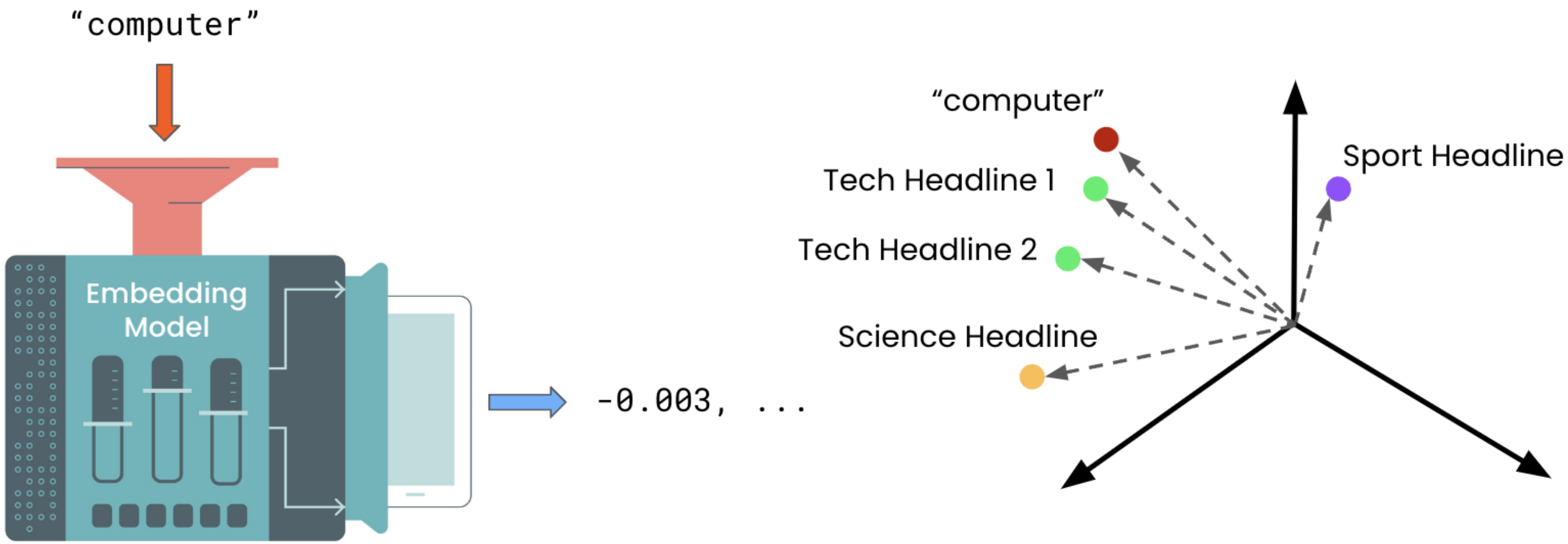
Semantic search and enriched embeddings

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API

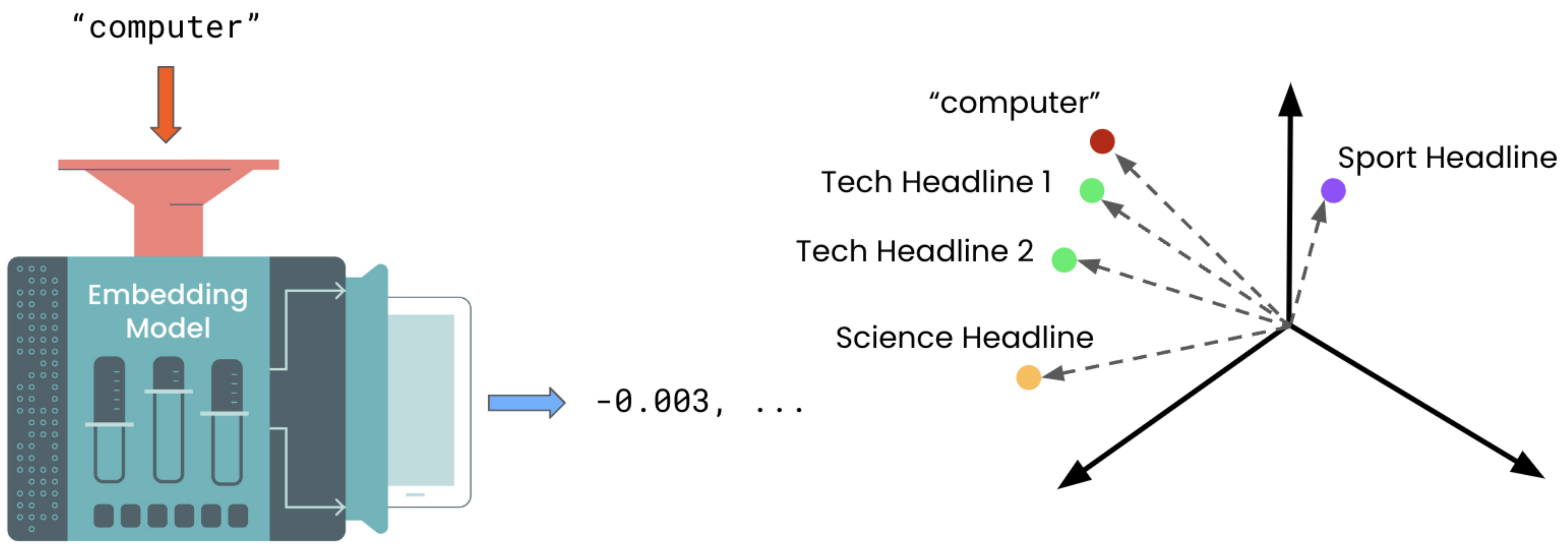
Emmanuel Pire
Senior Software Engineer, DataCamp

Semantic search

- Use embeddings to return *most similar* results to a search query
- **Example:** Semantic search for online news website



Semantic search



1. **Embed** the search query and other texts
2. Compute the **cosine distances**
3. **Extract** the texts with the *smallest* cosine distance

Enriched embeddings

```
articles = [
    {"headline": "Economic Growth Continues Amid Global Uncertainty",
     "topic": "Business",
     "keywords": ["economy", "business", "finance"]},
    ...
    {"headline": "1.5 Billion Tune-in to the World Cup Final",
     "topic": "Sport",
     "keywords": ["soccer", "world cup", "tv"]}
]
```

```
Headline: Economic Growth Continues Amid Global Uncertainty
Topic: Business
Keywords: economy, business, finance
```

Combining features with F-strings

```python
articles = [..., {"headline": "1.5 Billion Tune-in to the World Cup ",
                  "topic": "Sport",
                  "keywords": ["soccer", "world cup", "tv"]}]

def create_article_text(article):
  return f"""Headline: {article['headline']}
Topic: {article['topic']}
Keywords: {', '.join(article['keywords'])}"""

print(create_article_text(articles[-1]))
```

```
Headline: 1.5 Billion Tune-in to the World Cup Final
Topic: Sport
Keywords: soccer, world cup, tv
```

Creating enriched embeddings

```python
article_texts = [create_article_text(article) for article in articles]

article_embeddings = create_embeddings(article_texts)
print(article_embeddings)
```

```
[[-0.019609929993748665, -0.03331860154867172, ...],
 ...,
 [..., -0.014373429119586945, -0.005235843360424042]]
```

Computing distances

```
from scipy.spatial import distance

def find_n_closest(query_vector, embeddings, n=3):
  distances = []
  for index, embedding in enumerate(embeddings):
    dist = distance.cosine(query_vector, embedding)
    distances.append({"distance": dist, "index": index})
  distances_sorted = sorted(distances, key=lambda x: x["distance"])
  return distances_sorted[0:n]
```

Returning the search results

```
query_text = "AI"
query_vector = create_embeddings(query_text)[0]

hits = find_n_closest(query_vector, article_embeddings)

for hit in hits:
  article = articles[hit['index']]
  print(article['headline'])
```

```
Tech Giant Buys 49% Stake In AI Startup
Tech Company Launches Innovative Product to Improve Online Accessibility
India Successfully Lands Near Moon's South Pole
```

Let's practice!

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API

Recommendation systems

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API

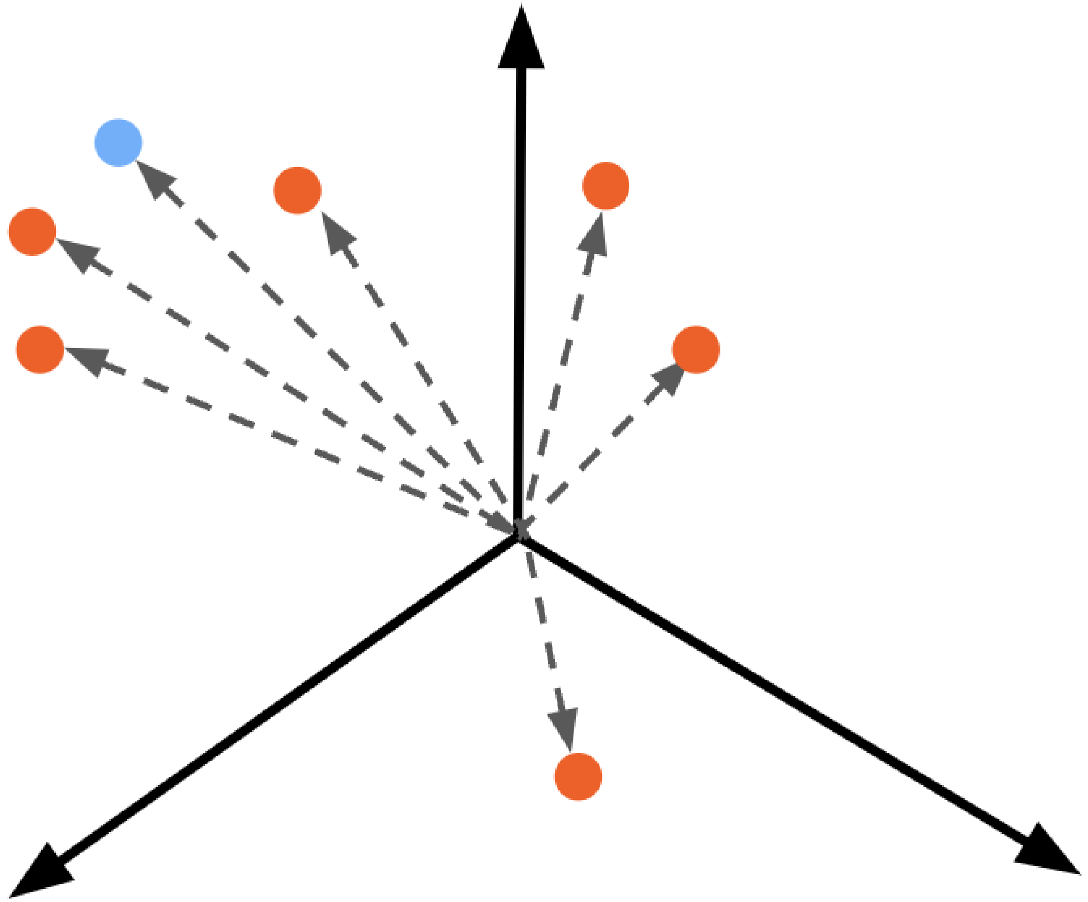
Emmanuel Pire
Senior Software Engineer, DataCamp

Recommendation systems with embeddings

- Very similar to semantic search!

Process:

1. Embed the potential recommendations and data point

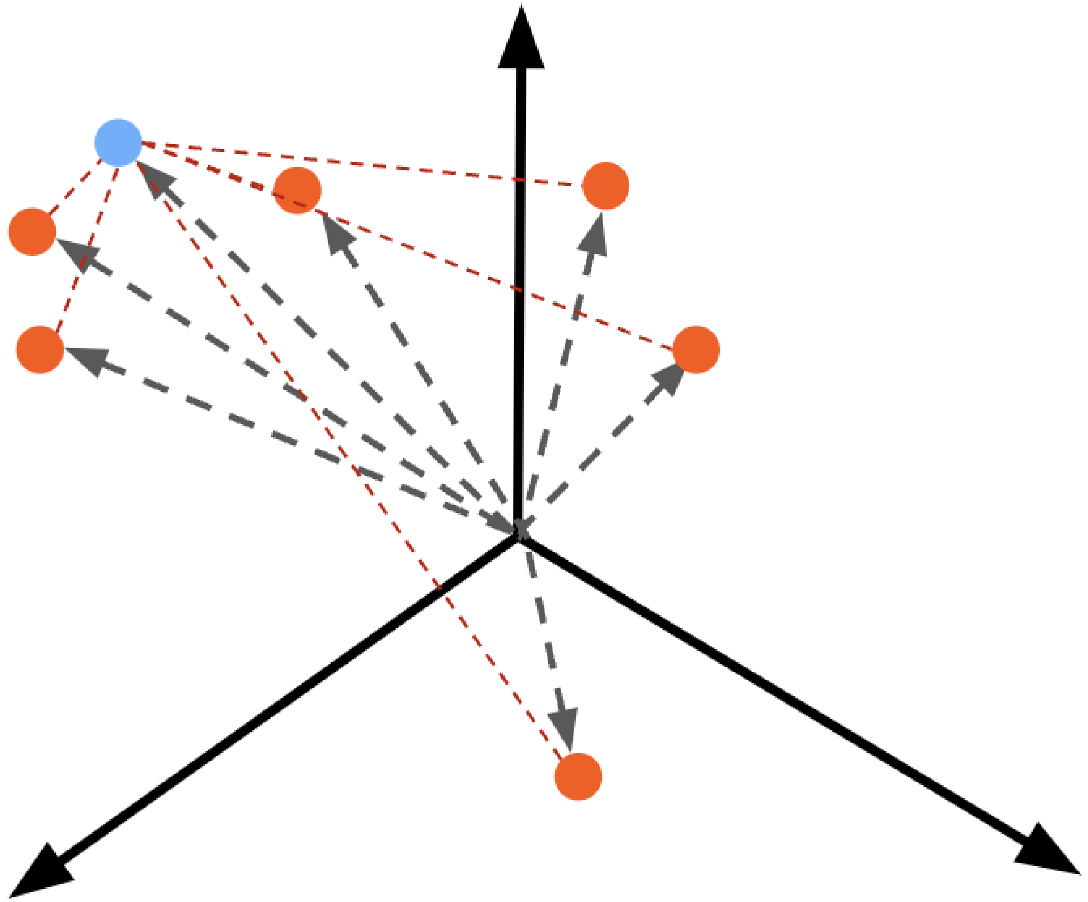


Recommendation systems with embeddings

- Very similar to semantic search!

Process:

1. Embed the potential recommendations and data point
2. Calculate cosine distances

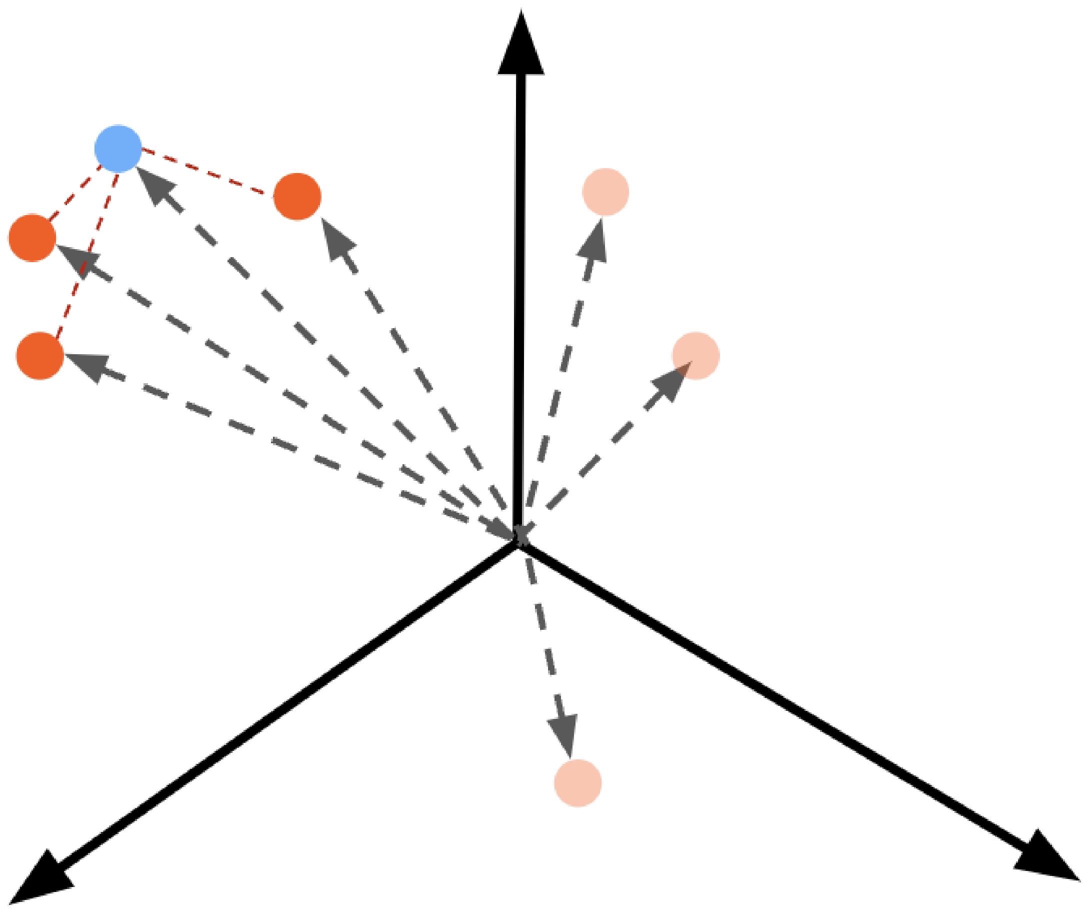


Recommendation systems with embeddings

- Very similar to semantic search!

Process:

1. Embed the potential recommendations and data point
2. Calculate cosine distances
3. Recommend closest items



Example: Recommended articles

```
articles = [
    {"headline": "Economic Growth Continues Amid Global Uncertainty",
     "topic": "Business",
     "keywords": ["economy", "business", "finance"]},
    ...
    {"headline": "1.5 Billion Tune-in to the World Cup Final",
     "topic": "Sport",
     "keywords": ["soccer", "world cup", "tv"]}
]

current_article = {"headline": "How NVIDIA GPUs Could Decide Who Wins the AI Race",
                   "topic": "Tech",
                   "keywords": ["ai", "business", "computers"]}
```

Combining features

```python
def create_article_text(article):
  return f"""Headline: {article['headline']}
Topic: {article['topic']}
Keywords: {', '.join(article['keywords'])}"""
```

```python
article_texts = [create_article_text(article) for article in articles]
current_article_text = create_article_text(current_article)
print(current_article_text)
```

```
Headline: How NVIDIA GPUs Could Decide Who Wins the AI Race
Topic: Tech
Keywords: ai, business, computers
```

Creating Embeddings

```
def create_embeddings(texts):
  response = openai.Embedding.create(
    model="text-embedding-ada-002",
    input=texts
  )
  response_dict = response.model_dump()

  return [data['embedding'] for data in response_dict['data']]
```

```
current_article_embeddings = create_embeddings(current_article_text)[0]
article_embeddings = create_embeddings(article_texts)
```

Finding the most similar article

```
def find_n_closest(query_vector, embeddings, n=3):
  distances = []
  for index, embedding in enumerate(embeddings):
    dist = spatial.distance.cosine(query_vector, embedding)
    distances.append({"distance": dist, "index": index})
  distances_sorted = sorted(distances, key=lambda x: x["distance"])
  return distances_sorted[0:n]
```

```
hits = find_n_closest(current_article_embeddings, article_embeddings)

for hit in hits:
  article = articles[hit['index']]
  print(article['headline'])
```

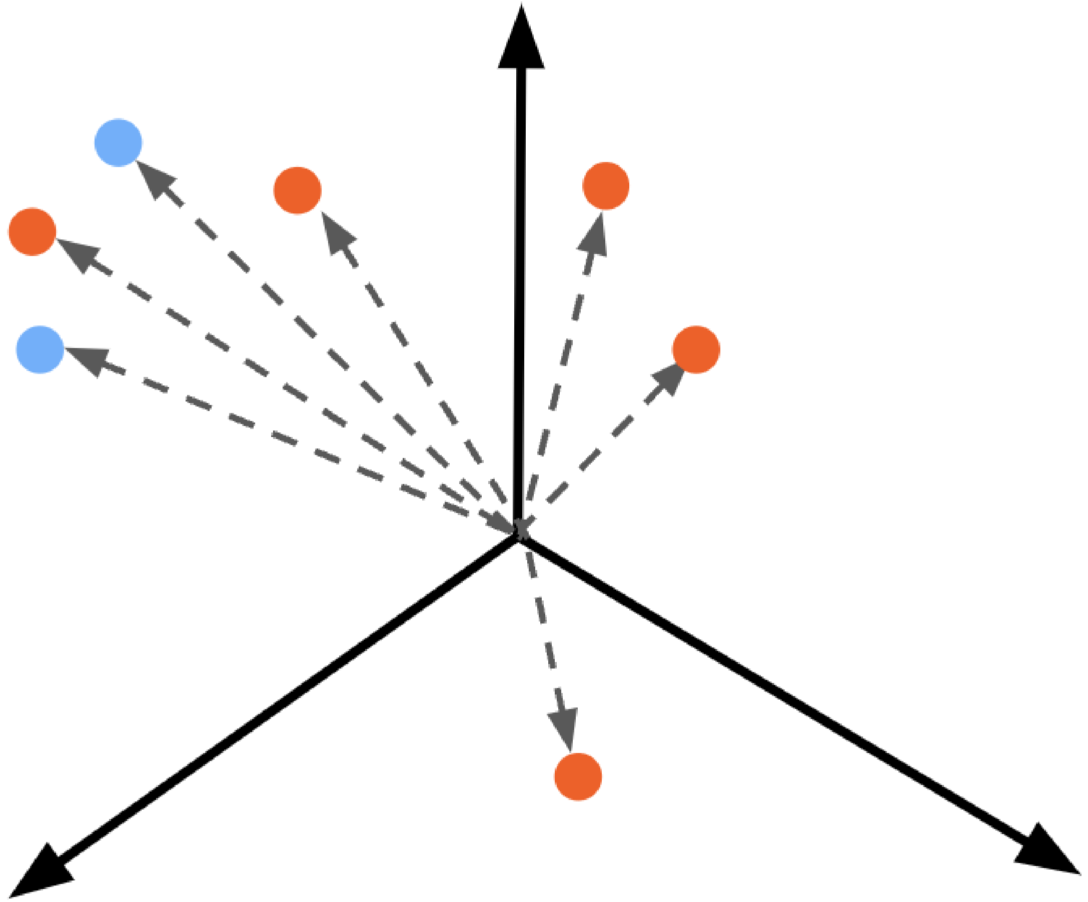
Finding the most similar article

```
Tech Giant Buys 49% Stake In AI Startup
Tech Company Launches Innovative Product to Improve Online Accessibility
Scientists Make Breakthrough Discovery in Renewable Energy
```

Adding user history

```
user_history = [
    {"headline": "How NVIDIA GPUs Could Decide Who Wins the AI Race",
     "topic": "Tech",
     "keywords": ["ai", "business", "computers"]},
    {"headline": "Tech Giant Buys 49% Stake In AI Startup",
     "topic": "Tech",
     "keywords": ["business", "AI"]}
]
```

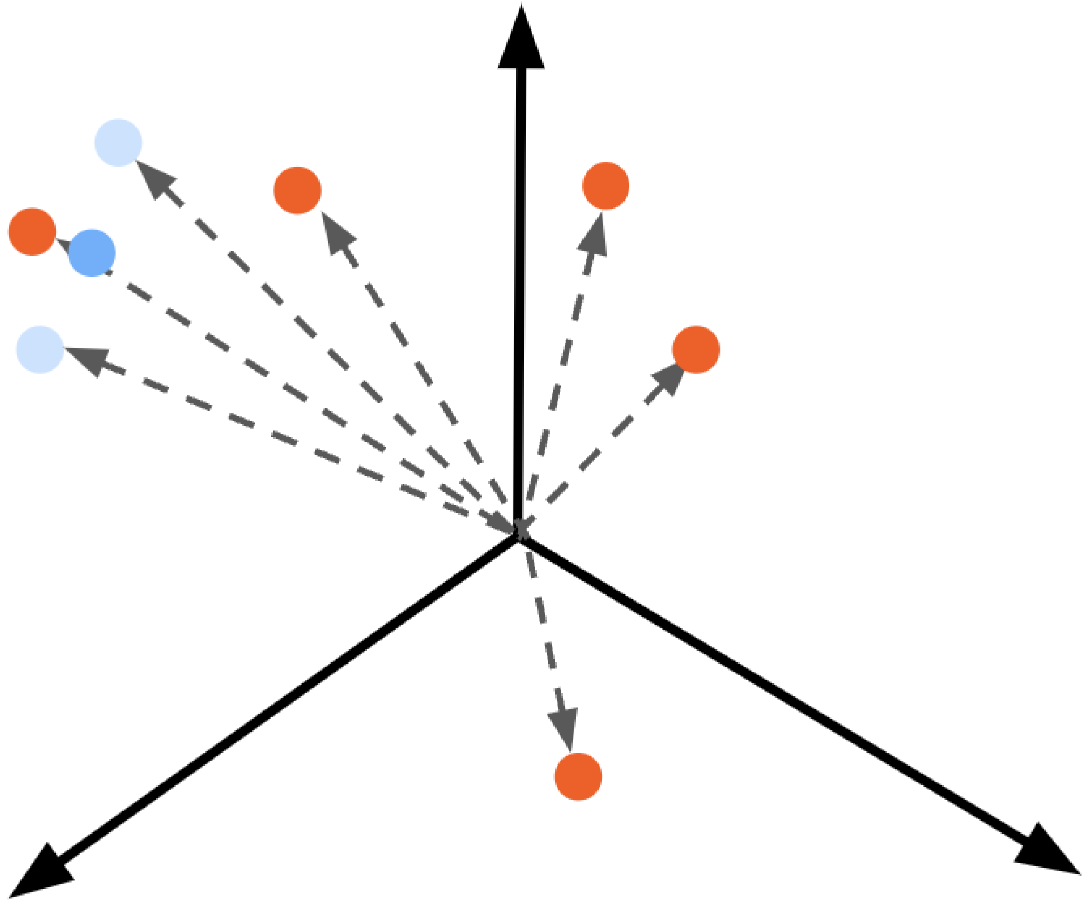
Recommendations on multiple data points



Recommendations on multiple data points

Process:

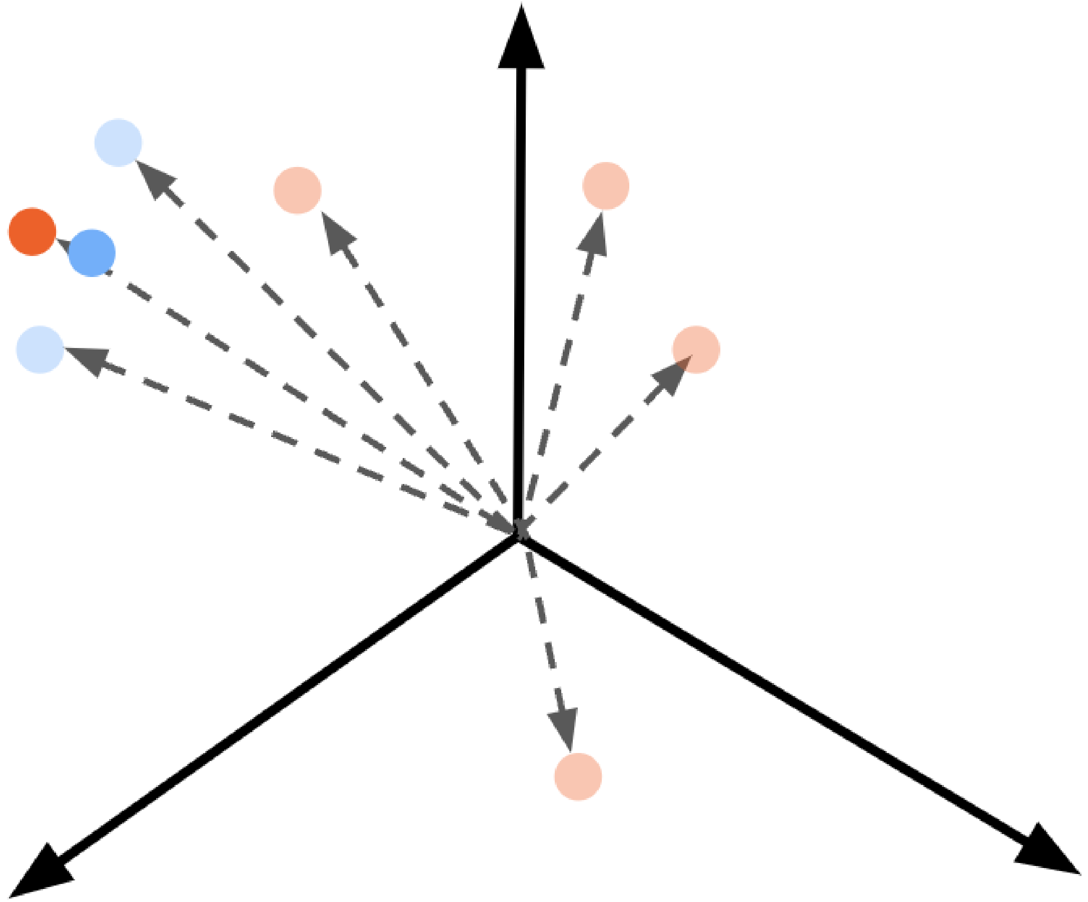
- Combine multiple vectors into one by taking the **mean**
- Compute cosine distances



Recommendations on multiple data points

Process:

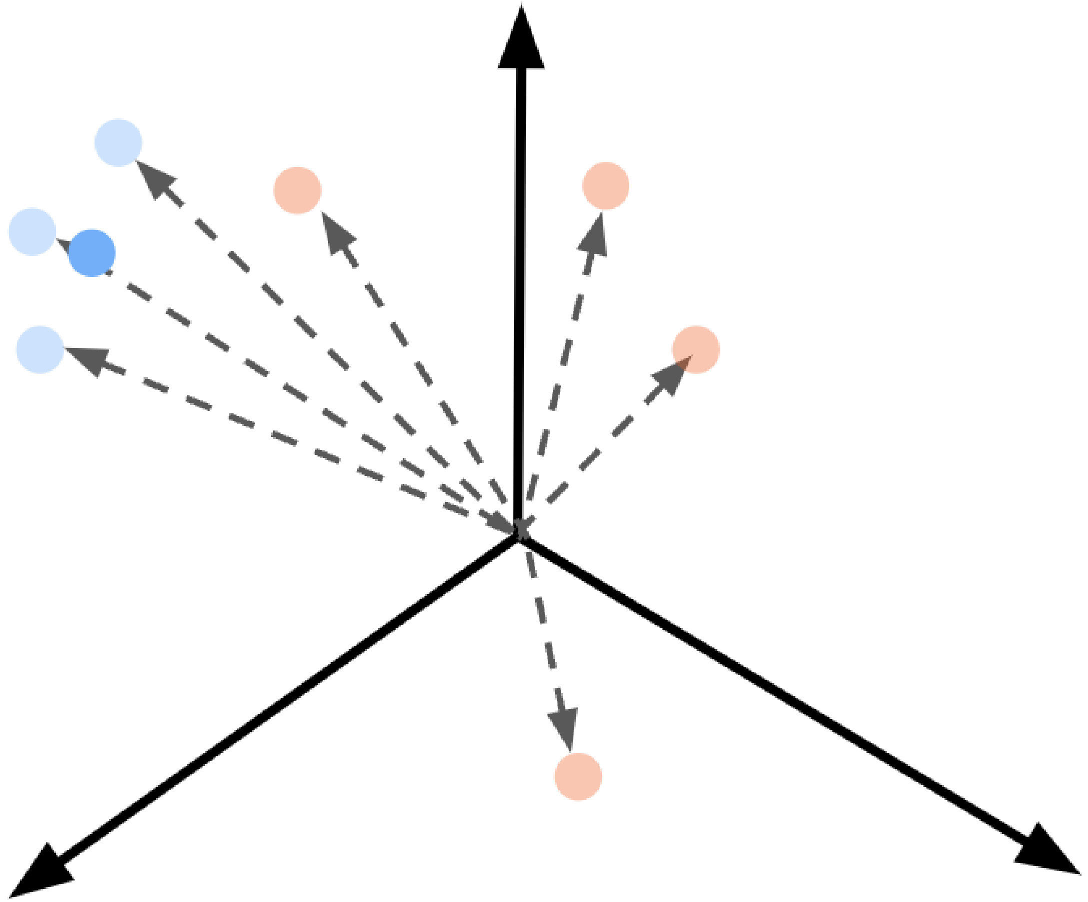
- Combine multiple vectors into one by taking the **mean**
- Compute cosine distances
- Recommend closest vector



Recommendations on multiple data points

Process:

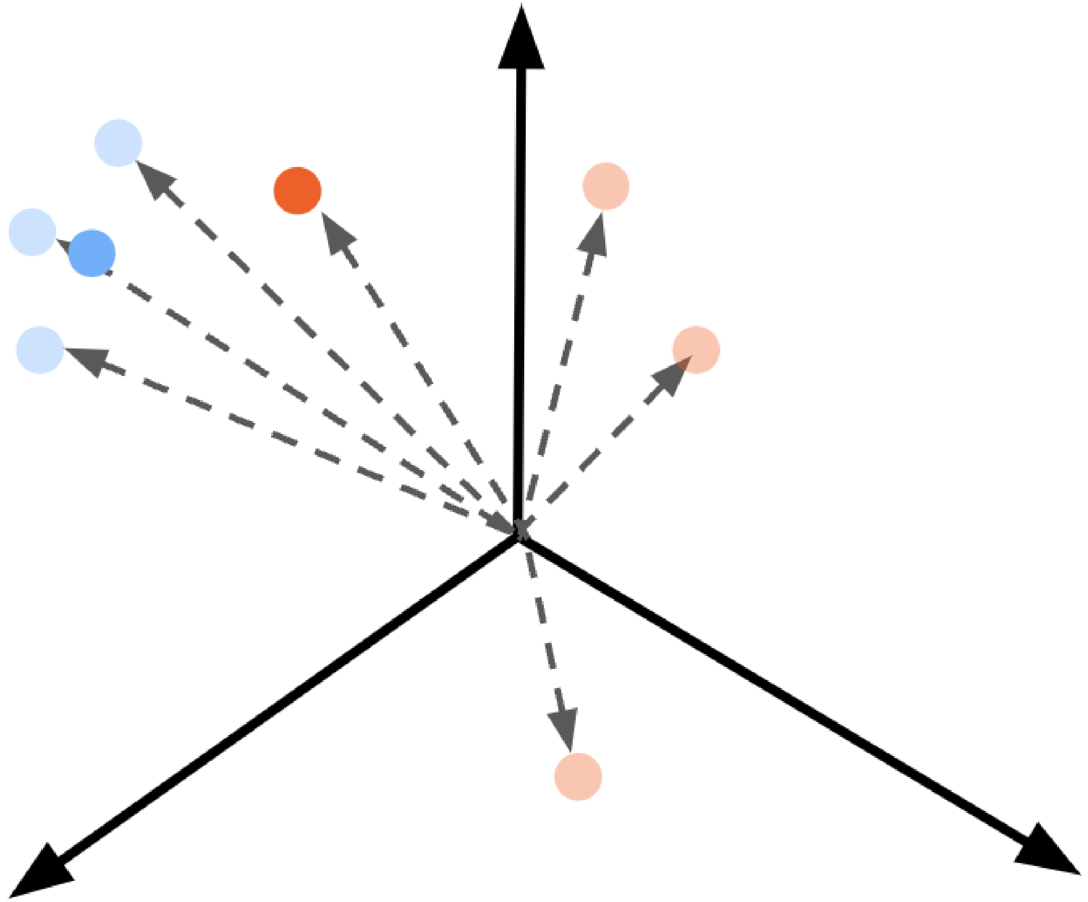
- Combine multiple vectors into one by taking the **mean**
- Compute cosine distances
- Recommend closest vector



Recommendations on multiple data points

Process:

- Combine multiple vectors into one by taking the **mean**
- Compute cosine distances
- Recommend closest vector
  - Ensure that it's unread



Recommendations on multiple data points

```
def create_article_text(article):
  return f"""Headline: {article['headline']}
Topic: {article['topic']}
Keywords: {', '.join(article['keywords'])}"""

history_texts = [create_article_text(article) for article in user_history]
history_embeddings = create_embeddings(history_texts)
mean_history_embeddings = np.mean(history_embeddings, axis=0)

articles_filtered = [article for article in articles if article not in user_history]
article_texts = [create_article_text(article) for article in articles_filtered]
article_embeddings = create_embeddings(article_texts)
```

Recommendations on multiple data points

```python
hits = find_n_closest(mean_history_embeddings, article_embeddings)

for hit in hits:
  article = articles_filtered[hit['index']]
  print(article['headline'])
```

```
Tech Company Launches Innovative Product to Improve Online Accessibility
New Social Media Platform Has Everyone Talking!
Scientists Make Breakthrough Discovery in Renewable Energy
```

Let's practice!

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API

Embeddings for classification tasks

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API

Emmanuel Pire
Senior Software Engineer, DataCamp

Classification tasks

Assigning **labels** to items

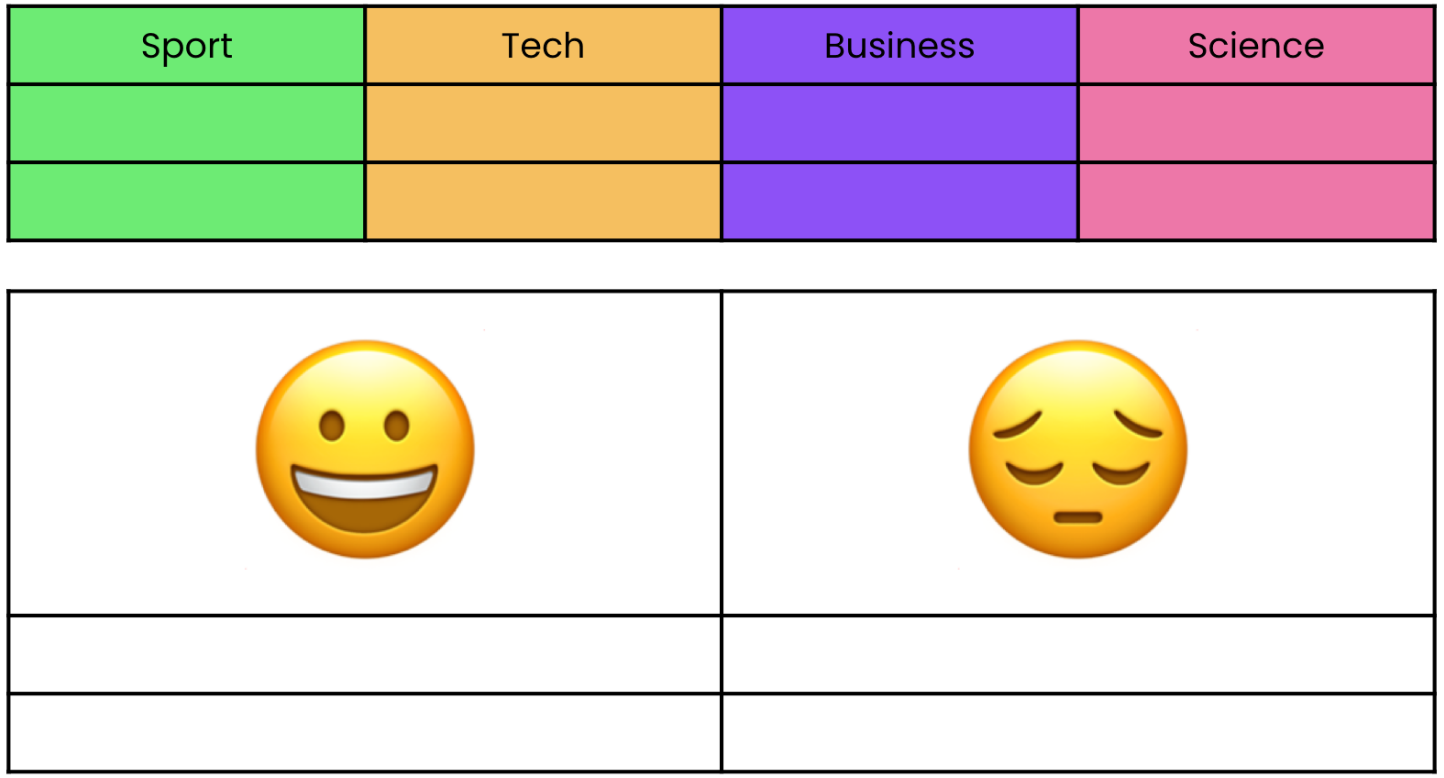
- Categorization
  - **Example:** headlines into topics
- Sentiment analysis

Sport	Tech	Business	Science

Classification tasks

Assigning **labels** to items

- Categorization
  - **Example:** headlines into topics
- Sentiment analysis
  - **Example:** Classifying reviews as positive or negative



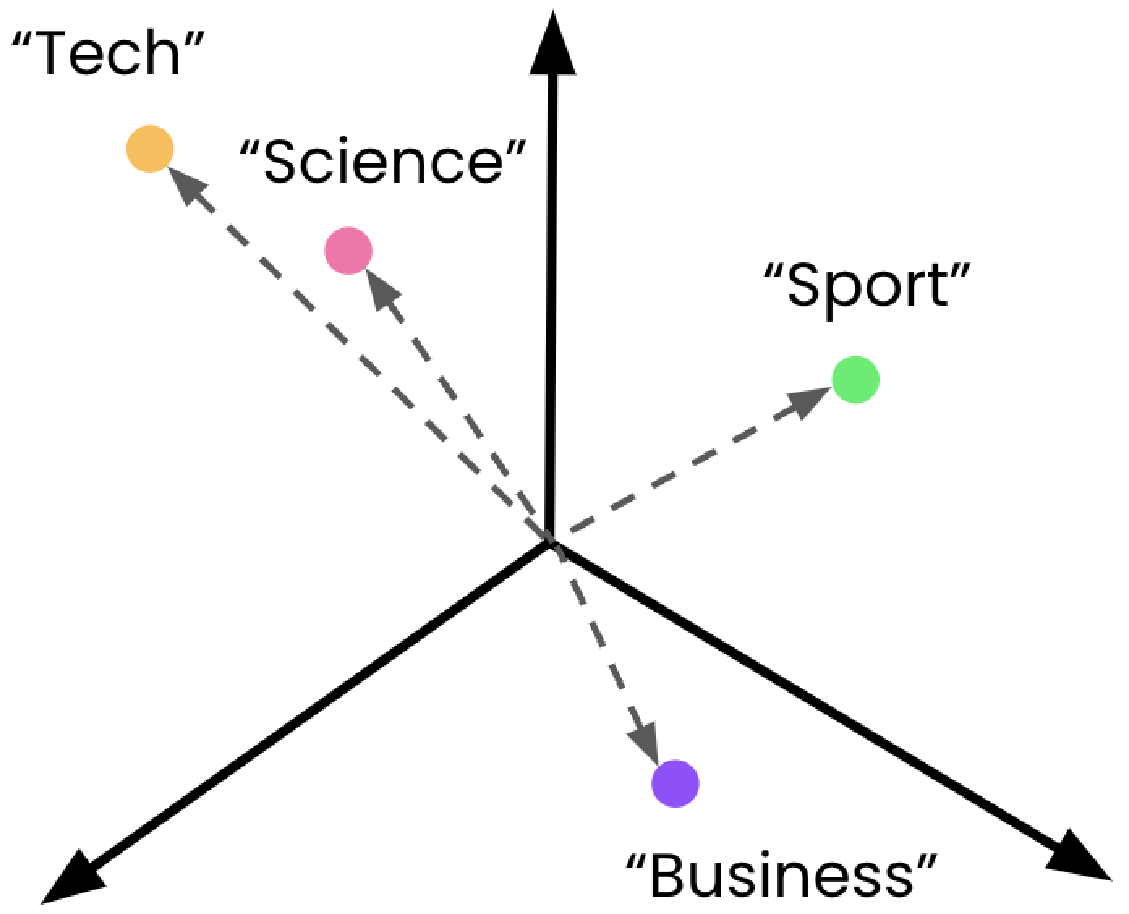
Embeddings capture *semantic* meaning

Classification with embeddings

- **Zero-shot classification:**
  - Not using labeled data

Process:

1. Embed class descriptions

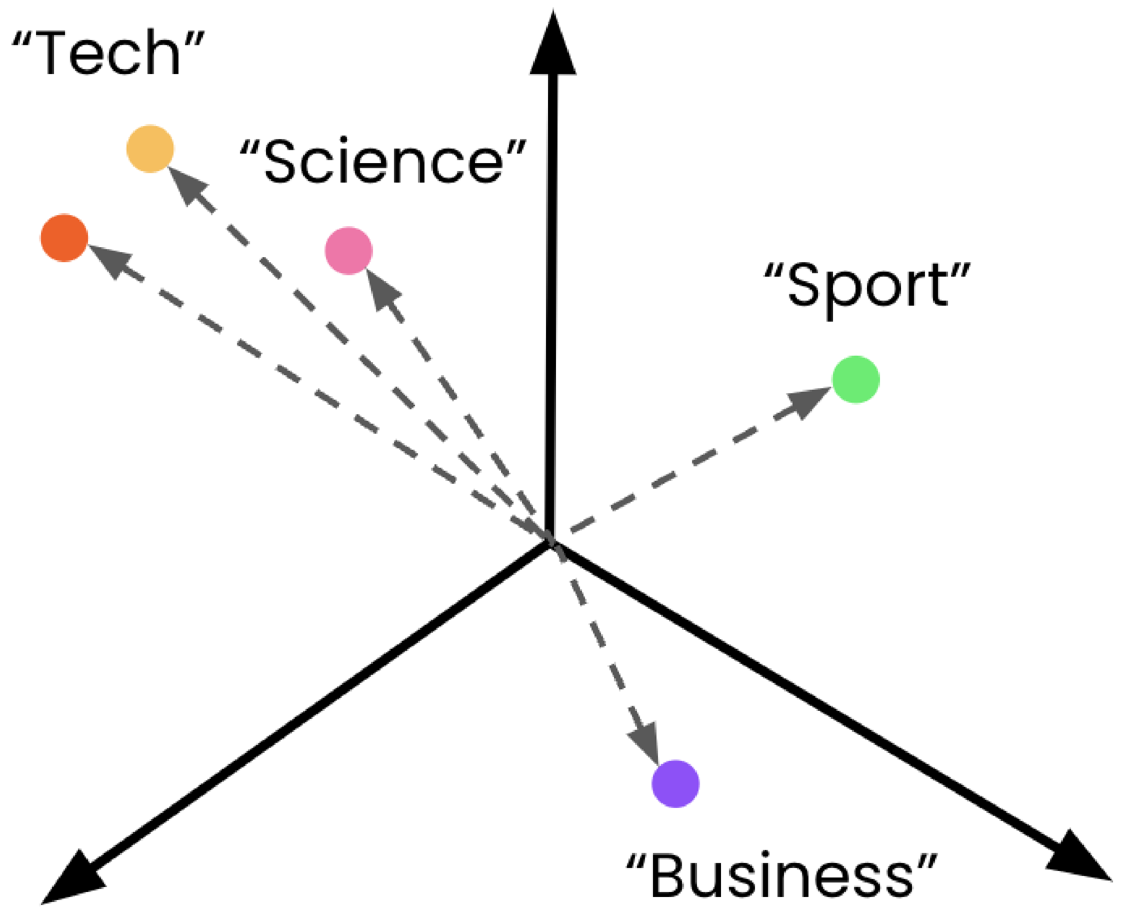


Classification with embeddings

- **Zero-shot classification:**
  - Not using labeled data

Process:

1. Embed class descriptions
2. Embed the item to classify
3. Compute cosine distances

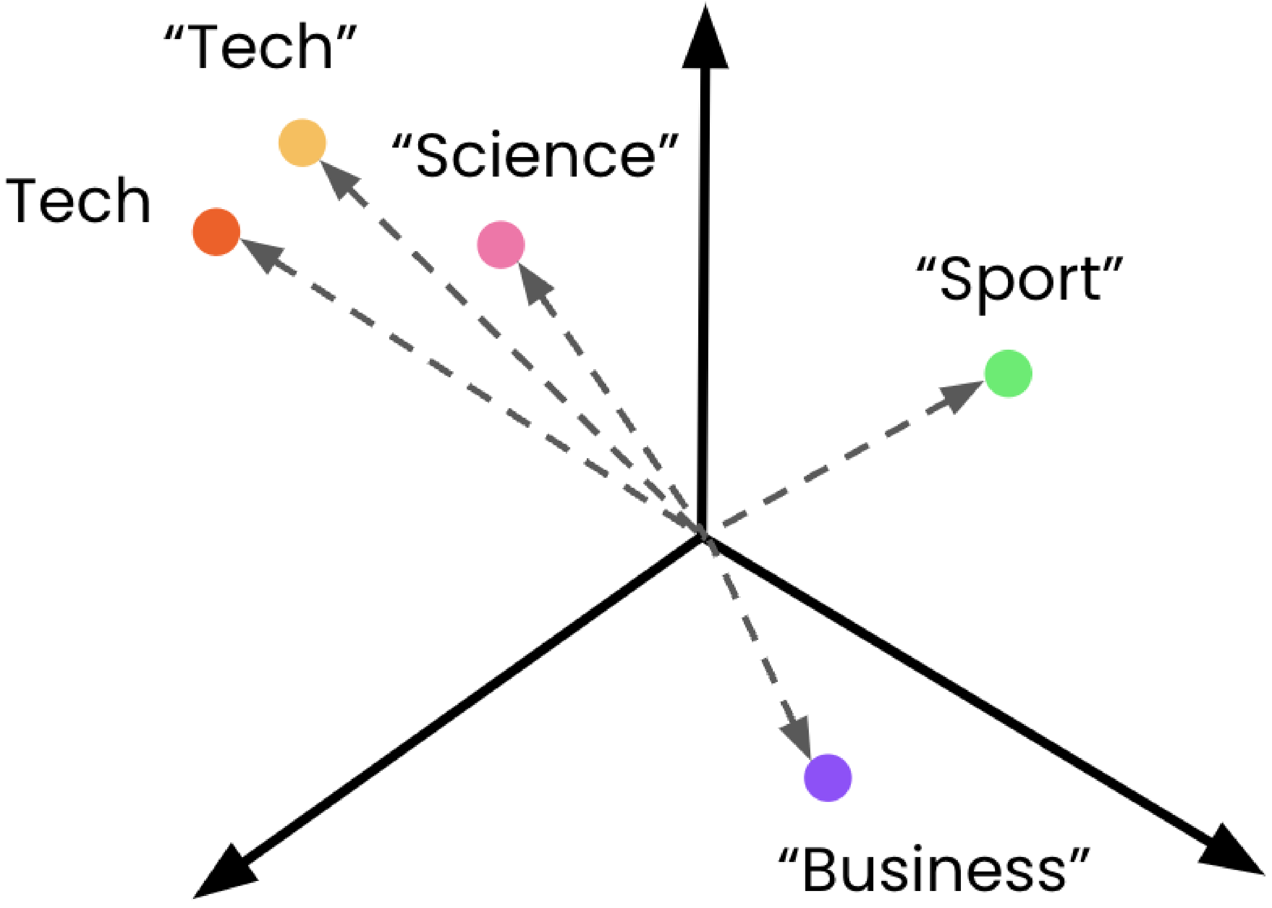


Classification with embeddings

- **Zero-shot classification:**
  - Not using labeled data

Process:

1. Embed class descriptions
2. Embed the item to classify
3. Compute cosine distances
4. Assign the most similar label



Embedding class descriptions

```
topics = [
  {'label': 'Tech'},
  {'label': 'Science'},
  {'label': 'Sport'},
  {'label': 'Business'},
]

class_descriptions = [topic['label'] for topic in topics]
class_embeddings = create_embeddings(class_descriptions)
```

Embedding item to classify

```
article = {"headline": "How NVIDIA GPUs Could Decide Who Wins the AI Race",
           "keywords": ["ai", "business", "computers"]}

def create_article_text(article):
  return f"""Headline: {article['headline']}
Keywords: {', '.join(article['keywords'])}"""

article_text = create_article_text(article)
article_embeddings = create_embeddings(article_text)[0]
```

Compute cosine distances

```python
def find_closest(query_vector, embeddings):
  distances = []
  for index, embedding in enumerate(embeddings):
    dist = distance.cosine(query_vector, embedding)
    distances.append({"distance": dist, "index": index})
  return min(distances, key=lambda x: x["distance"])

closest = find_closest(article_embeddings, class_embeddings)
```

Extract the most similar label

```
label = topics[closest['index']]['label']
print(label)
```

```
Business
```

```
article = {"headline": "How NVIDIA GPUs Could Decide Who Wins the AI Race",
           "keywords": ["ai", "business", "computers"]}
```

Limitation:

- Class descriptions lacked sufficient detail

More detailed descriptions

```
topics = [
  {'label': 'Tech', 'description': 'A news article about technology'},
  {'label': 'Science', 'description': 'A news article about science'},
  {'label': 'Sport', 'description': 'A news article about sports'},
  {'label': 'Business', 'description': 'A news article about business'},
]

class_descriptions = [topic['description'] for topic in topics]
class_embeddings = create_embeddings(class_descriptions)
[...]
label = topics[closest['index']]['label']
print(label)
```

```
Tech
```

Let's practice!

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API