MICES 2025

2025.06.18

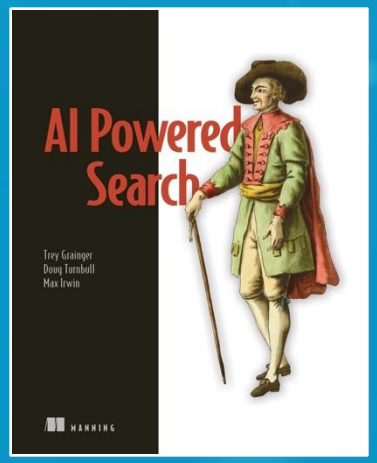
Personalizing e-commerce search with latent behavioral embeddings

Trey Grainger

Searchkernel

Founder / CTO

Author, *AI-Powered Search*



Searchkernel

Building the next generation of Search.

Career

Searchkernel
Founder / CTO

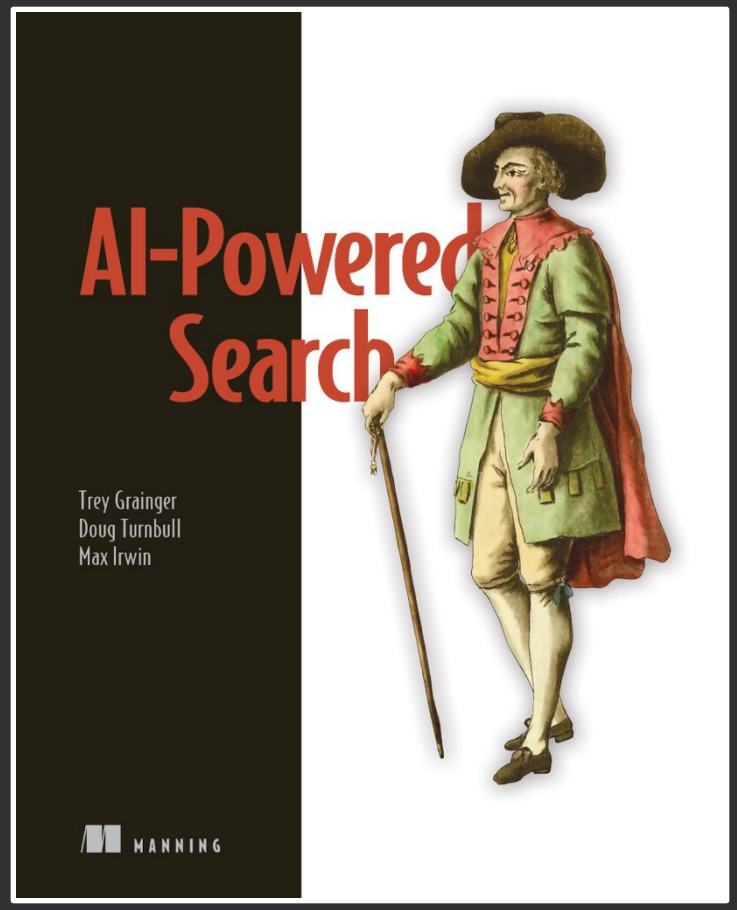
Presearch
Chief Technology Officer

Lucidworks
Chief Algorithms Officer
SVP Engineering

CAREERBUILDER
Director of Engineering, Search & Recommendations

Books

Education



(45% Discount Code: mices45)

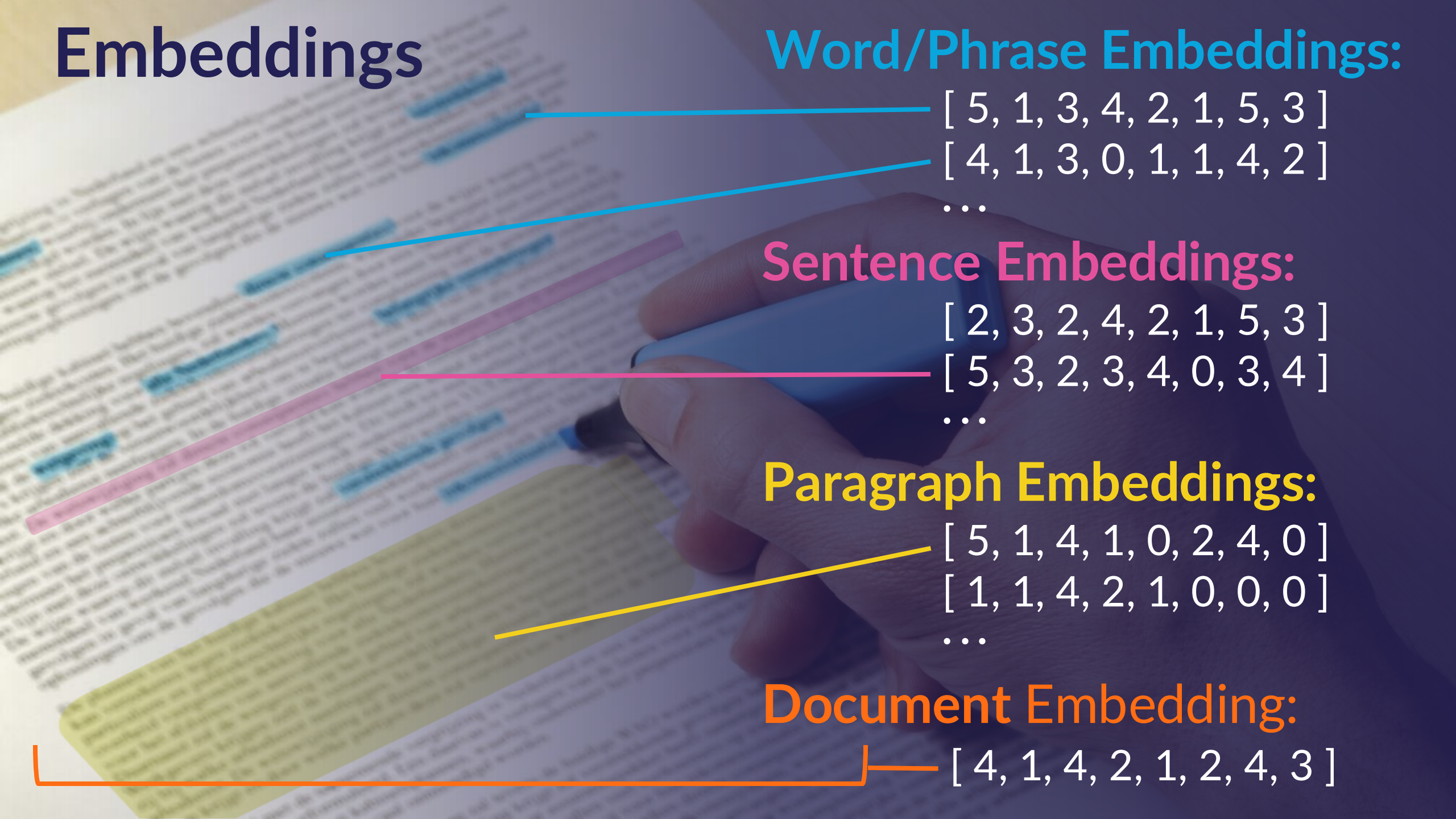
AI-Powered Search

- RAG (Retrieval Augmented Generation)
- Generative Search & Summarization
- Learning to Rank & implicit judgments
- Semantic Search
- Dense Vector Search
- Fine-tuning LLMs for Search
- Personalized Search & Recommendations
- Knowledge Graph Learning
- User signals boosting & click models
- Crowdsourced Relevance
- Quantization techniques
- Hybrid search
- Multimodal search

Get a copy @ *http://aiPoweredSearch.com*

Agenda

- Refresher on embeddings and latent features
- The personalization spectrum between search and recommendations
- Leveraging a user's signals to implement matrix factorization and personalization from latent features
- Using embedding vectors to generate personalization profiles
- Mixing user signals and content-based attributes to generate multimodal personalization
- Clustering products by embeddings to create personalization guardrails
- Avoiding the pitfalls of personalized search

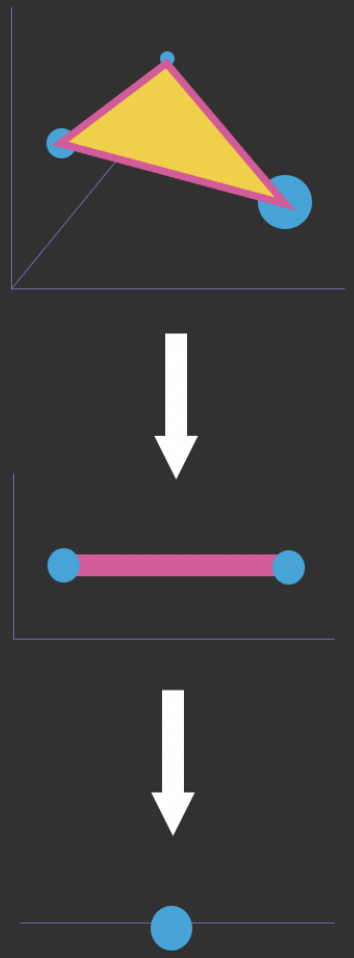

Embeddings
Word/Phrase Embeddings:
[5, 1, 3, 4, 2, 1, 5, 3]
[4, 1, 3, 0, 1, 1, 4, 2]
...
Sentence Embeddings:
[2, 3, 2, 4, 2, 1, 5, 3]
[5, 3, 2, 3, 4, 0, 3, 4]
...
Paragraph Embeddings:
[5, 1, 4, 1, 0, 2, 4, 0]
[1, 1, 4, 2, 1, 0, 0, 0]
...
Document Embedding:
[4, 1, 4, 2, 1, 2, 4, 3]

An **embedding** is a set of ***coordinates in vector space*** into which we map a concept.

An inverted index creates embeddings, with one dimension per term

	apple	caffeine	cheese	coffee	drink	donut	food	juice	pizza	tea	water	... *term N*
latte	0	0	0	0	0	0	0	0	0	0	0	...
cappuccino	0	0	0	0	0	0	0	0	0	0	0	...
apple juice	1	0	0	0	0	0	0	1	0	0	0	...
cheese pizza	0	0	1	0	0	0	0	0	1	0	0	...
donut	0	0	0	0	0	1	0	0	0	0	0	...
soda	0	0	0	0	0	0	0	0	0	0	0	...
green tea	0	0	0	0	0	0	0	0	0	1	0	...
water	0	0	0	0	0	0	0	0	0	0	1	...
cheese bread sticks	0	0	1	0	0	0	0	0	0	0	0	...
cinnamon sticks	0	0	0	0	0	0	0	0	0	0	0	...

Embedding (verb) maps concepts into another vector space (usually a lower-dimensional space)



	food	drink	dairy	bread	caffeine	sweet	calories	healthy
apple juice	0	5	0	0	0	4	4	3
cappuccino	0	5	3	0	4	1	2	3
cheese bread sticks	5	0	4	5	0	1	4	2
cheese pizza	5	0	4	4	0	1	5	2
cinnamon bread sticks	5	0	1	5	0	3	4	2
donut	5	0	1	5	0	4	5	1
green tea	0	5	0	0	2	1	1	5
latte	0	5	4	0	4	1	3	3
soda	0	5	0	0	3	5	5	0
water	0	5	0	0	0	0	0	5

We then leverage the vector space to explore similarity

Phrase:	Vector:
apple juice:	[0, 5, 0, 0, 0, 4, 4, 3]
cappuccino:	[0, 5, 3, 0, 4, 1, 2, 3]
cheese bread sticks:	[5, 0, 4, 5, 0, 1, 4, 2]
cheese pizza:	[5, 0, 4, 4, 0, 1, 5, 2]
cinnamon bread sticks:	[5, 0, 4, 5, 0, 1, 4, 2]
donut:	[5, 0, 1, 5, 0, 4, 5, 1]
green tea:	[0, 5, 0, 0, 2, 1, 1, 5]
latte:	[0, 5, 4, 0, 4, 1, 3, 3]
soda:	[0, 5, 0, 0, 3, 5, 5, 0]
water:	[0, 5, 0, 0, 0, 0, 0, 5]

Vector Similarity (a, b):

$$\cos(\theta) = \frac{a \cdot b}{|a| \times |b|}$$

Vector Similarity Scores:

Ranked Results: Cheese Pizza	
0.99	cheese bread sticks
0.91	cinnamon bread sticks
0.89	donut
0.47	latte
0.46	apple juice
...	...
0.19	water

Ranked Results: Green Tea	
0.94	water
0.85	cappuccino
0.80	latte
0.78	apple juice
0.60	soda
...	...
0.19	donut

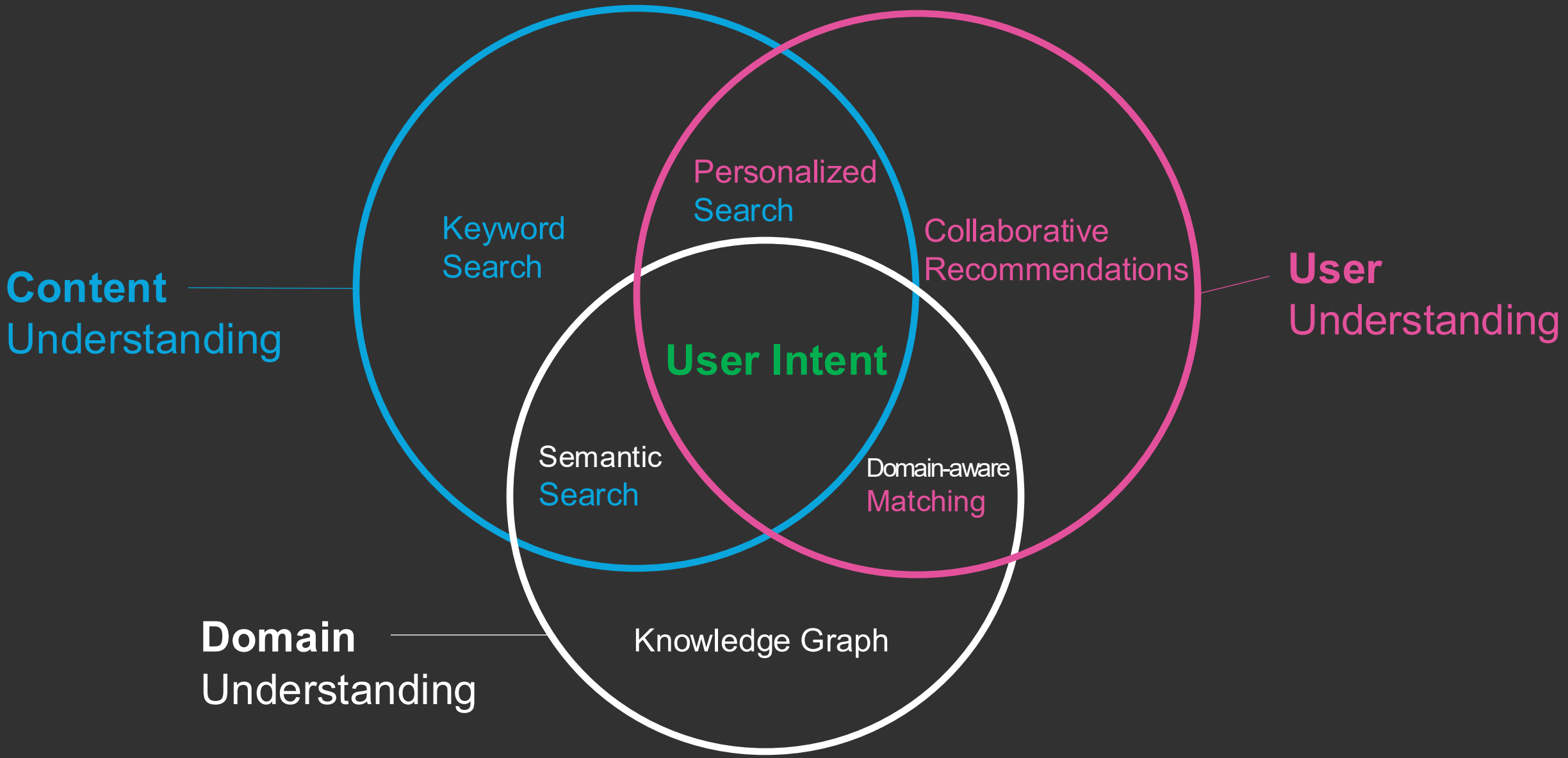
Embedding concepts into a vector space

https://atlas.nomic.ai/map/stablediffusion

With LLMs and other Foundation Models, the dimensions learned are Latent Features*.

📌 * We'll return to this later

Dimensions of User Intent



Personalization Spectrum

Traditional
Keyword Search
(Completely User-specified)

User-guided
Recommendations
(Mostly driven by user profile,
partially user-specified)

Personalized
Search
(Mostly user-specified,
partially driven by user profile)

Traditional
Recommendations
(Completely driven by user profile)

Recommendations Approaches

Content-based Matching (Content Filtering)

Recommendations:
-**fabric softener**
-**dryer sheets**

detergent

category: laundry
text: cleans clothes spotless and removes stains leaving a fresh citrus scent

fabric softener

category: laundry
text: softens clothes and leaves a fresh citrus scent

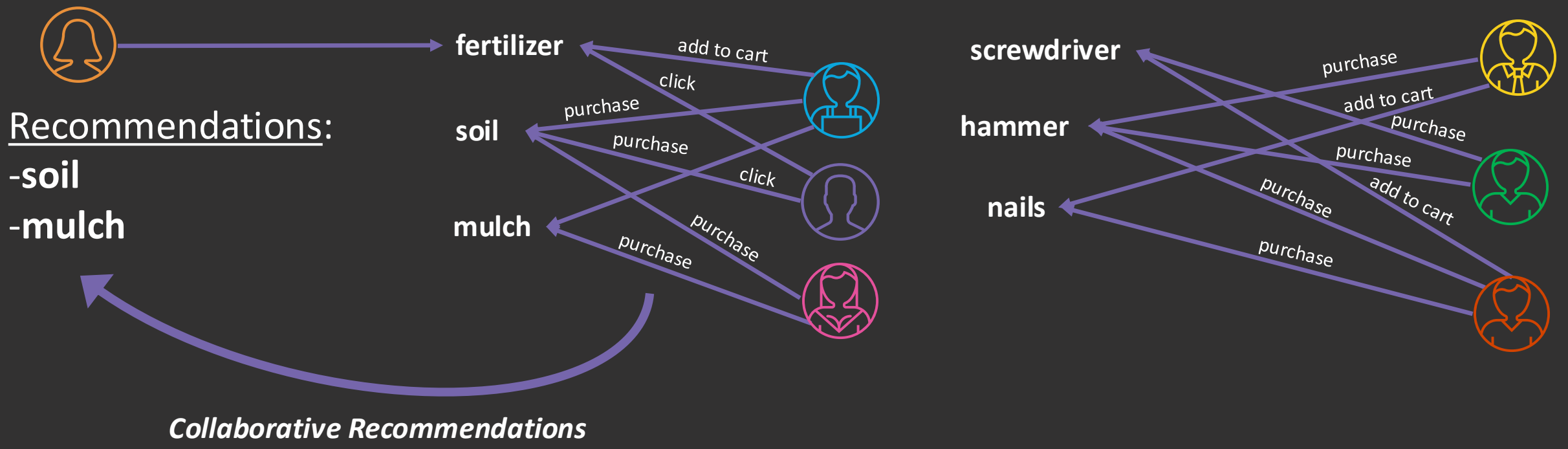
dryer sheets

category: laundry
text: removes static electricity from clothes and leaves a fresh citrus scent

Content-based Recommendations

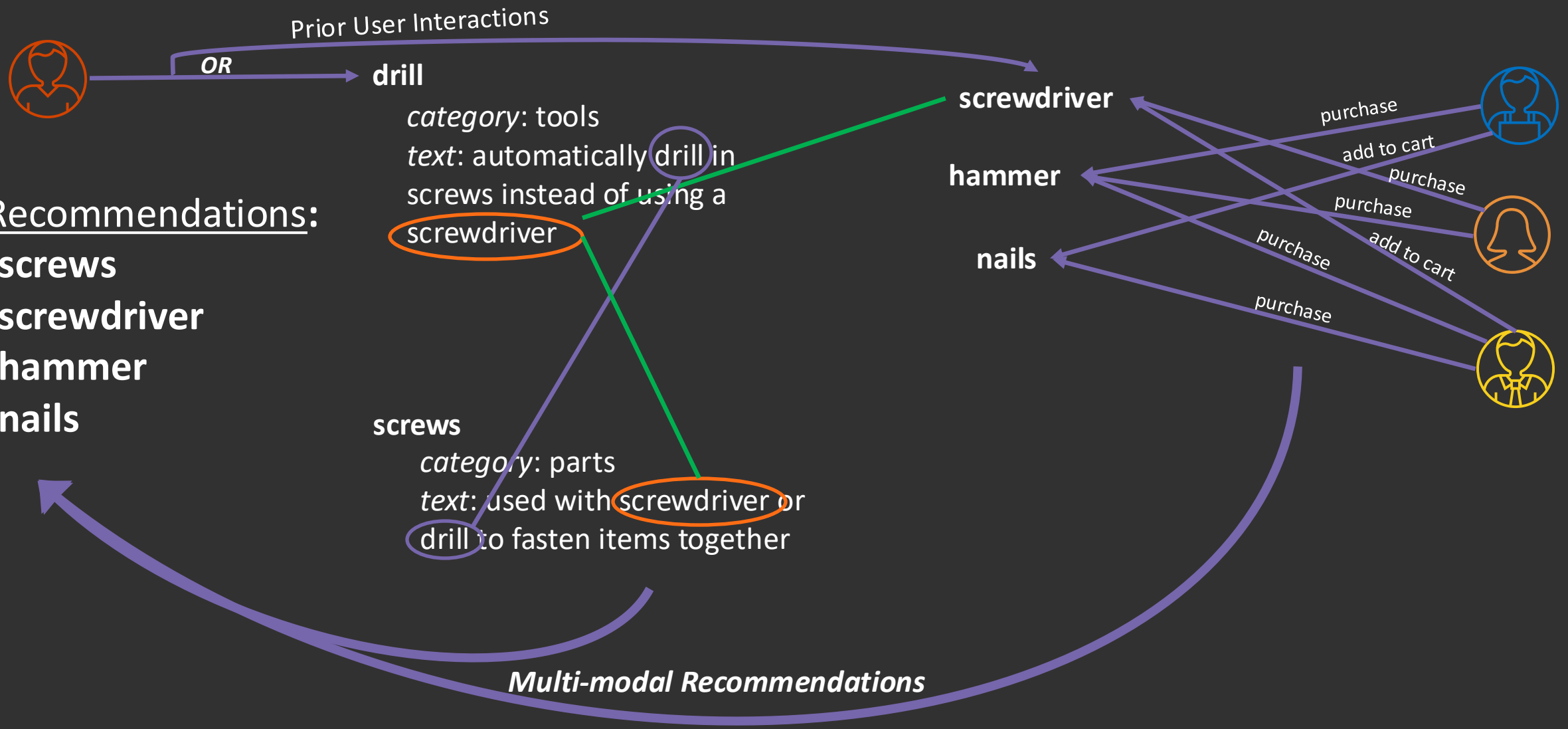
Recommendations Approaches

Behavior-based Matching (Collaborative Filtering)



Recommendations Approaches

Content-based Matching (Content Filtering) + **Behavior-based Matching (Collaborative Filtering)**



Recommendations:

- **screws**
- **screwdriver**
- **hammer**
- **nails**

Generating latent behavioral embeddings for personalization

Collaborative Filtering (Alternating Least Squares)

Alternating Least Squares (Matrix Factorization based Collaborative Filtering)

```python
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row

als = ALS(maxIter=3, rank=10, regParam=0.15, implicitPrefs=True,
          userCol="userIndex", itemCol="productIndex", ratingCol="rating",
          coldStartStrategy="drop", seed=0)

(training, test) = indexed_prefs.randomSplit([0.8, 0.2], 0)

model = als.fit(indexed_prefs) (1)
predictions = model.transform(test) (1)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print(f"Root-mean-square error = {rmse}")

indexed_user_recs = model.recommendForAllUsers(10)
indexed_user_recs.show(5)
```

Results:

```
Root-mean-square error = 0.9589

+---------+--------------------+
|userIndex|     recommendations|
+---------+--------------------+
|        0|[{11, 0.014824464...|
|        1|[{5, 0.010805087}...|
|        2|[{7, 0.060263287}...|
|        3|[{1, 0.02967207},...|
|        4|[{14, 0.007906279...|
+---------+--------------------+
only showing top 5 rows
```

```
User: u478462

Previous Searches:
--apple
--macbook

Previous Product Interactions:
--type: click, name: Apple® - iPad® 2 with Wi-Fi - 16GB - Black
--type: add-to-cart, name: Apple® - iPad® 2 with Wi-Fi - 16GB - Black
--type: purchase, name: Apple® - iPad® 2 with Wi-Fi - 16GB - Black
--type: click, name: Apple® - MacBook® Air - Intel® Core™ i5 Processor ...
```

Alternating Least Squares (Matrix Factorization based Collaborative Filtering)

```python
def get_query_time_boosts(user, boosts_collection):
  request = {"query": "",
             "return_fields": ["product", "boost"],
             "filters": [("user", user)] if user else [],
             "limit": 10,
             "order_by": [("boost", "desc")]}

  response = boosts_collection.search(request)
  signals_boosts = response["docs"]
  return " ".join(f'"{b["product"]}"^{b["boost"] * 100}' for b in signals_boosts)

def run_main_query(query, signals_boosts):
  request = product_search_request(query if query else "")
  request["query_boosts"] = signals_boosts if signals_boosts else "*"
  return products_collection.search(request)

recs_collection = engine.get_collection("user_item_recommendations")
user = "u478462"
boosts = get_query_time_boosts(user, recs_collection)

response = run_main_query(None, boosts)

print(f"Boost Query:\n{boosts}")
display_product_search("", response["docs"])
```

```
Boost Query:
"885909457588"^75.393623 "097360810042"^18.904798 "821793013776"^15.852094 "610839379408"^10.217768000000001 "635753493559"^9.087185 "885909395095"^8.304988 "885909457595"^7.9175640000000005 "885909431618"^7.375394 "885909459858"^6.592548 "885909436002"^6.1031554

Recommendations:
```

Search

Name: Apple® - iPad® 2 with Wi-Fi - 16GB - Black | **Manufacturer:** Apple®

9.7" widescreen display; 802.11a/b/g/n Wi-Fi; Bluetooth; iBooks support; measures just 0.34" thin and weighs only 1.35 lbs.

Name: HTC - Flyer Tablet with 16GB Internal Memory - White | **Manufacturer:** HTC

Android 2.3 Gingerbread operating system7" color touch screenWi-Fi16GB memoryHTC notes, Watch and Listen apps

Name: Asus - Eee Pad Transformer Tablet with 16GB Storage Memory - Brown/Black | **Manufacturer:** Asus

Android 3.0 Honeycomb10.1" WXGA IPS touch screen Wi-Fi16GB hard drive

Name: Samsung - Galaxy Tab 10.1 - 16GB - Metallic Gray | **Manufacturer:** Samsung

Android 3.1 (Honeycomb) operating system10.1" WXGA touch screenWi-Fi

Name: Apple® - iPod touch® 32GB* MP3 Player (4th Generation - Latest Model) - Black | **Manufacturer:** Apple®

FaceTime camera, HD video recording, Retina display, Multi-Touch interface; gorgeous 3.5" widescreen display; Wi-Fi Web browsing

Name: Apple® - iPad® 2 with Wi-Fi - 32GB - Black | **Manufacturer:** Apple®

9.7" widescreen display; 802.11a/b/g/n Wi-Fi; Bluetooth; iBooks support

Non-personalized Search Results

```
query = "tablet"

response = run_main_query(
  query, None, 7
)
print(f"Non-personalized Query")
display_product_search(
  query, response["docs"]
)
```

Non-personalized Query

tablet Search

Name: Init™ - Tablet Sleeve - Olive | **Manufacturer:** Init™

Fits most tablets with up to a 10" display; heavy-duty neoprene material

---

Name: Memorial Tablet - CD | **Manufacturer:** Ltm/cd41

Name: Stone Tablet - CD | **Manufacturer:** Important Records

---

Name: Sony - AC Power Adapter for Sony Tablet S | **Manufacturer:** Sony

Compatible with Sony Tablet S; charges your tablet

Personalized Search Results

```
query = "tablet"

recs_collection = engine.get_collection(
  "user_item_recommendations"
)
user = "u478462"
boosts = get_query_time_boosts(
  user, recs_collection
)

response = run_main_query(query, boosts, 7)
print(f"Personalized Query")
display_product_search(query, response["docs"])
```

Personalized Query

tablet Search

Name: Apple® - iPad® 2 with Wi-Fi - 16GB - Black | **Manufacturer:** Apple®

9.7" widescreen display; 802.11a/b/g/n Wi-Fi; Bluetooth; iBooks support; measures just 0.34" thin and weighs only 1.35 lbs.

Name: HTC - Flyer Tablet with 16GB Internal Memory - White | **Manufacturer:** HTC

Android 2.3 Gingerbread operating system7" color touch screenWi-Fi16GB memoryHTC notes, Watch and Listen apps

Name: Asus - Eee Pad Transformer Tablet with 16GB Storage Memory - Brown/Black | **Manufacturer:** Asus

Android 3.0 Honeycomb10.1" WXGA IPS touch screen Wi-Fi16GB hard drive

Name: Samsung - Galaxy Tab 10.1 - 16GB - Metallic Gray | **Manufacturer:** Samsung

Android 3.1 (Honeycomb) operating system10.1" WXGA touch screenWi-Fi

Name: Apple® - iPad® 2 with Wi-Fi - 32GB - Black | **Manufacturer:** Apple®

9.7" widescreen display; 802.11a/b/g/n Wi-Fi; Bluetooth; iBooks support

Name: Init™ - Tablet Sleeve - Olive | **Manufacturer:** Init™

Fits most tablets with up to a 10" display; heavy-duty neoprene material

Name: Memorial Tablet - CD | **Manufacturer:** Ltm/cd41

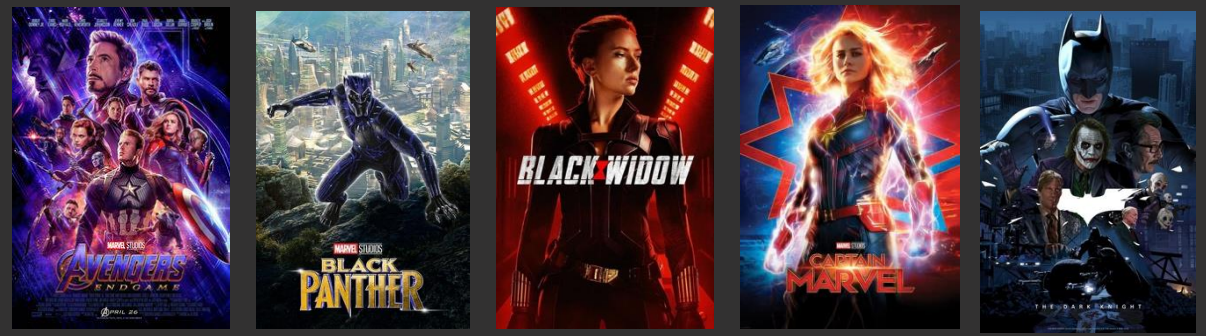
LLM-based embedding dimensions are latent features

Items

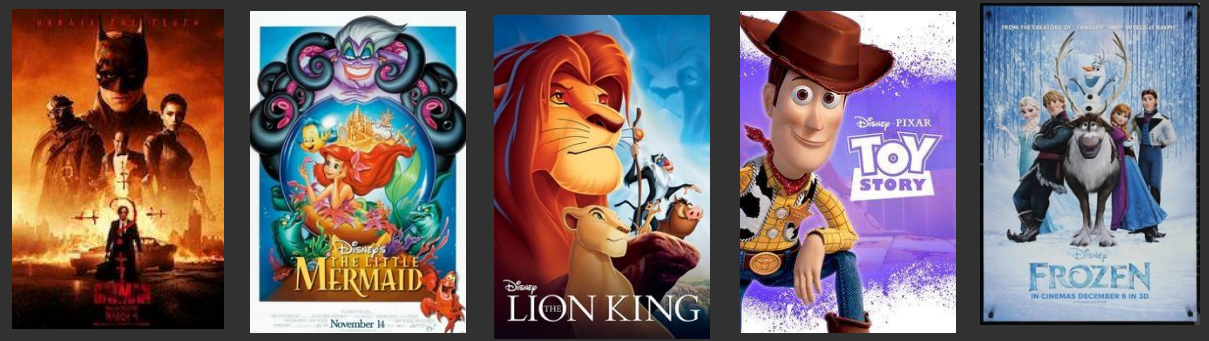
Latent Features (Content-based)								
	1	4.1	4.1	8.2	7.7	1.9	4.0	≈ size?
	2	9.8	1.2	2.2	0.8	1.2	0.3	≈ color?
	3	9.9	9.9	3.2	1.7	9.2	0.2	≈ computer-like?
	...	...	...	...	...	...	...	
	768	3.0	3.0	7.3	5.9	4.3	0.3	≈ price?

…and collaborative filtering **ALSO** relies on latent features

- User 1:
  - *Avengers: Endgame*
  - *Black Panther*
  - *Black Widow*
- User 2:
  - *Black Panther*
  - *Black Widow*
  - *Captain Marvel*
- User 3:
  - *Black Widow*
  - *The Dark Knight*
  - *The Batman*
- User 4:
  - *The Little Mermaid*
  - *The Lion King*
  - *Toy Story*
- User 5:
  - *The Lion King*
  - *Toy Story*
  - *Frozen*

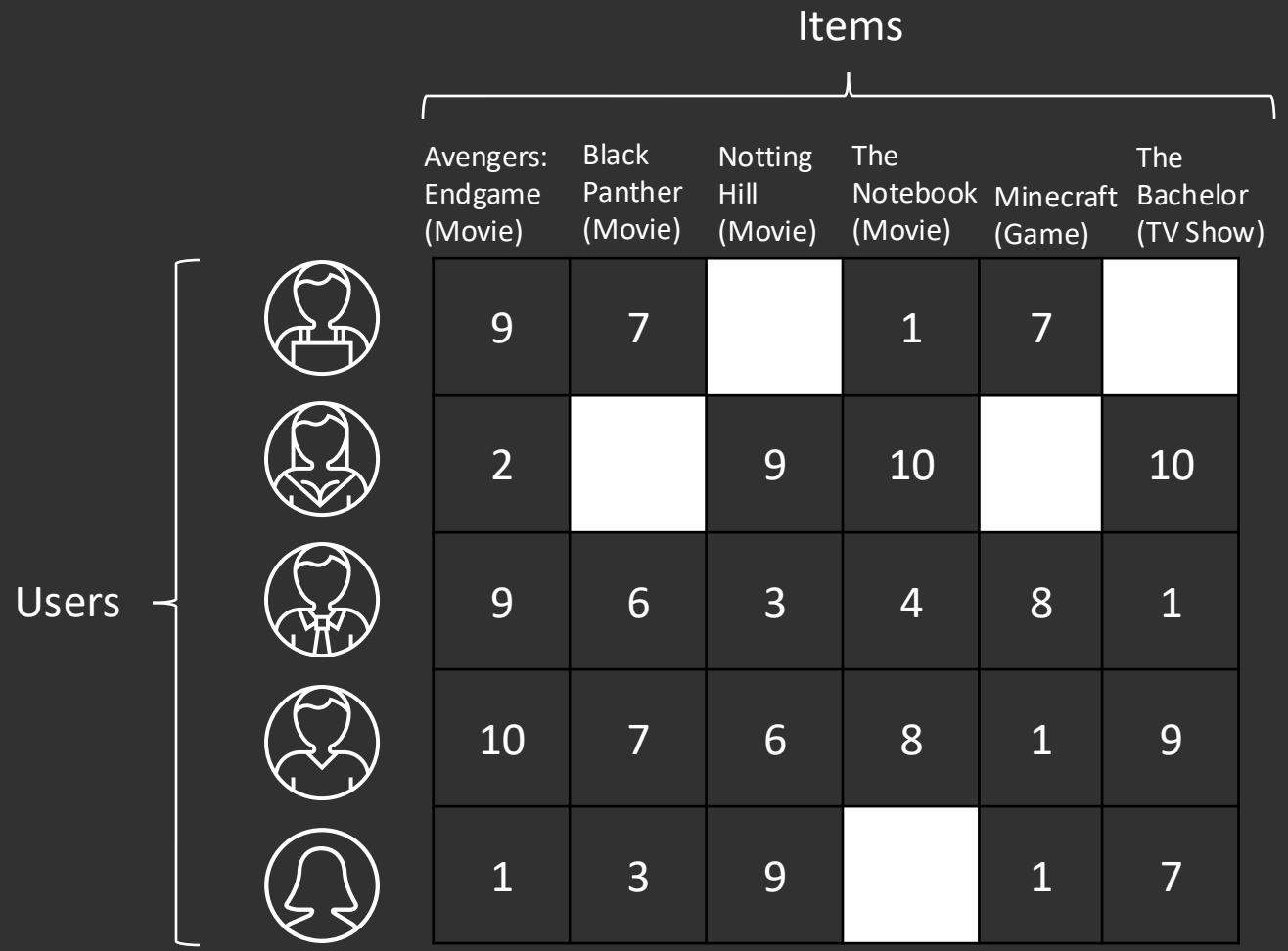


User1, User2, User3:
- All of these are movies about superheroes
- Most of them were made by Marvel Studios, though some were made by Warner Brothers (DC Comics)
- They are all action movies
- They are not suitable for small children due to violence and/or language



User4, User5:
- All of them are animated movies
- All of them are suitable for small children
- All of them are made by Disney/Pixar

Starting point for Collaborative Filtering:
User-Item Interaction Matrix



Matrix Factorization: learning latent features from user signals

User-Item Rankings (R)

User Feature Matrix (U)

Item Feature Matrix (I)

Generating recommendations using collaborative filtering

0.67	-0.51	2.81
1.13	3.18	-0.13
1.66	-0.08	2.50
-0.05	2.76	2.29
2.20	2.16	-0.20

×

Avengers: Endgame (Movie)	Black Panther (Movie)	Notting Hill (Movie)	The Notebook (Movie)	Minecraft (Game)	The Bachelor (TV Show)
0.09	0.71	1.91	1.74	1.80	0.28
0.75	0.75	2.17	2.54	-1.17	3.02
3.43	2.21	0.01	0.46	1.89	0.30

=

Avengers: Endgame (Movie)	Black Panther (Movie)	Notting Hill (Movie)	The Notebook (Movie)	Minecraft (Game)	The Bachelor (TV Show)
9.32	6.30	**0.20**	1.16	7.11	**-0.51**
2.04	**2.90**	9.06	9.98	**-1.93**	9.88
8.66	6.64	3.02	3.84	7.81	0.97
9.92	7.10	5.92	7.98	1.01	9.01
1.13	2.74	8.89	**9.22**	1.05	7.08

Calculating Preferences from Latent Factors:

Avengers: Endgame (Movie)

(0.67 * 0.09) + (-0.51 * 0.75) + (2.81 * 3.43) =

9.32

The Notebook (Movie)

(1.13 * 1.74) + (3.18 * 2.54) + (-0.13 * 0.46) =

9.98

BUT, we can also just use the latent features directly as behavioral embeddings!

Latent User Feature 1	Latent User Feature 2	Latent User Feature 3
0.67	-0.51	2.81
1.13	3.18	-0.13
1.66	-0.08	2.50
-0.05	2.76	2.29
2.20	2.16	-0.20

×

Avengers: Endgame (Movie)	Black Panther (Movie)	Notting Hill (Movie)	The Notebook (Movie)	Minecraft (Game)	The Bachelor (TV Show)	
0.09	0.71	1.91	1.74	1.80	0.28	Latent Item Feature 1
0.75	0.75	2.17	2.54	-1.17	3.02	Latent Item Feature 2
3.43	2.21	0.01	0.46	1.89	0.30	Latent Item Feature 3

user1: [0.67, -0.51, 2.81]
user2: [1.13, 3.18, -0.13]
...

avengers_endgame: [0.09, 0.75, 3.43]
the_notebook: [0.71, 0.75, 2.21]
...

Generating LLM-based embeddings for personalization

Naïve approach to generating user-based embeddings:

Average the embeddings for their past N-product interactions (purchases, add-to-carts, clicks, etc.)

We can generate real-time personalization vectors for our users based on both their current query AND their relevant previous behavior

Current search: microwave

Previous Interactions:

Hello Kitty Plush Toy	GE Electric Razor (Black)
GE Bright White Light Bulbs	Samsung Stainless-steel Refrigerator

Personalization based on irrelevant previous behavior

GE - 0.7 Cu. Ft. Compact Microwave - Black

Cook or reheat your meals in minutes with this compact 700-watt microwave oven that features instant-on LED controls which heat food with the simple touch of a button. It also provides peace of mind with a control lockout feature that locks the microwave when it's not in use.

Dept Appliance Brand GE Color Black

Hello Kitty - 0.7 Cu. Ft. Compact Microwave

Cook your favorite treats in this microwave that features 6 preprogrammable recipes for customized use and a compact design for optimal placement.

Dept Photo/Commodities Brand Hello Kitty

GE - 1.1 Cu. Ft. Mid-Size Microwave - White

Prepare hassle-free meals and snacks with this

vs.

Personalization based on relevant previous behavior

Samsung - Mid-Size Microwave - Stainless-steel

This mid-size microwave is large enough for almost any meal, but compact enough to fit into almost any space. Take an interactive product tour of this Samsung Microwave! (479KB Flash demo)

Dept Appliance Brand Samsung Color Stainless-Steel

Samsung - 1.7 Cu. Ft. Over-the-Range Microwave - Stainless-Steel

The capabilities of a toaster oven, convection oven, range hood and microwave combine in this microwave that features a 1.7 cu. ft. capacity for simultaneously cooking multiple dishes and sensor cook options to take the guesswork out of quick meal prep.

Dept Appliance Brand Samsung Color Stainless-

Better approach to generating user-based embeddings:

Find the relevant query context and generate user-embeddings dynamically based on that context.

Clustering products by embedding is an easy way to generate categories to scope query context

Clustering products to generate dynamic category contexts

```
def get_clusters(data, algorithm, args, kwds):
  return algorithm(*args, **kwds).fit(data)

def assign_clusters(labels, product_names):
  clusters = defaultdict(lambda:[], {})
  for i in range(0, len(labels)):
    clusters[labels[i]].append(product_names[i])
  return clusters

algo = get_clusters(product_embeddings, cluster.KMeans, (),
                    {"n_clusters":100, "n_init":10, "random_state":0}) #<1>
labels = algo.predict(product_embeddings)
clusters = assign_clusters(labels, product_names) #<2>

#<1> Generate 100 clusters using a KMeans clustering algorithms
#<2> Assign each product name to its corresponding cluster label
```

Visualizing the clusters

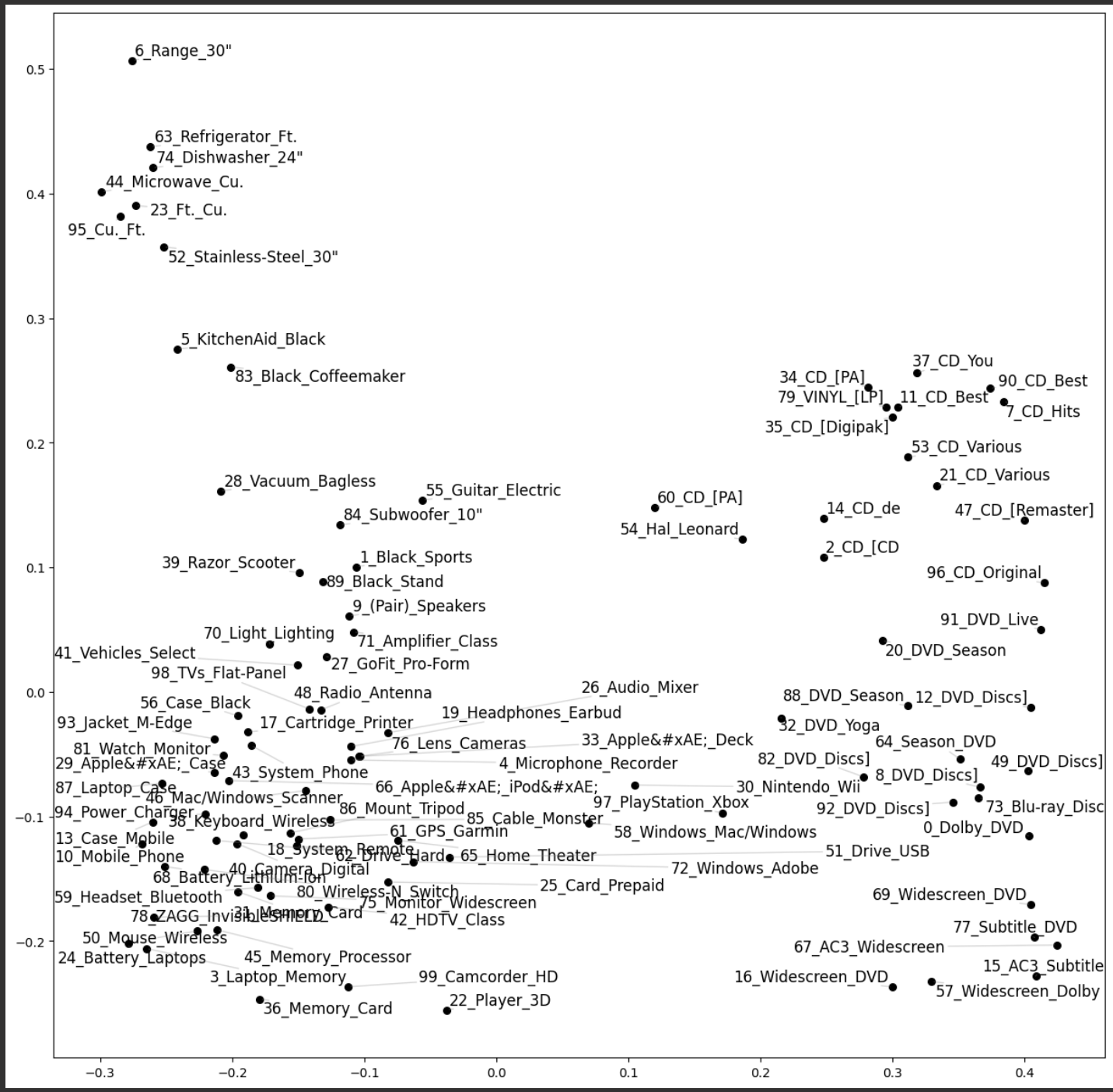
Visualize the category contexts

```
import collections, numpy as np, matplotlib.pyplot as plt
from adjustText import adjust_text
from sklearn.decomposition import PCA

plt.figure(figsize=(15, 15))
pca = PCA(100, svd_solver="full")
centers = algo.cluster_centers_
plot_data = pca.fit_transform(centers) #<1>

points = []
for i, cluster_name in enumerate(plot_data): #<2>
  plt.scatter(plot_data[i,0], plot_data[i,1], #<3>
              s=30, color="k")  (2)
  label = f"{i}_{"_".join(top_words(clusters[i], 2))}"
  points.append(plt.text(plot_data[i,0], #<3>
                         plot_data[i,1], #<3>
                         label, size=12)) #<3>
adjust_text(points, arrowprops=dict(arrowstyle="-", #<4>
                          color="gray", alpha=.3)) #<4>
plt.show()

#<1> Perform PCA to reduce to 2 dimensions for visualization
#<2> Loop through each cluster and plot it on the graph
#<3> Add a text label for each cluster with the cluster ID
#    and top-N words in each cluster
#<4> Display improvement: adjust the text labels to
#    minimize overlap
```



Mapping queries into clusters (multiple approaches)

Different approaches for mapping queries into clusters

```python
import sentence_transformer, heapq

def get_top_labels_centers(query, centers, n=2): #<4>
  q_emb = transformer.encode([query], convert_to_tensor=False)
  similarities = sentence_transformers.util.cos_sim(q_emb, centers)
  sim = similarities.tolist()[0]
  return [sim.index(i) for i in heapq.nlargest(n, sim)]

def get_query_cluster(query): #<5>
  q_emb = transformer.encode([query], convert_to_tensor=False)
  return algo.predict(q_emb)

def get_cluster_description(cluser_num):
  return "_".join(top_words(clusters[cluser_num], 5))

query = "microwave"
kmeans_predict = get_query_cluster(query)[0] #<1>
print("KMeans Predicted Cluster:")
print(f"    {kmeans_predict} ({get_cluster_description(kmeans_predict)})")

closest_sim = get_top_labels_centers(query, centers, 1)[0] #<2>
print(f"\nCosine Predicted Cluster:")
print(f"    {closest_sim} ({get_cluster_description(closest_sim)})")

knn_cosine_similarity = get_top_labels_centers(query, centers, 5) #<3>
print(f"\nKNN Cosine Predicted Clusters: {knn_cosine_similarity}")
for n in knn_cosine_similarity:
      print(f"    {n} ({get_cluster_description(n)})")

#<1> Option 1: Predict nearest cluster (KMeans)
#<2> Option 2: Find most-similar cluster (Cosine similarity)
#<3> Option 3 (recommended): Find N-most-similar clusters (Cosine similarity)
#<4> Get the top N clusters based on cosine similarity with the cluster centroids
#<5> Get the cluster based on the KMeans model's prediction
```

Results:

```
K-means Predicted Cluster:
  44 (Microwave_Cu._Ft._Stainless-Steel_Oven)

Cosine Predicted Cluster:
  44 (Microwave_Cu._Ft._Stainless-Steel_Oven)

KNN Cosine Predicted Clusters: [44, 52, 5, 83, 6]
  44 (Microwave_Cu._Ft._Stainless-Steel_Oven)
  52 (Stainless-Steel_30"_Black_Range_Cooktop)
  5 (KitchenAid_Black_White_Stand_Mixer)
  83 (Black_Coffeemaker_Maker_Coffee_Stainless-Steel)
  6 (Range_30"_Self-Cleaning_Freestanding_Stainless-Steel)
```

Approaches for integrating embedding-based personalization into search results

1. Perform a weighted average between the query vector (embedding for microwave) and the vectors for the user's previous interactions within the predicted clusters. This would generate a single vector representing a personalized version of the user's query, so all results would be personalized.

2. Perform a standard search, but then boost the results based on the average of the embeddings from a user's previous interactions within the predicted clusters. This would be a hybrid keyword and vector-based ranking function, where the keyword search would be the primary driver of the results, but the user's previous interactions would be used to boost related results higher.

Approaches for integrating embedding-based personalization into search results

3. Do one of the above, but then only personalize a few items in the search results instead of all the results. This follows a light-touch mentality so as not to disturb all of the user's search results, while still injecting novelty to enable the user to discover personalized items they may not have otherwise found.

4. Perform a standard search (keyword or vector), but then re-rank the results based on the weighted average between the query vector and the vectors for the user's previous interactions within the predicted clusters. This uses the original search to find the candidate results using the default relevance algorithm, but then those results are re-ranked to boost personalized preferences higher.

Approaches for integrating embedding-based personalization into search results

We'll do this one, because it's more generalizable across search engines for our example implementation. There are times when each of the approaches could make sense.

4. Perform a standard search (keyword or vector), but then re-rank the results based on the weighted average between the query vector and the vectors for the user's previous interactions within the predicted clusters. This uses the original search to find the candidate results using the default relevance algorithm, but then those results are re-ranked to boost personalized preferences higher.

Generating personalization vectors for a user's query

Generating a personalization vector for the user's query

```python
import pandas, numpy

def get_user_embeddings(products=[]): #<1>
  values = []
  for p in products:
    values.append([product_ids_emb[p],
                   top_clusters_for_embedding(product_ids_emb[p], 1)[0]])
  column_names = ["embedding", "cluster"]
  return pandas.DataFrame(data=numpy.array(values), index=products,
                          columns=column_names)

def get_personalization_vector(query=None, #<2>
                               user_items=[],
                              query_weight=1, #<3>
                     user_items_weights=[]): #<4>
  query_embedding = transformer.encode(query) if query else None

  if len(user_items) > 0 and len(user_items_weights) == 0: #<3>
    user_items_weights = numpy.full(shape=len(user_items),
                                    fill_value=1 / len(user_items))

  embeddings = []
  embedding_weights = []
  for weight in user_items_weights: #<4>
    embedding_weights.append(weight) #<4>
  for embedding in user_items:
    embeddings.append(embedding)
  if query_embedding.any():
    embedding_weights.append(query_weight) #<4>
    embeddings.append(query_embedding)

  return numpy.average(embeddings, weights=numpy.array(embedding_weights),
                       axis=0).astype("double") if len(embeddings) else None

#<1> Returns a dataframe with the embedding and guardrail cluster for each product
#<2> Returns a vector that combines (weighted average) an embedding for the query
#    with the embeddings for the passed-in user_items
#<3> By default, the weight is split 1:1 (50% each) between the query embedding
#    and the user_items_weight
#<4> You can optionally specify a query_weight and user_item_weights to influence
#    how much each embedding influences the personalization vector
```

Generating contextual user personalization vectors

Contextual (filtered) vs. Non-contextual (unfiltered) personalization vectors

```python
product_interests = ["7610465823828", #hello kitty water bottle
                     "36725569478"] #stainless steel electric range

user_embeddings = get_user_embeddings(product_interests)
query = "microwave"

unfiltered_personalization_vector = get_personalization_vector(query=query,
  user_items=user_embeddings['embedding'].to_numpy()) #<1>
print("\nPersonalization Vector (No Cluster Guardrails):")
print(format_vector(unfiltered_personalization_vector))

query_clusters = get_top_labels_centers(query,
                                centers, n=5) #<2>
print("\nQuery Clusters ('microwave'):\n" + str(query_clusters))

clustered = user_embeddings.cluster.isin(query_clusters) #<3>
products_in_cluster = user_embeddings[clustered] #<3>
print("\nProducts Filtered to Query Clusters:\n" + str(products_in_cluster))

filtered_personalization_vector = get_personalization_vector(query=query,
  user_items=filtered['embedding'].to_numpy()) #<4>
print("\nFiltered Personalization Vector (With Cluster Guardrails):")
print(format_vector(filtered_personalization_vector))

#<1> Personalization vector with no guardrails
#    (uses query and all past item interactions)
#<2> Get the top 5 clusters for the query to use as guardrails
#<3> Filter down to only items in the guardrail query clusters
#<4> Generate a personalization vector with guardrails
#    (uses query and only items related to the query)
```

Results:

```
Products Interactions for Personalization:
product        embedding                                                cluster
7610465823828  [0.06417941, 0.04178553, -0.0017139615, -0.020...    1
36725569478    [0.0055417763, -0.024302201, -0.024139373, -0....    6

Personalization Vector (No Cluster Guardrails):
[0.016, -0.006, -0.02, -0.032, -0.016, 0.008, -0.0, 0.017, 0.011, 0.007 ...]

Query Clusters ('microwave'):
[44, 52, 5, 83, 6]

Products Filtered to Query Clusters:
product      embedding                                                cluster
36725569478  [0.0055417763, -0.024302201, -0.024139373, -0....     6

Filtered Personalization Vector (With Cluster Guardrails):
[0.002, -0.023, -0.026, -0.037, -0.025, 0.002, -0.009, 0.007, 0.033, -0 ...]
```

Run different personalization scenarios

Run Different Personalization Scenarios

```
def rerank_with_personalization(docs, personalization_vector):
  result_embeddings = numpy.array([product_ids_emb[docs[x]["upc"]]
                                   for x in range(len(docs))]).astype(float)
  similarities = sentence_transformers.util.cos_sim(
      personalization_vector, result_embeddings).tolist()[0]
  reranked = [similarities.index(i)
              for i in heapq.nlargest(len(similarities), similarities)]
  reranked, v = zip(sorted(enumerate(similarities),
                           key=itemgetter(1), reverse=True))
  return [docs[i] for i in reranked]

query = "microwave"
request = {"query": query,
           "query_fields": ["name", "manufacturer"],
           "return_fields": ["upc", "name", "manufacturer", "score"],
           "limit": 100,
           "order_by": [("score", "desc"), ("upc", "asc")]}

response = products_collection.search(*request)
docs = response["docs"]
print("No Personalization:")
display_product_search(query, docs[0:4])

print("Global Personalization (no category guardrails):")
reranked_seach_results_no_guardrails = \
  rerank_with_personalization(docs,
    unfiltered_personalization_vector)
display_product_search(query, reranked_seach_results_no_guardrails[0:4])

print("Contextual Personalization (with category guardrails):")
reranked_seach_results_with_guardrails = \
  rerank_with_personalization(docs,
    filtered_personalization_vector)
display_product_search(query, reranked_seach_results_with_guardrails[0:4])
```

No Personalization

microwave Search

Name: LG - Trim Kit for Select LG Microwave Ovens - Stainless-Steel
Manufacturer: LG

Name: GE - Profile Advantium 1.7 Cu. Ft. Over-the-Range Microwave - Stainless-Steel/Black
Manufacturer: GE

Name: LG - 1.7 Cu. Ft. Over-the-Range Microwave - Stainless-Steel
Manufacturer: LG

Name: GE - 1.1 Cu. Ft. Mid-Size Microwave - Black
Manufacturer: GE

Naive Personalization (No Guardrails)

microwave Search

Name: Hello Kitty - 0.7 Cu. Ft. Compact Microwave
Manufacturer: Hello Kitty

Name: Panasonic - 1.2 Cu. Ft. Mid-Size Microwave - Stainless-Steel
Manufacturer: Panasonic

Name: Panasonic - 1.2 Cu. Ft. Mid-Size Microwave - Stainless-Steel
Manufacturer: Panasonic

Name: Panasonic - 1.2 Cu. Ft. Mid-Size Microwave - White
Manufacturer: Panasonic

Contextual Personalization (with Guardrails)

microwave Search

Name: Samsung - 1.8 Cu. Ft. Over-the-Range Microwave - Stainless-Steel
Manufacturer: Samsung

Name: Samsung - 1.8 Cu. Ft. Over-the-Range Microwave - Stainless-Steel
Manufacturer: Samsung

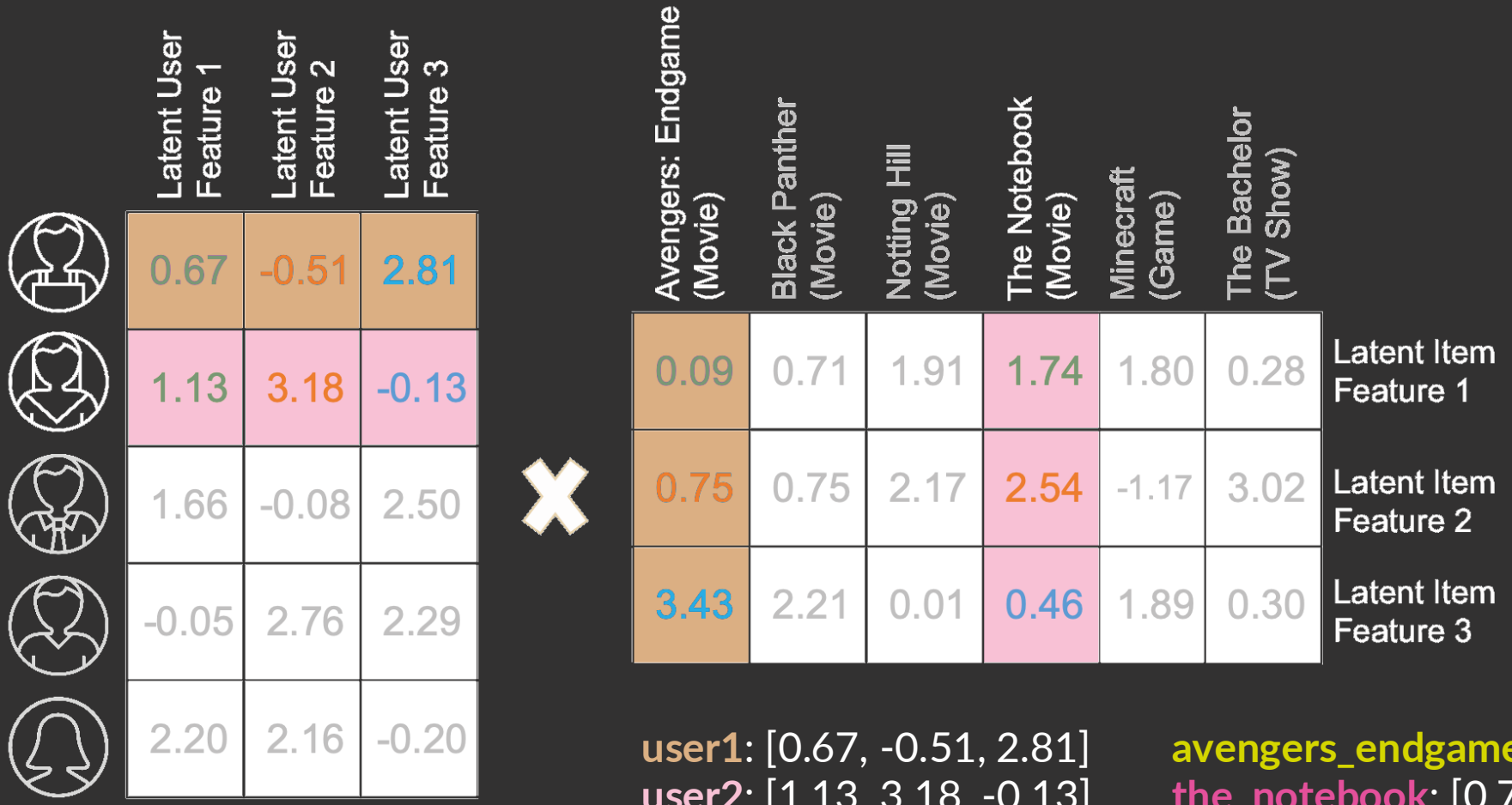
Name: Samsung - 1.6 Cu. Ft. Over-the-Range Microwave - Stainless-Steel
Manufacturer: Samsung

Name: Samsung - 1.7 Cu. Ft. Over-the-Range Microwave - Stainless-Steel
Manufacturer: Samsung

Avoiding the pitfalls of personalized search

1. Account for the Cold Start Problem
2. Guardrails are important
3. Overpersonalization is frustrating!
4. Feedback loops are critical
5. Privacy can be a concern
6. You should apply personalization with a light touch

Multimodal personalization: given latent behavior embeddings and conten based embeddings, we can integrate the dimensions from both to discover new insights and enhance relevance



user1: [0.67, -0.51, 2.81]
user2: [1.13, 3.18, -0.13]
...

avengers_endgame: [0.09, 0.75, 3.43]
the_notebook: [0.71, 0.75, 2.21]
...

Multimodal Vector Search

Image Encoder Layer

[0.19, 82.3, 0.02, 0.0, 0.0, 99.1, ...],

Text Encoder Layer

[0.00, 1.3, 26.9, 0.23, 0.0, 1.3, ...]]

... "but I like to be here. Oh, I like it a lot!" said the Cat in the Hat to the fish in the pot...

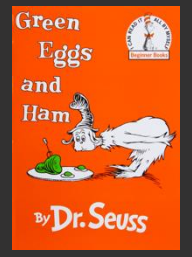
Concatenate

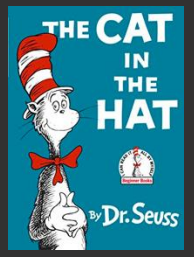
[0.00, 1.3, 26.9, 0.23, 0.0, 1.3, ..., 82.3, 0.02, 0.0, 0.0, 99.1, ..., 0.00, 1.3, 26.9, 0.23, 0.0, 1.3, ...]

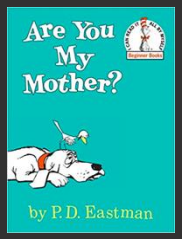
Multimodal Encoder / Dimensionality Reduction

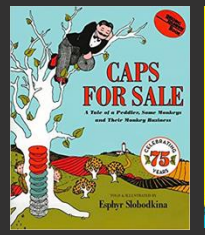
[[0.00, 1.3, 26.9, 0.23, 0.0, 1.3, ...],

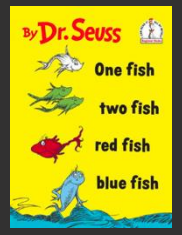
Collaborative User Signals Encoder Layer

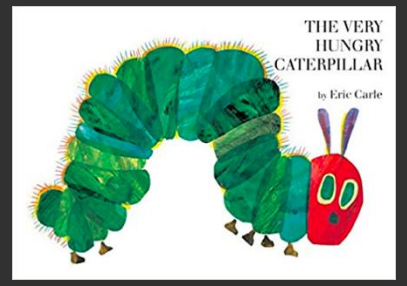




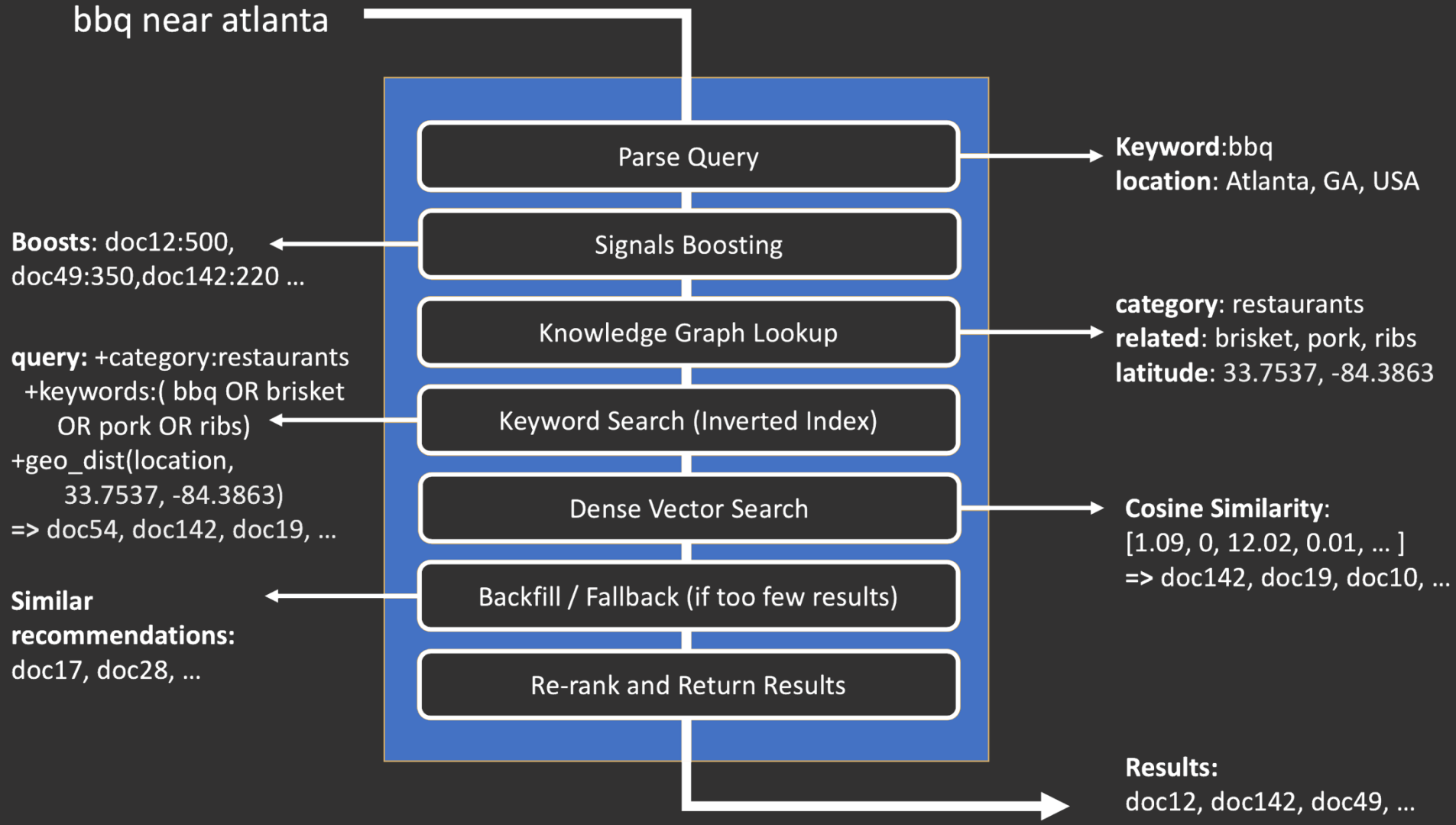




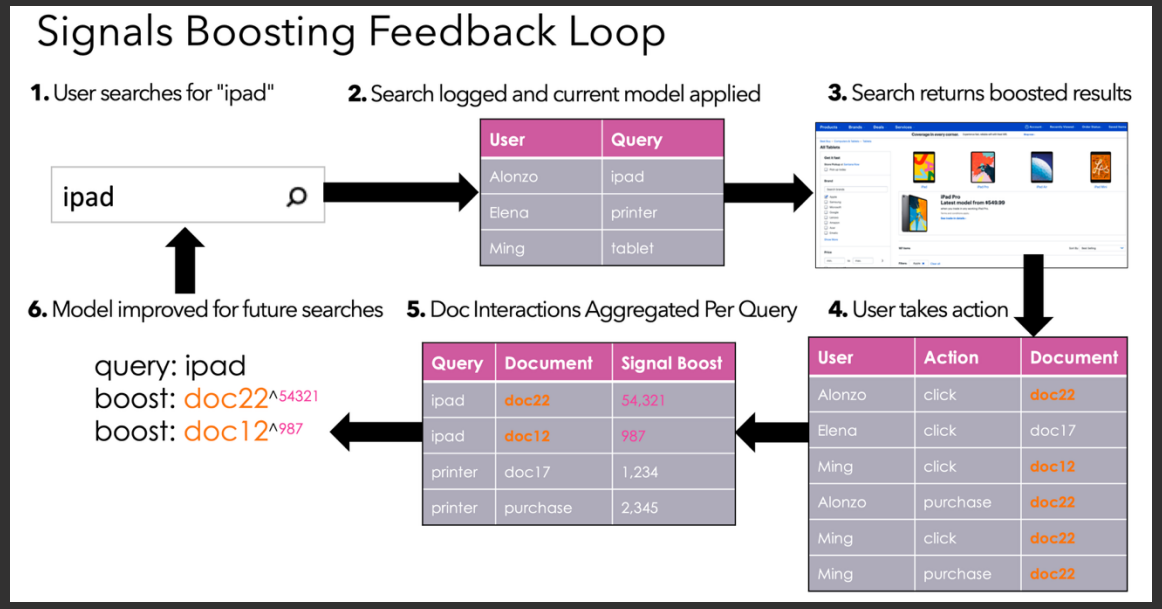


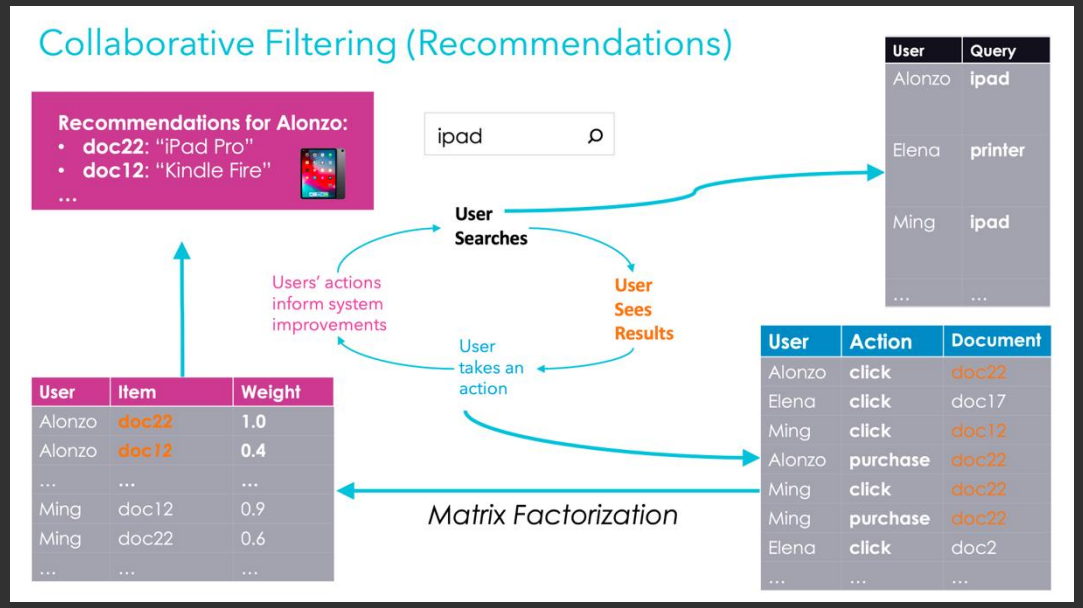


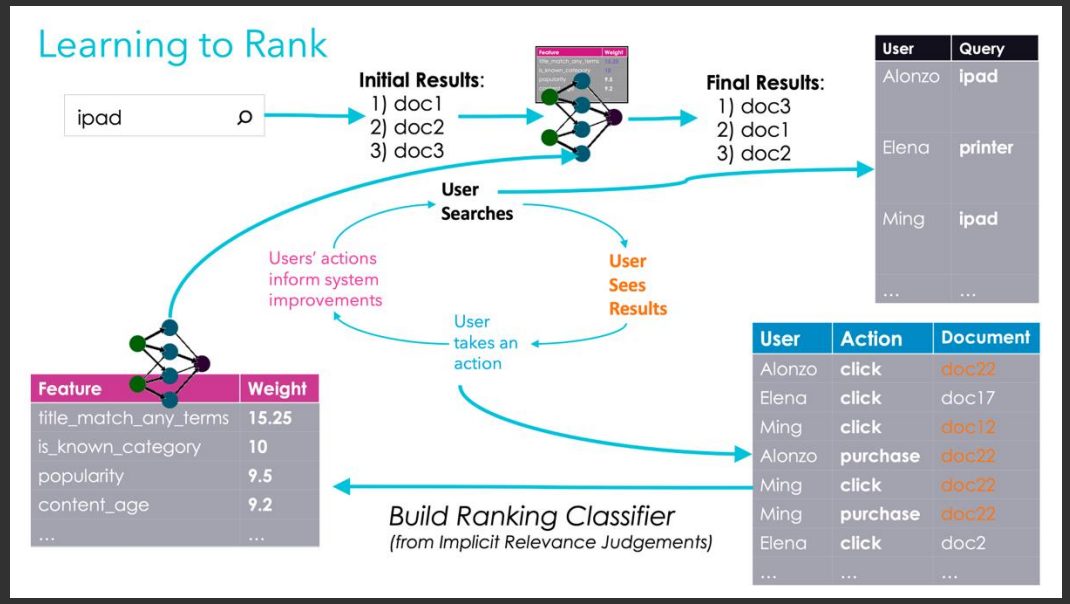
It's also powerful to use many *query modalities* to optimize search quality

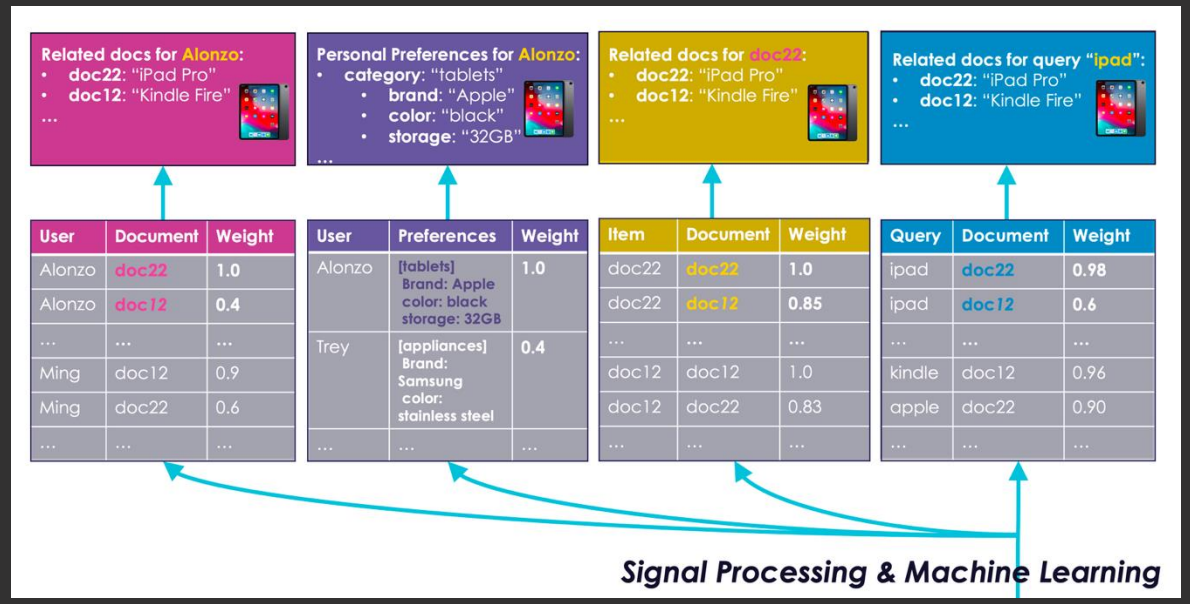


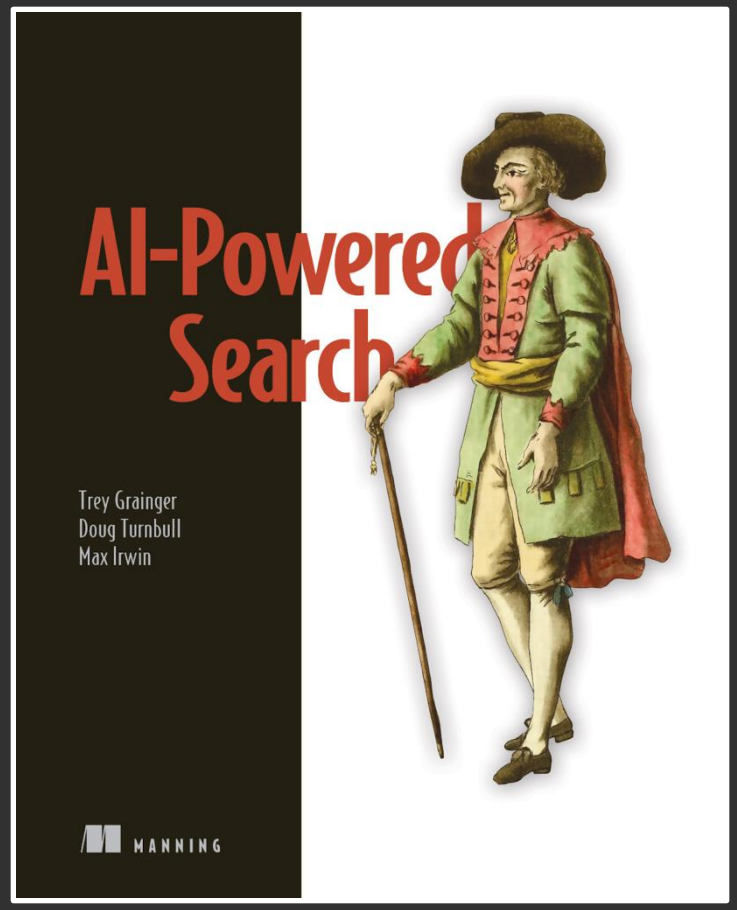
Reflected Intelligence: So many ways to integrate user signals…











(45% Discount Code: mices45)

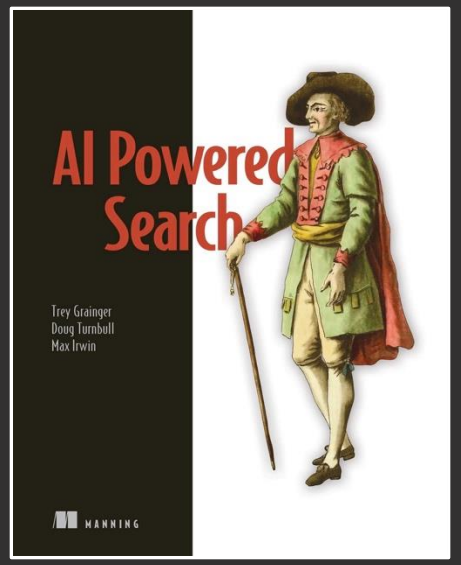
AI-Powered Search

- RAG (Retrieval Augmented Generation)
- Generative Search & Summarization
- Learning to Rank & implicit judgments
- Semantic Search
- Dense Vector Search
- Fine-tuning LLMs for Search
- Personalized Search & Recommendations
- Knowledge Graph Learning
- User signals boosting & click models
- Crowdsourced Relevance
- Quantization techniques
- Hybrid search
- Multimodal search

Get a copy @ *http://aiPoweredSearch.com*

Thank You!

http://**aiPoweredSearch**.com



(45% Discount Code: **mices55**)

Trey Grainger

trey@searchkernel.com
@treygrainger

Other presentations:

http://treygrainger.com

Founder / CTO

Searchkernel

Technical Advisor:

OpenSource Connections

Continue the Conversation

Join the AI-Powered Search Community

aipoweredsearch.com/community