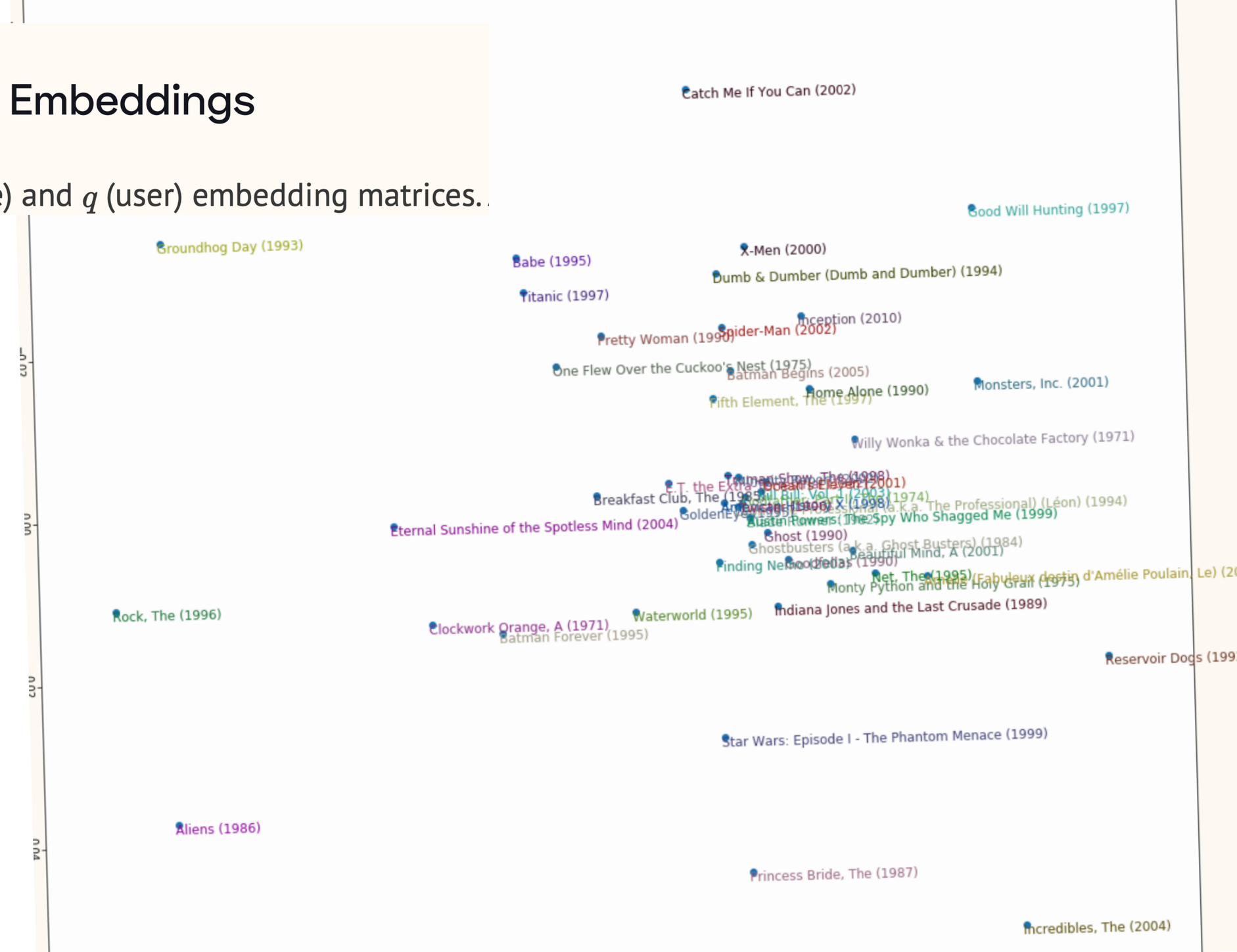
```
dfuser3predsbygenre[dfuser3predsbygenre['genre:War']==True][['movieId', 'predictions', 'movie','
```

	movieId	predictions	movie	userId	genre:War	genre:Adventure
2179	2395	4.306676	Lawrence of Arabia (1962)	3	True	True
4045	4261	4.298936	Paths of Glory (1957)	3	True	False
506	722	4.281205	Dr. Strangelove or: How I Learned to Stop Worr...	3	True	False
2013	2229	4.262089	Gallipoli (1981)	3	True	False
1928	2144	4.247703	Great Escape, The (1963)	3	True	True



Embeddings

Now lets turn to the p (movie) and q (user) embedding matrices.



cosine similarity for movie No Man's Land (2001)

```
sim = cosine_similarity(embeds_dict[459],
        [embeds_dict[x] for x in movieIds]).numpy()
```

Sort to get the top 10.

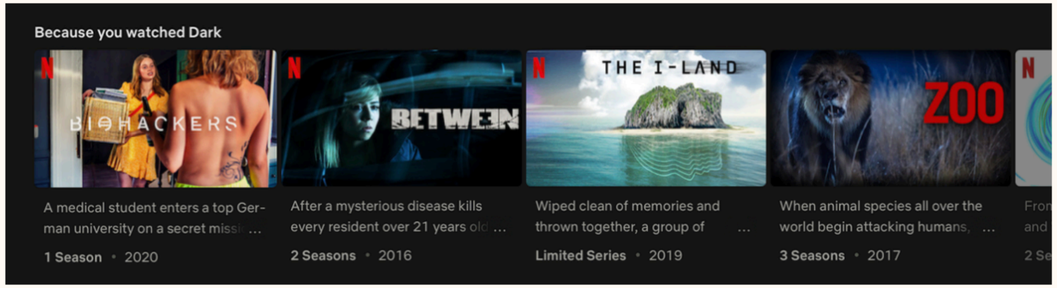
If you can figure what the latent factors roughly mean, you could even craft queries based on the kind of movie you want to see. This could even be done by averaging movies over genres and computing *average embeddings*.

	sim	movieId	movie
1059	0.992056	1059	(500) Days of Summer (2009)
2232	0.890796	2232	Phone Booth (2002)
997	0.884676	997	Splash (1984)
1812	0.849182	1812	Knight's Tale, A (2001)
385	0.848195	385	Pretty in Pink (1986)
5320	0.845839	5320	Mystery Train (1989)
487	0.833043	487	American President, The (1995)
469	0.810570	469	Quiz Show (1994)
1836	0.747937	1836	Sandlot, The (1993)
3082	0.740793	3082	Road, The (2009)

Particularizing to a user

	sim	movieId	movie	predictions
1059	0.992056	1059	(500) Days of Summer (2009)	3.651469
3082	0.740793	3082	Road, The (2009)	3.625063
487	0.833043	487	American President, The (1995)	3.533409
5320	0.845839	5320	Mystery Train (1989)	3.412999
469	0.810570	469	Quiz Show (1994)	3.359421
385	0.848195	385	Pretty in Pink (1986)	3.230702
1836	0.747937	1836	Sandlot, The (1993)	3.222170
1812	0.849182	1812	Knight's Tale, A (2001)	3.174309
997	0.884676	997	Splash (1984)	3.046751
2232	0.890796	2232	Phone Booth (2002)	2.966445

The order in which results are shown does not have to be similarity. It can be, for a given set of simlar results (say 10), ranked by the score the user would have given those similar movies...



	sim	userId
297	0.890576	297
50	0.890151	50
497	0.880803	497
154	0.783779	154
419	0.782829	419
267	0.778596	267
79	0.774985	79

Cosine similarity on users

You can take dot products of users, as well.

```
sim = cosine_similarity(user_embeds_dict[3],
        [user_embeds_dict[x] for x in userIds]).numpy()
```

A movie-cafe based dating site, anyone?

Scoped by follow circles or existing searches, this is how sites recommend new groups or new topics you might be interested in.